1. GİRİŞ....3

2. TEMEL İŞLETİM SİSTEMLERİ KAVRAMLARINA GİRİŞ....4

2.1 İşletim Sistemi Nedir?....4
2.2 İşletim Sisteminin Temel Fonksiyonları....4
2.3 İşletim Sistemleri Kavramları....5
2.3.1 Süreçler....5
2.3.2 Dosyalar....5
2.3.3 Komut Yorumlayıcısı....5
2.3.4 Sistem Çağırımları....5
2.4 Çok Programlı İşletim Sistemleri....6
2.5 İşletim Sistemlerinde Süreç Yönetimi....7
2.6 İşletim Sistemlerinde Hafıza Yönetimi....8
2.6.1 Sayfalama Mekanizması....9

3. FAT12 DOSYA SİSTEMİ....11

3.1 Disklerin Yapısı....11
3.2 FAT12 Yapısı....12
3.2.1 Açılış Sektörü....13
3.2.2 Kök Dizini....14
3.2.3 FAT Bölgesi....14

4. İNTEL 386 AİLESİ MİMARİSİ VE KORUMALI MOD....18

4.1 Genel Mimari....18
4.2 Programlama Modeli....20
4.3 Çalışma Modları....21
4.4 Korumalı Mod Hafıza Yönetimi....21
4.5 Sistemdeki Kontrol Yazmaçları....23
4.6 Segmentasyon....24
4.6.1 Tanımlayıcı Tabloları ve Tanımlayıcılar....25
4.6.2 Tanımlayıcı girdileri....26
4.6.3 Segment Tanımlayıcıları (Kod veya Veri Segmenti Tanımlayıcıları)....27
4.6.4 Sistem Segment Tanımlayıcıları....29
4.6.5 Selektörler (Seçiciler)....30
4.7 Sayfalama....32
4.7.1 Sayfa Tabloları ve Sayfa Dizini....33
4.7.2 Sayfa Dizini ve Sayfa Tabloları Girdileri....34
4.8 Koruma Mekanizması....35
4.8.1 Limit Kontrolü....36
4.8.2 Tip Kontrolü....36
4.8.3 Ayrıcalık Düzeyleri....37
4.8.4 Kod Segmentlerini Operand Olarak Alan Jump Komutları....39
4.8.5 Kapı Tanımlayıcıları....39
4.8.6 Çağırım Kapıları....40

4.8.7 Yiğit Değişimi....42
4.8.8 Sayfa Seviyesinde Koruma....43
4.9 Kesme ve İstisna Yönetimi....44
4.9.1 Kesme Yönetiminde Yiğit Yapisi....46
4.9.2 İntel Mimarisindeki İstisnalar....47
4.10 Süreç Yönetimi....48
4.10.1 TSS Yapisi....49
4.10.2 TSS Tanimlayicisi....50
4.10.3 TR Yazmaçi....51
4.10.4 Görev Kapisi Tanimlayicilari....51
4.10.5 Süreçler Arasi Geçiş....52

5. GERÇEK ZAMANLI KORUMALI MOD 32 BİT BİR İŞLETİM SİSTEMİ GERÇEKLEŞTİRİMİ....54

5.1 İşletim Sistemi Açiliş Adimlari....55
5.1.1 İşletim Sisteminin Hafizaya Yüklenmesi (Boot.asm)....55
5.1.2 İşletim Sistemini İlkleme İşlemleri (Initsys.asm)....56
5.1.3 Ana Çekirdek İlklemeleri (Start.asm)....57
5.1.4 Çekirdeğe Giriş....57
5.2 İşletim Sistemi Temel Veri Yapilari ve Sabitleri....58
5.2.1 Tanimlayici ve Kapi Yapilari....58
5.2.2 Sayfa Tablolarini Doldurmak İçin Kullanilan Sabitler....60
5.2.3 Adres Sahasi Yapisi....62
5.2.4 Fiziksel Bellek Takipi İçin Kullanilan Yapi....62
5.2.5 Süreç Durum Sabitleri....63
5.2.6 Süreç Durumu Yapisi....63
5.2.7 Süreç Yapisi....64
5.2.8 Süreç Listeleri....65
5.2.9 Sistem Çağirimlari Tablosu....66
5.3 İşletim Sisteminde Süreç Yönetimi....67
5.3.1 Sistemdeki Listeler....67
5.3.2 İşlemci Dağiticisi (Scheduler)....68
5.3.3 Süreç Yaratilmasi ve Exec Fonksiyonu....69
5.3.4 Süreçlerin Sistemden Çikmasi ve Exit Fonksiyonu....72
5.3.5 Süreçlerin Kullanici Tarafindan Öldürülmesi ve killProcess....73
5.4 İşletim Sisteminde Hafiza Yönetimi....73
5.4.1 Fizikse Hafiza Takipi ve allocPages Fonksiyonu....74
5.4.2 Sistemdeki Sayfa Dizin Tablosu ve Kernel Sayfa Tablolari....75
5.4.3 Sayfa Tablolarina Girdi Eklemek -mapPages....77
5.4.4 Kullanici Adres Bölgesinden Bellek İsteminde Bulunma - allocUserPages....78
5.4.5 Kernel Adres Bölgesinden Bellek İsteminde Bulunma - allocKernelPages....79
5.4.6 Tahsis Edilmiş Bir Sayfayi Sisteme Geri Vermek – freePages....79
5.4.7 Süreçlere Ait Sayfa Tablolarinin Doldurulmasi - createPageTables....81
5.4.8 Süreçlere Ait Bellek Bölgelerinin Sisteme Geri Verilmesi – deleteProcess....82
5.5 İşletim Sisteminde Sistem Çağirimlari....83
5.5.1 Sistem Çağirimlari -SystemCall....83

6. SONUÇ....85

6.1 Gelecek Çalişmalar....85

1. GİRİŞ

Hazırlanan tezin gerçek amacı, intel 386 ailesi işlemcileri üzerinde gerçek anlamda çalışabilecek bir işletim sistemi yazımı idi. Bu sayede, geliştirilmeye ve büyümeye açık, ileride bilgisayar mühendisleri tarafından büyütülebilecek bir projenin başlangıcını oluşturulabilecekti.

Bu amaçların gerçekleştirilmesi için öncelikle temel işletim sistemleri kavramları hakkında hem pratik hem de teorik bilgiler toplandı. Gerçekleştirilecek işletim sistemine ait basit tasarımlar yapıldı ve işletim sisteminin fonksiyonları belirlendi.

Bu teorik araştırmalardan sonra, işletim sisteminin yazılacağı ortan olan intel 386 ailesi işlemcilerinin içsel yapısı, çalışma mekanizmaları ve programlanma modeli hakkında bilgi toplandı. Bunun için mikroişlemciler ve assembly dili ile ilgili kaynaklar da incelendi.

Sonraki adımda ise gerçek işletim sistemleri kaynak kodları incelendi. Başta linux olmak üzere çok sayıda işletim sistemi kaynak kodu, hafıza yönetimi ve süreç yönetimi konularında incelendi. Özellikle bu konudaki kaynak eksikliği, tezin bu kısmının hayli uzun olmasına yol açmıştır.

Tüm bu incelemelerden sonra kod yazımına geçilmiştir. Öncelikle C ve Assembly dilleri arasındaki ilişki, C kodundan assembly kodu çağırmak ve assembly kodundan C kodu çağırmak gibi kavramlar incelendi. AT&T assembly kodlama stili hakkında bilgi toplandı. GCC , LD ve NASM gibi GNU lisansına sahip ücretsiz yazılımların işletim sistemi yazımında nasıl kullanıldığı hakkında bilgi toplanıldı. Son olarak kodlanan işletim sistemi, değişik test aşamalarından geçirilmiştir.

Raporda, incelenen tüm konular hakkında toplanan bilgiler yer almaktadır.Çalışma sırasında çok zaman kaybına yol açan ayrıntılar bu raporda ayrıntılı açıklanmıştır. Dolayısıyla, ilerdeki zaman kayıpları engellenecektir. İçerik olarak sırası ile temel işletim kavramları, işletim sistemi yüklenmesinde kullanılan FAT12 dosya sistemi, intel korumalı mod mimarisi raporda yer almaktadır.

2. TEMEL İŞLETİM SİSTEMLERİ KAVRAMLARINA GİRİŞ

2.1 İşletim Sistemi Nedir?

Bilgisayar sistemini kullanan kullanıcılar ile sistem kapsamındaki donanım aygıtları arasındaki **arayüz işlevini gören temel program**, işletim sistemidir. İşletim sistemleri donanıma yönelik alt seviyeli işlemlerden kullanıcıları soyutlar. Örneğin; kullanıcıların disk donanımına erişmesini, mantıksal dosya isimleri ile (donanım ayrıntısını gizleyerek) sağlar. Donanım kesmelerini, zamanlayıcıları, hafıza ve işlemci gibi hayati birimleri yöneten; işletim sistemidir. Bu kapsamda, kullanıcıların gözünde işletim sistemi , donanım programlamadan daha kolay programlanabilen ve donanımın sağladığı servisleri sağlayabilen bir **sanal makina** olarak düşünülebilir.

İşletim sistemi, soyutlama işleminin yanısıra, **sistemdeki kaynakların da dağıtıcısıdır**. Sistemdeki hafıza, işlemci, giriş/çıkış cihazları gibi birimlerin, süreçler arasındaki paylaşımını sağlayar. Kaynakların hangi süreçler tarafından kullanıldığının takipini yapar, aynı anda sistemde bulunan bir kaynağı isteyen süreçlerin isteklerinin düzenlenmesini ve planlamasını yerine getirir.

İşletim sistemi, bir **servis sağlayıcısıdır**. Süreçler , işletim sisteminin sağladığı servisleri kullanarak, sistem içerisindeki varlıklarını sürdürmektedirler. Sistemdeki kaynaklara ulaşma ancak işletim sisteminin sağladığı servisler sayesinde olmaktadır.

2.2 İşletim Sisteminin Temel Fonksiyonları

İşletim sistemlerinin sağladığı temel fonksiyonlar şöyle sıralanabilir:

- hafıza yönetimi
- süreç yönetimi
- dosya sistemi
- ağ yönetimi
- giriş/çıkış aygıtı yönetimi
- koruma ve güvenlik

Bir işletim sistemi, kullanıcının hatalı işlem yapmasını engellemek için, tüm giriş/çıkış işlemlerini üstlenmiştir. Kullanıcı süreçlerinin , diğer süreçlerin adres sahası içinde olan bölgelere yazma ve okuma işlemleri, işletim sisteminin kontrolü altında olmalıdır. Örneğin, sistem için hayati önem taşıyan kesme vektörü ve kesme servisleri, kesinlikle korunmalıdır. Sistemdeki tüm süreçlerin yaratılışı, hayat döngüsü ve sonlanışı, tamamiyle işletim sisteminin denetimindedir. İşlemcinin bu süreçler arasında paylaşımı, işletim sisteminin en önemli fonksiyonudur.

2.3 İşletim Sistemleri Kavramları

2.3.1 Süreçler

Süreç, diskten okunarak hafızaya yüklenmiş, işletim sistemi tarafından gerekli veri yapıları yaratılmış ve bu sayede sistem içerisindeki aktivitelerinin takipinin yapılabildiği, aktif programa denmektedir.

2.3.2 Dosyalar

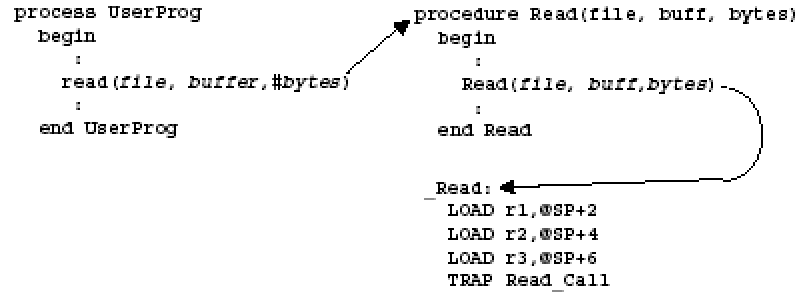
İşletim sistemleri, disk ve giriş/çıkış cihazlarının kendilerine ait özelliklerini kullanıcılarıdan saklayarak, onlara cihazlardan bağımsız bir dosya sistemi sunar. Dosya sistemleri sayesinde, dosyaların takipi yapılır, diske okuma ve yazma işlemleri, mantıksal dosyalar üzerinden gerçekleşir.

2.3.3 Komut Yorumlayıcısı

İşletim sisteminin bir parçası olmamasına rağmen, işletim sisteminin özellikleri ve sağladığı servisler ile terminal başındaki işletim sistemi kullanıcıları arasında bir arayüzdür. Temel işletim sistemi servislerinin çağırılmasını sağlar.

2.3.4 Sistem Çağırımları

Bir sürecin, işletim sisteminin sunduğu bir servisi kullanabilmesi, sistem çağırımı yardımı ile olmaktadır.



Şekil 2.1 : Bir Kullanıcı Süreci ve Sistem Çağırımları

Kullanıcı programları, işletim sistemini bir servis sağlayıcı olarak gördüğü için; servislerden faydalanabilmek ancak sistem çağırımları yardımı ile olmaktadır. Sistem çağırımı yapıldığı anda, işletim sistemi çalışmayı o anki programdan alır ve gerekli servisi sunduktan sonra ; istemde bulunan program tekrar kaldığı yerden çalışmaya başlar.

2.4 Çok Programlı İşletim Sistemleri

Tek programlı işletim sistemlerinde, sistem ancak tek bir programın aktif olmasına izin vermektedir. Program, çalışması süresince hafızada kalmaktadır. Bazı sistemler yer değiştirme (swapping) mekanizmasını uygulamaktadır. Bu sistemlerde, sistem hafızada aktif bir programı ve disk üzerinde birden çok programı barındırır. O an çalışmakta olan program giriş/çıkış işlemi yapmak isterse, o program hafızadan diske alınır ve diğer bir program hafızaya yüklenir.

Çok programlı işletim sistemlerinde ise , işletim sistemi hafızada birden fazla süreci barındırır ve işlemci, giriş/çıkış aygıtları vb. kaynakları süreçlere paylaştırır. Zaman paylaşımlı işletim sistemleri, çok programlı işletim sistemlerinin bir çeşitidir. Burada işletim sistemi, hafızada aktif olan her sürecin işlemciyi belirli bir süre kullanmasına izin verir. Böylelikle, kullanıcılar birden fazla sürecin aynı anda çalıştığını zannederler.

Eğer işletim sistemi bir sürece işlemciyi verdikten sonra o süreç işlemciyi bırakıncaya kadar çalışmasına izizn veriyorsa; buna işlemciyi ele geçirmeyen (non preemptive) işletim sistemleri denmektedir. Eğer işletim sistemi o süreçten belirli bir süre sonra işlemciyi alıyorsa, buna işlemciyi ele geçiren (preemptive) işletim sistemleri denmektedir.

Çok programlı işletim sistemlerinde kullanılan temel işlemci dağıtım algoritmaları FCFS (ilk gelene ilk servisi ver) ve Round Robin algoritmalarıdır.

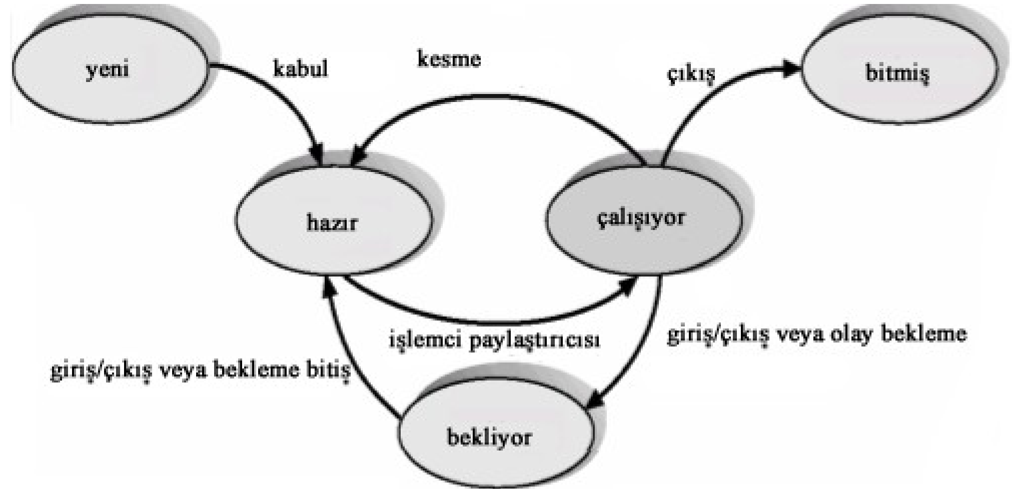
2.5 İşletim Sistemlerinde Süreç Yönetimi

Her süreç, bir adres sahasına sahiptir. Adres sahası kavramı, o sürece tahsis edilmiş, o sürecin okuyup yazabileceği bellek bölgelerini göstermektedir. Adres sahası, o sürece ait kod, veri ve yığıt bölümlerini içermektedir.

Ayrıca o sürece ait durumun saklanabilmesi için, sürece ait yazmaçların (IP, SS, DS, CS ...) ve diğer bilgilerin tutulduğu bir veri yapısı , işletim sistemi tarafından, süreç yaratıldığı , tahsis edilmelidir. Böylelikle, eğer o süreç , askıya alınırsa, sürecin tüm bilgileri kaydedilecek; daha sonra tekrar aktif hale geçtiğinde, bu bilgiler kullanılarak sürecin çalışmasına kalınan yerden devam edilecektir.

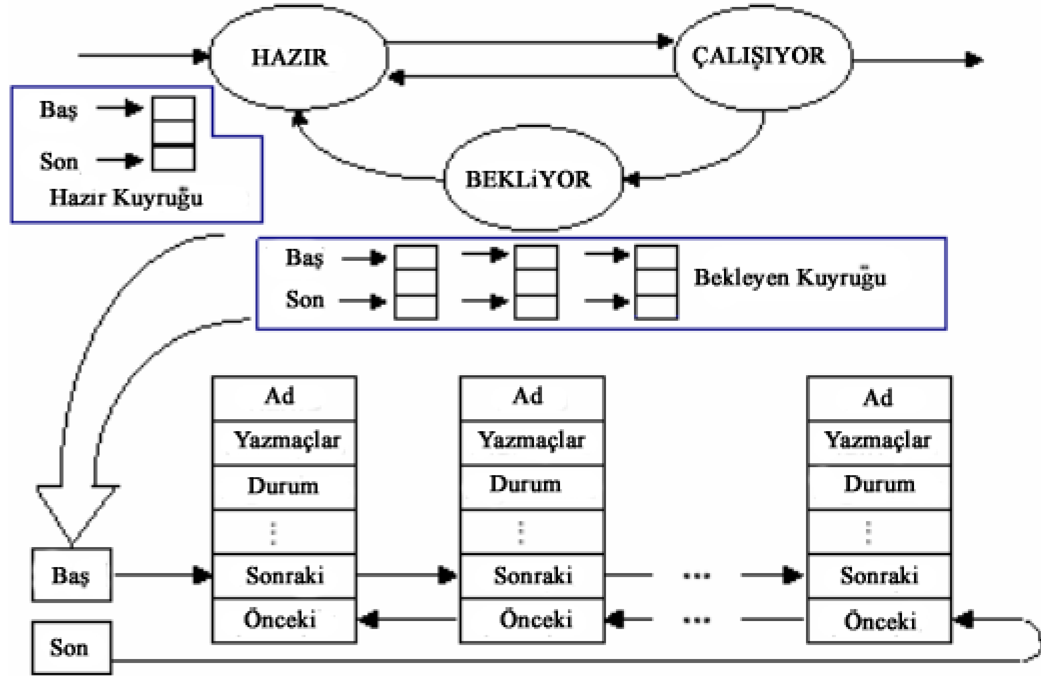
İşletim sistemi, bir sürecin diğer sürecin adres sahasına erişimesini kesinlikle denetlemeli ve uygun olmayan erişimlere izin vermemelidir.

Süreçler sonlanıncaya kadar bir durumdan diğer duruma geçerler. Dolayısıyla bilgileri de bu değişiklikler ile değişmektedir.



Şekil 2.2 : Süreçler ve Yaşam Döngüleri

Süreçlerin durum değişiklikleri ve bu değişimlerde bilgilerinin tutulması işletim sistemi içerisindeki süreç tabloları ile olmaktadır. Süreç tabloları ise bir süreç listesi ile gerçekleştirilebilir.



Şekil 2.3: Süreç Listeleri

Bir işletim sistemi, süreç yönetimi kapsamında

- Süreç yaratılması ve sistemden silinmesi için servisler
- Süreçlerin durdurulması ve tekrar çalıştırılabilmesi için servisler
- Süreçler arası iletişim için servisler
- Süreçlerin senkronizasyonu için servisler

gibi servisleri sağlamalıdır.

2.6 İşletim Sistemlerinde Hafıza Yönetimi

Hafıza yönetimi kapsamında işletim sistemleri hafızanın takipini yapmak, hafızanın hangi bölümleri kullanılıyor, hangi bölümleri boş gibi işlemleri yerine getirir. Ayrıca süreçlere istemde bulundukları anda bellek atamak ve atanmış belleği

sisteme geri vermek en önemli görevdir. Hafıza ile disk arasında süreç taşınması işlemini yapmak böylelikle bir sürecin çalışabilmesi için yeterli belleği sağlamak da hafıza yönetimi kapsamına girmektedir.

Hafıza yönetimi ile, süreçlerin birbirlerinin adres sahasına veri yazmaları önlenmiş olur. Süreçlerin ortak bellek bölgeleri üzerinde işlem yapmaları, hafıza yönetim biriminin kontrolü sayesinde olmaktadır. Ayrıca süreçlerin diskten okunup fiziksel belleğe yüklenmesi yine hafıza yönetim birimi ile olmaktadır.

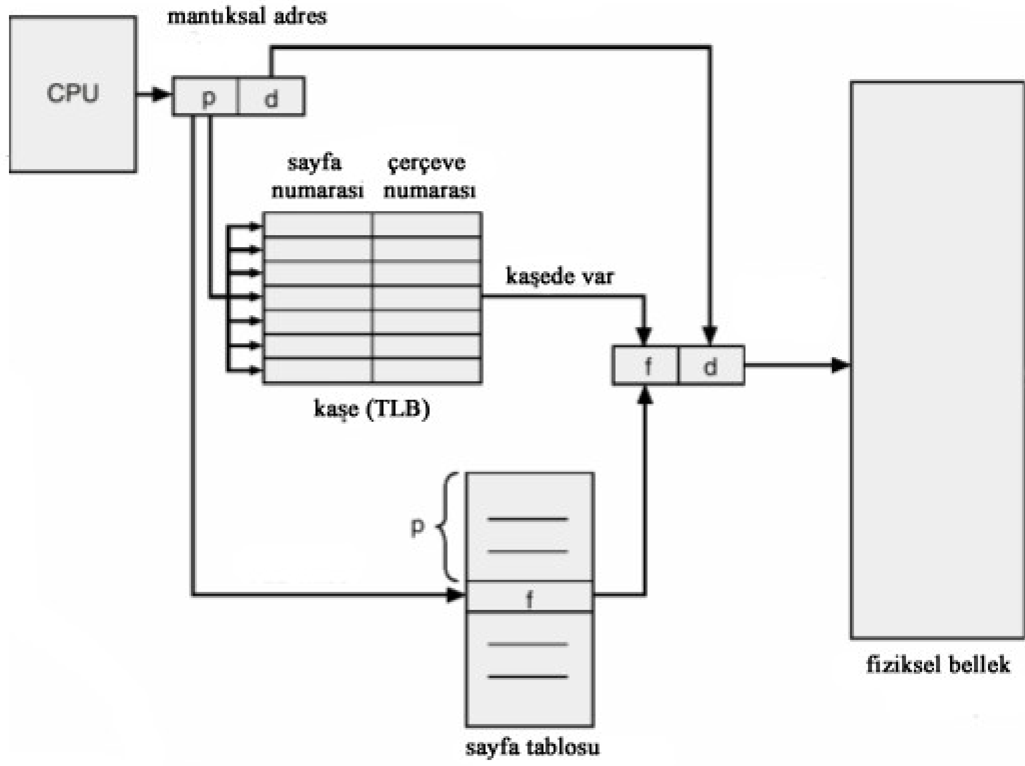
Hafıza yönetimi sayesinde, süreçlerin her seferinde aynı fiziksel adrese yüklenmesi sorunu ortadan kalkmış olmaktadır.

2.6.1 Sayfalama Mekanizması

Sistemde sürece yetecek kadar bellek varsa ancak bu bellek sahası sürekli değilse, bu bellek sahası o sürece atanamaz. Bu sorun, sayfalama mekanizması ile çözülmüştür. Sayfalama ile, fiziksel hafıza eşit ve sabit uzunluklu bloklara bölünmüştür. Bunlara çerçeve denir. Mantıksal hafıza da aynı uzunluklu parçalara bölünmüştür. Bunlara ise sayfa denmektedir.

Sayfa tablosu, o sürece ait mantıksal adres sahası ile ona karşılık gelen fiziksel adres dönüşümünü yapabilmek için kullanılır.Genellikle, sistemlerde her sürece ait bir sayfa tablosu vardır. İşletim sistemleri her sürece ait bir sayfa tablosu tuttuğu için, bu tabloyu da süreç yapıları içerisinde tutmalıdır. Bu sebeple , sayfalama süreçler arası geçiş işlemine ek yük getirmektedir.

İşlemci tarafından üretilen adresin 2 parçası vardır. Bunlar sayfa numarası ve ofsettir.



Şekil 2.4: Sayfalama Mekanizması

Sisteme bir süreç girdiğinde, uzunluğu sayfa sayısı olarak belirlenir. Eğer süreç n adet sayfadan oluşuyor ise, n adet boş çerçeve bulunmalıdır. Bulunan bu boş çerçeveler, sürecin sayfa tablosuna koyulur.

Sayfa tablosu girdilerinde, bir bit koruma biti olarak bulunur. Bu sayede o hafıza bölgesine okuma / yazma işlemleri denetlenir. Böylelikle, süreç kendi sayfa tablosu girdilerinden başka bellek bölgesine erişemez. Koruma sağlanmış olur.

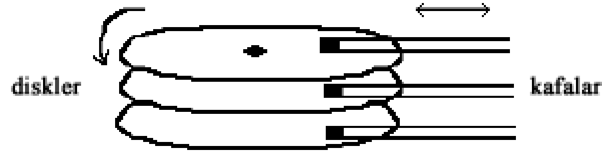
İşletim sistemleri hangi çerçevelerin tahsis edilip hangilerinin edilmediğini tutmak için çerçeve tablosu tutmaktadır.

3. FAT12 DOSYA SİSTEMİ

Dosya Ayırım Tabloları (FAT), MS-DOS işletim sisteminin disklerdeki fiziksel verilere ulaşmak için kullandığı basit bir bağlı liste yapısıdır. Disketlerde kullanılan dosya sistemi ise FAT12 adıyla isimlendirilmektedir. FAT12 en fazla 8 MB veriyi adresleyebilmektedir.

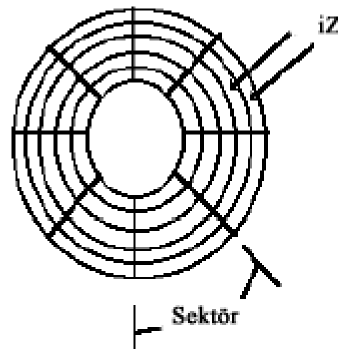
3.1 Disklerin Yapısı

Disklerde bulunan okuma/yazma kafaları, verinin diske yazılıp okunmasını sağlayan temel mekanizmalardır. Diskler kendi eksenlerinde dönerken, okuma/yazma kafaları ileri geri hareket etmektedir.



Şekil 3.1 : Disklerin Fiziksel Yapısı

Diskler birbirine komşu dairelere bölünmüştür ve bunlara iz (track) denmektedir. İzler ise eşit açılı dilimlere bölünmüştür ve bunlara ise sektör (sector) denmektedir.



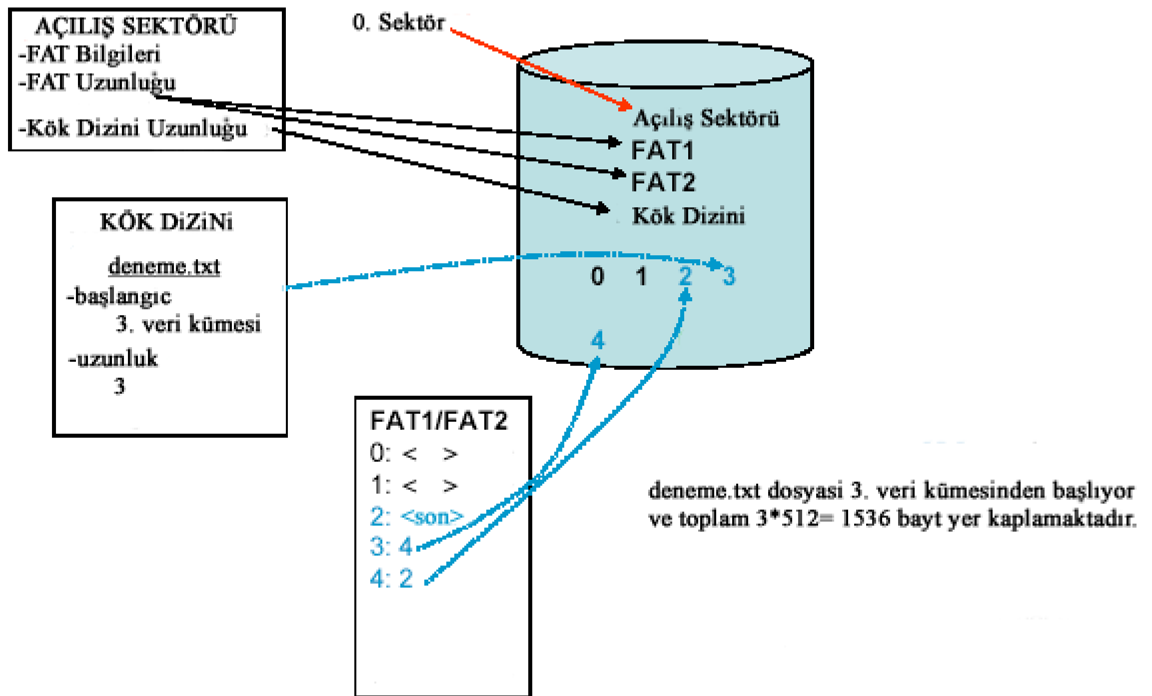
Şekil 3.2 : Disklerin Mantıksal Yapısı

Sektörlerin bir araya gelmesi ile oluşan sektör grubuna iseveri kümesi (cluster) denmektedir. Diğer bir tanımla, yazma ve okuma işlemlerinde üzerinde işlem yapılacak en küçük veri grubuna veri kümesi denmektedir.Veri kümelerin

içerdiği sektör sayısı işletim sisteminden işletim sistemine değişmektedir. Örneğin FAT 12 ‘de bir veri kümesi bir sektör içermektedir.

3.2 FAT12 Yapısı

FAT12 de disk üzerinde bazı sektörler dosya sistemi tarafından özel amaçlarla kullanılmaktadır. Bunlardan 0. sektörde açılış (boot) sektörü bulunur.Açılış sektörü FAT tabloları hakkında bilgi içerir. Açılış sektörünü FAT1 ve FAT2 dosya sistemi tabloları takip etmektedir. Bu tablolar sistemdeki veri kümelerini bir bağlı liste ile tutmaktadırlar.FAT2’den sonra ise Kök Dizini (Root Directory) gelmektedir. Kök dizini ise dosya adlarını, dosyaların başlangıç veri kümelerini ve uzunlukları tutmaktadır.



Şekil 3.3 : FAT12 Disk Bölgeleri

Görüldüğü gibi, sistemde 4 önemli bölge vardır. Bu bölgeler ayrılmış olan sektörler aşağıdaki şekilde belirtilmiştir.

SEKTÖR	ADRES	İÇERİK
0	0x0000-0x01ff	Açılış Sektörü
1-9	0x0200-0x13ff	FAT1 (Ana Tablo)
10-18	0x1400-0x25ff	FAT2 (Yedek Tablo)
19-32	0x2600-0x41ff	Kök Dizini
33-2879	0x4200-0x167fff	Dosya Veri Bölgesi

Şekil 3.4 : FAT12 Sektör İçerikleri

FAT12 ‘de her bir FAT girdisi için 12 bit ayrılmıştır. Dolayısıyla 4096 adet veri kümesi adreslenebilir. Disket sürücülerde bir veri kümesi bir sektöre eşit olup, sektör de 512 bayttır. Dolayısıyla 2880 sektör yani 1.44 MB veri disketlerde tutulabilir.

3.2.1 Açılış Sektörü

Açılış sektörü, dosya sisteminin kaydedildiği diskin fiziksel bilgilerini tutar ve ona erişimi sağlar. Açılış sekörü, diskin 0. sektörüdür ve işletim sisteminin yüklenmesinde de önemli rol oynamaktadır.

Çizelge 3.1: FAT12 Açılış Sektörü Yapısı

Adres	Adresteki içerik	Uzunluk (Byte)
00h	Atlanacak boot fonksiyonu	3
03h	Üretici Bilgisi	8
0Bh	Bir sektörün kapladığı bayt	2
0Dh	Bir veri kümesinin kapladığı sektör	1
0Eh	Ayrılmış sektör sayısı	2
10h	FAT tablosu sayısı	1
11h	Kök dizinindeki girdi sayısı	2
13h	Diskteki sektör sayısı	2
15h	Disket tanımlayıcı bilgisi	1
16h	FAT başına düşen sektör sayısı	2
18h	İz başına düşen sektör	2
1Ah	Okuma/yazma kafası sayısı	2
1Ch	Saklı sektör sayısı	2
1Eh	Boot fonksiyonu	Variable

512 bayt uzunluğunda olan açılış sektörünün önemli sahalrından biri atlanacak açılış fonksiyonuna ait adresin tutulduğu bölgedir. Bu bölgede gösterilen adrese atlanarak,işletim sistemi yükleme işlemine başlanmış olunur.

3.2.2 Kök Dizini

Kök dizini 19. sektörden başlar. Dosyaların ve alt dizinlerin bilgilerini tutar. Kök dizinindeki her dosya veya alt dizin girdisi 32 baytlık uzunluğa sahiptir. Bu girdiler yardımı ile dosyanın verilerine ulaşılır.

Çizelge 3.2: Kök Dizini Girdileri

BYTE	DOSYA ADI [8]
BYTE	UZANTI [3]
BYTE	ÖZELLiKLER
BYTE	AYRILMIS [10]
WORD	ZAMAN
WORD	TARiH
WORD	VERi KÜMESi (BASLANGiC)
DWORD	BÜYÜKLÜK

Burada önemli noktalardan birisi dosya isminin ilk karakteridir.

Çizelge 3.3 : Dosya İsimlerinin İlk Karakterlerinin Anlamları

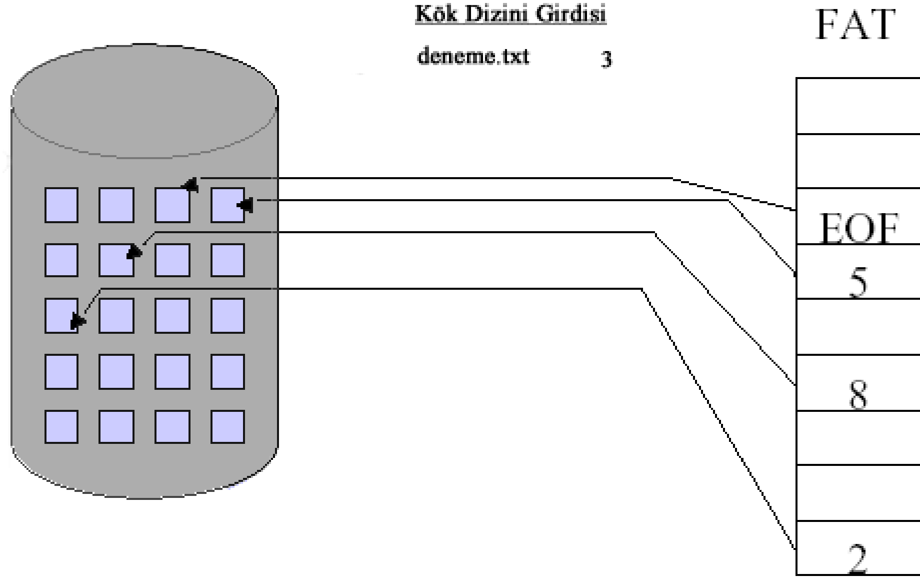
Kod	Anlam
00h	Dizindeki son girdi
05h	Dosya adının ilk karakteri E5h ASCII değerine sahip
E5h	Dosya silinmiş

Eğer bu karakter 0x00 ise o girdi kullanılmamaktadır. Eğer 0xE5 ise, o dosya silinmiştir.

3.2.3 FAT Bölgesi

FAT dosyalara ait veri kümelerinin bağlı bir liste gibi tutulduğu dosya sistemi bölgesidir. Sistemde 2 adet FAT vardır. FAT1 ana dosya tablosudur ve FAT2 ise yedek tablo olarak kullanılmaktadır.

Bir dosyaya ulaşılırken önce kök dizininden o dosyaya ait ilk veri kümesi bulunur. Daha sonra o veri kümesindeki değer kullanılarak FAT tablosundaki sonraki veri kümelerine ulaşılır.



Şekil 3.5: FAT Kullanılarak Dosyalara Erişim

FAT her veri kümesi için 12 bit yer ayırmıştır. Yani, 2 FAT girdisi toplam 3 bayt yer kaplamaktadır. İlk iki FAT giridisi kullanılmamaktadır. FAT girdilerinin özel anlamları vardır.

Çizelge 3.4 : Veri Kümesi Değerleri

0	Kullanılmayan Veri Kümesi
0xFF0-0xFF6	Ayrılmış veri kümesi
0xFF7	Bozuk veri kümesi
0xFF8-0xFFF	Veri kümesi zincirinin sonu (EOF)
Other	Dosyaya ait diğer veri kümesi

Eğer girdi 0 ise o veri kümesi kullanılmıyor demektir. Eğer 0xFFF ise , o dosyaya ait son veri kümesidir.

Kök dizininden başlangıç veri kümesi numarası bulunduktan sonra, o veri kümesi numarası FAT tablosundaki bir girdi indeksine çevirilmelidir. Bu çevirim şöyle gerçekleştirilir:

FAT'teki indeks = (veri kümesi numarası) * 3/2 + 1

Her bir girdi 12 bit yani 1.5 bayt yer kapladığı için, 1.5 ile çarpıp 1 eklediğimiz zaman FAT girdi numarasını bulmuş oluruz.

2 FAT girdisi 3 bayt yer kapladığı için , örneğin xyz ve XYZ içeriğinde iki adet veri kümesi girdimiz var ise, bunlar FAT tablosuna şu şekilde yerleştirilmişlerdir:

YZ-**z**X-**xy**

Şekil 3.6 : FAT Tablosundaki Girdi Formatı

Dolayısıyla, FAT girdisi bulunduktan sonra, doğru veriyi elde etmek için şu algoritma kullanılır:

```
sektor = 0

// mantıksal veri kümesi numarası, sektör değerine çevirilir.
FatIndeks= (mantıksal veri kümesi numarası * 3 ) / 2

Eğer ( mantıksal veri kümesi numarası çift ise)
{
       sektor=FAT[FatIndeks + 1] & 0x0f
       sektor=sektor <<8
       sektor=sektor|FAT[FatIndeks]
}
Değilse
{
      sektor=FAT[FatIndeks + 1]
      sektor=sektor <<4
      sektor=sektor| ((FAT[FatIndeks ] & 0xf0) >>4
}
```

Şekil 3.7: FAT Tablosu Kullanılarak Okunacak Diğer Sektörün Belirlenmesi

Algoritmada, önce veri kümesi değeri uygun FAT girdisine çevirilmekte sonra, o girdideki değer; veri kümesi numarasının çift olup olmamasına göre değerlendirilmektedir.

FAT'ten mantıksal veri kümesi numaraları alındıktan sonra, bunların gerçek sektör numaralarına çeviriminin yapılması gerekmektedir.

İlk iki FAT girdisi kullanılmadığı için veri kümesi numarasından 2 çıkartılır, elde edilen sayı veri kümesi başına düşen sektör sayısı ile çarpılır. Elde edilen değer ise, dosya verilerinin başladığı ilk sektör numarası ile toplanır.

FAT12'de 1 veri kümesi 1 sektöre eşit olduğundan ve ilk veri sektörü 33 olduğundan (açılış sektörü (1) + FAT1 (9) + FAT2(9) + Kök Dizini(14)); elde edilen veri kümesi numarasını 31 ile toplamak yeterlidir.

4. İNTEL 386 AİLESİ MİMARİSİ VE KORUMALI MOD

İntel 32 bit işlemci mimarisi,

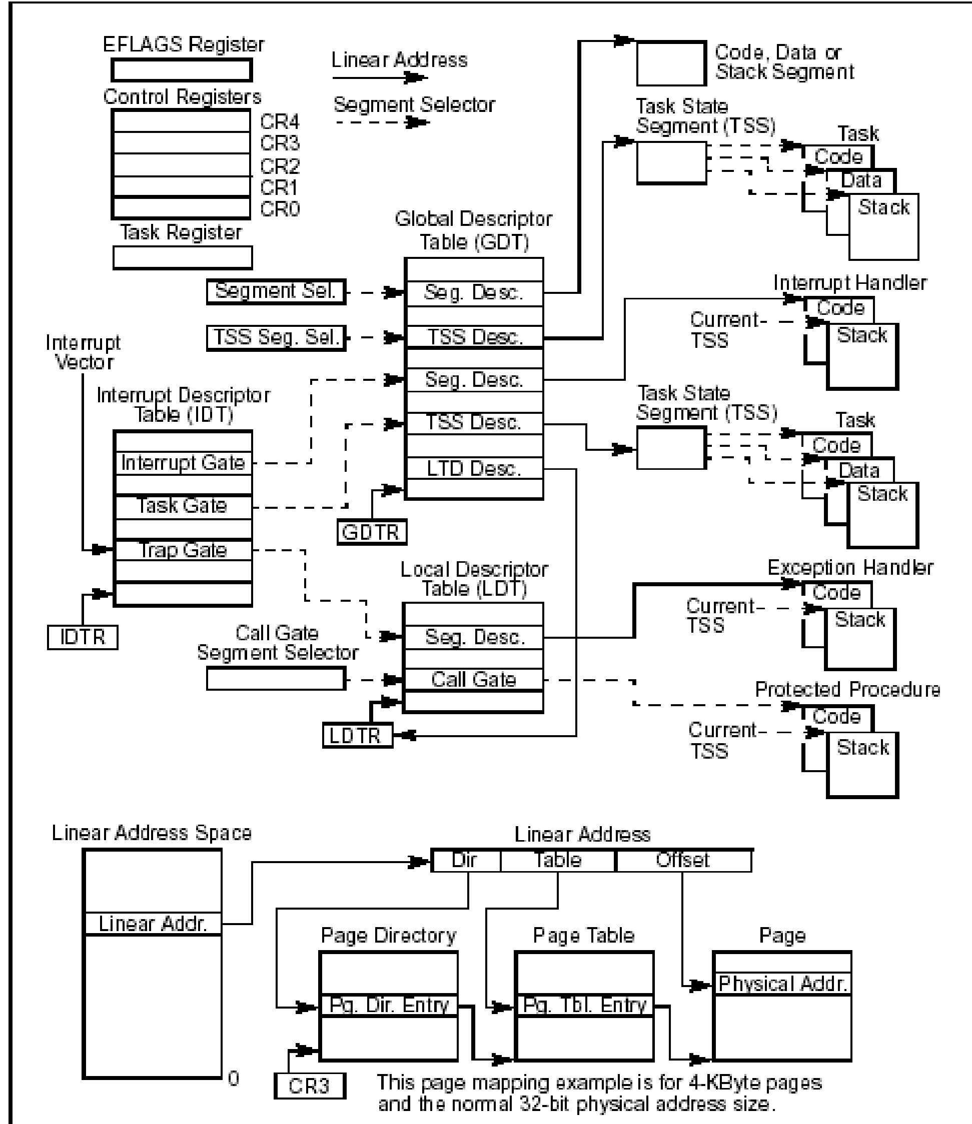
- Hafıza yönetimi
- Koruma mekanizması
- Çoklu Programlama
- Kesme yönetimi
- Kaşe yönetimi

gibi önemli fonksiyonlar içermektedir. Bu fonksiyonların nasıl gerçekleştirildiği ve yönetimi için sistemin genel mimarisine bakmak faydalı olacaktır.

4.1 Genel Mimari

İntel 386 ailesi işlemci mimarisi, aşağıdaki şekildeki gibi özetlenebilir. Mimari, görüldüğü gibi, bazı özel sistem yazmaçlarını (LDTR,GDTR,IDTR) içermektedir. Bu yazmaçlar, sistem için hayati önem taşıyan tablolara işaretçidirler. Ayrıca, mimari bazı kontrol yazmaçlarını da (CR0,CR1,CR2,CR3,CR4) içermektedir. Bunlar, sistemin işleyişi hakkında bilgiler veren ve içerdiği bitler sayesinde, sistemin çalışmasını etkileyen önemli yazmaçlardır.

Sistem yazmaçları , sadece işletim sistemi gibi düşük önceliğe sahip programlar tarafından kullanılabilir. Yani kullanıcı süreçlere görünür değillerdir. Bu, sistemin kontrolünün sistem programlarının daha fazla elinde olmasını sağlamaktadır.



Şekil 4.1: İntel 386 Ailesi Mimarisi

Sistemin genel mimarisinde görüleceği gibi, tüm hafıza erişimleri GDT ve LDT isimli tablolar üzerinden olmaktadır.

Kapılar (call gate, interrupt gate, trap gate, task gate), değişik ayrıcalık düzeyinde bulunan programların , sistem yönetim fonksiyonlarını korumalı ve güvenli bir şekilde kullanabilmesini sağlamaktadır.

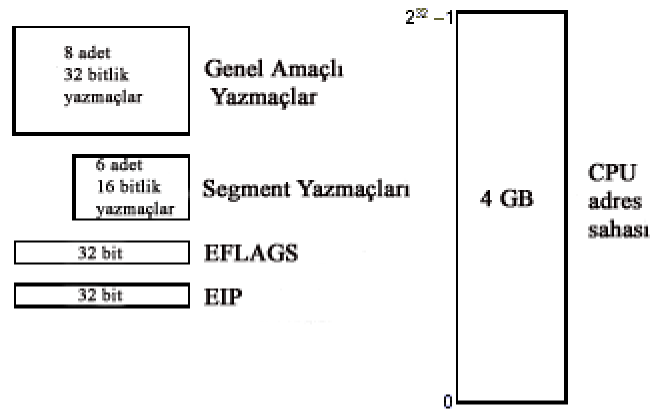
TSS'ler (Task State Segment), bir programın çalışma sahasını tanımlamaktadır. O programa ait yazmaçlar ve bilgiler TSS'lerde saklanır.

Sistemdeki kesme mekanizması IDT (Interrupt Descriptor Table) kesme tablosu üzerinden çalışmaktadır. 256 adet tanımlayıcı içeren bir tablodur.

GDTR yazmaçı GDT tablosuna, LDTR yazmaçı LDT tablosuna ve IDTR yazmaçı IDT tablosuna işaretçidir. TR yazmaçı o an çalışmakta olan programın TSS bölgesine işaretçidir.

Sistemin genel mimarisine baktıktan sonra, gerçek anlamda işleyişine ve hafıza modeline ayrıntılı olarak bakmak, ortamın tam olarak anlaşılmasını sağlayacaktır.

4.2 Programlama Modeli



Şekil 4.2 : İntel 386 Ailesi Programlama Modeli

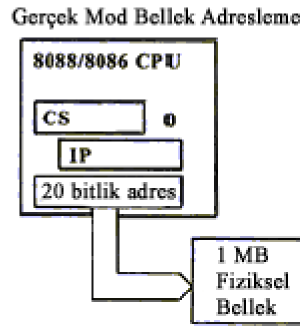
İşlemcinin genel amaçlı yazmaçları 32 bit genişliğindedir. Bu yazmaçlar, 8086 ailesindeki yazmaçlarla aynı isme sahiplerdir. Tek farkları, önlerine “E” harfi getirilerek adlandırılmışlardır. Bu yazmçlar sırası ile EAX , EBX , ECX , EDX , ESI , EDI , EBP, ESP ‘dir .

Sistem, ayrıca 16 bitlik segment yazmaçları da içermektedir. Bu yazmaçlar yine 8086 mimarisindeki yazmaçlarla aynıdır. Yani CS,DS,ES,SS yazmaçları bu mimaride de vardır. Bu yazmaçlara ek olarak, FS ve GS isimli iki yazmaç daha ilave edilmiştir.

İşlemcinin komut kümesi de 32 bittir. Dolayısıyla 8086 mimarisindeki IP yazmaçı 32 bite çıkmış ve EIP olarak adlandırılmaktadır. Hafıza yolu da 32 bitlik olduğu için, işlemci 4 GB fiziksel belleği adresleyebilir.

4.3 Çalışma Modları

İntel işlemcisi, gerçek mod ve korumalı mod olmak üzere iki temel işletim moduna sahiptir. Gerçek modda, işlemcimiz hızlı bir 8086 işlemcisinden farklı değildir. Sadece 1 MB'lık bellek bölgesini adresleyebilir, dolayısyla bellek adresleri 20 bitlik adreslerdir.. Tüm bellek erişimleri bir segment yazmaçı ile yapılmaktadır. Segment yazmaçı, adresin 16 bitlik kısmını tutmaktadır. Bu adres değeri 4 bit sola kaydırılıp bir ofset değeri ile toplanınca, gerçek fiziksel adres elde edilir. Aşağıda , örnek bir hafıza erişim işlemi gösterilmiştir.



Şekil 4.3 : Gerçek Mod Bellek Erişimi

Korumalı modda ise, hafıza erişimleri, GDT ve LDT tabloları üzerinden olmaktadır. Eğer sayfalama mekanizması aktiflenirse, sayfa tabloları da fiziksel hafızaya erişimde bir diğer basamak olmaktadır.

Sistem açıldığında , işlemci gerçek moda girmektedir. CR0 yazmaçının içindeki bir bit 1'lendiği zaman; işlemci korumalı moda geçmektedir.

Bunların dışında, bir de sanal 8086 modu (Virtual 8086 Mode) , 8086 programlarını çalıştırmak için işlemci yapısında vardır.

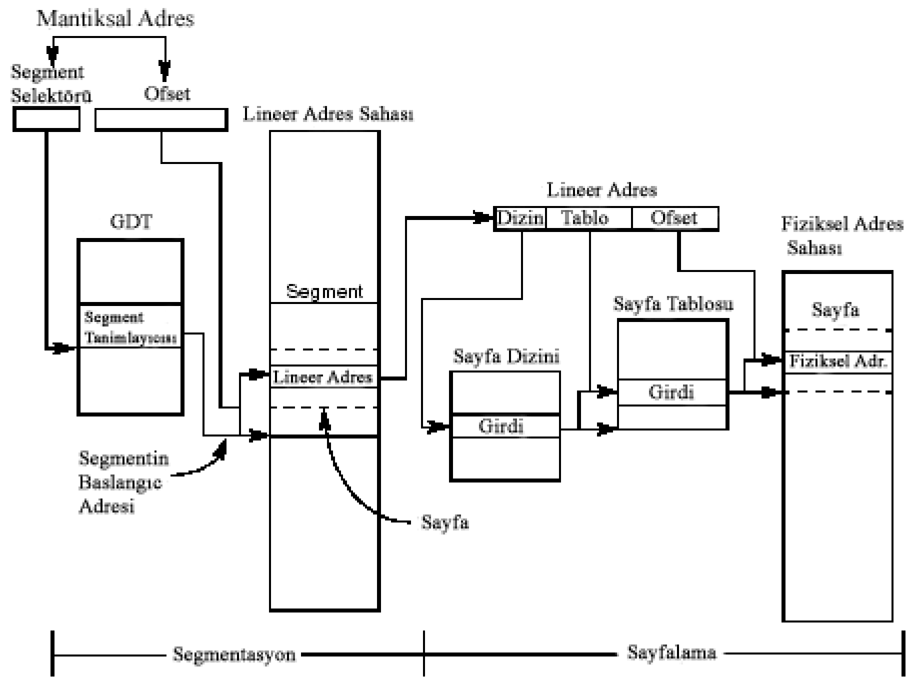
4.4 Korumalı Mod Hafıza Yönetimi

Korumalı modda hafıza erişimi sayfalama ve segmentasyon mekanizmaları ile sağlanmaktadır.

Segmentasyon ile her programın kendi kod, veri ve yığıt bölgeleri diğer programlardan soyutlanmaktadır. Böylece birden fazla program, aynı işlemci üzerinde, birbirlerinden bağımsız olarak çalışabilmektedir.

Sayfalama mekanizması ile, süreçlere bir sanal bellek servisi sunulmaktadır. Böylelikle, programın çalışma süresi içerisinde, adres sahası istenildiği anda fiziksel bellekle eşleştirilir.

İşlemci çalışmaya başlandığı anda, segmentasyon mekanizması zaten aktiftir ve sistem bizlere bu mekanizmayı kapatmak için bir servis sunmamaktadır. Ancak, sayfalama mekanizması, işlemci çalışmaya başladığı anda aktif değildir. Bu mekanizma, sistemde bulunan kontrol yazmaçları sayesinde aktif hale getirilmektedir. Sayfalama mekanizması, işlemci gerçek modda çalışıyorken de aktif hale getirilebilir. Yani, korumalı moda özgü bir mekanizma değildir.



Şekil 4.4 : Adres Dönüşümleri

İntel 386 ailesi işlemcileri için, genel belleğe erişim görüntüsü şekildeki gibidir. Sürecin ürettiği adrese , mantıksal adres denmektedir. Bu adres 16 bitlik bir selektör ile 32 bitlik bir ofset değerini içermektedir. Mantıksal adres, segmentasyon mekanizmasından geçer. Gerekli tablolardan girdilere ulaşılır ve segment bilgileri sonucunda, lineer adres üretilir. Lineer adres, sayfalama mekanizmasından geçecek adrestir. Bu adres ise 32 bitlik bir tamsayı değeridir. Eğer sayfalama mekanizması aktif değilse, lineer adresler fiziksel adreslere eşittir. Eğer sayfalama mekanizması

aktif ise, lineer adresler, gerekli sayfa tabloları yardımı ile gerçek fiziksel adreslere dönüştürülür. Yani adres dönüşümü, aşağıdaki gibi özetlenebilir.

Şekil 4.5 : İntel Mimarisindeki Adres Geçişleri

4.5 Sistemdeki Kontrol Yazmaçları

Bellek yönetiminden önce, sistemdeki kontrol yazmaçlarını incelemek gerekmektedir.

Kontrol yazmaçları, işlemcinin hangi işletim modunda çalıştığını ve o an çalışan süreçin karakteristiğini belirlerler. Sistemde 5 adet kontrol yazmaçı vardır.

CR0 işlemcinin işletim modunu ve işlemcinin durumunu içeren yazmaçtır.

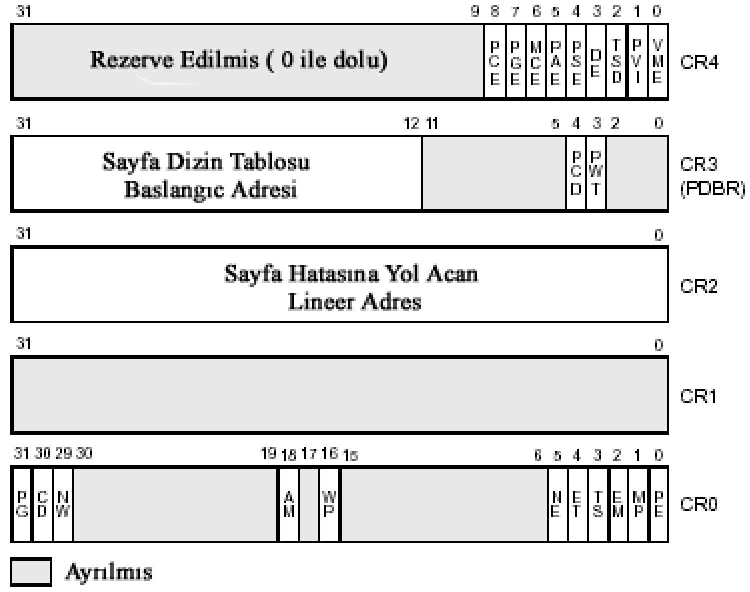
CR1 intel tarafından ayrılmıştır. Kullanılamaz.

CR2 sayfa hatasına neden olan lineer adresi tutar.

CR3 Sayfa dizin tablosunun fiziksel başlangıç adresini tutar ve kaşe yönetimini sağlar.

CR4 mimariye özgü değişik seçenekler sunar.

Korumalı modda, uygulama programları kontrol yazmaçlarını yükleyemezler ancak bu yazmaçları okuyabilirler. Ayrıca bu yazmaçların bitlerini değiştirirken, diğer bitlerin değerleri kesinlikle korunmalıdır.



Şekil 4.6: Sistemdeki Kontrol Yazmaçları

Kontrol yazmaçlarındaki bazı bitlerin işlevleri aşağıdadır.

PG (Paging) biti : Sayfalama mekanizmasını aktiflemek için kullanılır. PG=1 ise mekanizma çalışır, 0 ise lineer adresler fiziksel adreslere eşittir.

CD (Cache Disable) biti : Fiziksel belleğin işlemcinin içindeki kaşelerde (L1 ve L2) kaşelenmesini engeller. Bu bit 1 ise, kaşede iligili veri varsa, bilgi kaşeden alınır. Ancak kaşede ilgili veri yok ise, fiziksel bellekten okunan veri kaşeye yazılmaz. Eğer kaşeyi tamamen aradan çıkarmak istersek, tüm kaşe girdilerini geçersiz yaparak kaşe erişimini engellemiş oluruz.

PE (Protection Enable) : Eğer 1 ise işlemcinin korumalı modda çalışmasını sağlar. İşlemcinin işlev modunu belirler.

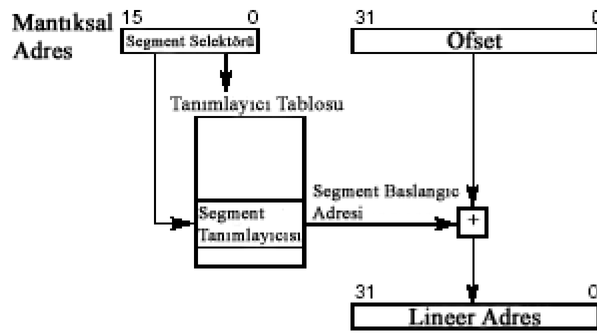
PSE (Page Size Extensions): Eğer 1 ise işlemci sayfa büyüklüğünü 4MB olarak, 0 ise 4KB olarak görür.

4.6 Segmentasyon

Segmentasyon ile, işlemcinin adres sahası küçük adres sahalarına ayrılır ve bunların her birine segment denmektedir. Daha önce de belirtildiği gibi, segmentasyon ile, her sürece kendi adres sahası atanır ve süreç bu adres sahası kapsamı içerisinde çalışmasını süredürür. Diğer süreçlerden bağımsız ve onların

adres sahalarını etkilemeyecek şekilde çalışır. Segmentler o sürece ait veri, yığıt ve kod kısımlarını içereceği gibi, sistemin o süreç için tuttuğu TSS ve LDT gibi sistem yapılarını da içerebilmektedir.

Korumalı modda, segment yazmaçları selektör denen ve bir tanımlayıcı tablosundaki bir tanımlayıcıyı seçmek için kullanılan indeks değerini içerir. Tanımlayıcı ise, hafıza segmenti hakkındaki bilgileri tutan bellek bölgesidir. Bu bilgiler arasında, o hafıza bölgesinin başlangıç adresi, uzunluğu ve erişim hakları gibi bilgiler bulunur.



Şekil 4.7: Matıksal Adresten Lineer Adrese Dönüşüm

Korumalı modda, segmentasyon ile bellek erişimi, aynen gerçek moddaki gibidir. Yine bir segmet yazmaçı ve bir ofset değeri kullanılır. Gerçek moddan farkı, segment yazmaçı içerisindeki değerin, işlemci tarafından farklı şekilde yorumlanmasıdır. Bu yorumlama işleminden sonra, ilgili segmentin başlangıç adresi bulunur ve ofset değeri buna eklenir. Çeşitli kontrol mekanizmaları sonucu, eğer sayfalama mekanizması aktif değilse, fiziksel adres; aktif ise sayfalama mekanizmasından da geçecek olan sanal adres elde edilir.

4.6.1 Tanımlayıcı Tabloları ve Tanımlayıcılar

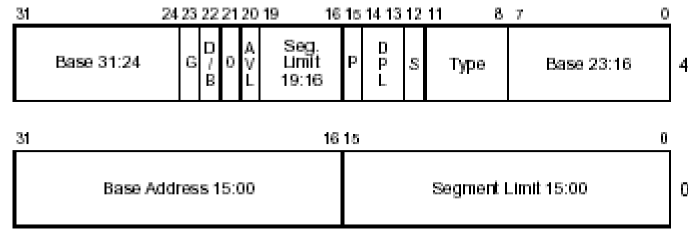
Korumalı modda, sistemde Genel Tanımlayıcı Tablosu (Global Descriptor Table – GDT) ve Yerel Tanımlayıcı Tablosu (Local Descriptor Table – LDT) olmak üzere iki tablo bulunmaktadır. Tanımlayıcı tabloları, segment tanımlayıcılarını tutan bir dizi gibi düşünülebilir.

GDT, sistemdeki tüm programlar için geçerli olan segment bilgilerini tutarken, LDT ise genellikle sistemdeki her program için farklıdır ve o programa

özgü segment bilgilerini tutmaktadır. Sistemde bir adet GDT bulunurken, hiçbir LDT bulunmayabilir veya sistemdeki süreçler sadece bir tane LDT'yi paylaşabilir.

Her tablo 8192 adet tanımlayıcı içerebilir ve her tanımlayıcı 8 byte yer kaplamaktadır. Böylelikle sistem, her uygulamaya toplam 16384 tanımlayıcı sunabilmektedir.

Tanımlayıcılar, işlemciye o segmentin yeri, büyüklüğü, erişim hakları ve durumu hakkında bilgi verirler ve 8 bayt uzunluktadır.



Şekil 4.8: Bir Tanımlayıcı Yapısı

4.6.2 Tanımlayıcı girdileri

Base sahası: segmentin hafızadaki başlangıç adresini tutmaktadır.

Limit sahası: segmentin uzunluğunu tutmaktadır.

G (Granularity) biti: G biti, segement limiti sahasının içerisindeki değerin byte mı yoksa sayfa sayısı mı olduğunu belirlemek için kullanılır. Eğer G=1 ise, limit değeri sayfa sayısı cinsinden doldurulmuştur.

D/B biti : Eğer 1 ise, segment 32 bitlik kod veya veri segmenttir. Eğer bu bit 0 ise , segment 16 bitlik bir segment olarak algılanır. Bu bit sayesinde, işlemci örneğin bir kod segment ise, o segmentten aldığı komutları 16 bit veya 32 bit uzunluğunda komutlar olarak değerlendirir.

AVL (Available) biti: Segmentin sistem programları tarafından kullanılıp kullanılamayacağını belirleyen bittir.

P (Present) biti: Segmentin hafızada olup olmadığını belirleyen bittir. Eğer bu bit 0 ise ve program o segmente erişmek isterse, işlemci "Segment Not Present" yazılım istisnasını oluşturur.

DPL (Descriptor Privilege Level) sahası: Tanımlayıcı ayrıcalık düzeyi, o segmente erişmek için kullanılır ve segmente kontrollü erişimi sağlar. Sistemde 3 adet ayrıcalık düzeyi vardır ve bunlar sırasıyla DPL 0, DPL 1, DPL 2 ve DPL 3'tür. Bunlardan en fazla erişim hakkına sahip olunulduğu düzey DPL 0'dır. Daha düşük bir öncelik düzeyine sahip bir program, daha yüksek öncelik düzeyine sahip bir segmente erişemez.

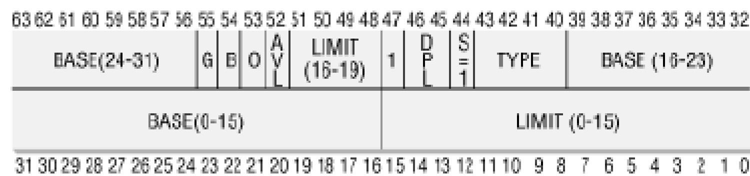
S biti : O segmentin bir sistem segmenti mi yoksa bir kod veya veri segmenti mi olduğunu belirleyen bittir. S= 0 ise segment bir sistem segmentidir.

Type sahası: O segmentin tipini belirler. Örneğin kod segmenti mi yoksa veri segmenti mi olduğunu bu saha ortaya koyar.

Tanımlayıcılar , sistem-segmenti tanımlayıcıları ve segment tanımlayıcıları olmak üzere iki tiptedir ve bunlar segmentler hakkında değişik bilgileri içermektedirler.

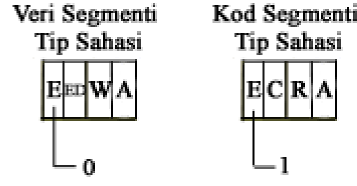
4.6.3 Segment Tanımlayıcıları (Kod veya Veri Segmenti Tanımlayıcıları)

Eğer tanımlayıcının S biti 1 ise, o tanımlayıcı bir kod veya veri segmenti bilgisi içeriyor demektir.Bir segment tanımlayıcısının içeriği aşağıdaki gibidir.



Şekil 4.9 : Segment Tanımlayıcısı Yapısı

Tip sahasının içerdiği bilgi, o segmente nasıl erişileceği ve o segmente nasıl davranılacağı hakkında bilgiler içermektedir. Tip sahası 4 bitlik bir sahadır ve her bit, o segmentin özelliklerini belirlemek için kullanılır.



Şekil 4.10 : Segment Tanımlayıcısı Tip Sahası

E (Executable) biti : O segmentin veri (ya da yığıt) veya kod segmenti olup olmadığını belirlemek için kullanılır. Eğer E = 1 ise , o segment kod segmentidir, değilse o segment kod veya yığıt segmentidir.

A (Accessed) biti : Bu bit, mikroişlemci o segmente her eriştiğinde 1 yapılmaktadır. Bu sayede, o segmentin kullanımı hakkında bilgi elde edilebilinir.

Veri Segmenti Tip Bitleri

ED (Expand downward) biti: Bu bit o segmentin yığıt olup olmadığıdı belirlemek için kullanılır. ED=1 ise segment yığıt segmentidir. ED=0 ise segment veri segmentidir.

W (Write) biti: O segmente veri yazılıp yazılamayacağını belirtir. Eğer W=1 ise, o segment hem okunabilir hem de yazılabilir bir segmenttir. Eğer W=0 ise, o segment sadece okunabilir bir segmenttir.

Kod Segmenti Tip Bitleri:

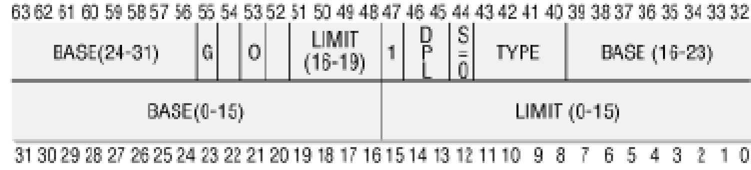
C (Conforming) biti : Daha önce belirtildiği gibi, her segment tanımlayıcısının bir tanımlayıcı ayrıcalık düzeyi vardı ve o segmente erişim ancak o ayrıcalık düzeyinden daha düşük veya eşit öncelikli programlar tarafından yapılıyordu. Eğer C=0 ise, o segmente erişimde ayrıcalik düzeyi göz önüne alınmaz. C=1 ise , segmente erişim de ayrıcalık düzeyi göz önüne alınır.

R(Read) biti : O kod segmentinin okunup okunamayacağını belirleyen bittir. R=1 ise kod segmenti süreçler tarafından okunabilir, eğer R=0 ise sadece işletilebilir.

Kod segmenti ve veri segmenti tanımlayıcıları, GDT ve LDT tablolarının elemanı olabilirler. Sistem yazılımları veya işletim sistemi, programlara ait segment bilgilerini oluştururlar ve bu bilgiler sistedeki bu tablolara işlenirler.

4.6.4 Sistem Segment Tanımlayıcıları

Eğer tanımlayıcı girdisindeki S biti 0 ise, o tanımlayıcı bir sistem segmenti ile ilgili bilgi içeriyor demektir. Bir sistem segmenti tanımlayıcısının içeriği aşağıdaki gibidir.



Şekil 4.11 : Sistem Segment Tanımlayıcısı Yapısı

Sistem segmentleri, sistemdeki önemli veri yapılarını içeren segmentlerin bilgisini tutar. Görüleceği gibi, tanımlayıcı yapısı kod veya veri segmenti tanımlayıcı yapısıyla aynıdır. Yine tip sahası bilgisi, o sistem segmenti tanımlayıcısının hangi tür sistem veri yapısını tuttuğu bilgisini içermektedir.

Çizelge 4.1: Sistem Segmenti Tanımlayıcısı Tip Sahası

Sistem Segmenti Tanımlayıcısı Tip Sahası

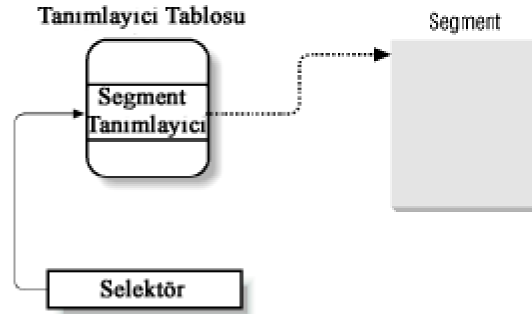
0	0	0	0	Reserved
0	0	0	1	16-Bit TSS (Available)
0	0	1	0	LDT
0	0	1	1	16-Bit TSS (Busy)
0	1	0	0	16-Bit Call Gate
0	1	0	1	Task Gate
0	1	1	0	16-Bit Interrupt Gate
0	1	1	1	16-Bit Trap Gate
1	0	0	0	Reserved
1	0	0	1	32-Bit TSS (Available)
1	0	1	0	Reserved
1	0	1	1	32-Bit TSS (Busy)
1	1	0	0	32-Bit Call Gate
1	1	0	1	Reserved
1	1	1	0	32-Bit Interrupt Gate
1	1	1	1	32-Bit Trap Gate

Görüldüğü gibi, sistem segment tanımlayıcıları; süreçlere özel LDT tabloları, süreçlere ait bilgilerin tutulduğu TSS (Task State Segment) yapısı, yazılım ve donanım kesmeleri

için kapılar ve çağırım kapıları gibi sistem tarafından kullanılan veri yapılarına ait segment bilgilerini içermektedir.

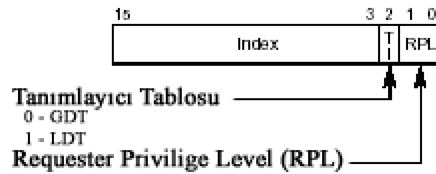
4.6.5 Selektörler (Seçiciler)

Selektörler; segment tanımlayıcı tablolarındaki segment tanımlayıcılarını seçmeye yarayan ve bu tanımlayıcılar üzerinden segmentlere erişimi sağlayan 16 bitlik değerlerdir.



Şekil 4.11 : Selektör Kullanımı

Segmentlere ancak selektörler yardımı ile erişilebilir. Selektörler, segment yazmaçlarına yüklenirler. Bir başka değişle, segment yazmaçları, gerçek moddaki gibi segmentin fiziksel olarak başlangıç adresini tutmaz. Bunun yerine, segment tanımlayıcı tablosundaki bir girdiyi işaret eder. Böylelikle, o segmente ilişkin bilgiler, işaret edilen tablonun elemanından elde edilirler. Bu sebeple, korumalı modda, segment yazmaçları selektör değerini tutarlar.



Şekil 4.12 : Selektör Yapısı

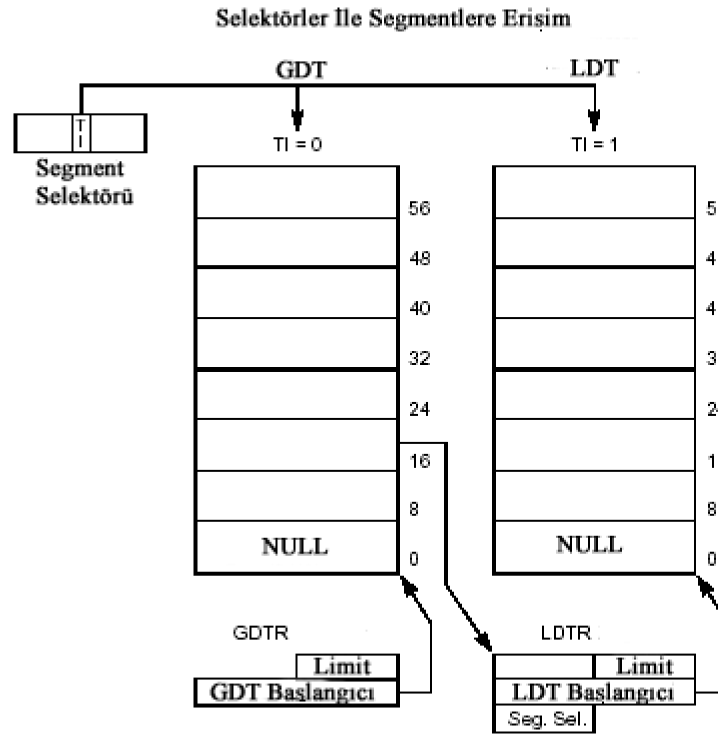
Bir segment selektörü; tanımlayıcı indeksi, tablo indeksi ve istemci ayrıcalık düzeyi bilgilerini içermektedir.

<u>İndeks Sahası :</u> GDT veya LDT tablosundaki 8192 segment tanımlayıcısından birini seçer. İşlemci, bu sahadaki veriyi 8 ile çarpıp GDTR yazmaçındaki değer ile toplar. Daha önce belirtildiği gibi, segment tanımlayıcıları 8 byte uzunluğundadır.

<u>TI (Table Indicator) Biti :</u> Selektörün GDT 'den mi yoksa LDT'den mi girdi seçtiğini gösteren bittir. TI=0 ise GDT girdisi, TI=1 ise LDT girdisi seçilmektedir.

<u>RPL (Requester Privilege Level) Sahası:</u> Selektörün ayrıcalık düzeyini belirtir.

Böylelikle, selektör yardımı ile segmentlere erişim aşağıdaki şekildeki gibi özetlenebilir.



Şekil 4.13 : Selektörler İle Segmentlere Erişim

Şekilde görüldüğü gibi, GDT ve LDT tablolarının ilk elemanları her zaman 0 ile doldurulur ve bu ilk elemana NULL tanımlayıcı denir. Selektörler ilgili tablo elemanı indeksi ile doldurulur. LDT 'ler GDT içerisinde bir sistem segment tanımlayıcısı ile tanımlandığı için, LDTR, GDT tablosundaki bir elemanı gösterir.

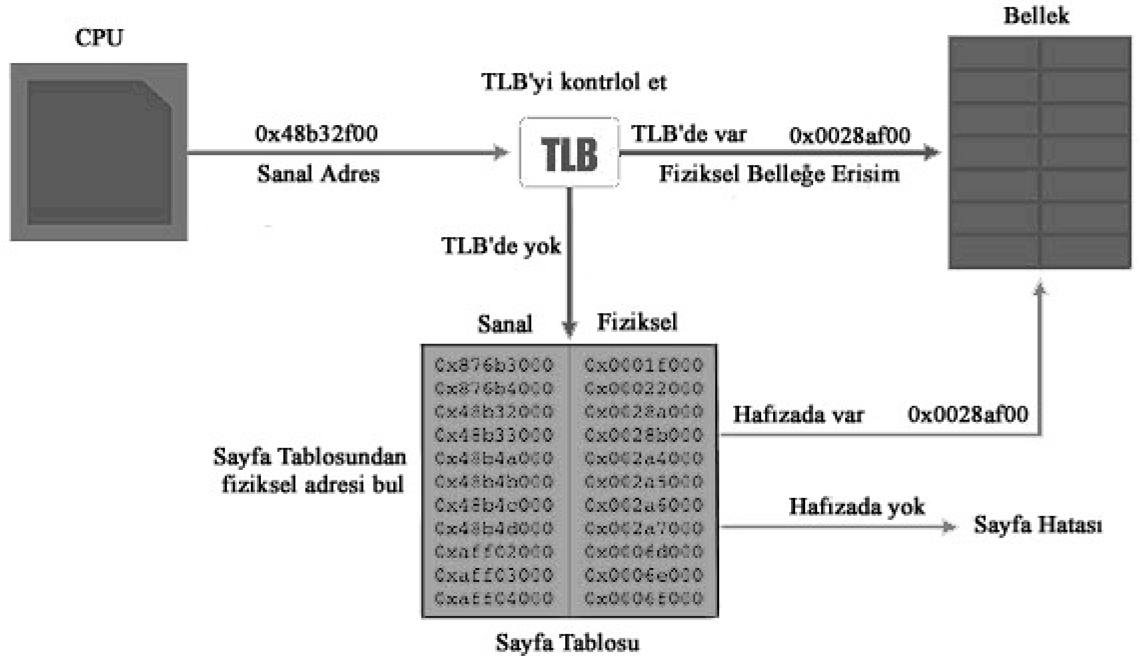
4.7 Sayfalama

Sayfalama sayesinde, intel işlemcisi büyük bir lineer adres bölgesini, daha küçük bir fiziksel adres sahası ile eşleyebilir. Buna sanal bellek denmektedir.

Sayfalama kullanıldığında, işlemci lineer adres sahasını eşit uzunluklu sayfalara böler ve bu sayfalar fiziksel adreslerle eşlenir. Sayfalama mekanizması aktiflendiğinde, işlemci üretilen mantıksal adresi lineer adrese dönüştürür. Daha sonra, lineer adresi sayfalama mekanizmasından geçirerek, doğru fiziksel adresi elde eder.

Eğer o lineer adresin eşleneceği sayfa hafızada değilse, işlemci bir "Sayfa Hatası" istisnası oluşturur. Bu istisna sayesinde, illetim sistemi ilgili sayfayı diskten okuyarak hafızaya yükler; hafızadaki bazı sayfaları ise diske yazar.

Sayfalama mekanizması içerisinde, oluşan lineer adreslerin fiziksel adreslere dönüştürülümesi için önce o sürecin sayfa tablosundan ilgili girdi bulunur ve bu girdi sayesinde ilgili fiziksel adres elde edilir. Bu işlemleri kısaltmak için, işlemcinin iç yapısı içerisinde TLB (Translation Lookaside Buffers) denen kaşe bulunmaktadır.



Şekil 4.13 : Sayfalama İle Bellek Erişimi

Adres dönüşümünde öncelikle bu kaşeye bakılmakta, eğer ilgili lineer adres burada mevcut değilse, sayfa tablolarına bakılmaktadır. Böylelikle adres dönüşüm daha hızlı yapılmış olur.

Sayfalama mekanizmasını kontrol eden 2 önemli bit şunlardır:

- PG (Paging) biti (CR0 'ın 31. biti)
- PSE (Page Size Extensions) biti (CR4'ün 4. biti)

PG biti sayfalama mekanizmasını aktfilemek için kullanılır. PSE biti sayfa boyutunu 4MB veya 4KB yapmak için kullanılır.

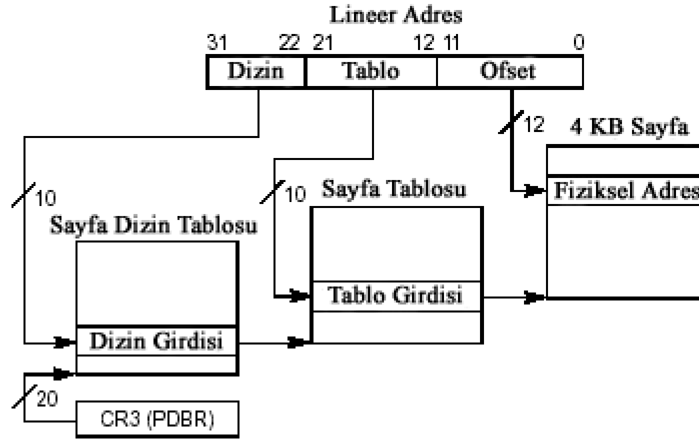
4.7.1 Sayfa Tabloları ve Sayfa Dizini

Sayfalama mekanizması sayfa tabloları ve sayfa dizini denen yapıları kullanmaktadır.

Sayfa dizin tablosu 32 bitlik girdilere sahiptir ve bu girdiler sayfa tablolarının fiziksel adresini tutarlar. Bu tablo en fazla 1024 girdiye sahip olabilir ve her sayfa tablosu 4 MB'lık fiziksel belleği adresleyebileceği için, sayfa dizin tablosu yapısı ile 4GB'lık bellek adreslenebilir.

Sayfa tabloları yine 32 bitlik girdilere sahiptirler ve her girdi 4KB'lık bellek bölgesinin başlangıç adresini tutar. Bu tablolarda en fazla 1024 girdi tutabilir. Böylelikle, her sayfa tablosu, 4MB'lık bellek bölgesini adreslemiş olur.

Sayfalama mekanizması aktiflendiğinde, lineer adres 3 parçadan oluşuyormuş gibi düşünülür. Bunlar sırasıyla sayfa dizin tablosu girdisi, ilgili sayfa tablosu girdisi ve ofset değeridir.



Şekil 4.14 : Lineer Adres Yapısı

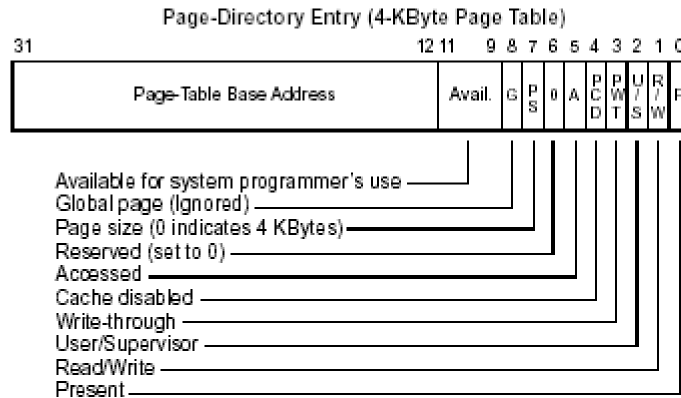
Lineer adres 32 bitlik bir sayıdır. Bunun 31-22 arasındaki bitleri sayfa dizin tablosundan bir girdinin ofsetini belirtir. Girdi, fiziksel adresin bulunacağı sayfa tablosunun fiziksel başlangıç adresini tutmaktadır.

21-12 bitleri arasındaki bitler, sayfa dizin tablosundan elde edilen sayfa tablosunun ilgili girdisinin ofsetini tutmaktadır. Bu girdiden, o sayfanın fiziksel adresi elde edilir.

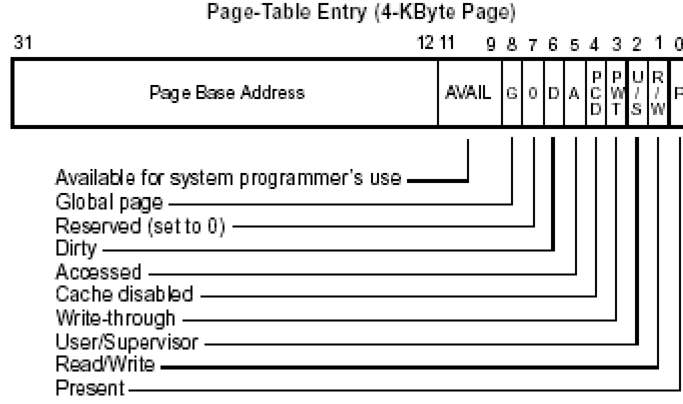
11-0 arasındaki bitler ise elde edilen fiziksel adrese eklenecek değerdir.

4.7.2 Sayfa Dizini ve Sayfa Tabloları Girdileri

Bir sayfa dizini ve sayfa tablosu girdisi 32 bit uzunluğundadır ve aşağıdaki formata sahiptir.



Şekil 4.15 : Sayfa Dizin Tablosu Girdisi Yapısı



Şekil 4.16 : Sayfa Tablosu Girdisi Yapısı

Bu girdilerdeki önemli bitler ve anlamları aşağıda açıklanmıştır.

<u>Page Table Base Address :</u> Sayfa Tablosu fiziksel başlangıç adresini tutmaktadır. Sayfa Tablosu ve Sayfa Dizini, 4KB 'ın katı şeklindeki fiziksel adreslerden başlamalıdır. Bu sebeple, 31-12 arasındaki bitler, 4KB'ın katı fiziksel adrese yerleştirilmiş sayfa tablosunun, 4KB'ın katı cinsinden fiziksel başlangıç adresini tutar.

<u>Page Base Address</u> : Sayfa fiziksel başlangıç adresini tutmaktadır.

<u>P (Present) Biti :</u> O sayfanın hazıfada olup olmadığını belirten bittir. Eğer hafızada değilse, işlemci "Sayfa Hatası" ististanısını oluşturur.

<u>R/W (Read/Write) Biti :</u> O sayfanın okunup yazılabilir olmasını sağlar.

<u>U/S (User/Supervisor) Biti :</u> Eğer bu bit 1 ise, o sayfaya kullanıcı programları tarafından erişilemez.

<u>A (Accessed) Biti :</u> O sayfaya her erişimde , bu bit 1 olmaktadır.

<u>D(Dirty) Biti :</u> O sayfaya her yazma işleminde bu bit 1 olmaktadır.

4.8 Koruma Mekanizması

Sayfalama, segmentasyon ve ayrıcalık seviyeleri sayesinde işlemci bize programların birbirlerini etkilemeden çalışabileceği bir ortam sunmaktadır. Koruma mekanizması ile, sistemdeki işletim performansını etkilemeden,

- Limit kontrolü
- Tip kontrolü
- Kullanılabilir komut kümesinin sınırlandırılması
- Adres sahasının sınırlandırılması

gibi kontroller yapılmaktadır. Bu kontroller adres dönüşümü sırasında hızlı bir şekilde yapılmaktadır.

Koruma mekanizması, denetlemeleri yaparken şu bit sahalarını denetlemektedir:

- Segment tanımlayıcısındaki S (System) biti
- Tip sahası
- Limit sahası
- Tanımlayıcı Ayrıcalık Düzeyi (DPL)
- İstemci Ayrıcalık Düzeyi (RPL)
- Anki Ayrıcalık Düzeyi (CPL)
- Sayfa Girdilerindeki U/S (User / Supervisor) biti
- Sayfa Girdilerindeki R/W (Read/Write) biti

4.8.1 Limit Kontrolü

Limit kontrolü kapsamında, o segmente erişirken o programın segment sınırları dışına çıkması engellenir. Eğer süreç limitin dışında bir bellek bölgesine erişmeye çalışırsa, işlemci “ Genel Koruma Hatası” istisnasını oluşturur.

4.8.2 Tip Kontrolü

Tip kontrolü kapsamında , işlemci o segment tanımlayıcısının S bitini ve tanımlayıcının tip sahasını kontrol eder. Kontrol mekanizmasına göre doğru işleyen bir sistemde bazı özellikler şunlardır:

Kod segmenti yazmaçı, kod segment tipindeki bir segment hakkında bilgi içeren bir segment tanımlayıcısını gösteren bir selektör ile yüklenmiş olmalıdır.

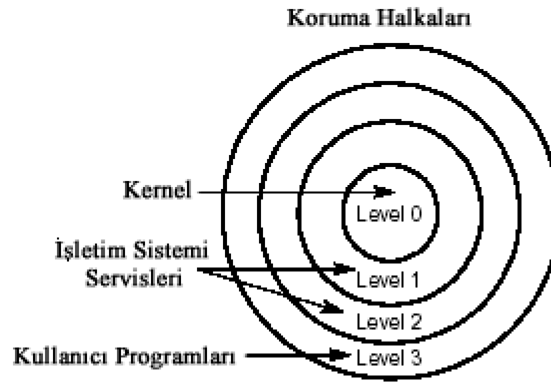
Kod segment selektörleri, veri ve yığıt segmentleri gibi veri yazılan segment yazmaçlarına yüklenemezler.

Sadece yazılabilir segment selektörleri yığıt yazmaçına yüklenebilir.

Bir CALL veya JMP komutunun operandı bir selektör ise, o selektör bir kod segmente, çağırım veya görev kapısına işaretçi olmalıdır.

4.8.3 Ayrıcalık Düzeyleri

İşletim sisteminin 0,1,2 ve 3 olmak üzere 4 ayrıcalık düzeyi bulunmaktadır. En içte işletim sistemi çekirdeği çalışmaktadır ve en düşük ayrıcalık düzeyine sahiptir. İşletim sistemi çekirdeği, sistemdeki her şeyin kontrolüne sahiptir ve diğer tüm progranlar onadan daha yüksek ayrıcalık seviyesine sahiptirler.



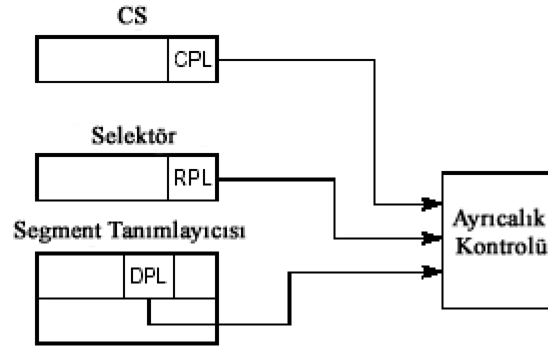
Şekil 4.17 : Sistem Ayrıcalık Düzeyleri

Burada CPL, DPL ve RPL kavramlarını incelemek gerekmektedir.

CPL (Current Privilege Level) : O anki ayrıcalık düzeyi, o an çalışmakta olan programın kod segment yazmaçındaki selektörün gösterdiği kod segmenti tanımlayıcısının tutmuş olduğu tanımlayıcı ayrıcalık düzeyidir.

DPL (Descriptor Privilege Level) : Bir segmentin ayrıcalık seviyesidir. O segmentin tanımlayıcısının DPL sahasında tutulmaktadır.

RPL (Requestor Privilege Level) : İstemci ayrıcalık düzeyi, selektörlerde saklanan ayrıcalık düzeyi sahasındaki değerdir.

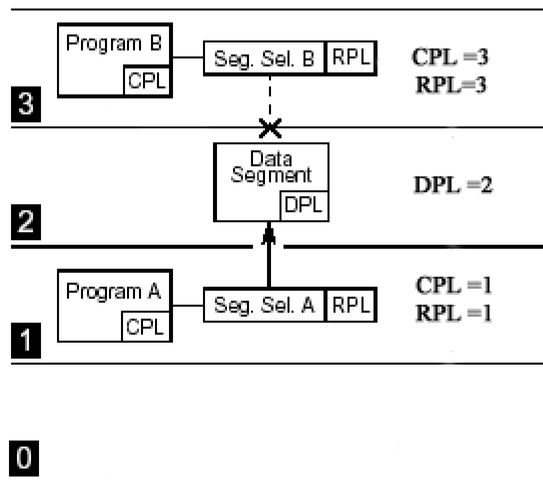


Şekil 4.18 : Ayrıcalık Düzeyleri ve Ayrıcalık Kontrolü

İşlemci RPL,DPL ve CPL değerlerini kontrol ederek işletime izin verir veya vermez. Örneğin RPL yeterli değilse, programın o segmente erişimine izin verilemez.

Aşağıdaki örnekte, 3. ayrıcalık seviyesinde çalışan B programı, yine 3. istemci ayrıcalık seviyesi ile , tanımlayıcı ayrıcalık düzeyi 2 olan veri segmentine erişmeye çalışmaktadır. B programı daha yüksek ayrıcalık seviyesinde olduğu için, işlemci programın o segmente erişmesine izin vermez.

Ancak, 1. öncelik seviyesinde çalışan A programı, yine aynı istemci öncelik seviyesinden aynı veri segmentini çağırmaktadır. Daha düşük önceliğe sahip olduğu için, işlemci koruma mekanizması bu erişime izin verecektir.

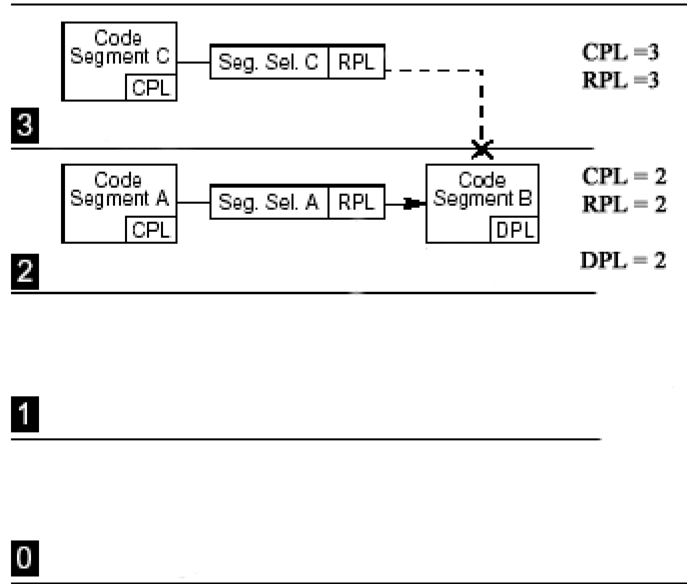


Şekil 4.19 : Ayrıcalık Düzeyleri ve Segment Erişimleri

4.8.4 Kod Segmentlerini Operand Olarak Alan Jump Komutları

Jump komutuna yakın adresler parametre olarak verildiğinde, o segment içerisindeki adreslere atlanır, dolayısıyla öncelik düzeyi kontrol edilmez. Ancak segmentler arası işletilen JMP komutu, öncelik seviyesi kontrolünden geçer.

Korumalı modda, daha önce bahsedildiği gibi segment yazmaçları selektör değerlerini içerir. Ayrıca JMP komutuna operand olarak bir kod segmenti selektörü verilmelidir.



Şekil 4.20 : Ayrıcalık Düzeyleri ve Ayrıcalık Kontrolü

Yukarıda, bir kod segmentine atlayan 2 program gösterilmektedir. C kod segmentini işleten program, 3. istemci ayrıcalık düzeyi ile B kod segmentine atlamak istemektedir. Ancak daha düşük bir tanımlayıcı ayrıcalık seviyesinde bulunan B kod segmentine erişimine izin verilmeyecektir.

2. ayrıcalık seviyesinde çalışan A programı ise, B kod segmentine erişebilir ve oradaki kodu çalıştırabilir.

4.8.5 Kapı Tanımlayıcıları

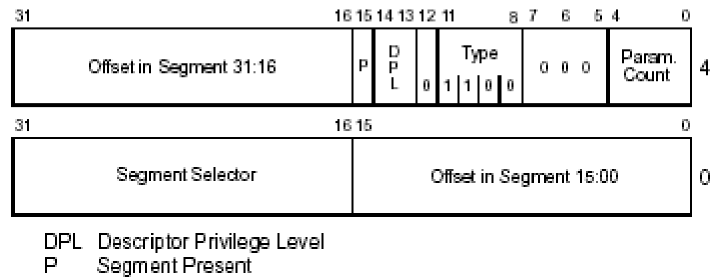
İntel işlemcisi, farklı ayrıcalık seviyesinde bulunan kod segmentlerinin birbiriyle ilişkide bulunabilmesi için kapı denen bir yapıyı ortaya koymuştur. Bunlar

- Çağırım kapıları
- Yazılım Kesmesi Kapıları
- Donanım Kesmesi Kapıları
- Görev Kapıları 'dır

4.8.6 Çağırım Kapıları

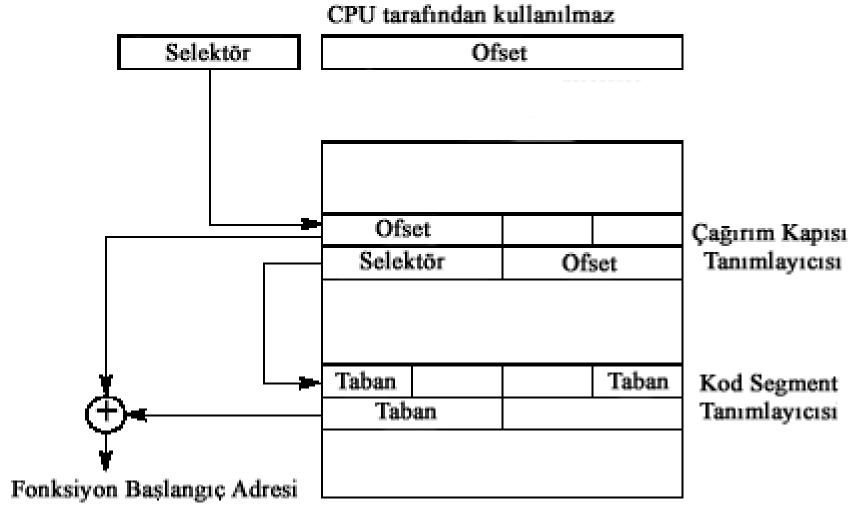
Çağırım kapıları, farklı ayrıcalık seviyesinde çalışan programların birbirlerinin kodlarına erişimini sağlarlar.

Çağırım kapıları da tanımlayıcılar ile ifade edilirler. Çağırım kapılarına ait tanımlayıcılar GDT ve ya LDT'de bulunabilir. Bir çağırım kapısına ait tanımlayıcı yapısı aşağıdaki gibidir.



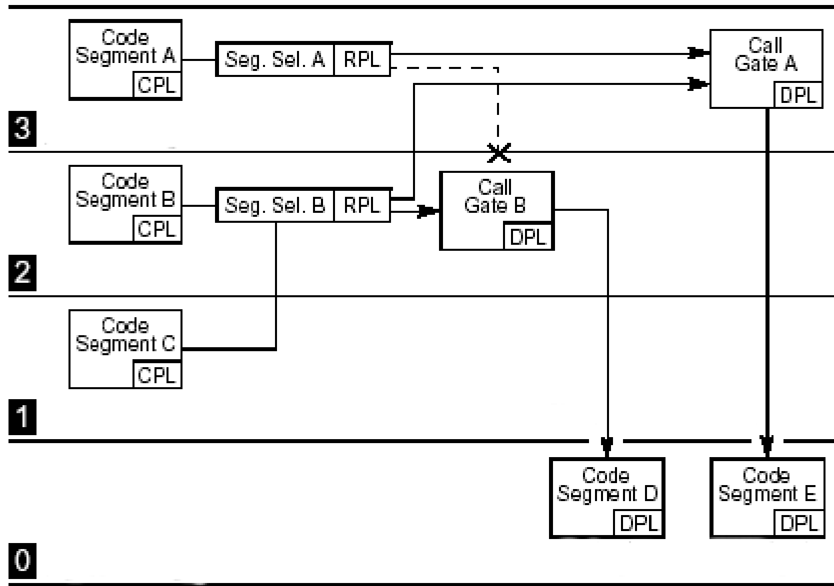
Şekil 4.21 : Kapı Tanımlayıcısı Yapısı

Segment selektörü sahası, erişilecek kod segmente ait selektördür. Bu selektör yardımı ile, GDT veya LDT 'deki segmente erişilir. Parametre sayısı sahası ise, bir yığıt değişimi olduğunda kaç adet parametrenin, yeni yığıta atılacağını belirler.



Şekil 4.22 : Kapılar Yardımı İle Bellek Bölgelerine Erişim

Aşağıdaki şekilde görüldüğü gibi, farklı ayrıcalık düzeyinde çalışan bir program , kendisinden daha düşük bir ayrıcalık düzeyinde bulunan kod parçasını, çağırım kapıları sayesinde kontrollü bir şekilde çağırabilmektedir.



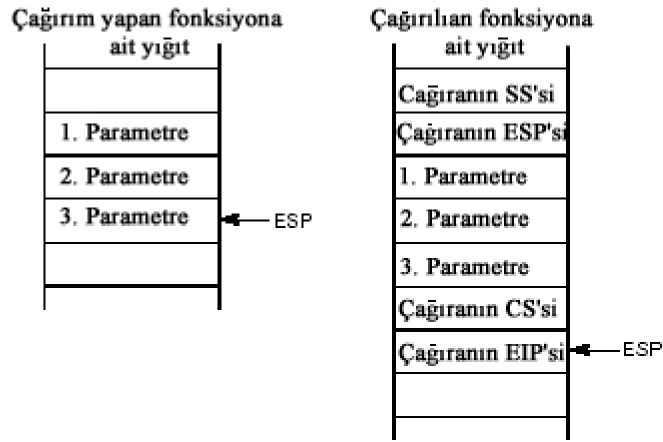
Şekil 4.23 : Kapılar ve Erişim Kontrolleri

Burada, çağırım yapan süreçin CPL değeri, çağırım kapısının DPL değerinden küçük veya ona eşit olmalıdır.Eğer çağırım, daha öncelikli bir kod segmentine yapılıyorsa ve bu kod segmentine erişimde önceliğin göz önüne alınacağını ifade eden C (conforming) biti 1 ise, bir yığıt değişim işlemi olmaktadır.

4.8.7 Yığıt Değişimi

Yığıt değişimi, daha öncelikli olan fonksiyonların, daha az öncelikli fonksiyonlar tarafından çağırılırken, oluşabilecek hataları önlemek için yapılmaktadır. Örneğin, çağıran fonksiyonun yeterli yığıtı yoksa, daha öncelikli fonksiyon çökebilir.Ayrıca, yığıt değişimi ile, daha öncelikli bir fonksiyonun, daha az öncelikli bir fonksiyon tarafından okunup yazılması önlenmiş olur.

Intel 386 mimarisinde, her sürece 4 adet yığıt atanmalıdır. Her yığıt , o öncelik seviyesindeki işlemlerde kullanılır.Bu yığıtlara ait işaretçiler, ö sürece ait TSS yapısında tutulurlar. Süreçlere bu yığıtların atanması, işletim sisteminin sorumluluğudur.



Şekil 4.24 : Kapılar ve Erişim Kontrolleri

Bir süreç, çağırım kapısı yardımı ile daha öncelikli bir süreci çağırırsa, işlemci yığıt değişimin uygulamaktadır.Bu işlem sırasındaki adımlar şu şekildedir:

1. Öncelikle , atlanacak kod segmentine ait kod segment tanımlayıcısının DPL sahasını kullanarak, hangi seviyedeki yığıtın yüklenileceğini öğrenir.
2. Çağırım yapan sürecin TSS'sinden, yüklenecek yığıtı alır. Bu arada erişim kontrollerini de yapmaktadır.
3. O an kullanılan yığıt değerlerini TSS'ye kaydeder.
4. Yeni yığıta ait segment selektörünü SS'ye, ofset değerini de ESP'ye

yükler.

5. Çağırım yapan fonksiyona ait SS ve ESP değerlerini yeni yüklenen yığıta atar.
6. Çağırım kapısının parametre sayısı sahasındaki değer kadar parametreyi yeni yığıta yükler.
7. Çağırım yapan sürece ait CS ve EIP değerlerini sırası ile yeni yığıta atar.
8. Çağırılan fonksiyona ait CS ve EIP değerlerini ilgili yazmaçlara yükler.

4.8.8 Sayfa Seviyesinde Koruma

Sayfalama ile koruma sayesinde, daha öncelikli kod bölgelerinin kullanıcı programlarından korunması sağlanır.

Sayfalama mekanizması ile, hafızaya her erişimde, erişim kontrol edilir. Eğer erişimde herhangi bir ihlal varsa, sayfa hatası istisnası oluşturulur. Sayfalar yazmaya karşı korumalı olabilir. Veya sayfa daha fazla bir öncelik düzeyinde olabilir. İşlemci ,sayfa düzeyinde korumada

- Ayrıcalık düzeylerini
- Sayfanın yazmaya karşı korumalı olup olmadığını kontrol eder.

Sayfanın sayfa tablosundaki girdisinin bitleri ile sayfa seviyesinde koruma sağlanır. U/S biti seviye kontrolü için kullanılır. U/S biti 0 ise, o sayfa işletim sistemi veya sistem verisini içeren sayfadır. Bunun için CPL 0,1 veya 2'dir. Eğer U/S biti 1 ise, sayfa kullanıcı verisi içindir. Bunun için CPL ise 3'tür.

Ayrıca R/W biti de önemlidir. Eğer R/W biti 0 ise, sayfa yazmaya karşı korumalı; eğer 1 ise sayfa yazılabilirdir.

Sayfalama mekanizması, segmentasyon ile kullanılırsa, önce segmentlere yönelik erişim kontrolü yapılır, daha sonra ise sayfa seviyesinde erişim kontrolü yapılır.

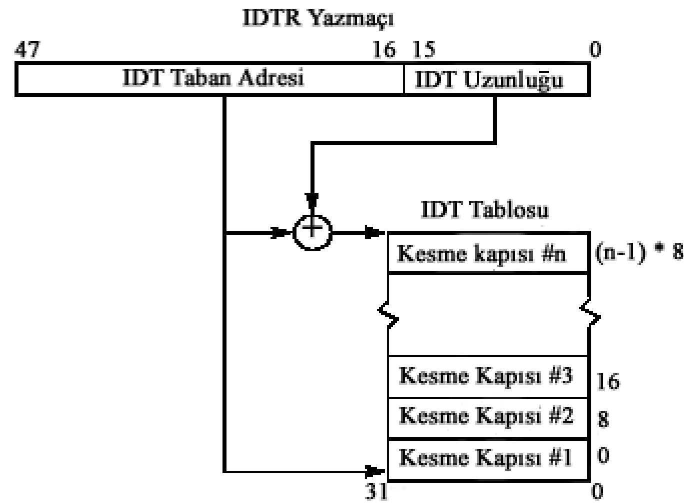
4.9 Kesme ve İstisna Yönetimi

Donanım kesmeleri, bir programın çalışması sırasında herhangi bir anda gelen donanım sinyalleri ile meydana gelebilir. Programlar da INT komutu sayesinde bir kesme oluşturabilir. İstisnalar ise işlemcinin bir komutu işlerken sıfıra bölme gibi hatalı bir durum ortaya çıktığı anda oluşturulur.Bu hatalar arasına erişim ihlalleri, sayfa hataları gibi durumlar da girmektedir.

Bir kesme oluştuğu anda, o an çalışmakta olan program askıya alınır ve kesme yöneticisi çalışmaya başlar. Kesme yöneticisinin çalışması bittikten sonra, program çalışmaya kaldığı yerden devam eder.

İşlemci her bir kesme ve istisna için, ilgili yöneticilere ait adresleri içeren bir tablo tutmaktadır. Bu tabloya IDT (Interrupt Descriptor Table) denmektedir. GDT'de olduğu gibi, IDT kesmelere ait yöneticilere ait 256 adet tanımlayıcı içermektedir. GDT'nin ilk elemanı boş (null) eleman iken IDT'de böyle bir durum yoktur.

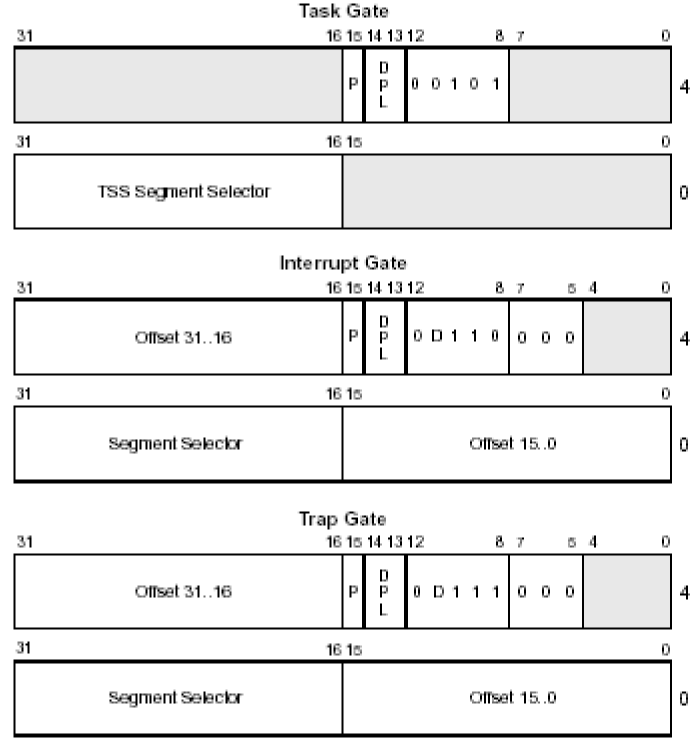
İşlemci IDT'ye IDTR (Interrupt Descriptor Table Register) yazmaçı ile ulaşmaktadır. Bu yazmaç IDT'ye ait 32 bir taban adresi ile 16 bitlik uzunluk değerini tutmaktadır.



Şekil 4.25 : IDT Tablosuna Erişim

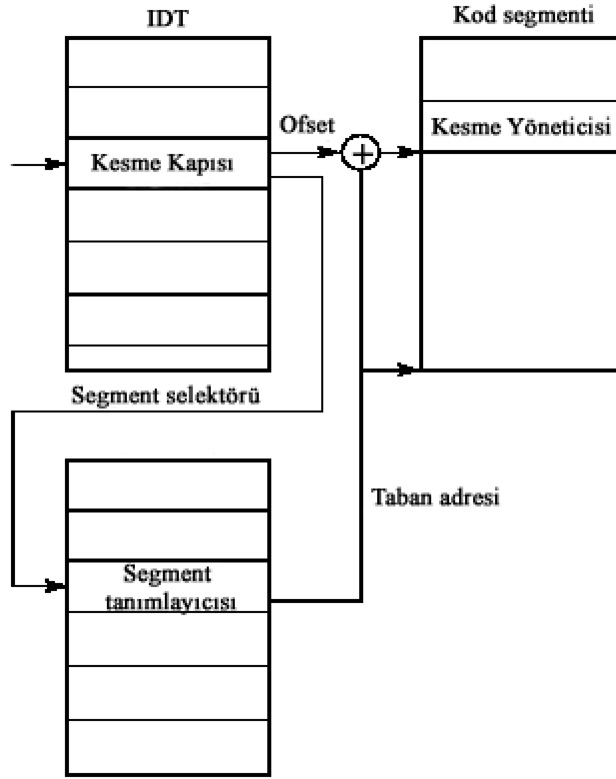
IDTR yazmaçı LIDT komutu ile yüklenmektedir. Bu komut ancak 0. ayrıcalık düzeyinde çalıştırılabilmektedir. IDTR yazmaçı genellikle işletim sistemi tarafından yüklenir.

IDT tablosu 3 çeşit tanımlayıcı içerebilmektedir. Bunlar görev kapısı tanımlayıcısı, kesme kapısı tanımlayıcısı ve yazılım kesmesi tanımlayıcısıdır.



Şekil 4.25 : IDT Tablosu Girdilerinin Yapıları

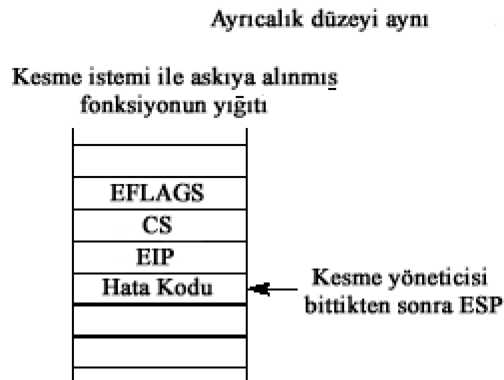
Görüldüğü gibi, IDT tanımlayıcıları çağırım kapıları ile benzer yapılara sahiptir. İşlemci, istisna ve kesme isteklerini çağırım kapılarına benzer bir şekilde işlemektedir. Bir kesme istemi geldiğinde, kesme numarasını kullanarak IDT'deki ilgili tanımlayıcıya ulaşır. Tanımlayıcıyı kullanarak kesme yöneticisinin adresini elde eder.



Şekil 4.25 : IDT Tablosu İle Bellek Erişimi

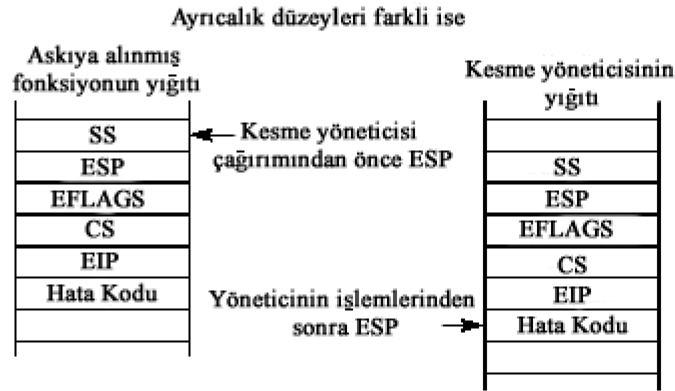
4.9.1 Kesme Yönetiminde Yığıt Yapısı

İşlemci, aynı ayrıcalık seviyesinde bir kesme isteği olduğunda EFLAGS,CS ve EIP yazmaçlarını yığıtta saklar. Eğer istisna sonucunda bir hata kodu da döndürülecekse, o da yığıtta saklanır. Kesme yöneticisinin yığıtı da, aşağıdaki şekildekiyle aynı görünümdedir.



Şekil 4.26 : Ayrıcalık Düzeyi Aynı İken Yığıtın Durumu

Eğer kesme aynı ayrıcalık düzeyinde işletilmeyecekse, bir yığıt değişimi olmalıdır.İşlemci, çalışması askıya alınmış olan fonksiyonun yığıtına EFLAGS,SS,ESP,CS,EIP ve hata kodunu, kesme yöneticisinin yığıtından kopyalar.



Şekil 4.27 : Ayrıcalık Düzeyi Farklı İken Yığıtın Durumu

Bir kesme yöneticisi fonksiyonunu sonlandırmak için IRET (Interrupt Return) komutu kullanılır. IRET komutunun RET komutundan farkı, daha önce saklanmış olan EFLAGS yazmaçını tekrar geri yüklemesidir. Ayrıca bir yığıt değişimi yapılacaksa, bu işlem gerçekleştirilir.

4.9.2 İntel Mimarisindeki İstisnalar

IDT tablosunun en fazla 256 eleman içerebileceği daha önceden belirtilmişti.Bu tablonun ilk 32 elemanı istisnalara ayrılmıştır. Aşağıda bu istisnalar listelenmiştir.

Bu istisnalara ait yöneticilere ait tanımlayıcılar IDT tablosuna yerleştirilir. Böylelikle, bu istisnalar meydana geldiği anda gerekli olan işlemler yapılabilir. IDT tablosunun geri kalan elemanları yazılım ve donanım kesmelerine ayrılmıştır. Sistemdeki PIC (Priority Interrupt Controller) çipi programlanarak, ilgili IDT elemanlarına yönlendirilir. IDT tablosu da ilgili tanımlatıcılar ile doldurularak kesmeleri yönetecek fonksiyonlar sisteme kaydedilir.

Çizelge 4.2: Sistemdeki İstisnalar

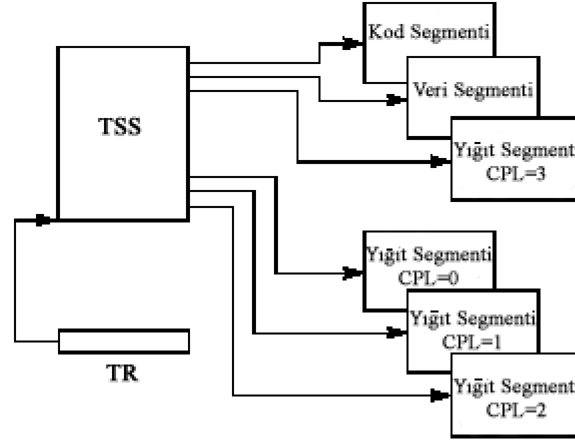
1. Kesme	Divide Error
2. Kesme	Debug exception
3. Kesme	NMI Interrupt
4. Kesme	Breakpoint
5. Kesme	INTO Detected Overflow
6. Kesme	BOUND Range Exceeded
7. Kesme	Invalid Opcode
8. Kesme	Coprocessor not available
9. Kesme	Double exception
10. Kesme	Coprocessor segment overrun
11. Kesme	Invalid Task State Segment
12. Kesme	Segment not present
13. Kesme	Stack Fault Exception
14. Kesme	General Protection Exception
15. Kesme	Page Fault Exception
16. Kesme	Rezerve Edilmiş
17. Kesme	Floating Point Error
18. – 32. Kesmeler	Rezerve Edilmiş

4.10 Süreç Yönetimi

İntel 386 ailesi mimarisi, süreçlerin durumunu saklamak ve süreçler arasındaki geçişleri sağlamak için bazı yapılar tutmaktadır.

Süreçler 2 parçadan oluşmaktadırlar. Bunlar bir çalışma sahası ve bir süreç durum yapısı (Task State Segment – TSS). Sürecin çalışma sahası onun kod, veri ,yığıt ve diğer segmentlerini içermektedir. TSS ise o sürece ait bilgilerin tutulduğu bir yapıdır.

Süreçler, onların TSS yapılarına ait selektörler ile ifade edilirler.Çalışan sürecin TSS yapısına ait selektör, görev yazmaçına (Task Register – TR) yüklenir.

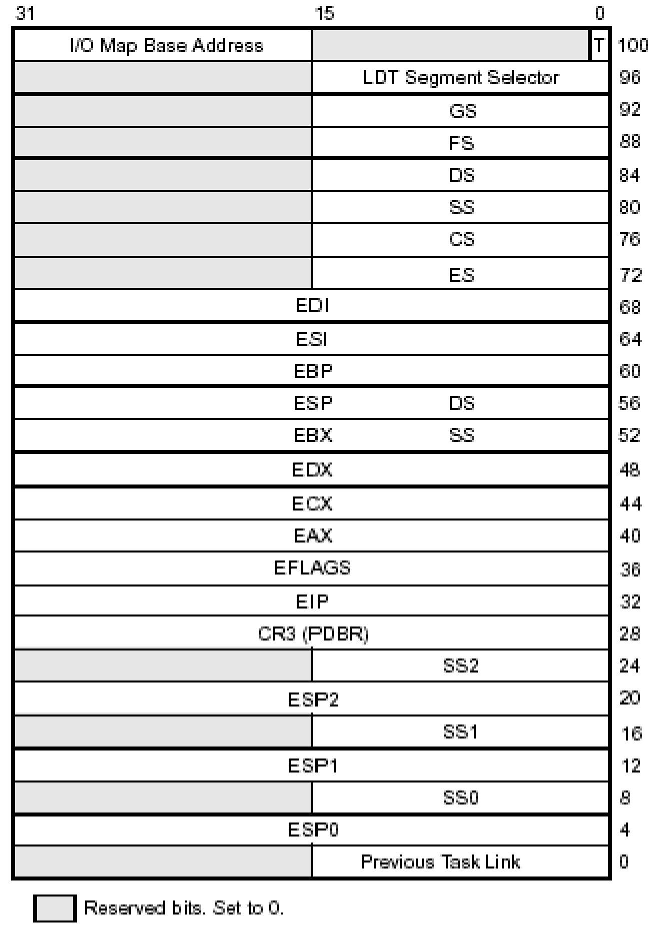


Şekil 4.28 : TSS Yapısı ve Bellek Bölgelerine Erişim

4.10.1 TSS Yapısı

Bir sürecin TSS yapısında şu bilgiler tutulmaktadır:

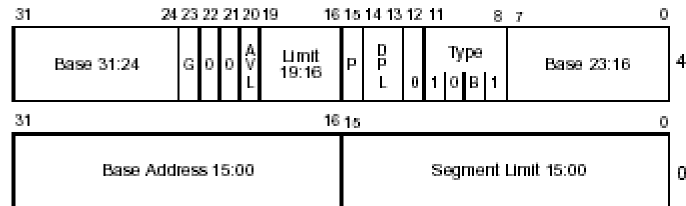
- Sürecin segment selektörleri ile belirlenmiş çalışma sahası (CS,DS,SS,ES,FS ve GS yazmaçları)
- Genel amaçlı yazmaçlar (EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP)
- EFLAGS yazmaçı
- EIP yazmaçı
- Sayfa dizin tablosu yazmaçı (CR3)
- TR yazmaçı
- LDTR yazmaçı
- Giriş/Çıkış bit haritası ve taban adresi
- 0., 1. ve 2. seviye yığıtlara ait yazmaçlar
- Bir önceki çalışan sürecin TSS'sine bağ



Şekil 4.29 : TSS Yapısı

4.10.2 TSS Tanımlayıcısı

TSS de , diğer segmentler gibi bir segment tanımlayıcısı ile ifade edilir.TSS'ye ait tanımlayıcılar sadece GDT tablosunda yer almaktadırlar.



Şekil 4.30 : TSS Tanımlayıcısı Yapısı

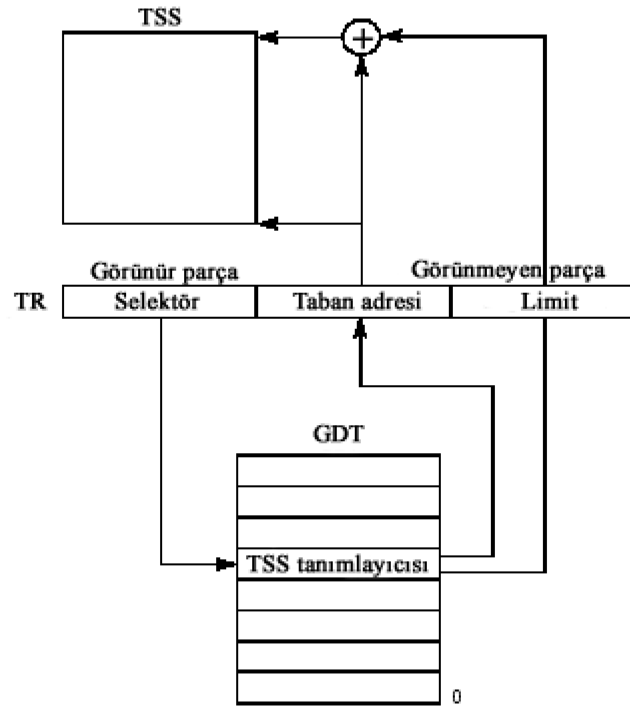
Tip sahasındaki B (Busy) biti, o sürecin o an çalışmakta olduğunu ya da çalışmayı beklediğini gösterir.

Limit sahası her zaman 67h değerine eşit veya uzun olmalıdır. Çünkü TSS yapısının kendisi zaten 67h uzunluğundadır.

4.10.3 TR Yazmaçı

TR (görev yazmaçı) , 16 bit segment selektörünü, 32 bit taban adresini, 32 bit segment limitini ve o an çalışmakta olan sürece ait TSS yapısının tanımlayıcı özelliklerini tutmaktadır.

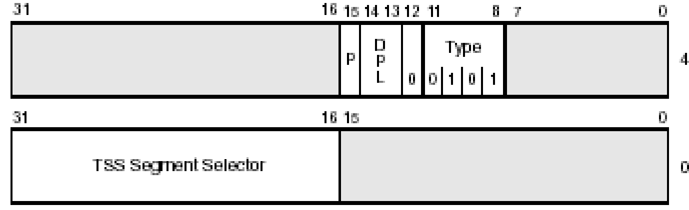
LTR komutu görev yazmaçını yüklemek için kullanılır.Bu komut da ayrıcalıklı bir komuttur ve 0. ayrıcalık düzeyinden çağırılmalıdır.



Şekil 4.31 : TR Yazmaçı ve TSS Tanımlayıcıları

4.10.4 Görev Kapısı Tanımlayıcıları

Bir görev kapısı, dolaylı yoldan bir göreve işaret etmektedir. Göre kapıları tanımlayıcıları GDT, LDT veya IDT'ye yerleştirilebilir.



Şekil 4.31 : Görev Kapısı Tanımlayıcısı Yapısı

Bir programın, bir görev kapısını paramtere olarak alan bir JMP veya CALL komutu işletmesi için, sürecin görev kapısının DPL'sinden daha öncelikli veya ona eşit olması gerekmektedir.

4.10.5 Süreçler Arası Geçiş

İşlemci, işletimi diğer bir sürece ancak 4 koşulda devreder:

- O an çalışan program GDT'deki bir TSS tanımlayıcısını parametre olarak alan bir CALL veya JMP komutunu işler
- O an çalışan program GDT'deki bir görev kapısı tanımlayıcısını parametre olarak alan bir CALL veya JMP komutunu işler
- IDT'deki bir tanımlayıcı bir görev kapısına işaret eder.
- O an çalışmakta olan program EFLAGS yazmaçındaki NT biti 1 iken bir IRET komutunu işletir.

İşlemci, süreçler arasında geçişi sağlamak için aşağıdaki adımları sırası ile işletmektedir:

1. O sürecin yeni sürece geçiş için gerekli koşulları sağlayıp sağlamadığı kontrol edilir. O sürece ait CPL , TSS tanımlayıcısının DPL'sine eşit veya ondan daha düşük olmalıdır.
2. Atlanacak TSS tanımlayıcısının o an hafızada olduğunu ve doğru bir limite sahip olduğunu kontrol edilir.

3. O an çalışmakta olan sürece ait TSS,yeni TSS ve geçişte kullanılacak tüm tanımlayıcıların hafıza sistemi tarafından sayfalandığı kontrol edilir.
4. O an çalışmakta olan sürecin durunu saklanır.
5. TR yazmaçı yeni sürece ait TSS ile yüklenir.
6. Yeni sürecin durumu TSS'den yüklenir. Genel amaçlı yazmaçlar, CR3 yazmaçı ve diğer bilgiler, ilgili yazmaçlara yüklenir.

5. GERÇEK ZAMANLI KORUMALI MOD 32 BİT BİR İŞLETİM SİSTEMİ GERÇEKLEŞTİRİMİ

Bir işletim sisteminin yazılabilmesi için öncelikle temel işletim sistemi kavramları üzerinde anlaşmaya varılmalıdır. İşletim sisteminin hangi özelliklere sahip olması gerektiği tespit edilmelidir.

İşletim sistemlerinin temel özelliklerinden biri onun işlemciyi dağıtma mekanizmasıdır. Gerçekleştirilen işletim sistemi dağıtık olmayan, tek kullanıcılı ve çok programlı bir işletim sistemi olarak tasarlanmıştır.Hafızada birden fazla süreç aynı anda çalışmaktadır. İşlemci paylaştırıcısı işlemciyi ele geçiren (preemptive) bir algoritma kullanmaktadır.

Hafıza yönetiminde süreçler arası koruma sayfalama mekanizması ile sağlanmaktadır. Her sürece bir sayfa tablosu atanmıştır ve süreçlerin bu hafıza tablosu ile belirtilen adres sahası dışında bölgelere erişimi kısıtlanmıştır.

Sistemdeki kesmelerin yönetimide ise hız ön planda yer almaktadır. Sistemdeki zamanlayıcı kesmesi üzerine yerleştirilen işlemci paylaştırıcısı kodunun kısa ve hızlı işletilebilir olmasına dikkat edilmişltir. Ayrıca klavye kesmesi yöneticisinin de aynı koşulları sağlaması ön planda tutulmuştur.

İşletim sistemi tasarlanırken sadece iki önemli fonksiyonu yönetmesi öngörülmüştür. Bunlar süreç yönetimi ile hafıza yönetimidir. Süreç yönetimi kapsamında işlemcinin süreçler arasında paylaştırılması ,süreçlerin yaratılması ve sonlandırılması, süreçlerin durdurulması ve tekrar çalıştırılması gerçekleştirilmiştir. Hafıza yönetimi kapsamında ise, hafızanın hangi bölümlerinin kullanılıp hangi bölümlerinin kullanılmadığının takipinin yapılması, süreçlere atanmış belleğin geri alınması ve süreçlere bellek tahsisi gerçekleştirilmiştir.

Gerçek anlamda intel 386 mimarisi işlemcileri üzerinde çalışan bir işletim sisteminin yazılabilmesi için öncelikle bu aileye ait işlemcilerin mimarisi ve çalışma mekanizmaları incelenmiştir. Ayrıca işletim sisteminin temel dosyalarının hafızaya alınması için FAT12 dosya sistemi incelenmiştir. Tüm bu incelemelerin sonucunda, gerçekleştirim içinm temel bilgiler elde edilmiş, **Gerçek Zamanlı Korumalı Mod 32 Bit Bir İşletim Sistemi Çekirdeği** kodlanmıştır.

5.1 İşletim Sistemi Açılış Adımları

5.1.1 İşletim Sisteminin Hafızaya Yüklenmesi (Boot.asm)

Boot.asm dosyası , işletim sistemini çekirdeği üzerinde işlem yapacak INITSYS.BIN isimli dosyayı ve işletim sistemi çekirdeği olan KERNEL.BIN isimli dosyayı hafızaya yüklemekten sorumlu, disketin başlangıç sektörüne (boot sector) yazılan dosyadır.

Bu işlemler yapılırken , sırası ile şu adımlar gerçekleştirilir:

1. Boot.asm dosyası disketin ilk sektörüne yazıldığı için, BIOS sistem ilklemelerini yaptıktan sonra bu dosyayı hafızanın 0x7C00 fiziksel adresine (07C0h:0000h) yükler ve buraya atlar.
2. Boot.asm dosyası FAT12 dosya sistemi yapısı içerisinde yer alan kök dizinini BIOS kesmeleri yardımkı ile okuyup hafızaya alır (2000h:0000h adresine). Kök dizini bilindiği gibi 19. sektördür.
3. Kök sizini hafızaya alındıktan sonra , FAT12 dosya sistemi yapısı içerisindeki FAT tablosu BIOS kesmeleri yardımı ile okunup hafızaya alınır (3000h:0000h adresine). İlk FAT tablosu 1. sektörden itibnaren başlamaktadır.
4. Yükleme işlemlerinden sonra, hafızaya alınmış olan kök dizini girdileri teker teker dolaşılarak INITSYS.BIN dosyası aranır ve bu dosyaya ait ilk veri kümesi bulunur.
5. İlk veri kümesi yardımı ile, hafızaya alınmış FAT tablosu kullanılarak dosyaya ait diğer veri kümeleri de bulunup hafızaya alınır (8000h:0000h adresine).

6. Yükleme işlemi KERNEL.BIN dosyası için de aynen tekrarlanır. (1000h:0000h adresine alınır.)
7. Yüklenmiş olan dosyalardan INITSYS.BIN dosyasına atlanarak (8000h:0000h adresine), işletim sistemi ilklemelerinin yapılması sağlanır.

Böylelikle sistemdek,i tüm dosyalar hafızaya yüklenmiştir ve sistem için ana ilkleme işlemleri yapılacaktır.

5.1.2 İşletim Sistemini İlkleme İşlemleri (Initsys.asm)

İşletim sistemini ilkleme ve korumalı moda geçiş işlemlerinden Initsys.asm dosyası sorumludur.

Bu dosya yardımı ile yapılan işlemler şunlardır:

1. Öncelikle tüm kesmeler kapatılır. Böylelikle sistem ilklemesi sırasında oluşabilecek hatalar önceden önlenmiş olur.
2. 1000h:0000h fiziksel adresine yüklenmiş olan çekirdek kodu, 0000h:0000h fiziksel adresine taşınır. Böylelikle çekirdek doğrusal ve güvenilir bir bölgeye kopyalanmış olur.
3. Geçici GDT ve IDT tabloları GDTR ve IDTR yazmaçlarına yüklenir.
4. 4 GB ‘lık bellek bölgesinin tamamına erişim için 32 adet adres bacağından 20. ‘sinin aktiflenmesi gerekmektedir. Yani A20 kapısı aktif hale getirilir.
5. Sistemdeki PIC (Priority Interrupt Controller) çipi programlanarak, sistemdeki donanım kesmelerinin (IRQ 0 –IRQ15) 0x70 ve 0x7f kesme vektörlerini kullanması sağlanır. Yani IRQ 0 kesmesini kullanan zamanlayıcı bir kesme isteğinde bulunduğunda, IDT tablosundaki 0x70

vektörü ile işaret edilen kesme yöneticisi fonksiyon çalışacaktır.

6. Korumalı moda geçilir.
7. Kernel koduna atlanır.

Yukardaki işlemler, ana sistem ilklemeleri olup, daha sayfalama mekanizması aktif hale getirilmemiştir.Çekirdek kodunun başlangıcında bulunan **start.asm** dosyası ile , sistemin esas tabloları oluşturulur..

5.1.3 Ana Çekirdek İlklemeleri (Start.asm)

İlk sistem ilklemesinden sonra, işletim sisteminin esas tablolarının oluşturulması işlevi yerine getirilmelidir. Start.asm dosyası ile sistemdeki ana GDT tablosu oluşturulur. IDT tablosu ise doldurulmamış halde, hafızada güvenli bir yeri işaret edecek şekilde atanmaktadır.

Sayfalama işleminde esas nokta sayfa tablolarıdır. Sayfa tablolarının oluşturulması ve sayfa dizin tablosu yazmaçının (CR3) yüklenmesi işlevi yerine getirilir.

En son işlem olarak çekirdek koduna atlanır.

5.1.4 Çekirdeğe Giriş

Çekirdek koduna atlandıktan sonra, sırası ile şu adımlar gerçekleştirilir:

1. Önce video sistemi ilklenmelidir. Video sistemi bellek bölgesi atamaları ve gerekli işaretçi ayarlamaları yapılır.
2. Sistemdeki IDT tablosuna, istisnaları işleyecek yöneticiler atanır.
3. Sistemdeki zamanlayıcı, 100 Hz frekansında çalışması için ayarlanır.
4. Sistemdeki donanım kesmelerine ilgili yönetici fonksiyonlar atanır.
5. İşlemci paylaştırıcısı, zamanlayıcı kesmesi üzerine yerleştirilir. Bu işlemde yine IDT tablosuna uygun girdiler eklenir.
6. Klavye yöneticisi IDT tablosuna eklenir.

7. Sayfalama mekanizması için gerekli tablolar doldurulup, bu mekanizma aktif hale geçirilir.
8. Kesmeler aktif hale getirilir.
9. Sistemde her zaman bulunacak init süreci oluşturulur.
10. Init süreci sistem komut yorumlayıcısını çalıştırır.

Tüm bu işlemlerden sonra, sistemimiz çalışmaya hazırdır. İşletim sistemi, kullanıcı komutlarını beklemektedir.

5.2 İşletim Sistemi Temel Veri Yapıları ve Sabitleri

İşletim sisteminin süreçlerin takipini yapması, süreçlere bellek ataması , sistemdeki hafıza bloklarının takipini yapması ; içsel olarak tuttuğu veri yapıları sayesinde olmaktadır. Öncelikle bu veri yapıları sırası ile incelenecektir.

5.2.1 Tanımlayıcı ve Kapı Yapıları

İncelenen intel-386 ailesi mimarisinde bellek bölgelerine tanımlayıcılar ve yardımı ile ulaşılıyordu. Tanımlayıcılar aşağıdaki gibi bir C dili yapısı ile ifade edilirler. Bu yapıdaki sahalar, bire bir intel mimarisindeki tanımlayıcı sahaları ile örtüşmektedir. Görüldüğü gibi tanımlayıcımız 8 byte yani bir işaretsiz tamsayı uzunluğundadır.

```
// intel 32 bit mimarisindeki Bellek Tanımlayıcısı (Descriptor) yapısı
struct i386_Descriptor{
unsigned int    limit_low:16 ,      // Limit 0-15
                base_low :16 ,      // Base 0-15
                base_mid :8  ,      // Base 16-23
                access_rights:8 ,   // Descriptor'un erişim hakları
                limit_high:4 ,      // Limit 16-19
                size:4 ,            // Descriptor'un büyüklük bilgileri
                base_high:8 ;       // Base 24-31
};
```

Sistemdeki IDT tablosunu ve istisna durumlarına ait yöneticileri kesme ve yazılım kesmeleri kapıları ile ifade ediyorduk. Dolayısıyla, IDT tablosunu doldurmak

için bir kapı yapısına ihtiyaç vardır. Aşağıda C dilindeki bir kapı yapısı gösterilmiştir.

```
// intel 32 bit mimarisindeki Kapı (Gate) yapısı
struct i386_Gate{
unsigned int   offset_low:16 ,      // Offset 0-15
               selector:16 ,        // Segment selector 0-15
               p_count:8 ,          // Parametre sayısı (4 bit)
               access_rights:8 ,    // Kapının erişim hakları
               offset_high:16  ;    // Offset 16-31
};
```

Bu kapı yapısı sayesinde sistem için hayati rol oynayan IDT tablosu doldurulacaktır. Kapı yapısı alt seviyeli assembly dilini kullanırken karşılaştığımız zorlukları önleyecektir.

Descriptor.h dosyasında yer alan sabitler yardımı ile kapılar ve tanımlayıcılar doldurulacaktır. Tanımlamalar sayesinde alt seviyeli assembly dilini kullanırken karşılaşılan zorluklardan soyutlanılarak daha rahat bir erişim ortamı sağlanmıştır.

Tanımlamalar, intel ailesine ait tanımlayıcıları doldurmak için gerekli olan bit değerlerini soyutlamıştır ve böylelikle daha okunur ve hataların daha çabuk algılanabileceği kod parçaları yazılabilmiştir.

```
// Bellek tanımlayıcısını doldurmak için gerekli olan sabitler aşağıda
// tanımlanmıştır...

// Segment büyüklüğü ile ilgili özellikler
#define PAGE_GRANULARITY  0x80    // segment büyüklüğü sayfa
                                  //ölçüsünde
#define SEGMENT_32_BIT    0x60    // 32 bit segment
#define AVAILABLE         0x10    // sistem yazılımları tarafından

                                  //kullanıma uygun

// Erişim hakları ile ilgili özellikler
#define PRESENT  0x80       // segment hafızada ve kullanılabilir.
#define DPL1     0x20       // Descriptor Privilege Level=1
#define DPL2     0x40       // Descriptor Privilege Level=2
#define DPL3     0x60       // Descriptor Privilege Level=3

// Code veya Data segmentleri için özellikler
#define DATA_READ 0x10                          // read only
#define DATA_READWRITE 0x12                     // read/write
#define STACK_READ 0x14                         // read only
#define STACK_READWRITE 0x16                    // read/write
#define CODE_EXEC 0x18                          // exec only
#define CODE_EXECREAD 0x1A                      // exec/read
#define CODE_EXEC_CONFORMING 0x1C               // exec only
conforming
#define CODE_EXECREAD_CONFORMING 0x1E   // exec/read
conforming

#define ACCESSED 0x01             // o bellek bölgesine erişim oldu mu?

// Sistem tanımlayıcıları için özellikler
#define LDT 0x02                    // Local Descriptor Table
#define TASK_GATE 0x05              // Sistem Görev Kapısı
#define TSS 0x09                    // Görev Durum Segmenti (TSS)
#define CALL_GATE 0x0C              // Çağırma Kapısı
#define INTERRUPT_GATE 0x0E         // Kesme Kapısı
#define TRAP_GATE 0x0F              // Yazılım Kesme Kapısı
```

5.2.2 Sayfa Tablolarını Doldurmak İçin Kullanılan Sabitler

İntel mimarisi içerisinde açıklanan sayfa tabloları ve sayfa dizin tablolarına ait girdileri doldurmak için **Memory.h** içerisinde bazı sabitler tanımlanmıştır.

Aşağıda bir sayfa dizin tablosunu doldurmak için kullanılan sabitler gösterilmiştir.

```
// Sayfa Dizin Tablolarini doldurmak ve maskeleme için kullanılan sabitler
#define PDE_PRESENT                  0x00000001
#define PDE_WRITE                    0x00000002
#define PDE_USER                     0x00000004
#define PDE_WRITE_THROUGH            0x00000008
#define PDE_CACHE_DISABLED           0x00000010
#define PDE_ACCESSED                 0x00000020
#define PDE_GLOBAL                   0x00000100
#define PDE_PT_BASE                  0xFFFFF000
```

Bir sayfa dizin tablosu girdisi doldurulurken , bu sabitlerin OR'lanması ile gerekli girdi oluşturulur. PDE_PT_BASE sabiti ise, sayfa dizin tablosu girdisinden sayfa tablosu taban adresini almak için kullanılır. Sayfa dizin tablosu ile PDE_PT_BASE sabiti AND'lenerek, o girdinin gösterdiği sayfa tablosu taban adresi elde edilir.

```
// Sayfa Tablolarini doldurmak ve maskeleme için kullanılan sabitler
#define PTE_PRESENT                  0x00000001
#define PTE_WRITE                    0x00000002
#define PTE_USER                     0x00000004
#define PTE_WRITE_THROUGH            0x00000008
#define PTE_CACHE_DISABLED           0x00000010
#define PTE_ACCESSED                 0x00000020
#define PTE_DIRTY                    0x00000040
#define PTE_GLOBAL                   0x00000100
#define PTE_P_BASE                   0xFFFFF000
```

Sayfa tabloları doldurulurken ise, yukarıdaki sabitler sayfa taban adresi ile OR'lanarak girdiler oluşturulur. PTE_P_BASE sabiti ile sayfa tablosu girdisi AND'lenerek, o sayfa tablosunun gösterdiği sayfanın taban adresi elde edilir.

Sayfa tablosu ve sayfa dizin tablosu girdileri üzerinde işlem yapmak için, hafıza yönetiminin kullandığı diğer sabitler ise şunlardır:

```
//Hafıza yönetimi sistemi için gerekli makro ve sabitler.
#define PAGE_SIZE        4096    //sayfa büyüklüğü
#define PDE_SHIFT        22      //sanal adresten PD'yi almak için
#define PDE_MASK         0x3FF   //PD'yi almak için gerekli maske
#define PTE_SHIFT        12      //sanal adresten PT'li almak için
#define PTE_MASK         0x3FF   //PT'yi almak için gerekli maske
```

Sayfa büyüklüğü 4KB boyutundadır. Bir sanal adresten o adrese ait sayfa dizin tablosu ve sayfa tablosu girdilerini öğrenmek için yine yukarıdaki sabitler kullanılır. Hatırlanacağı gibi intel mimarisinde sanal adres 32 bitten ve üç parçadan oluşuyordu. Bunlar sırası ile 10 bitlik sayfa dizin tablosu girdi numarası, 10 bitlik sayfa tablosu girdi numarası ve 12 bitlik ofset değeri idi. Bir sanal adrese ait sayfa dizin tablosu girdi numarasını bulmak için , sanal adresi 22 bit sağa kaydırıp 0x3FF ile OR'lamalıyız. Bu işlemler için gerekli sayısal değerler sabitler olarak tanımlanmış ve böylece işlemler daha okunaklı hale getirilmiştir.

5.2.3 Adres Sahası Yapısı

Bir sürece ait adres sahası , yani o sürecin yaşantısı boyunca erişebileceği adres bölgeleri, o sürece ait sayfa dizin ve sayfa tabloları ile belirlenmiştir. Bu tabloları tutmak için işletim sistemi adres sahası yapısını kullanmaktadır.

```
// süreçlerin sayfa ve sayfa dizin tablolarının tutulduğu yapı
struct address_space
{
        unsigned long pdir[1024];               //sayfa dizin tablosu
        unsigned long user_ptable_0[1024];      //kullanıcı sayfa tablosu
        unsigned long user_ptable_1[1024];      //kullanıcı sayfa tablosu
};
```

Görüldüğü gibi her sürece ait bir sayfa dizin tablosu ve iki adet sayfa tablosu verilmiştir. Bir sayfa tablosu ile 4MB adres sahası adreslenebileceği için, süreçlere 8MB'lık bellek bölgesi görünürdür. Her sayfa dizin tablosu ve sayfa tablosu 32 bitlik uzunluğa sahiptir.

Sistemde her süreç yaratıldığı anda, o sürece ait sayfa dizin ve sayfa tabloları doldurulmalıdır.

5.2.4 Fiziksel Bellek Takipi İçin Kullanılan Yapı

Sistemde bulunan çerçevelerin işletim sistemi tarafından takipinin yapılması için bir veri yapısı gerekmektedir. Bu veri yapısı, basit bir dizi olabilir.

```
static char FrameMap[NUM_FRAMES]={0,};
```

Eğer dizinin o elemanı 0 ise çerçeve boş, 1 ise çerçeve doludur. Örneğin dizinin 4. elemanı 1 ise, 4*4096 fiziksel başlangıç adresine sahip çerçeve o an bir süreç ya da işletim sistemi tarafından kullanılıyordur.

5.2.5 Süreç Durum Sabitleri

Süreçler yaşantıları boyunca çeşitli kuyruklarda tutulurlar ve bir durumdan öteki duruma geçerler. Süreçlere ait durum bilgilerini tutmak için içsel olarak bazı sabitlere gereksinim vardır.

```
#define TASK_RUNNING        1
#define TASK_READY          2
#define TASK_WAITING        3
#define TASK_TERMINATED     4
```

Süreçler hazır, çalışıyor, bekliyor ya da sonlanmış olabilirler. Bu durumların tutulması ile, süreçlerin hangi kuyruğa yerleştirilecekleri belirlenir.

5.2.6 Süreç Durumu Yapısı

İntel ailesi işlemcileri, bir sürece ait yazmaç değerlerini TSS yapısında tutuyordu. Bu yapı ile birebir öretüşen C yapısı ile, o süreçlerin takipinin C dili yapılması kolaylaşmıştır.

```
//Intel 386 ve sonrası işlemcileri için donanımsal olarak tanımlanmış
// TSS (task state segment) yani süreç durum bilgilerini tutan yapı
struct task_state_segment
{
        long    Previous_Link;
        long    ESP0;
        long    SS0;
        long    ESP1;
        long    SS1;
        long    ESP2;
        long    SS2;
        long    CR3;
        long    EIP;
        long    EFlags;
        long    EAX;
        long    ECX;
        long    EDX;
        long    EBX;
```

```
        long    ESP;
        long    EBP;
        long    ESI;
        long    EDI;
        long    ES;
        long    CS;
        long    SS;
        long    DS;
        long    FS;
        long    GS;
        long    LDT_selector;
        long    IO_Bitmap_Base_Adress;
};
```

Görüldüğü gibi o sürece ait tüm bilgiler bu yapı içerisinde tutulmaktadır. Yapı intel işlemcileri için tasarlanmışsa da, diğer işlemciler üzerinde süreç durumlarını saklamak için yazılımsal olarak da kullanılabilir.

TSS yapıları, GDT tablosuna tanımlayıcılar yoluyla işlenmelidir.

5.2.7 Süreç Yapısı

Sistemdeki süreçlerin takipinin yapılabilmesi için gerekli olan yapı bir süreç yapısıdır. Süreç yapısı, o süreçle ilgili tüm bilgileri tutmalıdır.

```
// İşletim sistemimizde yer alacak olan bir sürecin tamamen iç yapısını
// tutan C yapısı
struct Task
{
        //Sürece ait hafıza tablolarınu tutan yapı-------------
        struct address_space *addr_space;
        //---------------------------------------------------------------

        //sürece ait özel veriler------------------------------------
        unsigned long ID;                       //süreç ID
        char Durum;                             //süreçin durumu
        //-----------------------------------------------------------------

        //-----------------------------------------------------------------
        //sürecin klavye tamponu
        char keyboard_buffer[256];
        //-----------------------------------------------------------------

        // sürecin segment bilgileri--------------------------------
        //(fiziksel adres olarak)
        unsigned long code_segment_base;
```

```
	//---------------------------------------------------------------

	//sürecin durum bölgesi-------------------------------------
	struct task_state_segment Tss;	//sürecin durumu saklanacak
	//---------------------------------------------------------------

	//---------------------------------------------------------------
	// o sürecin sürec kuyruğundaki diğer elemanlar
	// ile bağlantısını sağlayan değişkenler
	struct Task *onceki_surec;
	struct Task *sonraki_surec;
};
```

Öncelikle, süreç yapısı o sürece ait adres sahası bilgisini tutmalıdır. Ayrıca o sürecin durumu, yazmaçları ve diğer durum bilgileri de bu yapı tarafından tutulur. Bu yapı içerisine, o sürece ait klavye tamponu da koyulmuştur. Önceki süreç ve sonraki süreç bağları ile, listedeki diğer süreçlere ulaşım da sağlanmıştır.

Süreç yapıları süreç listelerinde tutulurlar.

5.2.8 Süreç Listeleri

Süreçler durumlarına göre süreç listelerinde tutulurlar. Süreç listeleri, aynı durumda bulunan süreçleri bir bağlı liste yapısı içerisinde tutarlar.

```
// sistemdeki süreçleri tutan süreç listesi için bir yapı
struct Liste
{
	struct Task *liste_basi;
	struct Task *liste_sonu;
	unsigned long eleman_sayisi;
};
```

Sistemde çalışmaya hazır süreçleri tutan hazir süreç listesi, bir giris/çıkış işlemi üzerinde bekleyen süreçleri tutan bekleyen süreç listesi gibi listeler bulunmaktadır.

İşlemci dağıtıcısı süreçleri tutan bu listeleri dolaşarak uygun sürece işlemciyi vermektedir. Süreç listeleri sayesinde, süreçlerin takipi ve durumları kolayca izlenebilir.

5.2.9 Sistem Çağırımları Tablosu

İşletim sisteminin sunduğu servis fonksiyonlarına ait adresler, yine dizi gibi basit bir veri yapısı kullanılarak oluşturulabilir.

```
unsigned long Zeugma_Api_Table[]={
                    (unsigned long)Cls,                      //0
                    (unsigned long)Set_Cursor,               //1
                    (unsigned long)Print,                    //2
                    (unsigned long)Println,                  //3
                    (unsigned long)Exec,                     //4
                    (unsigned long)Exit,                     //5
                    (unsigned long)Set_Color,                //6
                    (unsigned long)Set_Background_Color,     //7
                    (unsigned long)Scanf,                    //8
                    (unsigned long)surecBilgisi,             //9
                    (unsigned long)killProcess,              //10
                    (unsigned long)Print_Sayi_Hex            //11
                                                         };
```

Servis dizisi, işletim sisteminin sunduğu tüm API’lerin adreslerini içermektedir. Bu sayede, tüm servis fonksiyonlarına yönelik referanslar sistem içerisinde tutulmaktadır.

```
#define API_CLS_INDEX                  0
#define API_SET_CURSOR_INDEX           1
#define API_PRINT_INDEX                2
#define API_PRINTLN_INDEX              3
#define API_EXEC_INDEX                 4
#define API_EXIT_INDEX                 5
#define API_SET_COLOR_INDEX            6
#define API_SET_BG_COLOR_INDEX         7
#define API_SCANF_INDEX                8
#define API_SUREC_BILGISI_INDEX        9
#define API_KILL_PROCESS_INDEX         10
#define API_PRINT_SAYI_HEX_INDEX       11
```

İşletim sisteminin sunduğu servislere ait numaralar ise sabitler olarak tanımlanmıştır. Bu sayede, istenen servis belirlenir ve dizinin o elemanına karşılık gelen fonksiyon adresine atlanır.

Yani servis tablomuz bir dağıtıcı sistem tablosudur.

5.3 İşletim Sisteminde Süreç Yönetimi

Sistemdeki veri yapıları içerisinde süreç yönetiminde kullanılanlar listeler ve süreçlere ait yapılar olarak sıralanabilir.

Süreç yönetimi kapsamında yeni süreçlerin yaratılması, süreçlerin sonlandırılması ve süreçlere işlemci dağıtımı gibi önemli fonksiyonlar gerçekleştirilmiştir.

5.3.1 Sistemdeki Listeler

Süreçlerin işletim sistemi içerisindeki listelerde tutulduğundan daha önce bahsedilmişti. Sistemde süreçlere ait üç önemli liste bulunmaktadır.

```
// Sistemdeki çalışmaya hazir tüm süreçlere ait süreç yapılarını
// tutan liste (Ready Queue)
struct Liste Hazir_Surec_Listesi;

// Sistemdeki IO bekleyen tüm tüm süreçlere ait süreç yapılarını
// tutan liste (Waiting Queue);
struct Liste Bekleyen_Surec_Listesi;

// Sistemdeki çalışmasını bitirmiş süreçleri tutan liste
// (Terminated Queue)
struct Liste Bitmis_Surec_Listesi;
```

Hazır süreç listesi, o an için çalışmaya hazır süreçleri tutmaktadır. Bekleyen süreç listesi ise bir giriş/çıkış işlemi üzerinde bekleyen süreçleri tutmaktadır. Bitmiş süreç listesi ise sonlanmış süreçleri tutmaktadır.

Sistemdeki temel süreç yönetim mekanizması bu listeler üzerinden gerçekleşmektedir.

```
void Remove_Task(struct Liste *lst,struct Task *task);
void Insert_Task(struct Liste *lst,struct Task *task);
```

Yukarıda süreç listelerine süreç ekleyen ve çıkartan iki önemli fonksiyon prototipi listelenmiştir. Temel liste işlemlerini yürüten bu fonksiyonlar, süreçler üzerinde işlem yapan fonksiyonlarca kullanılır.

```
//o an çalışan süreç
struct Task *aktif_surec;
```

Sistemde o an çalışan sürece ait referans da bir değişkende saklanmalıdır ki bu değişken çok önemlidir. Süreçler arasında geçiş mekanizmasında temel rol oynamaktadır.

5.3.2 İşlemci Dağıtıcısı (Scheduler)

Süreçler arası geçiş mekanizmasını sağlayan , işlemci dağıtıcısının sağladığı algoritma ile uyguladığı adımlardır.

Sistem çok programlı ve işlemciyi ele geçiren (preemptive) bir işlemci dağıtıcısı algoritmasına sahiptir. Algoritma FCFS ve Round-Robin algoritmalarını gerçekleştirmiştir.

İşletim sistemi ilklemeleri içerisinde, öncelikle işlemci dağıtıcısı kodu , sistem zamanlayıcısı kesme isteği vektörüne işlenmelidir (IRQ 0) . Yani IDT'ye ilgili girdi girilmelidir.

```
//Timer kesmesini işleyecek kod parçası IDT'ye işlensin
Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[0x70],
                    (unsigned long)Timer_Interrupt,    //handler
                    sel_KernelCS,                      //selector
                    0,                                 //parametre sayisi
                    INTERRUPT_GATE|PRESENT);  //access
```

İlgili ilkeleme işlemlerinden sonra , artık her zamanlayıcı kesmesinde (100Hz), işlemci dağıtıcısı aktif hale gelecek ve süreçlere işlemciyi dağıtacaktır.

İşlemci dağıtıcısı, ilk başta bitmiş süreç listesinde eleman olup olmadığını kontrol eder. Eğer eleman var ise,o süreç sonlandırılır.

```
// eğer var ise bitmiş süreç listesinden bir süreci çıkart ve onu
// sistemden tamamiyle sil
freeTask();
```

Bu işlemlerden sonra, hazır süreç listesinin ilk elemanı listeden çıkartılır.

```
// hazir süreç listesinden, liste başı sürecini çıkart
task=lst->liste_basi;
Remove_Task(lst,task);
```

Sistemde, o an çalışmakta olan süreç ise durumu "hazır" yapılarak bekleyen süreçler kuyruğuna koyulur.

```
//aktif süreç, artık çalışmıyor...
aktif_surec->Durum=TASK_READY;
//hazir süreç listesine ekle
Insert_Task(lst,aktif_surec);
```

Hazır süreç listesinden çıkartılan sürecin durumu "çalışıyor" yapılıp "switch_to" makrosu ile o sürece geçiş yapılır.

```
//çalışacak olan sürecimiz artık yeni süreçtir
aktif_surec=task;

//yeni süreç çalışacağı için durumu "ÇALIŞIYOR" yapılıyor
task->Durum=TASK_RUNNING;

//Task switching işlemi yapılıyor.
 switch_to(task->ID);
```

Yani esas işlem o an çalışmakta olan süreci hazir süreç listesine atmak ve hazir süreç listesinin başındaki süreci de çalıştırmaktır.

5.3.3 Süreç Yaratılması ve Exec Fonksiyonu

Süreç yaratılması için temel fonksiyon Exec fonksiyonudur. Exec fonksiyonu kendi içerisinde CreateProcess fonksiyonunu çağırmaktadır.

Süreç yaratılmasında temel işlemler ve adımlar şunlardır:

Öncelikle o sürece ait bilgilerin tutulduğu bir veri yapısı olan süreç yapısı için, hafıza yöneticisinden bellek isteminde bulunulmalıdır.

```
//Sürece ait bilgilerin işletim sistemi tarafından tutulması için gerekli
fiziksel //bellek ayırma işlemleri...
//boş bir sayfa al
//sürece ait bilgiler bu sayfa içerisinde tutulacak
yeni_surec=(struct Task *)allocKernelPages(1);
```

Sistemden boş bir sayfa istemi allocKernelPages fonksiyonu ile yapılmaktadır. (Bu fonksiyon hafıza yönetimi kapsamında incelenecektir.)

Süreç yapısı elde edildikten sonra, bu yapının doldurulması gerekir. Sürece ait sayfa tablolarının doldurulması bir sonraki adımdır.

```
//sürece ait sayfa tablolarını doldur
createPageTables(yeni_surec);
```

Sayfa tablolarının doldurulması, hafıza yönetimi kapsamında ayrıntılı incelenecektir.

Sayfa tabloları doldurulduktan sonra, o süreç için bir ID alınmalı ve o sürece ait yazmaçlar ve diğer bilgiler ilklenmelidir.

Süreç ID'si o an sistemde var olan süreç sayısı olarak atanır.

```
//süreçin ID'sini süreç listesindeki eleman sayısı olarak atanıyor
yeni_surec->ID=Surec_Sayisi;
```

O sürece ait durum bilgileri ise fill_TSS fonksiyonu ile doldurulur.

```
//segment selektörleri GDT değerlerini göstersin
task->Tss.CS=sel_UserCS;
task->Tss.DS=sel_UserDS;
task->Tss.ES=sel_UserDS;
task->Tss.FS=sel_UserDS;
task->Tss.GS=sel_UserDS;
task->Tss.SS=sel_UserDS;

//yığıt segmenti kullanıcı yığıt segmentini göstersin
task->Tss.ESP=(unsigned long)allocUserPages(1,
                                        task->addr_space-
>user_ptable_1)+4095;

//SS0 ve ESP0 kernel yığıt segmentini göstersin
task->Tss.SS0=sel_KernelDS;
task->Tss.ESP0=(unsigned long)allocKernelPages(1)+4095;

//sürecin tüm yazmaçlarını ilk başta sıfırla
task->Tss.EAX=0;
task->Tss.EBX=0;
task->Tss.ECX=0;
task->Tss.EDX=0;
```

```
task->Tss.ESI=0;
task->Tss.EDI=0;
task->Tss.EBP=0;

//flag değeri yazılıyor
task->Tss.EFlags=EFLG_IF | EFLG_IOPL3; //interrupt enable ve IOPL
task->Tss.EIP=0;
task->Tss.IO_Bitmap_Base_Adress=0;
task->Tss.LDT_selector=0;
```

Görüldüğü gibi sürece ait segment yazmaçları ve diğer genel amaçlı yazmaçlar ilklenmektedir. Sürecin sayfa dizin tablosu, CR3 yazma değeri olarak atanmaktadır. Tüm bu işlemlerden sonra sürecimiz ilklenmiştir. Sürece ait tüm veri yapıları oluşturulmuş ve bu veri yapıları başlangıç değerleri ile yüklenmiştir.

Sürece ait TSS yapısı oluşturulduktan sonra, bu yapı GDT tablosuna işlenmelidir. Bunun için fill_GDT fonksiyonu kullanılır. Sistemdeki GDT tablosu aşağıdaki yapıya sahiptir.

```
0->NULL desc.          Boş tanımlayıcı
1->KernelCS desc.      Kernel kod bölgesi tanımlayıcısı
2->KernelDS desc.      Kernel veri bölgesi tanımlayıcısı
3->UserCS desc.        Kullanıcı kod bölgesi tanımlayıcısı
4->UserDS desc.        Kullanıcı veri bölgesi tanımlayıcısı
5-> TSS0               0.süreç için TSS
6-> TSS1               1.süreç için TSS
 |
 |
 |
V
N-> TSS N              N.süreç için TSS
```

İlk yaratılan süreç 6. GDT girdisinden başlamak üzere süreçlere ait TSS tanımlayıcıları bu şekilde GDT’ye işlenir. Dolayısıyla, task->ID + 5 işleminini sonucu, sürece ait TSS tanımlayıcısının GDT’deki girdi numarasını vermektedir.

Sürece ait tüm işlemler tamamlandıktan sonra, EIP değeri, verilen adres değeri olarak atanır.

```
//süreç yapısı dolduruluyor...
 yeni_surec->code_segment_base=code_segment_base;
Yeni_surec->Tss.EIP=code_segment_base;
```

Süreç artık çalışmaya hazırdır. Bu nedenle, süreci hazir süreç kuyruğuna yerleştirirsek, işlemci dağıtıcısı işlemciyi belirli bir süre sonra sürece atayacak ve süreç çalışmaya başlayacaktır.

```
//süreç çalıştırılmaya hazır...
yeni_surec->Durum=TASK_READY;

.
.
.

Insert_Task((struct Liste *)phys_to_virt((unsigned long)
                                    &Hazir_Surec_Listesi),yeni_sure
c);

//süreç, süreç kuyruğuna yerleştirildi...Çalıştırılmaya hazır..
```

5.3.4 Süreçlerin Sistemden Çıkması ve Exit Fonksiyonu

Süreçler, çalışırken Exit sistem çağırımını yaparlarsa, çalışmaları sonlanır ve süreç yok edilir. Exit fonksiyonunun çalışma prensipi aşağıda açıklanmıştır.

Öncelikle o an çalışmakta olan ve Exit sistem çağırımını yapmış fonksiyon belirlenmelidir. Bunun için aktif_surec değişkeni kullanılır.

```
// o an aktif olan surecin sanal adresini al
surec=(struct Task **)phys_to_virt((unsigned long)&aktif_surec);
terminated_task=(*surec);
```

Bu işlemden sonra, aktif_surec değişkeni NULL yapılarak işlemci dağıtıcısı hazir süreç listesinden başka bir süreç almaya zorlanır.

```
// aktif sureci NULL yap
(*surec)=NULL;
```

En son adımda ise, süreç bitmiş süreçler listesine atılmalıdır.

```
// sureci bitmiş süreçler listesine koy
terminated_task->Durum=TASK_TERMINATED;
Insert_Task((struct Liste *)phys_to_virt((unsigned long)
                                        &Bitmis_Surec_Listesi),terminated_
task);
```

Süreç,bir sonraki adımda işlemci dağıtıcısı çalıştığında, freeTask fonksiyonu ile sistemden aldığı tüm bellek bölgelerini geri vererek sonlanacaktır.

5.3.5 Süreçlerin Kullanıcı Tarafından Öldürülmesi ve killProcess

ID'si verilen sürecin öldürülmesi işlemi, Exit fonksiyonu ile mantık olarak aynıdır. Öncelikle ID yardımı ile, sistemdeki tüm listeler dolaşılarak o süreç bulunur.

Bulunan süreç, bitmiş süreçler listesine yerleştirilerek işlemci dağıtıcısının bir sonraki adımında, tüm bellek bölgelerini sisteme freeTask fonksiyonu ile vererek sonlanacaktır.

```
// sureci bitmiş süreçler listesine koy
task->Durum=TASK_TERMINATED;
Insert_Task((struct Liste *)phys_to_virt((unsigned long)
                                        &Bitmis_Surec_Listesi),
task);
```

5.4 İşletim Sisteminde Hafıza Yönetimi

Sistemdeki veri yapıları içerisinde hafıza yönetiminde kullanılanlar adres sahası ve çerçeve takipinde kullanılan yapılar olarak sıralanabilir.

Hafıza yönetimi kapsamında fiziksel hafızanın takipi, sayfa tablolarının doldurulması, fiziksel belleğin sayfa tablolarına işlenmesi, süreçlere bellek dağıtımı ve süreçlerden alınan belleğin sisteme geri verilmesi gibi önemli fonksiyonlar gerçekleştirilmiştir.

5.4.1 Fizikse Hafıza Takipi ve allocPages Fonksiyonu

Sistemdeki fiziksel hafıza takipi, FrameMap veri yapısı ile yapılmaktadır. Daha önce bahsedildiği gibi çerçeve dizisinde bir eleman 0 ise o çerçeve boş, 1 ise o çerçeve doludur ve işletim sistemi veya bir süreç tarafından kullanılıyordur.

allocPages fonksiyonu sistemdeki çerçevelerin tahsisinden sorumludur. Parametre olarak verilen sayfa sayısı kadar ardışık çerçevenin fiziksel başlangıç adresini sürece döndürür ve o çerçevelere ait dizi elmanını 1 yapar.

```
// verilen sayfa sayısı kadar ardışık sayfayı, hafıza sistemi içe-
// risinde bul
for(i = 0; i < (NUM_FRAMES - num_pages); i++)
{
        //ardışık sayfaların da boş olup olmadığını kontrol et
        for(j = 0; j < num_pages; j++)
        {
                //eğer sayfa dolu ise, artık aramaya o sayfadan itibaren
                // başla
                if(FrameMap[i + j]==1)
                {
                        i += j;
                        break;
                }
        }
        // eğer istenen sayfa kadar ardışık sayfa var ise
        if(j == num_pages)
        {
                //sayfaları dolu olarak işaretle
                for(j = 0; j < num_pages; j++)
                {
                        FrameMap[i + j]=1;
                }

                //fiziksel adresi ata
                *physical_address = (void *) (i << 12);
                return STATUS_SUCCESS;
        }
}
```

Sayfa sayısı kadar ardışık eleman aranmaktadır. Eğer ardışık sayfa sayısı kadar çerçeve yoksa, hata mesajı verilir ve çerçeve tahsisi yapılamaz. Ardışık eleman bulunursa, o çerçevelere ait başlangıç fiziksel adres döndürülür.

5.4.2 Sistemdeki Sayfa Dizin Tablosu ve Kernel Sayfa Tabloları

İşletim sistemi içerisinde bir adet ana sayfa dizin tablosu bulunmaktadır.

```
// Sistemdeki kernel sayfa dizin tablosu (Start.asm'den)
// Sayfa dizin tablosunun fiziksel adresi = 0x000000
extern unsigned long Page_Directory_Table[1024];
```

Sayfa dizin tablosu işletim sistemi ilklemerinde önemli rol oynamaktadır.

İşletim sistemi çekirdeğinin de kendi içerisinde sayfa tabloları olmalıdır. Çünkü süreçlere ait sistem yapıları, işletim sistemi sayfa tablolarına işlenecektir ve diğer süreçlerin erişimi bu şekilde kısıtlanacaktır. İşletim sistemine ait üç adet sayfa tablosu sistem içerisinde yer almaktadır.

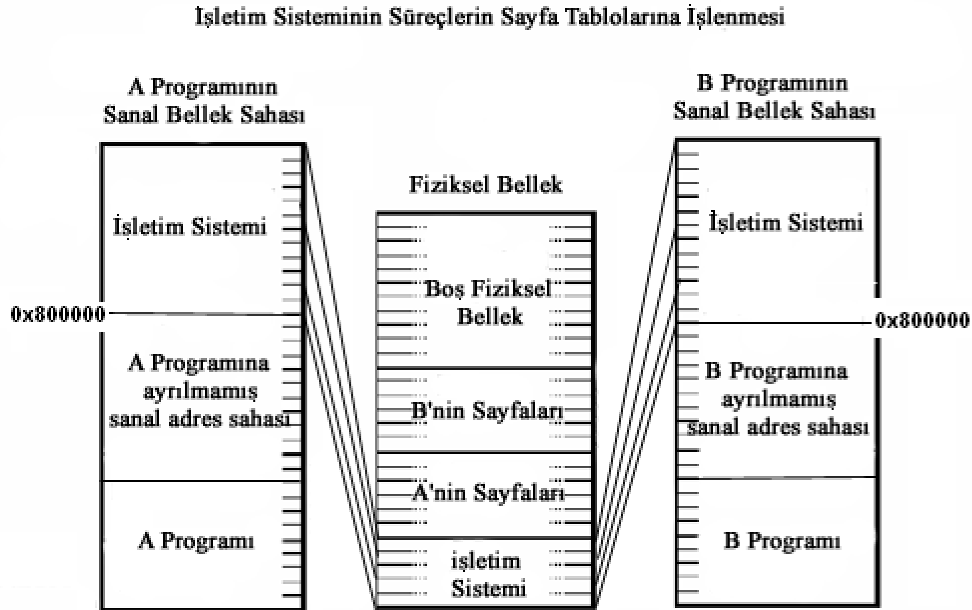
```
// 12 MB'lık bellek bölgesine erişim için 3 adet sayfa tablosu
// ayarlanmıştır.
unsigned long *KernelPageTable_Low_0;
unsigned long *KernelPageTable_Low_1;
unsigned long *KernelPageTable_High;

// Sistemdeki sayfa tablolarının fiziksel adresleri atanıyor.
// 12 MB'lık bellek adreslenebilecek...
KernelPageTable_Low_0  =(unsigned long *)0x00010000;
KernelPageTable_Low_1  =(unsigned long *)0x00011000;
KernelPageTable_High     =(unsigned long *)0x00012000;
```

İşletim sistemi çekirdeğine ait sayfa tabloları yukarıda blirtilen fiziksel adreslere atanmıştır. KernelPageTable_High, işletim sistemi çekirdeğinin ana sayfa tablosudur ve bu tablo her sürecin sayfa dizin tablosuna işlenir. Yani süreçlere ait ilk iki sayfa tablosu 8 MB'lık bellek bölgesine erişim için kullanılırken, sayfa dizin tablolarına işlenmiş 3. ayfa tablosu olan KernelPageTable_High işletim sisteminin ana sayfa tablosudur.

Bu sayfa tablosunun her sürecin adres sahasına işlenmesindeki amaç şudur: Süreçler her işletim seviyesi işletim seviyesine indiğinde, sayfa dizin tablosu yazmaçını değiştirmek, işletim sisteminin sayfa dizin tablosunu yüklemek performansı düşürmektedir. Ayrıca bu işlem her yapıldığında TLB kaşelerindeki veriler geçersiz yapılmaktadır ve bellek erişimi yavaşlamaktadır.

Eğer işletim sisteminin tabloları süreçlerin adres sahasında eklenirse, bu işlemlerden kurtulmuş olunur. Ayrıca bu tablo, sayfa dizin tablosuna eklendiğinde,

0x800000 adresinden yukarısını göstermektedir. Böylelikle işletim sistemi çekirdeği 0x800000 sanal adresinden yukarısında çalışıyormuş gibi gözükürken, kullanıcı süreçleri 0x000000-0x800000 sanal adresleri arasında çalışır gözükürler.



Şekil 5.1: Çekirdek Sayfa Tablolarının Süreçlerin Adres Sahasına Eklenmesi

İlk başta, çekirdeğe ait sayfa tabloları fiziksel adreslerle doldurlurken, süreçlerin adres sahasına eklencek tablo ise sadece kernel'e ait bölgelerin fiziksel adresleri ile doldurulup geri kalan kısmı boştur.

```
// Kernel Page Directory boşaltılıyor
// Sayfa tabloları bire bir fiziksel adresler ile dolduruluyor
for(i=0;i<1024;i++)
{
	Page_Directory_Table[i]=0;
	KernelPageTable_Low_0[i]=i*4096|PTE_PRESENT|PTE_WRITE|
PTE_USER;
	KernelPageTable_Low_1[i]=0;
	if((i<19) ||(i==0xb8))
		KernelPageTable_High[i]=i*4096|PTE_PRESENT|PTE_WRITE;
	else
		KernelPageTable_High[i]=0;
}
```

Görüldüğü gibi süreçlerin adres sahasına eklencek KernelPageTable_High sayfa tablosu, sadece kernel bölgelerine ait fiziksel adres sahalarını adreslemektedir ve geri kalan kısımlar boştur.

Sayfa tabloları doldurulduktan sonra, bu tablolara ait girdiler sayfa dizin tablosuna işlenmelidir.

```
//Kernel Sayfa dizin tablosuna gerekli girdiler konuluyor...
Page_Directory_Table[0]=(unsigned long)KernelPageTable_Low_0
                                        |PDE_PRESENT|PDE_WRITE;
Page_Directory_Table[1]=(unsigned long)KernelPageTable_Low_1
                                        |PDE_PRESENT|PDE_WRITE;
Page_Directory_Table[2]=(unsigned long)KernelPageTable_High
                                        |PDE_PRESENT|PDE_WRITE;
```

Bu işlemlerden sonra sayfalama mekanizması aktiflenebilir. Çünkü fiziksel hafızaya erişim ve sayfalama işlemi için gerekli sayfa dizin tablosu sağlanmıştır.

5.4.3 Sayfa Tablolarına Girdi Eklemek -mapPages

Sistemden fiziksel bellek allocPages fonksiyonu ile çağırıldıktan sonra, bu fiziksel belleğin süreçler tarafından kullanılabilmesi için o sürecin sayfa tablosuna eklenmesi gerekmektedir.

Bu işlem için mapPages fonksiyonu kullanılmaktadır. Bu fonksiyon verilen sayfa tablosuna, verilen fiziksel adres kadar sayfa ekler.

Bu işlemde öncelikle o sayfa tablosu dolaşılarak o sayfa sayısı kadar ardışık sayfa tablosu girdisi bulunur.

Ardışık boş sayfa tablosu girdileri bulunduktan sonra, fiziksel adresler sayfa tablosu girdisine işlenir ve sanal adres kullanıcı sürecine döndürülür.

```
// o sayfa tablosu içinde sayfa sayısı kadar,ardışık boş girdi bul
for(i = 0; i < (1024 - num_pages); i++)
{
        for(j = 0; j < num_pages; j++)
        {
                //eğer dolu ise döngüden çık
                if(page_table[i + j])
                {
```

```
                        i += j;
                        break;
                }
        }
        if(j == num_pages)
        {
                //eğer bulunduysa, sayfa tablosuna kopyala
                for(j = 0; j < num_pages; j++)
                {
                        page_table[i + j] = (((unsigned long)physical_address)

                                                + (j<<12)) | attributes;
                }
                // hesaplanan sanal adres, artık o fiziksel adresin, o sayfa
                // tablosuna karşılık gelen sanal adrestir.
                *virtual_address = (void *) (offset + (i<<12));

                return STATUS_SUCCESS;
        }
}
```

Dönen sanal adres, aslında sayfalama mekanizmasından geçirildikten sonra fiziksel adrese dönecektir. Sayfa tabloları, sürecin bellek erişimi için temel noktadır.

5.4.4 Kullanıcı Adres Bölgesinden Bellek İsteminde Bulunma - allocUserPages

Kullanıcı süreçler, kendi adres sahaları içerisinden bellek isteminde bulundukları zaman kernel'in değilde süreçlerin kendi sayfa tablolarına bu adreslerin işlenmesi gerekir. Bu işlem için, süreçler adres sahaları içerisindeki ilk iki sayfa tablosu kullanılır.

```
allocPages(num_pages,&physical_address);
mapPages(page_table,
           num_pages,
           PTE_PRESENT|PTE_WRITE|PTE_USER,
           0x400000,
           physical_address,
           &virtual_address);
```

Görüldüğü gibi öncelikle fiziksel çerçeve işletim sistemi tarafından alınmaktadır. Sonra, bu çerçeveye ait fiziksel adres, mapPages fonksiyonu ile sürecin sanal adres sahasına eklenmektedir.

Buradaki sayfa tablosu, sürecin ilk iki sayfa tablosundan biridir.

5.4.5 Kernel Adres Bölgesinden Bellek İsteminde Bulunma - allocKernelPages

İşletim sistemi adres sahası , her sürecin adres sahasına ekleniyordur. Ancak kullanıcı süreçlerin bu bellek bölgesine erişimi önlenmiştir.

Kernel adres sahasından bellek istemi ancak işletim sisteminin içsel olarak tutmuş olduğu sayfa tabloları sayesinde olur.

```
allocPages(num_pages,&physical_address);
mapPages((unsigned long *)phys_to_virt((unsigned long)
                                        KernelPageTable_High),
          num_pages,
          PTE_PRESENT|PTE_WRITE,
          0x800000,
          physical_address,
          &virtual_address);
```

Burada, yine fiziksel çerçeve atanması allocPages fonksiyonu ile yapılmakta ve dönen fiziksel adres çekirdek sayfa tablosuna eklenerek sanal adres döndürülmektedir.

Kernel sayfa tablosu kullanarak bellek tahsisi, işletim sistemi içsel yapılarını tutmak için kullanılır. Örneğin süreç oluşturma işleminde, bir süreç yapısı için yer ayarlanmalıdır. Bu işlem sonucunda alınan bellek bölgesi, sürecin kendi sayfa tablolarına değil, kernel sayfa tablosuna işlenmelidir. Çünkü kullanıcı süreçler bu veri yapısına hiçbir zaman ulaşmamalıdırlar ve bu veri yapısını sadece kernel kullanır.

```
//Sürece ait bilgilerin işletim sistemi tarafından tutulması için
//gerekli fiziksel bellek ayırma işlemleri...
//boş bir sayfa al
//sürece ait bilgiler bu sayfa içerisinde tutulacak
yeni_surec=(struct Task *)allocKernelPages(1);
```

Bu süreç yapısına ait adresler, kernel sayfa tablosuna işlenir. Daha önce belirtildiği gibi, kernel sayfa tablosu süreçlerin adres sahasına dahil edilmiştir.

5.4.6 Tahsis Edilmiş Bir Sayfayı Sisteme Geri Vermek – freePages

Sistemden alınmış bir sayfanın sisteme geri verilmesi için, o sayfanın başlangıç adresinden hangi bellek bölgesinden ayrıldığı öğrenilmelidir.

Süreçler iki bellek bölgesinden bellek tahsisi yapabilirler. Bunlar çekirdek adres sahası ve süreçlerin kendi adres sahalarıdır. Dolayısıyla, eğer sanal adres 0x000000 – 0x800000 adresleri arasında ise kullanıcı adres sahasından alınmıştır; 0x800000 değerinden büyükse çekirdek adres sahsından alınmıştır.

```
//eğer kernel adres sahası içinden bir adres sisteme verilecek ise
if((unsigned long)address >= 0x800000)
{
      index=(virt_to_phys((unsigned long)address) >>PTE_SHIFT) &
PTE_MASK;
      table=(unsigned long *)phys_to_virt((unsigned
long)KernelPageTable_High);
}
//eğer kullanıcı adres sahasından bir sayfa çıkartılacak ise
else
{
       if((unsigned long)address>=0x400000)
       {
          index=(((unsigned long)address-0x400000)>>PTE_SHIFT) &
PTE_MASK;
          table=addr_spc->user_ptable_1;
        }
        else
        {
           index=((unsigned long)address>>PTE_SHIFT) & PTE_MASK;
           table=addr_spc->user_ptable_0;
         }
}
```

O sanal adrese ait sayfa tablosu belirlendikten sonra, sayfa tablosu girdisinden fiziksel adres elde edilmelidir.

```
// o sanal adrese karşılık gelen fiziksel adresi, sayfa tablosunu
// kullanarak bul
physical=(void *)(table[index] & 0xFFFFF000);
```

O sanal adrese ait sayfa tablosu girdileri sıfırlanır.

```
// sayfa tablosunda, o elemana ait girdiyi 0'la
for(i=0;i<num_pages;i++)
{
     table[index+i]=0;
}
```

Fiziksel adres elde edildikten sonra, çrçeve bulunup, o çerçeveye ait FrameMap girdisi 0’lanmalıdır.

```
//frame takibini yapan listeden de o sayfayı çıkart.
j=(unsigned long)physical >> 12;

for(i=0;i<num_pages;i++)
{
        // o sayfa artik boş..
        FrameMap[j+i]=0;
}
```

Dolayısıyla, süreçlerin sayfa tabloları ve kernel sayfa tabloları her türlü bellek bölgesi erişiminde ve bellek tahsisinde kullanılmaktadır.

5.4.7 Süreçlere Ait Sayfa Tablolarının Doldurulması - createPageTables

Süreçlere ait adres sahası yapısı içerisinde yer alan tablolar yine işletim sisteminden bellek alınarak oluşturulmalıdır. Bu işlem için allocKernelPages fonksiyonu kullanılır. Çünkü bu tablolara sadece işletim sistemi erişecektir.

```
//sürece ait sayfa dizin tablosu ve sayfa tabloları doldurulmalı.
//Bu nedenle sürece ait sayfa dizin tablosu ve sayfa tabloları için
//gerekli yer, yine işletim sistemi tarafından atanmalı
allocPages(3,&physical);
mapPages(KernelPageTable_High,3,PTE_PRESENT|PTE_WRITE,
                VIRT_OFFSET,physical,&virt);
task->addr_space=(struct address_space *)virt;
```

Gerekli sayfa dizin ve sayfa tablosu değişkenleri sistemden alındıktan sonra, bu tablolar doldurulmalıdır. Öncelikle bu tablolar ilklenmelidir ve kullanıcı sayfa tablolari bire bir fiziksel adresler ile doldurulur.

```
// sayfa tabloları ilkleniyor...
for(i=0;i<1024;i++)
{
      task->addr_space->pdir[i]=0;
      task->addr_space->user_ptable_0[i]=i*4096 |PTE_PRESENT
                                                |PTE_WRITE |PTE_USER;
      task->addr_space->user_ptable_1[i]=0;
}
```

Bu işlemlerden sonra sürece ait sayfa dizin tablosu doldurulur. Sayfa dizin tablosuna sürecin sayfa tabloları dışında kernel sayfa tablosu girdisi de eklenir.

```
// sayfa dizininin ilk elemanı, sürece ait ilk sayfa tablosunu gösteriyor.
task->addr_space->pdir[0]= (unsigned long)(physical+4096)|
PDE_PRESENT
                                        |PDE_WRITE|PDE_USER ;
task->addr_space->pdir[1]= (unsigned long)(physical+4096*2)|
PDE_PRESENT
                                        |PDE_WRITE|PDE_USER;
// ikinci eleman ise kernel sayfa tablosunu gösteriyor(fiziksel adres)
task->addr_space->pdir[2]= (unsigned long)KernelPageTable_High
                                        |PDE_PRESENT |PDE_WRITE;
```

Tüm bu işlemler sonucunda, sürece ait sayfa dizin ve sayfa tabloları doldurulmuştur.

5.4.8 Süreçlere Ait Bellek Bölgelerinin Sisteme Geri Verilmesi – deleteProcess

Süreçler sonlandırılırken, onların daha önceden işletim sisteminden almış oldukları bellek bölgelerinin ve onlara ait veri yapılarının sisteme geri verilmesi gerekir.Bu işlem için kullanılan temel fonksiyon freePages fonksiyonu idi.

```
//süreç,o an zaten aktif süreç listesinden çıkartılmıştır ve ça-
//lışan süreçtir.dolayısıyla, sadece sürece ait bellek bölgeleri-
//ni geri vermemiz, yeterli olacaktir.
addr_spc=task->addr_space;

// sürece ait kullanıcı yığıtı sisteme veriliyor
freePages(addr_spc,1,(void *)task->Tss.ESP);

// sürece ait kernel yığıtı sisteme veriliyor
freePages(addr_spc,1,(void *)task->Tss.ESP0);


// sürece ait süreç yapısı sisteme veriliyor
freePages(addr_spc,1,(void *)task);

// sürece ait sayfa tabloları için ayırılmış bölgeler de tekrar
// sisteme veriliyor.
freePages(addr_spc,3,(void *)addr_spc);
```

Bu işlemlerde sırası ile kullanıcı sürece ait 3. seviye yığıt, 0. seviye yığıt, sürecin süreç veri yapısına ait bellek bölgesi ve sürecin adres sahasını tutan bellek bölgesi sisteme geri verilmektedir.

5.5 İşletim Sisteminde Sistem Çağırımları

İşletim sisteminin sunduğu servisleri 47h kesmesine yerleştirilmiştir. Kullanıcı süreçleri, INT 47h komutu ile işletim sistemi servislerinden faydalanabilir.

Sistemdeki işletim sistemi servislerine ait adresler bir dizi veri yapısında toplanmıştır. Kullanıcı sürecin EAX yazmaçına koyduğu değer, çağırılacak sistem çağırımına olan girdi numarasıdır.

5.5.1 Sistem Çağırımları -SystemCall

Sistem çağırımlarını yönetecek fonksiyon **int.asm** dosyasında belirtilen Zeugma_System_Call fonksiyonudur.

```
;API arayüzü  eax'te API  adresi var
_Zeugma_System_Call:
        EnterInterrupt

        ;kesme fonksiyonuna ait parametleri yığıta at
        push EDI                ;4. parametre
        push EDX                ;3. parametre
        push ECX                ;2. parametre
        push EBX                ;1. parametre

        call [_Zeugma_Api_Table + EAX*4 +0x800000] ;ilgli API'yi çağır

        ;atılan parametreleri geri al
        pop EBX
        pop ECX
        pop EDX
        pop EDI

        LeaveInterrupt
        IRET
```

Bu fonksiyon IDT tablosuna **interrupts.c** dosyasında şu şekilde işlenmiştir:

```
// Sistem çağırımlarının yapılacağı kesme numarası 47h olarak seçilmiştir
Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[0x47],
                (unsigned long)Zeugma_System_Call,      //handler
                sel_KernelCS,                           //selector
                0,                                      //parametre sayisi
                TRAP_GATE|PRESENT|DPL3);                //access
```

Bu IDT tablosu girdi işlemlerinden onra, artık kullanıcı süreçler sistem çağırımlarını yapabilmektedirler. Bunun için SystemCall fonksiyonu kullanılır.

```
long SystemCall(int API_Index,    //çağırılan fonksiyon indexi
                long param1,      //1. parametre
                long param2,      //2. parametre
                long param3,      //3. parametre
                long param4)      //4. parametre
{
        long ret;

        //ilgili API'yi parametreleri göndererek çağır
        __asm__ __volatile__ ("int $0x47 "
                               :"=a"(ret)
                               :"a"(API_Index),
                                "b"(param1),
                                "c"(param2),
                                "d"(param3),
                                "S"(param4));
        return ret;
}
```

Görüldüğü gibi, EBX,ECX,EDX ve ESI yazmaçları sırası ile 1., 2. , 3. ve 4. parametreleri tutmaktadır. EAX yazmaçı ise çağırılan sistem çağırımı numarasını tutmaktadır.

Örneğin API_Cls fonksiyonu, işletim sisteminin sağladığı bir API'dir ve içsel olarak SystemCall fonksiyonunu kullanır.

```
//Cls
void API_Cls()
{
        SystemCall(API_CLS_INDEX,0,0,0,0);
}
```

Cls fonksiyonunun hiç parametresi yoktur ve API girdi numarası ise API_CLS_INDEX yani 0'dır. EAX yazmaçına 0 değeri konularak INT 47h komutu ile Zeugma_System_Call fonksiyonuna dallanılır. Bu kesme yönetim fonksiyonu ise, Zeugma_Api_Table dizisini kullanarak, ilgili sistem servisine dallanır.

6. SONUÇ

Yapılan çalışma ile işletim sistemi çekirdeklerinin çalışma yapıları hakkında bilgi elde edinildi. Gerçek anlamda çalışabilir bir işletim sistemi ortaya konuldu ve bir işletim sistemi iskeleti gerçekleştirildi.

Özellikle alt seviye programlama konusunda tecrübe elde edildi. C ve Assembly dilleri programlama yeteneği daha da gelişti. Bilgisayar sistemi içerisinde bulunan programlanabilir çiplerin programlanması hakkında da bilgiler elde edildi.

Linux gibi bir işletim sistemi kaynak kodunda da faydalanılması, o işletim sisteminin içsel yapıları hakkında daha da fazla bilgi edinilmesine yol açtı. Ayrıca yaygın olarak kullanılan işletim sistemlerinin nasıl gerçekleştirildikleri hakkında bilgi sahibi olunuldu.

Kod incelemeleri sonucunda değişik kodlama stilleri öğrenildi. Bu da programlamaya olan bakış açısını daha da geliştirdi.

Bunların dışında, edinilen bilgilerin dökümantasyonu sonucu , döküman eksikliği olan bu konuda bir ek kaynak daha oluşturulmuş olundu.

Sonuç olarak yapılan çalışma ile, bu konu hakkında çalışma yapmak isteyenler için bir kaynak ortaya konulmuştur. Bu kaynağın, işletim sistemleri ile ilgilenenlere faydalı olacağını düşünüyorum.

6.1 Gelecek Çalışmalar

Gerçekleştirilen işletim sistemi çekirdeği, büyük bir işletim sistemi için başlangıç noktası teşkil edebilir. İleride geliştirilmek istenirse, Türkiye’deki bilgisayar mühendisleri tarafından ortaya konacak bir işletim sistemi oluşturulabilir.

Ayrıca sistemm programlama ve işletim sistemleri konusunda çalışma yapmak isteyenler de gerçekleştirilen koddan faydalanabilirler. Özellikle linux konusunda inceleme yapmak isteyenler için bir çıkış noktası olabilir.

EK (İşletim Sistemi Kaynak Kodu)

Boot.asm

```
[BITS 16]
[Org 0h]

                                                                ;OFFSETS
Boot_Routine                jmp short Start                    ;00h
                            NOP                                ;just for completing 3
bytes
Manifacturer                DB    'ZEUGMA  '                   ;03h
Bytes_per_Sector            DW    512                          ;0Bh
Sectors_per_Cluster         DB    1                            ;0Dh
num_Reserved_Sectors        DW    1                            ;0Eh
num_FATs                    DB    2                            ;10h
num_Root_Entry              DW    224                          ;11h
num_Sectors_in_Volume       DW    2880                         ;13h
Media_Descriptor            DB    0F0h                         ;15h
num_Sectors_per_FAT         DW    9                            ;16h
Sectors_per_Track           DW    18                           ;18h
num_Read_Write_Heads        DW     2                           ;1Ah
num_Hidden_Sectors          DD     0                           ;1Ch
num_Large_Sectors           DD     0                           ;20h No large sec.
(Sec.Over32MB)
DriveNumber                 DB     0                           ;24h Drive number 0 (A:)
Unused                      DB     0                           ;25h
Ext_Boot_Signature          DB     29h                         ;26h Extended Boot
Signature 29h
Serial_Number               DD     0                           ;27h Blank serial number
Volume_Name                 DB     'ZEUGMA                     ';2Bh Disk label
FileSystem                  DB     'FAT12   '                  ;36H Fat12 Filesystem
;--------------------------------------------------------------------------------
Start:
        mov AX,0x07C0
        mov DS,AX           ;Boot sektoru 0000h:7C00h adresine yuklendigi icin
        mov ES,AX           ;veri segmentini gosteren yazmaç ayarlanıyor.
        mov FS,AX
        mov GS,AX
        mov AX,0x9000       ;Yıgıt segmenti,9000h segmentine tasiniyor.
        mov SS,AX
        mov SP,0x8000
```

```
mov SI,Yukleniyor
call Mesaj_Yaz
;---------------------------------------------------------------------------
;Aşağıdaki işlemler kok dizininin hafızaya alınmasından
sorumludur.FAT12'de kok
;dizini 19. SEKTOR'den itibaren başlamaktadır. Kök Dizini 14 sektör yer
kaplar.
;Kök dizini 2000:0000h fiziksel adresine yükleniyor.
mov SI,CRLF
call Mesaj_Yaz
mov CL,14           ;CL=kaç sektör okunacak
mov BX,0x2000
mov ES,BX           ;ES -> kök dizininin yukleneceği segment
xor BX,BX           ;BX=0 -> kök dizininin yuklenecegi ofset
mov AX,19           ;AX=kok dizininin başlangıç sektoru
call Sektor_Yukle   ;kok dizininin hafıza alınması için ilgili fonksiyonu
                    ;çağır
;---------------------------------------------------------------------------
;Aşağıdak yer alan işlemler FAT tablosunun hafızaya alınmasından
sorumludur.
;FAT12'de FAT tablosu 1. sektörden başlar ve 9 sektör yer kaplamaktadır.
;FAT tablosu 3000:0000h fiziksel adresine yükleniyor.
mov SI,CRLF
call Mesaj_Yaz
mov CL,9            ;CL=kaç sektör okunacak
mov BX,0x3000
mov ES,BX           ;ES -> FAT tablosunun yukleneceği segment
xor BX,BX           ;BX=0
mov AX,1            ;AX=1 -> FAT'in başlangıç sektor numarası
call Sektor_Yukle
;---------------------------------------------------------------------------
;İşletim sistemini ilkleme işlemlerini yapacak olan 'INITSYS' dosyasını
;8000h:0000h adresine yükle...
mov SI,CRLF
call Mesaj_Yaz

mov SI,Initsys
Call Dosya_Ara      ;Dosyayı ara
mov BX,0x8000
mov ES,BX
call Dosya_Yukle
;---------------------------------------------------------------------------
;İşletim sistemi çekirdeğini 1000h:0000h fiziksel adresine yükle
mov SI,CRLF
call Mesaj_Yaz


mov SI,Kernel
```

```
	Call Dosya_Ara		;Dosyayı ara
	mov BX,0x1000
	mov ES,BX
	call Dosya_Yukle
	;---------------------------------------------------------------------------
	;İşletim sistemini ilkleme işlemlerini yapacak olan 'INITSYS' dosyasına atla
	mov SI,Mes_Initsys	;SI=mesajın başlangıç ofseti
	call Mesaj_Yaz

	mov AH,00h
	int 16h

	jmp DWORD 8000h:0000h	;Initsys dosyasina atlanıyor.
	Mes_Initsys  DB 0x0D, 0x0A,'INITSYS dosyasina atlanacak...',0

	;---------------------------------------------------------------------------
	;aşağıdaki kod parçası, IP yazmaçının ve CS kod segment yazmaçının
değerini
	;değiştirip bilgisayarın FFFF:0000 adresindeki kodu çalıştırmasını sağlar. Bu
	;adreste bulunan kod, bilgisayar her açıldığında çalıştırılan koddur ve bilgi-
	;sayarın tekrar başlamasını (reset) sağlar.
Reset:
	mov SI,Dosya_Bulunamadi		;SI=mesajın başlangıç ofseti
	call Mesaj_Yaz
	mov AH,00h
	int 16h
	jmp WORD 0FFFFh:0000h

;---------------------------------------------------------------------
; Fonksiyon Adı:
;   "Dosya_Ara"
; Açıklama:
;   Fonksiyon kök dizini içerisinde, verilen dosya adını arar ve eğer bulursa
;   o dosyaya ait ilk cluster'ı döndürür...
; Parameterler:
;   SI -> Aranacak dosyanın adını işaret eder.
; Geri Dönüş Değeri:
;   AX yazmaçında dosyanın ilk cluster'ı döndürülür.
;---------------------------------------------------------------------
Dosya_Ara:
	;---------------------------------------------------------------------
	;İlkleme işlemleri...

	mov AX,0x2000
	mov ES,AX
	xor DI,DI			;DI=0
	mov CX,[num_Root_Entry]		;Kök dizinindeki eleman sayısı kadar
					;karşılaştırma yapılacak
```

```
	;--------------------------------------------------------------------
	;Kök dizini, verilen dosya adı için aranıyor. Dosya adı 8 adet byte'tan olu-
	;şur ve bu nedenle Kök Dizinini gösteren [ES:DI], [DS:SI] ile 8 byte karşı-
	;laştırılır...
.karsilastir:
	push CX		;sayacımızı sakla
	push SI		;dosya adı karakter dizisine olan işaretçiyi sakla
	push DI		;kök dizinindeki konumu sakla

	cld		;ileri doğru arama yap
	mov CX,8	;CX=8 çünkü kok dizininde bulunan elemanlar daha önce
			;belirtildiği gibi 32 byte'tır ve ilk 8 byte dosya ismini
			;içerir
	repe cmpsb	;Dosya adına karşılık gelen 8 karakteri karşılaştır

	pop DI		;yığıta atılan yazmaçları yığıttan çek
	pop SI
	pop CX		;sayacı geri al

	je .bulundu	;Eğer aynı ise bulunmuştur.
	;--------------------------------------------------------------------
	;Eğer o kök dizin elemanındaki dosya ismi , verilen dosya ismi ile aynı değil
	;ise, kök dizininin bir sonraki kaydına geç ve tekrar arama yap...
	add DI,0x20
	loop .karsilastir
	jmp Reset
	;--------------------------------------------------------------------
	;Dosya bulunduysa, o dosyaya ait ilk cluster AX yazmaçına yükleniyor...
.bulundu:
	add DI,1Ah		;dosya bulunduysa ilk cluster bilgisi,o kök dizini kaydı-
				;nın 1Ah ofsetindedir...
	mov AX,[ES:DI]	;AX yazmaçı dosyanın ilk cluster'ını tutuyor.

	ret
;--------------------------------------------------------------------
; Fonksiyon Adı:
;   "Dosya_Yukle"
; Açıklama:
;   Aşağıdaki fonksiyon kök dizininde ilk clusteri bulunan dosyanın hafızada
;   belirtilen yere alınmasından sorumludur.
; Parameterler:
;   ES -> Dosyanın yükleneceği segment
;   AX -> Dosyanın ilk cluster'ı
; Geri Dönüş Değeri:
;   YOK
;--------------------------------------------------------------------
Dosya_Yukle:
	mov BX,0x3000
```

```
	mov FS,BX
.cluster_yukle:
	;----------------------------------------------------------------------
	;Cluster numarasını LBA'ya çevirme işlemleri...
	mov WORD [cluster],AX		;dosyanın ilk cluster'i saklanıyor
	add AX,31				;Cluster numarasını LBA'ya çevir ( 31
ekleyerek )
					;AX yazmaçında LBA değeri var
					;ES -> dosyanın yükleneceği segment
					;DI -> dosyanın yukleneceği offset
	;----------------------------------------------------------------------
	;LBA yardımı ile sektörün okunması
	push ES			;segment yazmaçını sakla

	xor CX,CX
	inc CX			;CX=1 sektor oku
	xor BX,BX
	call Sektor_Yukle	;AX=LBA,ES:BX dosya buffer'ı

	pop ES			;segment yazmaçını geri al
	;----------------------------------------------------------------------
	;Bellek bölgesinin sonraki 512 byte'ını yani sonraki sektorün okunacagi
	;alani göster...
	mov AX,ES
	add AX,0x20
	mov ES,AX
	;----------------------------------------------------------------------
	;SI=SI*3/2
	mov SI,WORD [cluster]
	add SI,SI
	add SI,WORD [cluster]
	shr SI,1		;SI FAT indeksini tutuyor (cluster*3/2)
	;----------------------------------------------------------------------
	;AX'e sonraki cluster numarasını at.Ancak bu deger daha tam olarak sonraki
	;cluster'ı göstermiyor. Üzerinde işlem yapılmalı...
	mov AX,[FS:SI]		;AX'te cluster numarası var
	;----------------------------------------------------------------------
	;Cluster numarası çift mi yoksa tek  mi?
	bt WORD[cluster],0	;BX=cluster numarası tek mi çift mi?
	jc .tek			;tek ise .tek isimli etikete git.
	;----------------------------------------------------------------------
	;Çift ise cluster numarası 11-0 bitlerindedir...
	and AH,0fh		;0-11 bitlerini çek
	jmp .isle
.tek
	;----------------------------------------------------------------------
	;Tek ise 15-4 bitlerindedir...
	shr AX,4		;4-15 bitlerini cek
```

```
.isle:
          cmp AX,0FF5h        ;eğer son cluster değil ise tekrar diğer cluster'in
          jb .cluster_yukle       ;yüklenmesi gerekmektedir.

          ret
cluster     DW 0

;---------------------------------------------------------------------
; Fonksiyon Adı:
;   "Sektor_Yukle"
; Açıklama:
;   Bu fonksiyon, parametre olarak girilen mantıksal sektor adresini (LBA)
;   CHS adresine çevirerek yine parametre olarak girilen segment ve offset
;   numaralarına girilen sektor sayısı kadar sektoru transfer eder
; Parameterler:
;   ES:BX = Dosyayı yuklemek icin ofset ve segment
;   AX = Mantıksal sektor numarası (LBA)
;   CL = Okunacak Sektor sayısı
; Geri Dönüş Değeri:
;   YOK
; Not:
;   HPC = okuma yazma kafası sayısı
;   SPT = track basina dusen sektor sayisi
;   Sector = ( LBA mod SPT ) + 1
;   Head = ( LBA / SPT )  mod HPC
;   Cylinder = ( LBA / SPT ) / HPC
;---------------------------------------------------------------------
Sektor_Yukle:
            ;---------------------------------------------------------------------
            ;LBA'yı CHS'te çevirme işlemleri...
            xor DX,DX                      ;DX-AX yazmaç çifti LBA'yı içeriyor
            div WORD [Sectors_per_Track]   ;kalan DX'te, Bölüm AX'te
            inc DL
            mov [sektor],DL
            xor DX,DX                      ;DX-AX yazmaç çifti LBA/SPT'yi
içeriyor
            div WORD [num_Read_Write_Heads]
                                           ;kalan DX'te, Bölüm AX'te
                                           ;DX==DL -> Kafa numarası
                                           ;AX=Silindir numarası
            ;---------------------------------------------------------------------
            ;Sektörü BIOS kesmesi yardımı ile oku
            push AX                        ;Silindir dumarasını sakla
            mov AL,CL                      ;sektor sayısı
            pop CX
            xchg CL,CH
            shl CL,6
            or CL,BYTE [sektor]
```

```
            xchg DH,DL                          ;DH=kafa
            xor DL,DL                           ;A sürücüsü
            mov AH,02h                          ;Sektor Oku fonksiyonu (BIOS)
            int 13h                             ;BIOS çağırımı

            ;------------------------------------------------------------------
            ;Ekrana nokta koy
            mov SI,Islem
            call Mesaj_Yaz

            ret
sektor      DB 0

;------------------------------------------------------------------
; Fonksiyon Adı:
;   "Mesaj_Yaz"
; Açıklama:
;   DS:SI'deki karakter dizisini ekrana yazar.Karakter dizisi 0h ile bitmelidir.
; Parametreler :
;   DS:SI -> Mesaj
; Geri Dönüş Değeri:
;   YOK
;------------------------------------------------------------------
Mesaj_Yaz:
            lodsb                 ;AX yazmaçına karakteri al
            or AL,AL              ;karakterin 0 olup olmadığını kontrol et
            jz .bitir
            mov ah, 0x0E          ;BIOS teletype
            mov bh, 0x00          ;0. görüntü sayfası
            mov bl, 0x07          ;text özelliği
            int 0x10
            jmp Mesaj_Yaz
.bitir:
            ret

;----------------------------------------------------------------------------
; VERI ALANI
Initsys             DB 'INITSYS',0x20
Kernel              DB 'KERNEL',0x20,0x20
Yukleniyor          DB  0x0D, 0x0A,'Sistem aciliyor...', 0x0D,0x0A, 0x00
CRLF                DB  0x0D, 0x0A, 0x00
Islem               DB '.',0x00
;Sistem açılış mesajları
Dosya_Bulunamadi    DB 'Sistem yuklenemiyor...',0
;----------------------------------------------------------------------------
TIMES 510-($-$$)  DB  0
          DW  0AA55h ; Boot İmzası
```

Descriptor.inc

```
; Descriptor.inc dosyası, sistemde bulunan GDT,IDT ve LDT tablolarını doldurmak
için gerekli olan tanımlamaları içerir.

%ifndef DESCRIPTOR
%define DESCRIPTOR

;---------------------------------------------------------------------------
; Aşağıdaki tanımlamalar , bir segment tanımlayıcısının özelliklerini gösteren
; bitlere ait değerlerdir. Bir segment tanımlayıcısı oluşturulurken bu değerler
; kullanılır ve OR işleminden geçirilirir.
;---------------------------------------------------------------------------
; Segment büyüklüğü ile ilgili özellikler
PAGE_GRANULARITY EQU 10000000b      ;segment büyüklüğü sayfa ölçüsünde
SEGMENT_32_BIT       EQU 01000000b      ;32 bit segment
AVAILABLE            EQU 00010000b      ;sistem yazılımları tarafından
kullanıma uygun
;---------------------------------------------------------------------------
;Erişim hakları ile ilgili özellikler
PRESENT   EQU 10000000b      ;segment hafızada ve kullanılabilir.
DPL1      EQU 00100000b      ;Descriptor Privilege Level=1
DPL2      EQU 01000000b      ;Descriptor Privilege Level=2
DPL3      EQU 01100000b      ;Descriptor Privilege Level=3


;---------------------------------------------------------------------------
;Code veya Data segmentleri için özellikler
DATA_READ                        EQU 00010000b      ;read only
DATA_READWRITE                   EQU 00010010b      ;read/write
STACK_READ                       EQU 00010100b      ;read only
STACK_READWRITE                  EQU 00010110b      ;read/write
CODE_EXEC                        EQU 00011000b      ;exec only
CODE_EXECREAD                    EQU 00011010b      ;exec/read
CODE_EXEC_CONFORMING             EQU 00011100b      ;exec only
conforming
CODE_EXECREAD_CONFORMING         EQU 00011110b      ;exec/read
conforming

ACCESSED                         EQU 00000001b
;---------------------------------------------------------------------------
;Sistem tanımlayıcıları için özellikler
LDT                              EQU 00000010b
TASK_GATE                        EQU 00000101b
TSS                              EQU 00001001b
CALL_GATE                        EQU 00001100b
```

```
INTERRUPT_GATE                  EQU 00001110b
TRAP_GATE                       EQU 00001111b

%endif
```

GDT.inc

```
; GDT.inc dosyası Global Descriptor Table içerisinde bulunan tanımlayıcıların tablo
içerisin-
; deki offsetlerini tutar.

%ifndef GDT
%define GDT

;---------------------------------------------------------------------------
; GDT tablosunda bulunan tanımlayıcılar.
sel_Null            EQU 0
sel_KernelCS        EQU 8
sel_KernelDS        EQU 16
sel_UserCS          EQU 24
sel_UserDS          EQU 32

%endif
```

Ports.inc

```
; PORTS.Inc dosyası sisteme ait portların değerlerini tutan sistem dosyasıdır. Bu
portlar
; yardımı ile , sistem açılışında gerekli olan ilklemeler yapılır.

%ifndef PORTS
%define PORTS
;---------------------------------------------------------------------------
;PIC programlanması için gerekli olan port adresleri
;Bu port numaraları kullanılarak donanım kesmeleri taşınacaktır. Ayrıca sistemin
kesme-
;lere tam olarak yanıt vermesi için, PIC programlamasının yapılması ve gereken
paramet-
;relerin verilmesi şarttır.BIOS bunu açılışta yapsa da , uyumluluk açısından tekrar ya-
;pılması daha uygundur.
%define PIC_MASTER_PORT_0 20h    ;ICW1'in gönderileceği port numarası
(PIC1)
```

```
%define PIC_MASTER_PORT_1 21h    ;ICW2,ICW3 ve ICW4'ün gönderileceği
port
                                 ;numarası (PIC1)
%define PIC_SLAVE_PORT_0 0A0h    ;PIC2 için ilk port
%define PIC_SLAVE_PORT_1 0A1h    ;PIC2 için 2. port
;-----------------------------------------------------------------------------
;PIT programlanması için gerekli portlar
%define PIT_1_COUNTER_0  40h
%define PIT_1_COUNTER_1  41h
%define PIT_1_COUNTER_2  42h
%define PIT_1_CONTROL_REGISTER 43h
%define PIT_2_COUNTER_0  48h
%define PIT_2_COUNTER_1  49h
%define PIT_2_COUNTER_2  4Ah
%define PIT_2_CONTROL_REGISTER 4Bh
;-----------------------------------------------------------------------------
;8042 Klavye denetleyicisinin programlanabilmesi için gerekli olan portlar
;Bu port numaraları A20 adres bacağının aktif hale getirilmesi için kullanılacaktır.
%define KEYBOARD_DATA_REGISTER  60h
%define KEYBOARD_COMMAND_REGISTER  64h  ;komut yazma
%define KEYBOARD_STATUS_REGISTER  64h   ;durum bilgisi alma

%endif
```

---

initsys.asm

```
[BITS 16]
[ORG 0h]

%include "ports.inc"      ;port tanımlamalarının olduğu dosya
%include "descriptor.inc" ; descriptor tanımlamalarının olduğu dosya

; INITSYS.asm dosyası , işletim sistemi çekirdeğini hafızaya yükleme işleminden
;sorumludur.
; Bu işlemle birlikte sırasıyla şu adımlar gercekleştirilir:
; -Kesmeler kapatılır.
; -10000h fiziksel adresindeki Kernel.Bin 0h fiziksel adresine taşınır.
; -Floppy motoru kapatılır.
; -Kod içindeki GDT'nin fiziksel adresi hesaplanıp yine kod içinde ilgili bölgeye
yazılır
; -GDTR yazmaçı GDT tablosu ile yüklenir.
; -IDTR yazmaçı henüz var olmayan IDT bilgisi ile yüklenir.
; -A20 adres bacağı aktif hale getirilir.
; -PIC programlanır.
; -Korumalı moda geçmek için gereken işlem yapılır ve Kernel'e atlanır.
;
```

```
; Inıtsys.bin 8000h:0000h fiziksel adresindedir.
;
; Inıtsys yüklendiği andaki sistem hafıza haritası:
; 1000h:0000h -> Kernel
; 8000h:0000h -> Initsys dosyası
;
; Bu aşamadan sonra:
; 0h--> adresinde Kernel vardır.
;

        jmp BASLA
;--------------------------------------------------------------------------------
; VERI ALANI
Imlec_X         DB 1      ;x koordinatımızı tutar
Imlec_Y         DB 1      ;y koordinatımızı tutar
;--------------------------------------------------------------------------------
;Açılış mesajları
Mesaj_Basla             DB 'INITSYS isletiliyor...',0
Mesaj_Kernel            DB 'Kernel Hafizaya alindi...',0
Mesaj_INT               DB 'Kesmeler kapatildi...',0
Mesaj_Tasindi           DB 'Kernel sifir fiziksel adresine tasindi...',0
Mesaj_A20               DB 'A20 adres pini aktif hale getirildi...',0
Mesaj_PIC               DB 'Priority Interrupt Controller programlandi...',0
Mesaj_GDTR              DB 'GTDR yazmaci gecici GDT tablosu ile yuklendi...',0
Mesaj_IDTR              DB 'IDTR yazmaci gecici IDT tablosu ile yuklendi...',0
Mesaj_Olum              DB 'Sistem acilamiyor...',0

;--------------------------------------------------------------------------------
; Geçici Sistem Tabloları (Sadece başlangıç için. Korumalı moda geçilince
değiştirilecek.)
GDT_Limit       DW 3*8-1    ;Limit (24 çünkü şu anda sadece 3 girdi var.)
GDT_Base        DD 0        ;Base (daha sonra kod içinde yazılacak
IDT_Limit       DW 0        ;Limit
IDT_Base        DD 0        ;Base
;--------------------------------------------------------------------------------
;Geçici GDT tablosu. Bu tablo sadece sistem yüklenirken geçici olarak
kullanılmaktadır.
;Korumalı moda geçtikten sonra, Kernel daha uygun bir GDT tablosu oluşturacaktır.

GDT_Start       DW 0,0,0,0      ;Null Descriptor

                DW 0FFFFh       ;Code Read/Execute Base=0 Limit=4GB 32 bit
                DW 0000h
                DW 9A00h
                DW 00CFh

                DW 0FFFFh       ;Data Read/Write   Base=0 Limit=4GB 32 bit
                DW 0000h
```

```
                    DW 9200h
                    DW 00CFh
;--------------------------------------------------------------------------------
BASLA:
        ;----------------------------------------------------------------------
        ;Yazmaçlarımızı ayarla
        mov AX,CS
        mov DS,AX
        mov ES,AX
        mov FS,AX
        mov GS,AX

        call Ekrani_Temizle
        ;----------------------------------------------------------------------
        ;Başlangıç mesajını yaz...
        mov SI,Mesaj_Basla          ;SI=mesajın başlangıç ofseti
        mov DL,00000001b            ;karakter rengi
        call Mesaj_Yaz
        ;----------------------------------------------------------------------
        ;Kesmeleri kapat...
        cli
        mov    al,80h               ;NMI interuptlarını boot süresince kapat
        out    70h,al
        ;----------------------------------------------------------------------
        mov SI,Mesaj_INT
        mov DL,00000001b
        call Mesaj_Yaz
        ;----------------------------------------------------------------------
        ;Yüklenen kernel'i 0h fiziksel adresine taşı. Burada kernel'in maximum
        ;64K olabileceği varsayılmıştır.
        push CS
        cld
        xor SI,SI
        xor DI,DI
        xor AX,AX
        mov ES,AX                  ;ES:DI ->0000h:0000h -> kernelin kopyalanacağı
adres
        mov AX,1000h
        mov DS,AX                  ;DS:SI -> 4000h:0000h -> yüklediğimiz kernel
        mov ECX,0f000h
        rep movsb                   ;tasi...
        pop DS
        ;----------------------------------------------------------------------
        mov SI,Mesaj_Tasindi
        mov DL,00000001b
        call Mesaj_Yaz
        ;----------------------------------------------------------------------
        ;Floppy motorunu kapat
```

```
mov DX,3f2h
mov AL,0
out DX,AL                ;floppy motorunu kapat...
;-----------------------------------------------------------------------
;GDTR yazmaçını yüklemek için gereken işlemler
xor EAX,EAX
xor EBX,EBX
mov AX,GDT_Start
mov BX,CS
shl EBX,4
add EBX,EAX              ;EBX yazmaçında GDT'nin fiziksel adresi var
mov [GDT_Base],EBX

lgdt [GDT_Limit]         ;GDTR yüklendi...
;-----------------------------------------------------------------------
mov SI,Mesaj_GDTR
mov DL,00000001b
call Mesaj_Yaz
;-----------------------------------------------------------------------
;Interrupt Descriptor Tablosunu IDTR yazmaçına yükle...
lidt [IDT_Limit]         ;IDTR yüklendi...
;-----------------------------------------------------------------------
mov SI,Mesaj_IDTR
mov DL,00000001b
call Mesaj_Yaz
;-----------------------------------------------------------------------
;A20 pinini aktif hale getir.
call A20_Ac              ;A20 adres bacağını aktif hale getiren fonksiyonu
                         ;çağır.
;-----------------------------------------------------------------------
mov SI,Mesaj_A20         ;SI=mesajın başlangıç ofseti
mov DL,00000001b         ;karakter rengi
call Mesaj_Yaz
;-----------------------------------------------------------------------
;Priority Interrupt Controller'ı programla
call PIC_Programla
;-----------------------------------------------------------------------
mov SI,Mesaj_PIC
mov DL,00000001b
call Mesaj_Yaz
;-----------------------------------------------------------------------
;Korumalı moda gec...
mov EAX,CR0
or EAX,1
mov CR0,EAX
;-----------------------------------------------------------------------
;16 bit komutlari içeren komut ön belleğini (cache) boşalt
jmp .atla
```

```
	;-------------------------------------------------------------------
	;Kernel'e atla

.atla:
	jmp DWORD 08h:0000h	;Kernel'e atla
;-------------------------INITSYS işlemleri burada bitti---------------------------

;-------------------------------------------------------------------
; Fonksiyon Adı:
;   "A20_Ac"
; Açıklama:
;   Aşağıdaki kod parçaları A20 adres bacağının aktif hale getirilmesinden sorumlu-
;   dur. 16 bitlik mimaride işlemcinin adresleyebileceği en yüksek hafıza 1 MB'tan
;   ibarettir. Bu nedenle A20 adres bacağına hafıza işlemlerinde gerek duyulmadığı
;   için, işlemci real modda çalışırken bu bacak aktif değildir. Korumalı modda
;   bu bacağın aktif hale getirilmesi gerekir. Çünkü işlemci 4GB'lık bellek böl-
;   gesini adresleyebilir ve bu 32 bitlik bir adres demektir. Bu nedenle A20 kapı-
;   sının aktif hale getirilmesi gerekir.
; Parametreler :
;   YOK
; Geri Dönüş Değeri:
;   YOK
; Not:
;   A20 adres bacağı, klavye denetleyicisinin çıkış tamponu (output buffer) ile AND-
;   lenmiştir. Bu nedenle , bu bacak, klavye çıkış tamponunun ilgili biti değiştirile-
;   rek aktif hale getirilir.
;-------------------------------------------------------------------
A20_Ac:
	call Bekle_8042
	;-------------------------------------------------------------
	;komut yazacağımızı klavye denetleyicisine ilet
	mov AL,0d1h
	out KEYBOARD_COMMAND_REGISTER,AL
	call Bekle_8042
	;-------------------------------------------------------------
	;veri portuna A20 kapısının açılması için gerekli olan bit dizisini yaz
	mov AL,0dfh
	out KEYBOARD_DATA_REGISTER,AL
	call Bekle_8042

	ret

;-------------------------------------------------------------------
; Fonksiyon Adı:
;   "Bekle_8042"
; Açıklama:
;   Aşağıdaki kod parçası, 8042 klavye denetleyicisinin giriş tamponunun
boşalmasını,
```

```
;   çıkış tamponunu boşalttıktan sonra beklemektedir.
; Parametreler :
;   YOK
; Geri Dönüş Değeri:
;   YOK
;-------------------------------------------------------------------
Bekle_8042:
        jmp short $+2
        jmp short $+2
        ;-----------------------------------------------------------------
        ;çıkış tamponunun boş olup olmadığı kontrol ediliyor
        in AL,KEYBOARD_STATUS_REGISTER
        test AL,1
        jz .cikis_bos
        jmp short $+2
        jmp short $+2
        ;-----------------------------------------------------------------
        ;veri portundaki veriyi oku
        in AL,KEYBOARD_DATA_REGISTER
        jmp Bekle_8042
.cikis_bos:
        ;-----------------------------------------------------------------
        ;cikis portu bosaldıktan sonra giris portunun bos olup olmadığını kontrol
        ;et
        test AL,1
        jnz Bekle_8042

        ret
;-------------------------------------------------------------------
; Fonksiyon Adı:
;   "PIC_Programla"
; Açıklama:
;   Bu fonksiyon gerçek moddaki (real-mode) donanım kesmelerine ait vektörlerin
;   taşınmasını sağlar. Çünkü korumalı moda geçildiği anda, işlemcinin üreteceği
;   yazılım kesmeleri bu vektörlerle çakışacaktır. Bu nedenle donanım kesmelerine
;   ait vektörlerin taşınması gerekir.
; Parametreler :
;   YOK
; Geri Dönüş Değeri:
;   YOK
; Not:
;   IRQ 0-7 70h-77h vekrötlerine; IRQ 8-15 ise 78h-7fh indekslerine yönlen-
;   dirilmiştir.
;-------------------------------------------------------------------
PIC_Programla:
        ;-----------------------------------------------------------------
        ;ICW1
        mov AL,00010001b        ; ICW1
```

```
out PIC_MASTER_PORT_0,AL    ; ICW1 PIC1 0. Portuna iletiliyor.
jmp short $+2
jmp short $+2
out PIC_SLAVE_PORT_0,AL     ; ICW1 PIC2 0. Portuna iletiliyor.
jmp short $+2
jmp short $+2
;-----------------------------------------------------------------
;ICW2
mov AL,01110000b            ; 01110xxx -> 70h-77h
out PIC_MASTER_PORT_1,AL    ; ICW2 PIC1 1. Portuna iletiliyor.
jmp short $+2
jmp short $+2
mov AL,01111000b            ; 01111xxx -> 78h-7fh
out PIC_SLAVE_PORT_1,AL     ; ICW2 PIC2 1. Portuna iletiliyor.
jmp short $+2
jmp short $+2
;-----------------------------------------------------------------
;ICW3
mov AL,00000100b            ; IRQ2'ye PIC2'ye bagli
out PIC_MASTER_PORT_1,AL    ; ICW3 PIC1 1. Portuna iletiliyor.
jmp short $+2
jmp short $+2
mov AL,00000010b            ; PIC2 slavedir.
out PIC_SLAVE_PORT_1,AL     ; ICW3 PIC2 1. Portuna iletiliyor.
jmp short $+2
jmp short $+2
;-----------------------------------------------------------------
;ICW4
mov AL,00000001b            ; normal EOI,8086/8080 mode
out PIC_MASTER_PORT_1,AL    ; ICW4 PIC1 1. Portuna iletiliyor.
jmp short $+2
jmp short $+2
out PIC_SLAVE_PORT_1,AL     ; ICW4 PIC2 1. Portuna iletiliyor.
jmp short $+2
jmp short $+2
;-----------------------------------------------------------------
;Bütün kesmeleri maskele
mov AL,0ffh
out PIC_MASTER_PORT_1,AL
jmp short $+2
jmp short $+2
out PIC_SLAVE_PORT_1,AL

ret

;-----------------------------------------------------------------
; Fonksiyon Adı:
;   "Mesaj_Yaz"
```

```
; Açıklama:
;   Bu ekrana ilgili karakter dizisini yazmak için çağırılan fonksiyondur.
;   Karakter dizisi 0h ile bitmelidir.
; Parametreler :
;   Mesaj ofseti = SI
;   karakter özelliği = DL
; Geri Dönüş Değeri:
;   YOK
;
;---------------------------------------------------------------------
Mesaj_Yaz:
        mov AX,0B800H
        mov ES,AX               ;Video bellek bolgesini goster
        ;---------------------------------------------------------------------
        ;Y koordinatı ile 160'ı çarp
        mov CL,160              ;1 satır 160 byte yer kaplıyor
        xor AX,AX
        mov AL,[Imlec_Y]        ;Y kordinatını AX yazmaçına taşı (AL)
        dec AL
        mul CL                  ;sonuç AX yazmaçında
        ;---------------------------------------------------------------------
        ;X koordinatı ile 2'yi çarp
        xor BH,BH
        mov BL,[Imlec_X]
        dec BL
        shl BL,1                ;2 ile çarp
        ;---------------------------------------------------------------------
        ;Y koordinatı ile 160'ı çarp
        add AX,BX               ;AX yazmaçı,video bellek bölgesinde mesajın yazıl-
                                ;maya başlanacağı ofseti tutmaktadır.
        mov CL,DL               ;basılacak karakterin özelliği
        mov DI,AX               ;DI video ram'i, SI string'i işaret eder
.tekrar:
        mov DL,[SI]
        cmp DL,0
        je .son
        mov [ES:DI],DL          ;ASCII kodunu video belleğine taşı
        inc DI
        mov [ES:DI],CL          ;karakter özelliğini video bölgesine taşı
        inc DI
        inc SI                  ;SI yazmaçını bir sonraki karakteri almak için 1
                                ;arttır.
        jmp .tekrar
.son:
        mov AL,[Imlec_Y]
        inc AL
        cmp AL,25
        jbe .dogru_deger
```

```
        xor AL,AL
        inc AL
.dogru_deger:
        mov BYTE [Imlec_Y],AL
        ret
;-----------------------------------------------------------------------
; Fonksiyon Adı:
;   "Ekrani_Temizle"
; Açıklama:
;   Bu fonksiyon ekranı temizleme işlemini yapmaktadır.
; Parametreler :
;   YOK
; Geri Dönüş Değeri:
;   YOK
;-----------------------------------------------------------------------
Ekrani_Temizle:
        mov AX,0B800H
        mov ES,AX               ;Video bellek bolgesini goster
        ;-----------------------------------------------------------------------
        ;Veri bölgesini 0'la
        cld
        mov CX,80*25
        xor DI,DI
        xor AX,AX
        rep stosw

        xor AL,AL
        inc AL
        mov BYTE [Imlec_X],AL
        mov BYTE [Imlec_Y],AL

        ret
;-----------------------------------------------------------------------
; Fonksiyon Adı:
;   "Reset"
; Açıklama:
;   aşağıdaki kod parçası, IP yazmaçının ve CS kod segment yazmaçının değerini
;   değiştirip bilgisayarın FFFF:0000 adresindeki kodu çalıştırmasını sağlar. Bu
;   adreste bulunan kod, bilgisayar her açıldığında çalıştırılan koddur ve bilgi-
;   sayarın tekrar başlamasını (reset) sağlar.
; Parametreler :
;   YOK
; Geri Dönüş Değeri:
;   YOK
;-----------------------------------------------------------------------
Reset:
        ;-------------------------------------------------------------
        ;Kullanıcıyı reset düğmesine basma zahmetinden kurtar :)
```

```
mov SI,Mesaj_Olum        ;SI=mesajın başlangıç ofseti
mov DL,01000001b         ;karakter rengi
call Mesaj_Yaz
mov AH,00h
int 16h                  ;bir tuşa basılmasını bekle
;----------------------------------------------------------
;Sistemi tekrar başlat...
jmp WORD 0FFFFh:0000h
```

---

start.asm

```
;ORG 0h
%include "GDT.inc"            ; GDT tanımlayıcılarına ait offset bilgileri tutan dosya
%include "Descriptor.inc"

; Artık korumalı moda geçmiş, işletim sistemi çekirdeğini hafızaya yüklemiş
durumdayız. Şu
; an kernel kodu çalışmakta.
; Saatler süren hata ayıklama işleminden sonra en sonunda çekirdeğe ve korumalı
moda atlaya-
; bildik. Artık işletim sistemi kodu C ile yazılmaya başlanabilir.
;
; |  Bu aşamada ortaya çıkan hafıza haritası şu şekildedir:
; |  0 - ffffh -> Kernel'e ayrılmıştır...
; |  10000h - A0000h -> Boş bellek
; |  A0000h - C0000h -> Video       ---> Hiçbir sürece atanamaz.Kernelin kontrolü
altında...
; V  C0000h - fffffh -> Rom system  ---> Rom kodunun bulunduğu bellek bölgesi...
;-------------------------------------------------------------------------------------
[section .text]
[BITS 32]

EXTERN     _Zeugma_Main                    ;Main.c içerisinde tanımlanmıştır
GLOBAL     _Global_Descriptor_Table        ;GDT tablosu
GLOBAL     _Interrupt_Descriptor_Table     ;IDT tablosu
GLOBAL     _Kernel_Stack                   ;Kernel yığıt bölgesi(kernel işlemleri
için)
GLOBAL     _Page_Directory_Table           ;Sistemdeki ana Sayfa Dizin Tablosu
;-------------------------------------------------------------------------------------
;Aşağıdaki kodlar tıpkı Linux'te olduğu gibi, sayfa tabloları ile
kaplanacaktır.Aşağıdaki
;kodlar işletildikten sonra, sayfalama işlemi için gerekli olan tablolardan Sayfa Dizin
;Tablosu (PageDirectory) , bu kodların üzerine yazılacaktır. Sayfa Dizin Tablosu
4MB'lık
```

```
;bellek bölgelerini seçmek için, Sayfa Tabloları 4MB'lık bellek bölgeleri içinden
4KB'lık
; bellek bölgeleri seçmek için kullanılır.
;
;Sayfa Dizin Tablosu için buradaki bölge uygun görülmüştür.C koduna geçildikten
sonra bu
;tabloya 2 adet girdi eklenecektir. Bu sayede 8MB'lık bellek bölgesi adreslenecektir.
;Bu sistemin ilklenmesi ve ilk işlemler için yeterlidir.
;
align 4096
_Page_Directory_Table:
        ;------------------------------------------------------------------
        ;Başlangıçta segment yazmaçlarını uygun değerlerle yükle...
        mov AX,10h          ;4GB data
        mov DS,AX
        mov ES,AX
        mov FS,AX
        mov GS,AX
        ;------------------------------------------------------------------
        ;Tüm sistem yaşantısı boyunca kullanılacak GDT tablosunu oluştur. IDTR ve
        ;GDTR yazmaçlarına uygun değerleri yükle
        Call GDT_Olustur
        lgdt [_GDT_Limit]
        lidt [_IDT_Limit]
        ;------------------------------------------------------------------
        ;Tüm tablolar yüklendikten sonra artık diğer segment yazmaçlarına uygun
        ;değerleri yeniden yükle
        mov AX,sel_KernelDS
        mov DS,AX
        mov ES,AX
        mov FS,AX
        mov GS,AX
        mov SS,AX
        ;------------------------------------------------------------------
        ;A20 kapısının aktif hale getirilip getirilmediğini kontrol et...
        ;Bu işlemde 0h fiziksel adresindeki veri 1 arttırılıp 100000h fiziksel
        ;adresine yazılıyor. Sonra 0h fiziksel adresindeki veri tekrar okunup
        ;kontrol ediliyor. Eğer o an okunan değer ile 1 arttırılmış değer aynı ise
        ;A20 kapısı aktif hale getirilememiştir...
        mov EAX,[0h];           ;0h fiziksel adresindeki bilgiyi al
        inc EAX                 ;ve o bilgiyi 1 arttır
        mov [100000h],EAX       ;arttırdığımız değeri 100000h fiziksel adresine yaz
        mov EBX,[0h]
        cmp EAX,EBX             ;eğer 2 bilgi aynı ise A20 kapısı aktif değildir...
        je $                    ;kullanıcı reset tuşuna basıncaya kadar bekler:)
        ;------------------------------------------------------------------
        ;Yığıt kernel stack'i göstersin...
        mov EAX,_Kernel_Stack
```

```
	add EAX,4096		;yığıtımız Kernel yığıtını gösteriyor.
	mov ESP,EAX
	;-----------------------------------------------------------------
	;Sayfalama mekanizmasını aktiflemeden önce, Sayfa Dizin Yazmaçını ,
	;Sayfa Dizinini gösterecek şekilde ayarla
	mov EAX,_Page_Directory_Table
	mov CR3,EAX		;cr3=Page Directory
	;-----------------------------------------------------------------
	;İşletim sistemini ilkleyecek ve süreçleri başlatacak olan fonksiyonu çağır
	xor EAX,EAX		;EAX=0
	push EAX		;Kernel_Main'in parametreleri
	push EAX
	push EAX
	call _Zeugma_Main
	;-----------------------------------------------------------------
	;Artık C dilini kullanarak yazmış olduğumuz fonksiyonlara atladık.
	;Artık alt satıra kesinlikle atlayamayız.(En azından öyle umuyoruz :))


;-------------------------------------------------------------------
; Fonksiyon Adı:
;   "GDT_Olustur"
; Açıklama:
;   Fonksiyon, korumalı modda hafızaya erişmek için kullanılacak olan selektörlerin
;   gösterdiği descriptor'ların Global Descriptor Table 'a doldurulmasını sağlar.
; Parametreler :
;   YOK
; Geri Dönüş Değeri:
;   YOK
;-------------------------------------------------------------------
GDT_Olustur:
	;-----------------------------------------------------------------
	;null tanımlayıcısı dolduruluyor.
	mov EDI,sel_Null
	xor EBX,EBX
	xor ECX,ECX
	xor DX,DX
	call Descriptor_Doldur
	;-----------------------------------------------------------------
	;KernelCS tanımlayıcısı dolduruluyor. Bu tanımlayıcı, kernelin tüm bellek
	;bölgesine erişimini sağlar.
	;4GB, Read,Write,
	mov EDI,sel_KernelCS
	mov DL,PRESENT + CODE_EXEC
	mov DH,PAGE_GRANULARITY + SEGMENT_32_BIT + AVAILABLE
	xor EBX,EBX
	mov ECX,0fffffh
	call Descriptor_Doldur
```

```
;----------------------------------------------------------------
;KernelDS tanımlayıcısı dolduruluyor. Bu tanımlayıcı, kernelin tüm bellek
;bölgesine erişimini sağlar.
;4GB, Read,Write,
mov EDI,sel_KernelDS
mov DL,PRESENT + DATA_READWRITE
mov DH,PAGE_GRANULARITY + SEGMENT_32_BIT + AVAILABLE
xor EBX,EBX
mov ECX,0fffffh
call Descriptor_Doldur
;----------------------------------------------------------------
;UserCS tanımlayıcısı dolduruluyor. Bu tanımlayıcı,kullanıcının tüm lineer
;bellek bölgesine erişimini sağlar.
;4GB, Read,Write,
mov EDI,sel_UserCS
mov DL,PRESENT + CODE_EXEC + DPL3
mov DH,PAGE_GRANULARITY + SEGMENT_32_BIT + AVAILABLE
xor EBX,EBX
mov ECX,0fffffh
call Descriptor_Doldur
;----------------------------------------------------------------
;UserDS tanımlayıcısı dolduruluyor. Bu tanımlayıcı,kullanıcının tüm lineer
;bellek bölgesine erişimini sağlar.
;4GB, Read,Write,
mov EDI,sel_UserDS
mov DL,PRESENT + DATA_READWRITE  + DPL3
mov DH,PAGE_GRANULARITY + SEGMENT_32_BIT + AVAILABLE
xor EBX,EBX
mov ECX,0fffffh
call Descriptor_Doldur

ret
```

--

<u>int.asm</u>

--

```
%include "GDT.inc"

; Genel donanım ve yazılım kesmelerini yönetmek için gerekli olan fonksiyonlar bu
dosyada ta-
; nımlanmıştır. Sistemdeki tüm donanım ve yazılım kesmelerinin işletim sistemi
tarafından ele
; alınabilmesi için, her donanım ve yazılım kesmelerine o işi yapacak bir fonksiyon
atanmalı-
; dır.

 [SECTION .text]
[BITS 32]
```

```
;---------------------------------------------------
;diğer modullerden alınmış gerekli global değişkenler
EXTERN      _Kernel_Stack          ;Kernel yığıt bölgesini al
EXTERN      _Report_Exception      ;(Interrupt.c)


EXTERN      _Scheduler             ;Scheduler.c
EXTERN      _Keyboard_Handler      ;Keyboard.c
EXTERN      _Default_Handler       ;Interrupts.c
EXTERN      _Zeugma_Api_Table      ;Api_Table.c
;---------------------------------------------------

;---------------------------------------------------
;diğer modullere aktarılacak global değişkenler
GLOBAL      _Timer_Interrupt
GLOBAL      _Keyboard_Interrupt
GLOBAL      _Default_Interrupt
GLOBAL      _Zeugma_System_Call

GLOBAL _DivideError
GLOBAL _DebugException
GLOBAL _NMIInterrupt
GLOBAL _BreakPointException
GLOBAL _OverFlowException
GLOBAL _BoundRangeExceeded
GLOBAL _InvalidOpcodeException
GLOBAL _DeviceNotAvailable
GLOBAL _DoubleFaultException
GLOBAL _CoprocessorSegmentOverrun
GLOBAL _InvalidTSSException
GLOBAL _SegmentNotPresent
GLOBAL _StackFaultException
GLOBAL _GeneralProtectionException
GLOBAL _PageFaultException
GLOBAL _Reserved
GLOBAL _FloatingPointError



;-----------------------------------INTERRUPT
BOLGESI-----------------------------------; Sistemde meydana gelen donanım kesmeleri
yani IRQ'lardan sadece timer ve keyboard IRQ'ları
;şu an kullanılacağı için bunları yöneteck olan C fonksiyonları çağırılacaktır.
;-----------------------------------------------------------------------------------------
;her donanım kesmesi başında yazmaçların saklanması ve gerekli segment
yazmaçlarının
;uygun selektörlerle yüklenmesi gerekmektedir.Aşağıdaki makro bu işlemleri yapar
```

```
%macro	EnterInterrupt 0
	pushad				;tüm yazmaçları sakla
	push DS				;segment yazmaçlarını sakla
	push ES
	push FS
	push GS

	push EAX
	mov AX,sel_KernelDS		;yazmaçları ilgili selektörlerle yükle
	mov DS,AX
	mov ES,AX
	mov FS,AX
	mov GS,AX
	pop EAX
%endmacro
;----------------------------------------------------------------------------------------
;her donanım kesmesi sonunda yazmaçların kesme başlangıcından önceki konumuna
getirilmesi ge-
;reklidir.
%macro	LeaveInterrupt 0

	pop GS			;sakladığımız yazmaçları geri al
	pop FS
	pop ES
	pop DS

	popad
%endmacro

;--------------------------------------
;API arayüzü
;eax'te API  adresi var
_Zeugma_System_Call:
	EnterInterrupt

	;kesme fonksiyonuna ait parametleri yığıta at
	push EDI		;4. parametre
	push EDX		;3. parametre
	push ECX		;2. parametre
	push EBX		;1. parametre

	call [_Zeugma_Api_Table + EAX*4 +0x800000] ;ilgli API'yi çağır

	;atılan parametreleri geri al
	pop EBX
	pop ECX
	pop EDX
	pop EDI
```

```
            LeaveInterrupt
            IRET

;-----------------------------------------
;Scheduler'in çağırılacağı kod parçası
_Timer_Interrupt:
        EnterInterrupt

        mov AL,20h                ;EOI (End Of Interrupt) sinyali PIC'e gönderiliyor.
        out 20h,AL

        call _Scheduler           ;Scheduler.c

        LeaveInterrupt
        IRET
;-----------------------------------------
;Klavye yöneticisinin çağırılacağı kod parçası
_Keyboard_Interrupt:
        EnterInterrupt

        call _Keyboard_Handler       ;Keyboard.c

        mov AL,20h                   ;EOI (End Of Interrupt) sinyali PIC'e
gönderiliyor.
        out 20h,AL

        LeaveInterrupt
        IRET
;-----------------------------------------
;Sistemdeki default interrupt handler
_Default_Interrupt:
        EnterInterrupt

        call _Default_Handler       ;Interrupts.c

        mov AL,20h                  ;EOI (EndOfInterrupt) sinyali PIC'e gönderiliyor.
        out 20h,AL

        LeaveInterrupt
        IRET

;-----------------------------------EXCEPTION
BOLGESI---------------------------------; Sistemde meydana gelen istisnalar aşağıda
tanımlanmış kod parçacıkları ve veri alanı saye-
;sinde ele alınıp değerlendirilir. Değerlendirme işlemi şu anda kullanıcıya mesaj
verme şek-
;lindedir.
```

```
;-----------------------------------------------------------------------------
;MainHandler, sistemde bulunan exception'ları yöneten ana fonksiyondur. Burada
;meydana gelen istisna indexi yığıta atılır ve Exception.c içindeki Report_Exception
;fonksiyonu çağırılır.
MainHandler:
        push EBP
        mov EBP,ESP

        ;-------------------------------------------------------
        ;kullanılan tüm yazmaçları sakla

    pushad
    push DS
    push ES
    push FS
    push GS

    push EAX

    mov AX,sel_KernelDS
    mov DS,AX
    mov ES,AX
    mov FS,AX
    mov GS,AX

    pop EAX
        ;-------------------------------------------------------

        ;-------------------------------------------------------
        ;parametreleri yığıta at ve ilgili fonksiyonu çağır
        ;aşağıda zaten yığıtta var olan değişkenler var

        push EAX
        push EBX
        push ECX
        push EDX
        push EDI
        push ESI
        mov EAX,[EBP+4]         ;Exception index
        push EAX
        mov EAX,[EBP+8]         ;EIP
        push EAX
        mov EAX,[EBP+12]        ;CS
        push EAX
        mov EAX,[EBP+16]        ;FLAGS
        push EAX
        mov EAX,[EBP+20]        ;ESP
```

```
    push EAX
    mov EAX,[EBP+24]        ;SS
    push EAX

  call _Report_Exception      ;Exception.c içinde
    ;-------------------------------------------------------


  mov AL,20h               ;EOI (End Of Interrupt) sinyali PIC'e gönderiliyor.
  out 20h,AL

    ;-------------------------------------------------------
    ;sakladığımız yazmaçları geri al

  pop GS
  pop FS
  pop ES
  pop DS

  popad
    ;-------------------------------------------------------

  mov ESP,EBP
  pop EBP

  add ESP,4               ;Exception index için

  IRET


;---------------------------------------------------------------------------
;Aşağıdak yer alan fonksiyonlar, sistemde bulunan istisnaları yönetecek olan kod
;parçalarıdır.
_DivideError:
        push DWORD 0
        jmp MainHandler

_DebugException:
        push DWORD 1
        jmp MainHandler

_NMIInterrupt:
        push DWORD 2
        jmp MainHandler

_BreakPointException:
        push DWORD 3
        jmp MainHandler
```

```
_OverFlowException:
        push DWORD 4
        jmp MainHandler

_BoundRangeExceeded:
        push DWORD 5
        jmp MainHandler

_InvalidOpcodeException:
        push DWORD 6
        jmp MainHandler

_DeviceNotAvailable:
        push DWORD 7
        jmp MainHandler

_DoubleFaultException:
        push DWORD 8
        jmp MainHandler

_CoprocessorSegmentOverrun:
        push DWORD 9
        jmp MainHandler

_InvalidTSSException:
        push DWORD 10
        jmp MainHandler

_SegmentNotPresent:
        push DWORD 11
        jmp MainHandler

_StackFaultException:
        push DWORD 12
        jmp MainHandler

_GeneralProtectionException:
        push DWORD 13
        jmp MainHandler

_PageFaultException:
        push DWORD 14
        jmp MainHandler

_Reserved:
        push DWORD 15
        jmp MainHandler
```

```
_FloatingPointError:
        push DWORD 16
        jmp MainHandler
```

AsmDefines.h

```
//  Assembly komutlarını C dili içinden inline olarak çağırmak için
//  kullanacağımız makrolar.

#ifndef ASM_DEF
#define ASM_DEF

//-------------------------------------------------------------------------
// Inline Assembly Kodları ve Makrolar
// Bu kodlar ve makrolar sayesinde C kodu içerisinden assembly
// seviyesindeki komutları çok rahat bir şekilde kullanabileceğiz


//-------------------------------------------------------------------------
// Önemli  Inline Assembly Kodları ve Makrolar

#define sti()                     __asm__("sti"::)
#define cli()                     __asm__("cli"::)
#define nop()                     __asm__("nop"::)

#define pushad()                  __asm__("pushal"::)
#define popad()                   __asm__("popal"::)

//-------------------------------------------------------------------------
// IDTR,LDTR ve GDTR yazmaçları üzerinde işlem yapmak için gerekli makrolar
#define lidt(mem48)  __asm__ __volatile__  ("lidt %0 \n"::"m" (mem48));
#define sidt(mem48)  __asm__ __volatile__  ("sidt %0 \n"::"m" (mem48));
#define lgdt(mem48)  __asm__ __volatile__  ("lgdt %0 \n"::"m" (mem48));
#define sgdt(mem48)  __asm__ __volatile__  ("sgdt %0 \n"::"m" (mem48));

//
// Sistem yazmaçlarından bilgi almak için gerekli makrolar
#define get_cr0()      ({ unsigned long res; \
                        __asm__ __volatile__ ("movl %%cr0, %0" : "=r" (res) :); \
                       res; })

#define get_cr2()      ({ unsigned long res; \
                       __asm__ __volatile__ ("movl %%cr2, %0" : "=r" (res) :); \
                       res; })
```

```
#define get_cr3()       ({ unsigned long res; \
                        __asm__ __volatile__ ("movl %%cr3, %0" : "=r" (res) :); \
                        res; })

#define set_cr0(mem32)  __asm__ __volatile__ ("movl %0,%%cr0"::"r"(mem32));
#define set_cr2(mem32)  __asm__ __volatile__ ("movl %0,%%cr2"::"r"(mem32));
#define set_cr3(mem32)  __asm__ __volatile__ ("movl %0,%%cr3"::"r"(mem32));

//-------------------------------------------------------------------------
// Segment yazmaçları üzerinde işlem yapmak için gerekli makrolar

// belirtilen segment yazmaçındaki değeri döndür
#define get_seg(segment) ( \
   { unsigned short sonuc;                    \
     __asm__ __volatile__ ("movw %%"segment",%0":"=r"(sonuc):); \
     sonuc; })

// belirtilen segment yazmaçına verilen değeri taşı
#define set_seg(segment,selector) __asm__ __volatile__ ("movw %0,%
%"segment::"r" (selector) );

// yukarıdaki makrolar kullanılarak yazmaçlar üzerinde işlemler yapılır...
#define set_ds(selector) set_seg("ds", selector)
#define set_es(selector) set_seg("es", selector)
#define set_fs(selector) set_seg("fs", selector)
#define set_gs(selector) set_seg("gs", selector)
#define set_ss(selector) set_seg("ss", selector)
#define get_cs() get_seg("cs")
#define get_ds() get_seg("ds")
#define get_es() get_seg("es")
#define get_fs() get_seg("fs")
#define get_gs() get_seg("gs")
#define get_ss() get_seg("ss")

//-------------------------------------------------------------------------
//Yığıt işlemleri için gerekli makrolar
#define push(reg)                   __asm__("pushw %"reg)
#define pop(reg)                    __asm__("popw %"reg)

#define push_ds()       push("ds")
#define push_es()       push("es")
#define push_fs()       push("fs")
#define push_gs()       push("gs")

#define pop_ds()        pop("ds")
#define pop_es()        pop("es")
#define pop_fs()        pop("fs")
```

```
#define pop_gs()        pop("gs")

//------------------------------------------------------------------------
//Port işlemleri
// outport işlemi
#define out_port(value,port) {__asm__("outb %%al,%%dx"::"a"(value),"d"(port)); \
                                                         nop();\
                                                         nop();}
#define in_port(port)  (\
                                                { unsigned char value; \
                                                __asm__("inb %%dx,%
%al":"=a"(value):"d"(port));\
                                                 nop(); \
                                                 nop(); \
                                                 value;})
//------------------------------------------------------------------------
//EFlags yazmaçı ile ilgili işlem
#define get_task_flags() (\
                                                 { long flag;                    \
                                            __asm__("pushfl ;"                   \
                                                       "popl %%ecx;"             \
                                                       "orl  $0x00000200,%%ecx;" \
                                                       "movl %0,%%ecx;"          \
                                                        :"=r"(flag)              \
                                                        :                        \
                                                        :"%ecx");                \
                                                  flag;})

#endif
```

Console.h

```
//  Video bellek bölgesindeki işlemleri yapabilmek için kullandığımız sabitler,
// yapılar ve makrolar burada tanımlanmıştır.

#ifndef CONSOLE
#define CONSOLE

void Console_Init(void);
void Cls(void);
void Set_Cursor(int koord_x,int koord_y);
void Set_Color(char color);
void Set_Background_Color(char color);
void Print(char *Karakter_Dizisi);
void Println(char *Karakter_Dizisi);
void Put_Char(char ch);
```

```
void Move_Cursor();
void Scroll();

#define VIDEO_MEMORY_BASE 0xB8000 // Video bellek bölgesinin başlangıcı

#endif
```

Console.c

```
//  Video bellek bölgesi üzerinde yapılacak işlemler burada tanımlanmıştır.

#include "Console.h"
#include "Descriptor.h"
#include "AsmDefines.h"

//-------------------------------------------------------------
//Global değişkenler
static char *Video;                    //Video bellek bölgesini gösteren işaretçi
static int Cursor_X,Cursor_Y;          //Cursor'un koordinatlarını tutan değişkenler
static char Color;                     //yazılacak yazının ön ve arka plan rengi
//-------------------------------------------------------------

//---------------------------------------------------------------------
// Fonksiyon Adı:
//   "Console_Init"
// Açıklama:
//   Fonksiyon ekran işlemlerinin  yapılmadan önce ilgili değişkenlerin
//   ilklenmesinden sorumludur.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//---------------------------------------------------------------------
void Console_Init(void)
{
  Video=(char *)VIDEO_MEMORY_BASE;
  Cursor_X=1;
  Cursor_Y=1;
  Color=0x07;
  Cls();
  Set_Color(6);
  Println("-------------------------------------------------------------------");
  Println("<Zeugma> Konsol ilklendi.");
}

//---------------------------------------------------------------------
```

```
// Fonksiyon Adı:
//   "Cls"
// Açıklama:
//   Fonksiyon ekranı siler, video bellek bölgesini boş karakterler ile
//   doldurur.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//----------------------------------------------------------------------
void Cls(void)
{
   int i;

   for(i=0;i<25*80*2;i=i+2)
   {
      Video[i]=' ';
      Video[i+1]=Color;
   }
}

//----------------------------------------------------------------------
// Fonksiyon Adı:
//   "Set_Cursor"
// Açıklama:
//   Fonksiyon video bellek bölgesine karakter dizilerini yazarken hangi
//   satıra ve hangi sütüna yazı yazılacağını belirleyen değişkenleri değiş-
//   tirmek için kullanılır.
// Parametreler :
//   koord_x -> X koordinatı
//   koord_y -> Y koordinatı
// Geri Dönüş Değeri:
//   YOK
//----------------------------------------------------------------------
void Set_Cursor(int koord_x,int koord_y)
{
   Cursor_X=koord_x;
   Cursor_Y=koord_y;

         Move_Cursor();
}

//----------------------------------------------------------------------
// Fonksiyon Adı:
//   "Set_Color"
// Açıklama:
//   Yazının rengini değiştirmeye yarar.
// Parametreler :
```

```
//   color -> RGB bileşeni
// Geri Dönüş Değeri:
//   YOK
//
//   7    6    5    4     3     2    1     0  |
// Blink Red Green Blue Intensity Red Green Blue |
//       |-----------|           |-----------| |
//        Background              Foreground   |
//--------------------------------------------------------------------
void Set_Color(char color)
{
        Color= (Color & 0xf8) + (color & 0x07);
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Set_Background_Color"
// Açıklama:
//   Yazının arka plan rengini değiştirmeye yarar.
// Parametreler :
//   color -> RGB bileşeni
// Geri Dönüş Değeri:
//   YOK
//
//   7    6    5    4     3     2    1     0  |
// Blink Red Green Blue Intensity Red Green Blue |
//       |-----------|           |-----------| |
//        Background              Foreground   |
//--------------------------------------------------------------------
void Set_Background_Color(char color)
{
        Color= (Color & 0x8f) + ((color & 0x07)<<4) ;
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Print"
// Açıklama:
//   Fonksiyon video bellek bölgesine karakter dizilerini yazmaktan so-
//   rumludur.
// Parametreler :
//   Karakter_Dizisi -> Yazılacak karakter dizisi
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void Print(char *Karakter_Dizisi)
{
   char c;        //video bellek bölgesine yazılacak karakter
```

```
	//-------------------------------------------------------------
	//karakteri ekrana yaz...
   while((c=(char)(*Karakter_Dizisi++))!='\0')
		Put_Char(c);
	//-------------------------------------------------------------

}

//---------------------------------------------------------------------
// Fonksiyon Adı:
//   "Println"
// Açıklama:
//   Fonksiyon video bellek bölgesine karakter dizilerini yazmaktan so-
//   rumludur.Print fonksiyonundan farkı yazma işleminden sonra 1 alt
//	 satıra geçilmesidir
// Parametreler :
//   Karakter_Dizisi -> Yazılacak karakter dizisi
// Geri Dönüş Değeri:
//   YOK
//---------------------------------------------------------------------
void Println(char *Karakter_Dizisi)
{
	Print(Karakter_Dizisi);

	//-------------------------------------------------------------
	//imlec ayarlaniyor
	Cursor_X=1;
	Cursor_Y++;
	if(Cursor_Y==25)
	{
		Cursor_Y--;
		Scroll();
	}
	//-------------------------------------------------------------
}
//---------------------------------------------------------------------
// Fonksiyon Adı:
//   "Put_Char"
// Açıklama:
//   Fonksiyon verilen karakteri, video bellek bölgesine yazar
// Parametreler :
//   ch -> yazılacak karakter
// Geri Dönüş Değeri:
//   YOK
//---------------------------------------------------------------------
void Put_Char(char ch)
{
```

```
  int index;

  switch(ch)
  {
          //----------------------------------------------------------
     case '\n': // ENTER tuşuna basıldıysa
                              Cursor_Y++;
                              Cursor_X = 1;
                              if(Cursor_Y==25)
                              {
                                      Cursor_Y--;
                                      //ekranı satır satır kaydır
                                      Scroll();
                               }

                              break;
          //----------------------------------------------------------

          //----------------------------------------------------------
     case '\b':
                              Cursor_X--;
                              if(Cursor_Y < 1)
                                      Cursor_Y = 1;

                              index = (Cursor_Y - 1) * 160 + (Cursor_X - 1) * 2;
                         Video[index] = 0;
                              break;
          //----------------------------------------------------------

          //----------------------------------------------------------
          default:
                              index = (Cursor_Y - 1) * 160 + (Cursor_X - 1) * 2;
                              Video[index] = ch;
                              Video[index+1] = Color;
                              Cursor_X++;
                              break;
          //----------------------------------------------------------
   }

  Move_Cursor();
}

//-------------------------------------------------------------------
// Fonksiyon Adı:
//   "Move_Cursor"
// Açıklama:
//   İmleçi belirtilen koordinatlara taşır
// Parametreler :
```

```
//   YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void Move_Cursor()
{
        int index;

        //------------------------------------------------------------
        //imleçin koordinatlarını hesapla
        index = (Cursor_Y - 1) * 160 + (Cursor_X - 1) * 2;
        //------------------------------------------------------------

        //------------------------------------------------------------
        // portlar yardımı ile imleçi taşı
        cli();
        out_port(14, 0x3d4);
        out_port(0xff&(index>>9), 0x3d5);
        out_port(15, 0x3d4);
        out_port(0xff&(index>>1), 0x3d5);
        sti();
        //------------------------------------------------------------
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Scroll"
// Açıklama:
//   Ekranı bir satır aşağıya kaydıran fonksiyondur
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void Scroll()
{
        int i;
        char ch;

        //------------------------------------------------------------
        // her satırı bir satır yukarı taşı
        for(i = 160; i <= 2 * 25 * 80; i=i++)
        {
          ch = Video[i];
          Video[i-160] = ch;
        }
        //------------------------------------------------------------
```

```
	//----------------------------------------------------------
	//son satırı boşalt
	for(i = 2 * 24 * 80; i <= 2 * 25 * 80 - 2; i+=2)
	{
	  Video[i] = 0;
	}
	//----------------------------------------------------------
}
```

Descriptor.h

```
//  Intel 32 bit mimarisindeki bellek tanımlayıcıları ve bu tanımlayıcılara
// ait özellikler ve sabitler bu başlık dosyasında tanımlanmıştır. Ayrıca
// LDT,GDT ve diğer elemanlar ( Gate,TSS vs...) üzerinde işlem yapacak olan
// basit makine dili komutlarını C dili ile kullanabilmek için gerekli olan
// tanımlamalar da yapılmıştır.
///////////////////////////////////////////////////////////////////////////

#ifndef DESCRIPTOR
#define DESCRIPTOR

//-----------------------------------------------------------------------
// intel 32 bit mimarisindeki Bellek Tanımlayıcısı (Descriptor) yapısı
struct i386_Descriptor{
   unsigned int    limit_low:16 ,    // Limit 0-15
                base_low :16 ,        // Base 0-15
                base_mid :8  ,        // Base 16-23
                access_rights:8 ,     // Descriptor'un erişim hakları
                limit_high:4 ,        // Limit 16-19
                size:4 ,              // Descriptor'un büyüklük bilgileri
                base_high:8 ;         // Base 24-31
};

//-----------------------------------------------------------------------
// intel 32 bit mimarisindeki Kapı (Gate) yapısı
struct i386_Gate{
   unsigned int    offset_low:16 ,   // Offset 0-15
                selector:16 ,         // Segment selector 0-15
                p_count:8 ,           // Parametre sayısı (4 bit)
                access_rights:8 ,     // Kapının erişim hakları
                offset_high:16  ;     // Offset 16-31
};

//-----------------------------------------------------------------------
// LDTR ve GDTR yazmaçlarını yükleyebilmek için gerekli olan yapı
struct i386_Memory_Register{
```

```
   unsigned short limit;   // Limit
   unsigned long base;     // taban adresi
};

//------------------------------------------------------------------------
// Bellek tanımlayıcısını doldurmak için gerekli olan sabitler aşağıda tanım-
// lanmıştır...

// Segment büyüklüğü ile ilgili özellikler
#define PAGE_GRANULARITY  0x80      // segment büyüklüğü sayfa ölçüsünde
#define SEGMENT_32_BIT    0x60      // 32 bit segment
#define AVAILABLE         0x10      // sistem yazılımları tarafından kullanıma
                                    // uygun
// Erişim hakları ile ilgili özellikler
#define PRESENT  0x80      // segment hafızada ve kullanılabilir.
#define DPL1     0x20      // Descriptor Privilege Level=1
#define DPL2     0x40      // Descriptor Privilege Level=2
#define DPL3     0x60      // Descriptor Privilege Level=3

// Code veya Data segmentleri için özellikler
#define DATA_READ 0x10                         // read only
#define DATA_READWRITE 0x12                    // read/write
#define STACK_READ 0x14                        // read only
#define STACK_READWRITE 0x16                   // read/write
#define CODE_EXEC 0x18                         // exec only
#define CODE_EXECREAD 0x1A                     // exec/read
#define CODE_EXEC_CONFORMING 0x1C              // exec only conforming
#define CODE_EXECREAD_CONFORMING 0x1E // exec/read conforming

#define ACCESSED 0x01 // o bellek bölgesine erişim oldu mu?

// Sistem tanımlayıcıları için özellikler
#define LDT 0x02                     // Local Descriptor Table
#define TASK_GATE 0x05               // Sistem Görev Kapısı
#define TSS 0x09                     // Görev Durum Segmenti (Task State Segment)
#define CALL_GATE 0x0C               // Çağırma Kapısı
#define INTERRUPT_GATE 0x0E // Kesme Kapısı
#define TRAP_GATE 0x0F               // Yazılım Kesme Kapısı

//------------------------------------------------------------------------
// Daha önce Inıtsys.asm tarafından doldurulmuş olan GDT 'ye ait selektörler
// INITSYS.asm dosyasında "GDT_Olustur" fonksiyonu ile GDT'ye
doldurulmuşlardır.
// Ayrıntılı bilgi için bu dosyaya bakınız...
#define sel_Null 0              // Null selektör
#define sel_KernelCS 8          // Tüm bellek bölgesine erişimi sağlar
#define sel_KernelDS 16         // Tüm bellek bölgesine erişimi sağlar
#define sel_UserCS 24 + 3       // Kullanıcının tüm bellek bölgesine erişimi
```

```
                              // RPL=3
#define sel_UserDS 32 + 3    // Kullanıcının tüm bellek bölgesine erişimi
                              // RPL=3
//------------------------------------------------------------------------
// EFLAGS yazmaçındaki bit haritası
//
#define EFLG_ID              (1<<21)
#define EFLG_VIP             (1<<20)
#define EFLG_VIF             (1<<19)
#define EFLG_AC              (1<<18)
#define EFLG_VM              (1<<17)
#define EFLG_RF              (1<<16)
#define EFLG_NT              (1<<14)
#define EFLG_IOPL0           (0<<12)
#define EFLG_IOPL1           (1<<12)
#define EFLG_IOPL2           (2<<12)
#define EFLG_IOPL3           (3<<12)
#define EFLG_IOPLSHFT        2
#define EFLG_OF              (1<<11)
#define EFLG_DF              (1<<10)
#define EFLG_IF              (1<<9)
#define EFLG_TF              (1<<8)
#define EFLG_SF              (1<<7)
#define EFLG_ZF              (1<<6)
#define EFLG_AF              (1<<4)
#define EFLG_PF              (1<<2)
#define EFLG_CF              (1<<0)

//------------------------------------------------------------------------
// Descriptor.c içindeki bulunan fonksiyon prototipleri
// Bu fonksiyonlar GDT,IDT ve LDT tablolarını dinamik olarak değiştirmek için
// kullanılırlar.Çalışma zamanında yeni eklemeler yapılır veya var olan değer-
// ler değiştirilebilir.
void Gate_Doldur(struct i386_Gate *gate,
            unsigned long offset,
            unsigned short selector,
            unsigned char parametre_sayisi,
            unsigned char access);

void Descriptor_Doldur(struct i386_Descriptor *desc,
                unsigned long limit,
                unsigned long base,
                unsigned char access,
                unsigned char size);
#endif
```

Descriptor.c

```
//  İşletim sistemimizde yer alan ve bellek erişimi için kullandığımız

#include "Descriptor.h"
#include "Kernel.h"

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Descriptor_Doldur"
// Açıklama:
//   Fonksiyon parametre olarak verilen bellek tanımlayıcısını, yine parametre
//   olarak verilen tanımlayıcı bilgileri ile doldurur.
// Parametreler :
//   desc -> doldurulacak tanımlayıcı
//   limit -> tanımlayıcı limiti
//   base -> tanımlayıcının işaret ettiği bellek bölgesinin başlangıç adresi
//   access -> tanımlayıcının erişim hakları
//   size -> tanımlayıcının büyüklüğü ile ilgili bilgiler
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void Descriptor_Doldur(struct i386_Descriptor *desc,
                    unsigned long limit,
                    unsigned long base,
                    unsigned char access,
                    unsigned char size)
{

   desc->limit_low = limit & 0xffff;
   desc->base_low = base & 0xffff;
   desc->base_mid = (base >> 16) & 0xff;
   desc->access_rights = access;
   desc->limit_high = (limit >> 16) & 0xf;
   desc->size = size & 0xf;
   desc->base_high = (base >> 24) & 0xff;
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Gate_Doldur"
// Açıklama:
//   Fonksiyon parametre olarak verilen Gate'i, yine parametre
//   olarak verilen bilgiler ile doldurur.
// Parametreler :
//   gate -> doldurulacak Gate
//   offset -> Gate'in o segment içinde işaret ettiği fonksiyon ofseti
```

```
//   selector -> Sistem bellek tablolarında işaret edilen segment
//   parametre_sayisi ->
//   access -> Gate erişim hakları
// Geri Dönüş Değeri:
//   YOK
//------------------------------------------------------------------
void Gate_Doldur(struct i386_Gate *gate,
             unsigned long offset,
             unsigned short selector,
             unsigned char parametre_sayisi,
             unsigned char access)
{
   gate->offset_low = offset & 0xffff ;
   gate->selector = selector ;
   gate->p_count = parametre_sayisi ;
   gate->access_rights = access | PRESENT ;
   gate->offset_high = (offset >> 16) & 0xffff;
}
```

Exceptions.h

```
//  Assembly komutlarını C dili içinden inline olarak çağırmak için
// kullanacağımız makrolar.

#ifndef EXCEPTIONS
#define EXCEPTIONS

void Init_Exceptions();

#endif
```

Exceptions.c

```
//  İşletim sisteminin istisnaların(Exceptions) yönetimini yapan fonksiyonların
//  IDT tablosuna yerleştirmesi ve bu fonksiyonlar yardımı ile sistemin
//  denetlemesi görevlerinden sorumludur.

#include "Console.h"
#include "Descriptor.h"
#include "Memory.h"
#include "AsmDefines.h"

//------------------------------------------------------------
// Exception'ları handle eden fonksiyonlar (interrupts.asm'den)
extern void DivideError();
```

```
extern void DebugException();
extern void NMIInterrupt();
extern void BreakPointException();
extern void OverFlowException();
extern void BoundRangeExceeded();
extern void InvalidOpcodeException();
extern void DeviceNotAvailable();
extern void DoubleFaultException();
extern void CoprocessorSegmentOverrun();
extern void InvalidTSSException();
extern void SegmentNotPresent();
extern void StackFaultException();
extern void GeneralProtectionException();
extern void PageFaultException();
extern void Reserved();
extern void FloatingPointError();
//------------------------------------------------------------

//------------------------------------------------------------
// Interrupt Descriptor Table tablosu (start.asm'den)
extern struct i386_Descriptor Interrupt_Descriptor_Table[256];
//------------------------------------------------------------

//------------------------------------------------------------
//Sistemde oluşan exception'ları ekrana yazmak için gerekli olan
//bir string dizisi aşağıdadır.
const char *Exception_List[]={"Divide error",                  // 0
                              "Debug exception",               // 1
                              "NMI Interrupt",                 // 2
                              "Breakpoint",                    // 3
                              "INTO Detected Overflow",        // 4
                              "BOUND Range Exceeded",          // 5
                              "Invalid Opcode",                // 6
                              "Coprocessor not available",     // 7
                              "Double exception",              // 8
                              "Coprocessor segment overrun",   // 9
                              "Invalid Task State Segment",    // 10
                              "Segment not present",           // 11
                              "Stack Fault Exception",         // 12
                              "General Protection Exception",  // 13
                              "Page Fault Exception",          // 14
                              "reserved",                      // 15
                              "Floating Point Error",          // 16
                              "reserved"                       // 17-32
                             };
//------------------------------------------------------------

//--------------------------------------------------------------------
```

```
// Fonksiyon Adı:
//   "Init_Exceptions"
// Açıklama:
//   Fonksiyon sistemde meydana gelebilecek olan exception'ları değerlen-
//         direcek olan fonksiyonların adreslerini IDT tablosuna işler.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void Init_Exceptions()
{
        int i;

        //------------------------------------------------------------
        //sistemde oluşabilecek exceptionları yönlendirecek olan IDT tablosu
        //sırası ile dolduruluyor.

        //0 ile bölme hatası
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[0],
                        (long)DivideError,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);

        //Debug Exception
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[1],
                        (long)DebugException,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);
        //NMI Interrupt
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[2],
                        (long)NMIInterrupt,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);
        //BreakPoint Exception
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[3],
                        (long)BreakPointException,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);
        //OverFlowException
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[4],
                        (long)OverFlowException,
                        sel_KernelCS,
                        0,
```

```
                        TRAP_GATE|PRESENT);
//BoundRangeExceeded
Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[5],
                        (long)BoundRangeExceeded,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);
//InvalidOpcodeException
Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[6],
                        (long)InvalidOpcodeException,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);
//DeviceNotAvailable
Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[7],
                        (long)DeviceNotAvailable,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);
//DoubleFaultException
Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[8],
                        (long)DoubleFaultException,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);
//CoprocessorSegmentOverrun
Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[9],
                        (long)CoprocessorSegmentOverrun,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);
//InvalidTSSException
Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[10],
                        (long)InvalidTSSException,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);
//SegmentNotPresent
Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[11],
                        (long)SegmentNotPresent,
                        sel_KernelCS,
                        0,
                        TRAP_GATE|PRESENT);
//StackFaultException
Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[12],
                        (long)StackFaultException,
                        sel_KernelCS,
                        0,
```

```
                              TRAP_GATE|PRESENT);
        //GeneralProtectionException
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[13],
                              (long)GeneralProtectionException,
                              sel_KernelCS,
                              0,
                              TRAP_GATE|PRESENT);
        //PageFaultException
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[14],
                              (long)PageFaultException,
                              sel_KernelCS,
                              0,
                              TRAP_GATE|PRESENT);
        //Reserved
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[15],
                              (long)Reserved,
                              sel_KernelCS,
                              0,
                              TRAP_GATE|PRESENT);
        //FloatingPointError
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[16],
                              (long)FloatingPointError,
                              sel_KernelCS,
                              0,
                              TRAP_GATE|PRESENT);
        //------------------------------------------------------------


        //------------------------------------------------------------
        //17 ve 32 numaralı kesmeler Intel tarafından daha sonra kullanılmak
        //için ayırılmıştır.Bu sebeple bu kesmelere ait fonksiyonlar sabit
        //bir yönetici fonksiyonu göstermektedir.
        for(i=17;i<32;i++)
                Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[i],
                                      (long)Reserved,
                                      sel_KernelCS,
                                      0,
                                      TRAP_GATE|PRESENT);
        //------------------------------------------------------------


        Println("<Zeugma> Sistemdeki yazilim kesmeleri ilklendi.");

}
//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Report_Exception"
// Açıklama:
```

```
//   Fonksiyon meydana gelen kesmeyi(exception) ve o anki yazmaç durumunu
//   kullanıcı ekranına yazar.
// Parametreler :
//   index -> meydana gelen exception numarası
// Geri Dönüş Değeri:
//   YOK
//------------------------------------------------------------------
extern void Report_Exception(unsigned long SS,
                             unsigned long ESP,
                             unsigned long EFLAGS,
                             unsigned long CS,
                             unsigned long EIP,
                             unsigned long exception_index,
                             unsigned long ESI,
                             unsigned long EDI,
                             unsigned long EDX,
                             unsigned long ECX,
                             unsigned long EBX,
                             unsigned long EAX)
{
        char **string;

        //sanal adrese çevirme işlemi
        string=(char **)phys_to_virt((unsigned long)Exception_List);

        Set_Color(12);
        //ekrana yaz
        Println((char *)string[exception_index]);
        Set_Color(5);
        // Register değerleri yazılacak
        Println(" -------------------");
        Print("|  CS     =");Print_Sayi_Hex(CS);Println(" |");
        Print("|  EIP    =");Print_Sayi_Hex(EIP);Println(" |");
        Print("|  EFLAGS =");Print_Sayi_Hex(EFLAGS);Println(" |");
        Print("|  ESP    =");Print_Sayi_Hex(ESP);Println(" |");
        Print("|  SS     =");Print_Sayi_Hex(SS);Println(" |");
        Print("|  EAX    =");Print_Sayi_Hex(EAX);Println(" |");
        Print("|  EBX    =");Print_Sayi_Hex(EBX);Println(" |");
        Print("|  ECX    =");Print_Sayi_Hex(ECX);Println(" |");
        Print("|  EDX    =");Print_Sayi_Hex(EDX);Println(" |");
        Print("|  ESI    =");Print_Sayi_Hex(ESI);Println(" |");
        Print("|  EDI    =");Print_Sayi_Hex(EDI);Println(" |");
        Println(" --------------------");

        if(exception_index==14)
        {
                Print_Sayi_Hex(get_cr2());
        }
```

```
        Print("surec beklemede...");
        for(;;);

}
```

Interrupts.h

```
//  İşletim sisteminin donanım kesmelerini yerleştirmesi ve bu kes-
//  meler yardımı ile sistemi denetlemesi görevlerinden sorumludur.

#ifndef INTERRUPTS
#define INTERRUPTS

void Init_Interrupts();

#endif
```

Interrupts.c

```
//  İşletim sisteminin donanım kesmelerini yerleştirmesi ve bu kes-
//  meler yardımı ile sistemi denetlemesi görevlerinden sorumludur.

#include "Console.h"            //console işlemleri için
#include "Descriptor.h"         //kullanılacak selektörler var
#include "AsmDefines.h"         //assembly makroları için
#include "Ports.h"              //PIC'e EOI gönderilecek.
#include "Keyboard.h"

//------------------------------------------------------------
//int.asm'de
extern void Default_Interrupt();
extern void Zeugma_System_Call();
//------------------------------------------------------------

//------------------------------------------------------------
// Interrupt Descriptor Table tablosu (start.asm'den)
extern struct i386_Descriptor Interrupt_Descriptor_Table[256];
//------------------------------------------------------------

//-------------------------------------------------------------------
// Fonksiyon Adı:
//   "Default_Handler"
// Açıklama:
//   Sistemde bir donanım kesmesi meydana gelirse,bu fonksiyon kesmenin
```

```
//          meydana geldiğini fakat bu kesmenin işlenmediğini bildirir.Yani,
//          varsayılan kesme fonksiyonudur.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//-----------------------------------------------------------------------
extern void Default_Handler()
{
        Println("Yönetilmeyen bir donanim kesmesi  olustu!!!");
}

//-----------------------------------------------------------------------
// Fonksiyon Adı:
//   "Init_Interrupts"
// Açıklama:
//   Fonksiyon sistemde meydana gelecek donanım kesmelerini yönetecek olan
//          fonksiyonları IDT tablosuna yerleştirmekle görevlidir.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//-----------------------------------------------------------------------
void Init_Interrupts()
{
        int i;

        //--------------------------------------------------------------
        //Geriye kalan interruptların hepsi DefaultHandler ile dolduru-
        //luyor.Daha sonra IDT'ye klavye ve timet kesmelerini yönetecek
        //fonksiyınlar işlenecek...
        for(i=32;i<256;i++)
                Gate_Doldur((struct i386_Gate )&Interrupt_Descriptor_Table[i],//gate
                                (unsigned long)Default_Interrupt,     //handler
                                sel_KernelCS,                         //selector
                                0,                                    //parametre sayisi
                                INTERRUPT_GATE|PRESENT);    //access
        //--------------------------------------------------------------


        //--------------------------------------------------------------
        // Sistem çağırımlarının yapılacağı kesme numarası 47h olarak seçilmiştir
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[0x47],
        //gate
                        (unsigned long)Zeugma_System_Call,        //handler
                        sel_KernelCS,                             //selector
                        0,                                        //parametre sayisi
                        TRAP_GATE|PRESENT|DPL3);                  //access
```

```
    //------------------------------------------------------------

    Println("<Zeugma> Sistemdeki donanim kesmeleri ilklendi.");
}
```

--
Kernel.h
--

```
// Kernel veri yapıları ve değişkenleri tanımlanmıştır.

#ifndef KERNEL
#define KERNEL

// GDT tablosunun büyüklüğü
#define GDT_SIZE 256

#endif
```

--
Kernel.c
--

```
// İşletim sistemi başlangıç noktası...
// İşletim sistemi yüklendikten sonra Zeugma_Main fonksiyonu çağırılır.Bu
// fonksiyon işletim sistemi ilklemeleri, IDT tablosunun oluşturulması ve
// diğer işlemleri yapar. Daha sonra işletim sistemi komut yorumlayıcısını
// çalıştırır.

#include "Console.h"
#include "Descriptor.h"
#include "AsmDefines.h"
#include "Exceptions.h"
#include "Interrupts.h"
#include "Timer.h"
#include "Scheduler.h"
#include "Keyboard.h"
#include "Memory.h"
#include "Process.h"
#include "Z_Api.h"

void init();
extern void Shell();

void Zeugma_Main (void)
{
    //------------------------------------------------------------
    //Console ilkleniyor...
```

```
        Console_Init();
        //------------------------------------------------------------

        //------------------------------------------------------------
        //Exception'lar ilkleniyor
        Init_Exceptions();
        //------------------------------------------------------------
        //------------------------------------------------------------
        //PIT ilkleniyor
        Init_Timer();
        //------------------------------------------------------------

        //------------------------------------------------------------
        //IDT ilkleniyor
        Init_Interrupts();
        //------------------------------------------------------------

        //------------------------------------------------------------
        //Scheduler fonksiyonu IDT'ye işlenip IRQ 0 aktif hale getiriliyor
        Init_Scheduler();
        //------------------------------------------------------------

        //------------------------------------------------------------
        //Klavye fonksiyonu IDT'ye işlenip IRQ 1 aktif hale getiriliyor
        Init_Keyboard();
        //------------------------------------------------------------

        //------------------------------------------------------------
        //Paging aktif hale gelsin
        initPaging();
        //------------------------------------------------------------

        //------------------------------------------------------------
        //Sistemdeki süreç sistemini ilkle ve ilgili GDT girdilerini koy
        initTask();
        //------------------------------------------------------------

        //------------------------------------------------------------
        sti();
        //------------------------------------------------------------
        Println("-----------------------------------------------------------------");
        //------------------------------------------------------------
        //init süreci yaratılıyor
        Exec((unsigned long)init);
        //------------------------------------------------------------
        for(;;);

}
```

```
//------------------------------------------------------------------
// Fonksiyon Adı:
//   "init"
// Açıklama:
//   Sistemde başlangıçta yaratılan ve hiçbir suretle sistemden atılma-
//         yan surec
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//------------------------------------------------------------------
void init()
{
        //----------------------------------------------------------
        // Shell'i çalıştır
        API_Set_Color(4);
        API_Println("------------------------------------------------------------------");
        API_Println("<init> Shell calistiriliyor");
        API_Println("------------------------------------------------------------------");
        API_Exec((unsigned long)Shell);
        //----------------------------------------------------------

        for(;;);
}
```

Keyboard.h

```
//  Klavye etkileşimini sağlayacak kod bölgesidir.

#ifndef KEYBOARD
#define KEYBAORD

void Init_Keyboard();
void Scanf(char *komut);

#endif
```

Keyboard.c

```
//  Klavye etkileşimini sağlayacak kod bölgesidir.

#include "Ports.h"
#include "Descriptor.h"
#include "AsmDefines.h"
#include "Console.h"
#include "Ports.h"
#include "Scheduler.h"
#include "Memory.h"
#include "System.h"
#include "Lib.h"

//-------------------------------------------------------------
//Int.asm'den
extern void Keyboard_Interrupt();
//-------------------------------------------------------------

//-------------------------------------------------------------
//(Scheduler.c)
extern struct Task *aktif_surec;
//-------------------------------------------------------------

//-------------------------------------------------------------
//(Scheduler.c)
extern struct Liste Bekleyen_Surec_Listesi;
extern struct Liste Hazir_Surec_Listesi;
//-------------------------------------------------------------

//-------------------------------------------------------------
// Interrupt Descriptor Table tablosu (start.asm'den)
extern struct i386_Descriptor Interrupt_Descriptor_Table[256];
//-------------------------------------------------------------

//-------------------------------------------------------------
//tuş durumlarını tutan bit değerleri
#define CAPS_ON         1
#define NUM_ON          2
#define SCROLL_ON       4
//tuş durumlarını tutan değişken
char state_NCS;
char NumLock;
char ScrollLock;
char CapsLock;
//-------------------------------------------------------------
```

```
//----------------------------------------------------------------
char KeyboardBuffer[128];   //klavye tampon bölgesi
char out;                   //klavye tamponunun hangi elemanına sonraki
                            //karakter yerleştirilecek?
unsigned char key;    //klavye tamponundan okunan tuş değeri
unsigned char ch;     //klavye tamponundan okunan tuşun ascii değeri
//----------------------------------------------------------------

//------------------------------------------------------------------------
// Fonksiyon Adı:
//   "removeFromWaitingQueue"
// Açıklama:
//         Klavye tamponundan veri okunmasi için bekleyen süreci, bekleyen
//  süreç listesinden çıkartır, komutu klavye tamponuna kopyalar ve
//  hazir süreç listesine kopyalar
// Parametreler :
//   komut -> tampondaki bilginin kopyalanacağı bellek bölgesi
// Geri Dönüş Değeri:
//   YOK
//------------------------------------------------------------------------
void removeFromWaitingQueue()
{
        struct Task *surec;
        struct Liste *lst;
        char *tampon;
        char *string;

        //----------------------------------------------------------------
        // bekleyen süreç listesinin sanal adresini al
        lst=(struct Liste *)phys_to_virt((unsigned long)&Bekleyen_Surec_Listesi);
        //----------------------------------------------------------------

        //----------------------------------------------------------------
        // listede eleman yok ise çık
        if(lst->eleman_sayisi==0)
                return;
        //----------------------------------------------------------------

        //----------------------------------------------------------------
        //sureci bekleyen süreç listesinden çıkart
        surec=lst->liste_basi;
        Remove_Task(lst,surec);
        //----------------------------------------------------------------

        //----------------------------------------------------------------
        //Klayve tamponundaki stringi, sürecin klavye tamponuna kopyala
        tampon=(char *)surec->keyboard_buffer;
        string=(char *)phys_to_virt((unsigned long)&KeyboardBuffer);
```

```
	while((*string)!=0)
	{
		(*tampon)=(*string);
		tampon++;
		string++;
	}
	(*tampon)='\0';
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	// süreci hazir sürec listesine ekle
	surec->Durum=TASK_READY;
	lst=(struct Liste *)phys_to_virt((unsigned long)&Hazir_Surec_Listesi);
	Insert_Task(lst,surec);
	//-------------------------------------------------------------

}
//----------------------------------------------------------------------
// Fonksiyon Adı:
//   "Scanf"
// Açıklama:
//          Klavye tamponundaki veriyi okur
// Parametreler :
//   komut -> tampondaki bilginin kopyalanacağı bellek bölgesi
// Geri Dönüş Değeri:
//   YOK
//----------------------------------------------------------------------
void Scanf(char *komut)
{
	struct Task **surec;
	struct Liste *lst;
	int i;

	//-------------------------------------------------------------
	// bekleyen süreç listesinin sanal adresini al
	lst=(struct Liste *)phys_to_virt((unsigned long)&Bekleyen_Surec_Listesi);
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	// o an aktif olan sureci al
	surec=(struct Task **)phys_to_virt((unsigned long)&aktif_surec);
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	//bekleyen süreç listesine koy
	(*surec)->Durum=TASK_WAITING;
	Insert_Task(lst,*surec);
	//-------------------------------------------------------------
```

```
	//------------------------------------------------------------
	// işlemci paylaştırıcısını çağır
	(*surec)=NULL;
	Scheduler();
	//------------------------------------------------------------

	//------------------------------------------------------------
	//Klayve tamponundaki stringi, verilen stringe kopyala
	strcpy(komut,(*surec)->keyboard_buffer);
	for(i=0;i<strlen((*surec)->keyboard_buffer);i++)
		(*surec)->keyboard_buffer[i]='\0';
	//------------------------------------------------------------
}

//---------------------------------------------------------------------
// Fonksiyon Adı:
//   "Wait_Keyboard"
// Açıklama:
//	Klavye giriş tamponunun boşalmasını bekler
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//---------------------------------------------------------------------
void Wait_Keyboard()
{
	unsigned char data;

	//------------------------------------------------------------
	do{
		//status bilgisini klavye denetleyisinin ilgili portundan
		//al
		data=in_port(KEYBOARD_STATUS_REGISTER);
		//2. bit set edilmiş ise giriş tamponu hala dolu.beklemeliyiz...
		data&=2;
	}while(data!=0);
	//------------------------------------------------------------

}

//---------------------------------------------------------------------
// Fonksiyon Adı:
//   "Set_Leds"
// Açıklama:
//	Sistemde bulunan CapsLock,NumLock ve Scroll Lock tuş ledlerinin
//	açılıp kapatılmasını sağlar.
// Parametreler :
```

```
//   YOK
// Geri Dönüş Değeri:
//   YOK
//-----------------------------------------------------------------------
void Set_Leds(unsigned char led_state)
{
	//-----------------------------------------------------------------
	//klavye giriş portunun boşalmasını bekle
	Wait_Keyboard();
	//-----------------------------------------------------------------

	//-----------------------------------------------------------------
	//klavyeye ledleri ayarlama komutunu gönder
	out_port(0xED,KEYBOARD_DATA_REGISTER);
	//-----------------------------------------------------------------

	//-----------------------------------------------------------------
	//klavye giriş portunun boşalmasını bekle
	Wait_Keyboard();
	//-----------------------------------------------------------------

	//-----------------------------------------------------------------
	//led durumunu klavyeye ilet
	out_port(led_state,KEYBOARD_DATA_REGISTER)
	//-----------------------------------------------------------------
}


//-----------------------------------------------------------------------
// Fonksiyon Adı:
//   "Keyboard_Interrupt"
// Açıklama:
//	 Herhangi bir klavye tuşuna basılınca meydana gelen kesmeyi yöneten
//	fonksiyondur. Tuşu yorumlar ve ilgili sürecin klavye tampon böl-
//	gesine iletir.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//-----------------------------------------------------------------------
extern void Keyboard_Handler()
{
	int i;
	//basılan tuşu klavye veri portundan al
	key = in_port(KEYBOARD_DATA_REGISTER);

	//-----------------------------------------------------------------
	//Led'li tuşlarımızdan herhangi birine basıldı mı? Eğer basıldı
```

```
//ise led'in açılıp kapanması gerekmektedir.
if(key==0x3A)        //Caps Lock
{
        if(CapsLock)
                state_NCS-=CAPS_ON;
        else
                state_NCS+=CAPS_ON;
        CapsLock=~CapsLock;
        Set_Leds(state_NCS);
}       else if(key==0x45) //Num Lock
        {
                if(NumLock)
                        state_NCS-=NUM_ON;
                else
                        state_NCS+=NUM_ON;
                NumLock=~NumLock;
                Set_Leds(state_NCS);
        }       else if(key==0x46) //Scroll Lock
                {
                        if(ScrollLock)
                                state_NCS-=SCROLL_ON;
                        else
                                state_NCS+=SCROLL_ON;
                        ScrollLock=~ScrollLock;
                        Set_Leds(state_NCS);
                }
//------------------------------------------------------------

//------------------------------------------------------------
//Ledli tuşlara basılmadıysa karakterleri kontrol et...
//basılan karakteri bul
switch(key)
{
        case 16: ch = 'q'; break;
   case 17: ch = 'w'; break;
   case 18: ch = 'e'; break;
   case 19: ch = 'r'; break;
   case 20: ch = 't'; break;
   case 21: ch = 'y'; break;
   case 22: ch = 'u'; break;
   case 23: ch = 'i'; break;
   case 24: ch = 'o'; break;
   case 25: ch = 'p'; break;
   case 30: ch = 'a'; break;
   case 31: ch = 's'; break;
   case 32: ch = 'd'; break;
   case 33: ch = 'f'; break;
   case 34: ch = 'g'; break;
```

```
    case 35: ch = 'h'; break;
    case 36: ch = 'j'; break;
    case 37: ch = 'k'; break;
    case 38: ch = 'l'; break;
    case 44: ch = 'z'; break;
    case 45: ch = 'x'; break;
    case 46: ch = 'c'; break;
    case 47: ch = 'v'; break;
    case 48: ch = 'b'; break;
    case 49: ch = 'n'; break;
    case 50: ch = 'm'; break;
    case 2: ch = '1'; break;
    case 3: ch = '2'; break;
    case 4: ch = '3'; break;
    case 5: ch = '4'; break;
    case 6: ch = '5'; break;
    case 7: ch = '6'; break;
    case 8: ch = '7'; break;
    case 9: ch = '8'; break;
    case 10: ch = '9'; break;
    case 11: ch = '0'; break;
    case 57: ch = ' '; break;
    case 14: ch = '\b'; break;
    case 28: ch = '\n'; break;
    case 185: ch = 0; break;
    case 142: ch = 0; break;
    case 156: ch = 0; break;
    case 13: ch = '='; break;
    case 26: ch = '['; break;
    case 27: ch = ']'; break;
    case 122: ch = '/'; break;
    case 43: ch ='/'; break;
    case 53: ch = '/'; break;
    case 51: ch = ','; break;
    case 52: ch = '.'; break;

    default: ch = 0; break;
}
//-------------------------------------------------------------

//-------------------------------------------------------------
//karakteri ekrana yaz ve klavye tamponuna kopyala
if(ch != 0)
{
Put_Char(ch);
                if(ch=='\b')
                {
                        KeyboardBuffer[--out]=0;
```

```
                }else if(ch!='\n') //ENTER değilse
                        {
                                KeyboardBuffer[out++]=ch;
                                //Stringin sonuna 0 yerleştir.
                                out=out%128;
                                KeyboardBuffer[out]=0;
                        }
        }
        //------------------------------------------------------------

        //------------------------------------------------------------
        // enter'a basildiysa, tuş bekleyen süreçi kuyruktan çıkart
        // ve adres sahasına komutu kopyala
        if(ch=='\n')
        {
                out=0;
                removeFromWaitingQueue();
                KeyboardBuffer[out]=0;
        }
        //------------------------------------------------------------
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Init_Keyboard"
// Açıklama:
//      Sistem açılışında tuşların konumlarını ilklemek için kullanılır.
//      Ayrıca klavye donanım kesmesini yönetecek olan fonksiyonu IDT'ye
//      kaydeder ve IRQ1'i aktif hale getirir.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void Init_Keyboard()
{
        char value;

        //------------------------------------------------------------
        //Keyboard interrupt'ını handle edeck kod parçası IDT'ye işlensin
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[0x71],  //gate
                        (long)Keyboard_Interrupt,               //handler
                        sel_KernelCS,                           //selector
                        0,                                      //parametre sayisi
                        INTERRUPT_GATE|PRESENT);                //access
        //------------------------------------------------------------

        //------------------------------------------------------------
```

```
	//IRQ 1 = Keyboard Interrupt aktif hale getiriliyor...
	//interrupt mask alınıyor
	value=in_port(PIC_MASTER_PORT_1);
	//IRQ 1 aktif hale gelsin
	out_port((value&0xfd),PIC_MASTER_PORT_1);
	//-----------------------------------------------------------

	//-----------------------------------------------------------
	//Num Lock'ı aç
	Set_Leds(NUM_ON);
	//-----------------------------------------------------------

	//-----------------------------------------------------------
	//sistemdeki tuşların açılış sonrası durumu
	state_NCS=NUM_ON;
	NumLock=0xff;
	ScrollLock=0;
	CapsLock=0;
	//-----------------------------------------------------------

	out=0;
	Println("<Zeugma> Klavye ilklendi.");

}
```

Lib.h

```
//  İşletim sisteminin sağladığı bazı özel kütüphane fonksiyonlarını
// içerir.
#ifndef Z_LIB
#define Z_LIB

void Print_Sayi_Hex(unsigned long sayi);
unsigned long strlen(char *str);
void strcpy(char *dest,char *src);
unsigned long strcmp(char *str1,char *str2,unsigned long length);

#endif
```

Lib.c

```
//  İşletim sisteminin sağladığı bazı özel kütüphane fonksiyonlarını
// içerir.

//-----------------------------------------------------------------------
// Fonksiyon Adı:
//   "Print_Sayi_Hex"
// Açıklama:
//        verilen sayıyı hexadesimal olarak ekrana yazar.
// Parametreler :
//   sayi -> ekrana yazılacak sayı
// Geri Dönüş Değeri:
//   YOK
//-----------------------------------------------------------------------
void Print_Sayi_Hex(unsigned long sayi)
{
        char string[9];
        char hex;
        int i;

        string[8]='\0';

        //-----------------------------------------------------------------
        //sayının hex olarak ekrana yazılmasını sağlayan kod bölgesi
        for(i=7;i>=0;i--)
        {
                hex=(char)(sayi & 0x0000000f);
                if(hex<10)
                        string[i]=hex +  0x30;
                else
                        string[i]=hex - 10 + 0x41;

                sayi= sayi >> 4;
        }
        //-----------------------------------------------------------------
        Print(string);
}

//-----------------------------------------------------------------------
// Fonksiyon Adı:
//   "strlen"
// Açıklama:
//        verilen stringin uzunluğunu bulur
// Parametreler :
//   str -> string
// Geri Dönüş Değeri:
```

```
//   YOK
//-----------------------------------------------------------------------
unsigned long strlen(char *str)
{
        unsigned long length=0;

        while((*str)!='\0')
        {
                length++;
                str++;
        }

        return length;

}

//-----------------------------------------------------------------------
// Fonksiyon Adı:
//   "strcpy"
// Açıklama:
//         verilen 2 stringi birbirine kopyalar
// Parametreler :
//   dest -> hedef string
//   src  -> kaynak string
// Geri Dönüş Değeri:
//   YOK
//-----------------------------------------------------------------------
void strcpy(char *dest,char *src)
{
        unsigned long i;

        for(i=0;i<strlen(src);i++)
                dest[i]=src[i];

        dest[i]='\0';
}

//-----------------------------------------------------------------------
// Fonksiyon Adı:
//   "strcmp"
// Açıklama:
//         verilen 2 stringi karşılaştırır kopyalar
// Parametreler :
//   str1 -> 1. string
//   str2 -> 2. string
// Geri Dönüş Değeri:
//   stringler eşit ise 1, değilse 0
//-----------------------------------------------------------------------
```

```
unsigned long strcmp(char *dest,char *src,unsigned long length)
{
        unsigned long i;

        for(i=0;i<length;i++)
        {
                if(src[i]!=dest[i])
                        return 0;
        }

        return 1;

}
```

--

Memory.h

--

```
//  Süreçler ile ilgili tanımlamaların yapıldığı kod bölgesidir.

#ifndef MEMORY
#define MEMORY

//-------------------------------------------------------------
#define NUM_FRAMES      3072
#define HIGH_MEM        0xC00000    //User   8 MB
#define LOW_MEM         0x400000    //Kernel 1 MB
#define VIRT_OFFSET     0x800000    //virtual olarak kernel 8.MB'tan başlıyor
//-------------------------------------------------------------

//-------------------------------------------------------------
// sanal - fiziksel adres dönüşümleri için gerekli makrolar
#define phys_to_virt(addr) (addr + VIRT_OFFSET)
#define virt_to_phys(addr) (addr - VIRT_OFFSET)
//-------------------------------------------------------------

//-------------------------------------------------------------------------
//Hafıza yönetimi sistemi için gerekli makro ve sabitler.
#define PAGE_SIZE  4096         //sayfa büyüklüğü
#define PDE_SHIFT  22           //fiziksel adresten PD'yi almak için
#define PDE_MASK  0x3FF         //PD'yi almak için gerekli maske
#define PTE_SHIFT  12           //fiziksel adresten PT'li almak için
#define PTE_MASK  0x3FF         //PT'yi almak için gerekli maske
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
//İntel işlemcilerinde sayfalama mekanizmasını kullanabilmek için gere
//ken sabitler
```

```
//--------------------------------------------------------------------------
// Sayfa Dizin Tablolarini doldurmak ve maskeleme için kullanılan sabitler
#define PDE_PRESENT                0x00000001
#define PDE_WRITE                  0x00000002
#define PDE_USER                   0x00000004
#define PDE_WRITE_THROUGH          0x00000008
#define PDE_CACHE_DISABLED         0x00000010
#define PDE_ACCESSED               0x00000020
#define PDE_GLOBAL                 0x00000100
#define PDE_PT_BASE                0xFFFFF000
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
// Sayfa Tablolarini doldurmak ve maskeleme için kullanılan sabitler
#define PTE_PRESENT                0x00000001
#define PTE_WRITE                  0x00000002
#define PTE_USER                   0x00000004
#define PTE_WRITE_THROUGH          0x00000008
#define PTE_CACHE_DISABLED         0x00000010
#define PTE_ACCESSED               0x00000020
#define PTE_DIRTY                  0x00000040
#define PTE_GLOBAL                 0x00000100
#define PTE_P_BASE                 0xFFFFF000
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
//Hafıza sistemi içerisinde bulunan fonksiyonların kullandığı sabitler
#define STATUS_SUCCESS                 0
#define STATUS_OUT_OF_MEMORY           1
#define STATUS_OUT_OF_VIRTUAL_MEM      2
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
// süreçlerin sayfa ve sayfa dizin tablolarının tutulduğu yapı
struct address_space
{
        unsigned long pdir[1024];                //sayfa dizin tablosu
        unsigned long user_ptable_0[1024];       //kullanıcı sayfa tablosu
        unsigned long user_ptable_1[1024];       //kullanıcı sayfa tablosu
};
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
// fonksiyon prototipleri
void initPaging();
unsigned long allocPages(unsigned long num_pages,
                                        void **physical_address);
```

```
unsigned long mapPages(unsigned long *page_table,
                                        unsigned long num_pages,
                                        unsigned long attributes,
                                        unsigned long offset,
                                        void *physical_address,
                                        void **virtual_address);
void *allocUserPages(unsigned long num_pages,
                                              unsigned long *page_table);
void *allocKernelPages(unsigned long num_pages);
void freePages(struct address_space *addr_spc,
                          unsigned long num_pages,
                          void *address);
unsigned long getPageDirectoryEntry(unsigned long *page_directory,
                                                           unsigned long
virtual_address);
unsigned long getPageTableEntry(unsigned long *page_directory,
                                                    unsigned long
virtual_address);
//------------------------------------------------------------------------

#endif
```

Memory.c

```
//  Sayfalama ve süreçler için bellek ayırma işlemlerinin yapıldığı
// modüldür.

#include "Memory.h"
#include "AsmDefines.h"
#include "Console.h"

//-----------------------------------------------------------
// sistemdeki fiziksel bellek parçalarının takibinin yapılabilmesi i-
// çin gereken dizi.İlk 16 sayfa Kernel kodu için ayrılmıştır.buradan sonraki
// 2 sayfa ise 2 adet sayfa tablomuza ayrılacaktır.
static char FrameMap[NUM_FRAMES]={0,};
//-----------------------------------------------------------

//-----------------------------------------------------------
// Sistemdeki kernel sayfa dizin tablosu (Start.asm'den)
// Sayfa dizin tablosunun fiziksel adresi = 0x000000
extern unsigned long Page_Directory_Table[1024];
//-----------------------------------------------------------

//-----------------------------------------------------------
// 12 MB'lık bellek bölgesine erişim için 3 adet sayfa tablosu
```

```
// ayarlanmıştır.
unsigned long *KernelPageTable_Low_0;
unsigned long *KernelPageTable_Low_1;
unsigned long *KernelPageTable_High;
//-------------------------------------------------------------

//-------------------------------------------------------------------
// Fonksiyon Adı:
//   "Init_Paging"
// Açıklama:
//       Sayfalama işleminin yapılabilmesi için gerekli olan tabloları dol-
// durur, sayfala mekanizmasını çalıştırır.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//-------------------------------------------------------------------
void initPaging()
{
        int i;

        //-------------------------------------------------------------
        // Sistemdeki sayfa tablolarının fiziksel adresleri atanıyor.
        // 12 MB'lık bellek adreslenebilecek...
        KernelPageTable_Low_0  =(unsigned long *)0x00010000;
        KernelPageTable_Low_1  =(unsigned long *)0x00011000;
        KernelPageTable_High   =(unsigned long *)0x00012000;
        //-------------------------------------------------------------

        //-------------------------------------------------------------
        // Kernel Page Directory boşaltılıyor
        // Sayfa tabloları bire bir fiziksel adresler ile dolduruluyor
        for(i=0;i<1024;i++)
        {
              Page_Directory_Table[i]=0;
              KernelPageTable_Low_0[i]=i*4096|PTE_PRESENT|PTE_WRITE|
        PTE_USER;
              KernelPageTable_Low_1[i]=0;

              if((i<19) ||(i==0xb8))
                 KernelPageTable_High[i]=i*4096|PTE_PRESENT|PTE_WRITE;
              else
                 KernelPageTable_High[i]=0;
        }
        //-------------------------------------------------------------

        //-------------------------------------------------------------
        //Kernel Sayfa dizin tablosuna gerekli girdiler konuluyor...
```

```
	Page_Directory_Table[0]=(unsigned long)KernelPageTable_Low_0|

	                              PDE_PRESENT|PDE_WRITE;
	Page_Directory_Table[1]=(unsigned long)KernelPageTable_Low_1
	                              |PDE_PRESENT|PDE_WRITE;
	Page_Directory_Table[2]=(unsigned long)KernelPageTable_High
	                              |PDE_PRESENT|PDE_WRITE;
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	//Kernel frame'leri ve PT frameleri dolu olarak işaretleniyor
	for(i=0;i<19;i++)
		FrameMap[i]=1;

	FrameMap[0xb8]=1; //video bellek bölgesi
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	//Artık 12MB'lık bellek bölgesine erişim için gerekli sayfalama tabloları oluş-
	//turuldu.

	//Sayfalama mekanizmasını aktif hale getir.
	//Start_Paging();
	__asm__ ("mov %0, %%cr3"::"r" (Page_Directory_Table));
	__asm__ ("mov %%cr0,%%eax ;"
	         "orl $0x80000000,%%eax ;"
	         "mov %%eax,%%cr0 ;"
	         "jmp 1f ;"
	         "1: movl $1f,%%eax ;"
	         " jmp *%%eax ;"
	         "1:  ":: );

	//-------------------------------------------------------------
	Println("<Zeugma> Hafiza sistemi ilklendi.");
	Println("<Zeugma> Sayfalama mekanizmasi aktif halde.");
}

//-------------------------------------------------------------------
// Fonksiyon Adı:
//   "allocPages"
// Açıklama:
//        İşletim sisteminden parametre olarak verilen kadar sayfa alınması
//  işlemini yapar.Sayfanın fiziksel başlangıç adresini döndürür.
// Parametreler :
//   num_pages -> kaç adet sayfa alınacak
//   physical_address -> dönen sayfaların fiziksel başlangıç adresi
// Geri Dönüş Değeri:
//   YOK
```

```
//-------------------------------------------------------------------
unsigned long allocPages(unsigned long num_pages,
                                             void **physical_address)
{
        unsigned long i, j;

        //-----------------------------------------------------------
        // verilen sayfa sayısı kadar ardışık sayfayı, hafıza sistemi içe-
        // risinde bul
        for(i = 0; i < (NUM_FRAMES - num_pages); i++)
        {
                //-----------------------------------------------------------
                //ardışık sayfaların da boş olup olmadığını kontrol et
                for(j = 0; j < num_pages; j++)
                {
                        //eğer sayfa dolu ise, artık aramaya o sayfadan itibaren
                        //başla
                        if(FrameMap[i + j]==1)
                        {
                                i += j;
                                break;
                        }
                }
                //-----------------------------------------------------------

                //-----------------------------------------------------------
                // eğer istenen sayfa kadar ardışık sayfa var ise
                if(j == num_pages)
                {
                        //sayfaları dolu olarak işaretle
                        for(j = 0; j < num_pages; j++)
                        {
                                FrameMap[i + j]=1;
                        }

                        //fiziksel adresi ata
                        *physical_address = (void *) (i << 12);

                        return STATUS_SUCCESS;
                }
                //-----------------------------------------------------------
        }
        //-----------------------------------------------------------

        Println("<allocPage>: Hafıza doldu!!!");
        return STATUS_OUT_OF_MEMORY;
}
```

```
//------------------------------------------------------------------------
// Fonksiyon Adı:
//   "mapPages"
// Açıklama:
//         verilen sayfa tablosuna, verilen fiziksel adres kadar sayfa ekle
// Parametreler :
//   page_table -> girdilerin ekleneceği sayfa tablosu
//   num_pages -> kaç adet girdi var
//         attributes -> girdi özellikleri
//   offset -> sanal adrese eklenecek ofset
//   physical_address -> eklenecek fiziksel adres girdisi
//   virtual_address -> eklenen girdilere göre dönen sanal adres
// Geri Dönüş Değeri:
//   YOK
//------------------------------------------------------------------------
unsigned long mapPages(unsigned long *page_table,
                                        unsigned long num_pages,
                                        unsigned long attributes,
                                        unsigned long offset,
                                        void *physical_address,
                                        void **virtual_address)
{

        unsigned long i,j;

        //-------------------------------------------------------------
        // o sayfa tablosu içinde sayfa sayısı kadar,ardışık boş girdi bul
        for(i = 0; i < (1024 - num_pages); i++)
        {
                for(j = 0; j < num_pages; j++)
                {
                        //eğer dolu ise döngüden çık
                        if(page_table[i + j])
                        {
                                i += j;
                                break;
                        }
                }

                if(j == num_pages)
                {
                        //-------------------------------------------------------------
                        //eğer bulunduysa, sayfa tablosuna kopyala
                        for(j = 0; j < num_pages; j++)
                        {
                                page_table[i + j] = (((unsigned long)physical_address)
+ (j<<12)) | attributes;
                        }
```

```
                        //-----------------------------------------------------------

                        //-----------------------------------------------------------
                        // hesaplanan sanal adres, artık o fiziksel adresin, o sayfa tab-
                        // losuna karşılık gelen sanal adrestir.
                        *virtual_address = (void *) (offset + (i<<12));
                        //-----------------------------------------------------------

                        return STATUS_SUCCESS;
                }
        }
        //-----------------------------------------------------------

        return STATUS_OUT_OF_VIRTUAL_MEM;

}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "allocateUserPages"
// Açıklama:
//        sistemden boş sayfa alınır ve verilen sayfa tablosuna girdi olarak
//   eklenir.
// Parametreler :
//   num_pages -> kaç adet girdi eklenecek
//   page_table -> girdilerin ekleneceği sayfa tablosu
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void *allocUserPages(unsigned long num_pages,
                                        unsigned long *page_table)
{
        void *physical_address;
        void *virtual_address;

        allocPages(num_pages,&physical_address);
        mapPages(page_table,
                        num_pages,
                        PTE_PRESENT|PTE_WRITE|PTE_USER,
                        0x400000,
                        physical_address,
                        &virtual_address);

        return virtual_address;
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
```

```
//   "allocateKernelPages"
// Açıklama:
//         sistemden
// Parametreler :
//   num_pages -> kaç adet girdi eklenecek
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void *allocKernelPages(unsigned long num_pages)
{
          void *physical_address;
          void *virtual_address;

          allocPages(num_pages,&physical_address);
          mapPages((unsigned long *)phys_to_virt((unsigned
long)KernelPageTable_High),
                              num_pages,
                              PTE_PRESENT|PTE_WRITE,
                              0x800000,
                              physical_address,
                              &virtual_address);

          return virtual_address;
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "freePages"
// Açıklama:
//         İşletim sisteminden daha önce alinmiş bir sayfayı, tekrar sisteme
//         geri vermek için kullanılır. Bu işlemde sürecin sayfa tablolarında,
//  kernel sayfa tablolarında ve sistemde frame takibi yapan listede
//  değişiklikler meydana gelebilir.
// Parametreler :
//   page_table ->hangi sayfa dizin tablosundaki o adrese ait girdi si-
//                                  linecek
//   num_pages -> ardışık kaç sayfa silinecek
//   address   -> sisteme verilecek adres başlangıcı
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void freePages(struct address_space *addr_spc,
                              unsigned long num_pages,
                              void *address)
{
          unsigned long index,i,j;
          void *physical;
          unsigned long *table;
```

```
	//------------------------------------------------------------
	// hangi sayfa tablosundan o elemanın çıkartılacağı ve o elemana
	// karşılık gelen fiziksel adres bulunuyor.

	//eğer kernel adres sahası içinden bir adres sisteme verilecek ise
	if((unsigned long)address >= 0x800000)
	{
		index=(virt_to_phys((unsigned long)address) >>PTE_SHIFT) &
PTE_MASK;
		table=(unsigned long *)phys_to_virt((unsigned
long)KernelPageTable_High);
	}
	//eğer kullanıcı adres sahasından bir sayfa çıkartılacak ise
	else
	{
		if((unsigned long)address>=0x400000)
		{
			index=(((unsigned long)address-0x400000)>>PTE_SHIFT) &
PTE_MASK;
			table=addr_spc->user_ptable_1;
		}
		else
		{
			index=((unsigned long)address>>PTE_SHIFT) &
PTE_MASK;
			table=addr_spc->user_ptable_0;
		}

	}
	//------------------------------------------------------------

	//------------------------------------------------------------
	// o sanal adrese karşılık gelen fiziksel adresi, sayfa tablosunu
	// kullanarak bul
	physical=(void *)(table[index] & 0xFFFFF000);
	//------------------------------------------------------------

	//------------------------------------------------------------
	// sayfa tablosunda, o elemana ait girdiyi 0'la
	for(i=0;i<num_pages;i++)
	{
		table[index+i]=0;
	}
	//------------------------------------------------------------

	//------------------------------------------------------------
	//frame takibini yapan listeden de o sayfayı çıkart.
```

```
	j=(unsigned long)physical >> 12;

	for(i=0;i<num_pages;i++)
	{
		// o sayfa artik boş..
		FrameMap[j+i]=0;
	}
	//-------------------------------------------------------------
}
//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "getPageDirectoryEntry"
// Açıklama:
//        Verilen sayfa dizin tablosundan,verilen adrese ilişkin sayfa tablo-
//        sunun fiziksel adresini döndürür.
// Parametreler :
//   page_directory   -> sayfa dizin tablosu
//   virtual_address  -> sanal adres
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
unsigned long getPageDirectoryEntry(unsigned long *page_directory,
                                                              unsigned long
virtual_address)
{
	unsigned long page_table_address;

	//-------------------------------------------------------------
	//sayfa dizin tablosunadaki, o adrese karşılık gelen elemanı döndür
	virtual_address=(virtual_address>>PDE_SHIFT) & PDE_MASK;
	page_table_address = page_directory[virtual_address] & PDE_PT_BASE;
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	return page_table_address;
	//-------------------------------------------------------------
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Get_Page_Table_Entry"
// Açıklama:
//        Verilen sayfa dizin tablosundan,verilen adrese ilişkin sayfa tablo-
//        sundaki fiziksel adres değerini döndürür.
// Parametreler :
//   page_directory   -> sayfa dizin tablosu
//   virtual_address  -> sanal adres
// Geri Dönüş Değeri:
```

```
//   YOK
//--------------------------------------------------------------------
unsigned long getPageTableEntry(unsigned long *page_directory,
                                                                unsigned long
virtual_address)
{
        unsigned long *page_table;

        //-------------------------------------------------------------
        //o adrese ait sayfa dizin tablosu girdisini al
        page_table=(unsigned long *)getPageDirectoryEntry(page_directory,

                  virtual_address);
        page_table=(unsigned long *)phys_to_virt((unsigned long)page_table);
        //-------------------------------------------------------------

        virtual_address=(virtual_address>>PTE_SHIFT) & PTE_MASK;
        //-------------------------------------------------------------
        // sayfa dizin tablosu girdisindeki değeri döndür.
        return (page_table[virtual_address] & PTE_P_BASE);
        //-------------------------------------------------------------
}
```

Ports.h

```
//  Sistem genelinde kullanılan önemli tanımlamaları ve sabitleri içerir.

#ifndef PORTS
#define PORTS

//--------------------------------------------------------------------------
//PIC programlanması için gerekli olan port adresleri
//Bu port numaraları kullanılarak donanım kesmeleri taşınacaktır. Ayrıca sistemin
kesme-
//lere tam olarak yanıt vermesi için, PIC programlamasının yapılması ve gereken
paramet-
//relerin verilmesi şarttır.BIOS bunu açılışta yapsa da , uyumluluk açısından tekrar
ya-
//pılması daha uygundur.
#define PIC_MASTER_PORT_0 0x20    //ICW1'in gönderileceği port numarası
(PIC1)
#define PIC_MASTER_PORT_1 0x21    //ICW2,ICW3 ve ICW4'ün gönderileceği
port numarası (PIC1)
#define PIC_SLAVE_PORT_0 0x0A0    //PIC2 için ilk port
#define PIC_SLAVE_PORT_1 0x0A1    //PIC2 için 2. port
#define EOI    0x20
```

```
//---------------------------------------------------------------------------
//PIT programlanması için gerekli portlar
#define PIT_1_COUNTER_0  0x40
#define PIT_1_COUNTER_1  0x41
#define PIT_1_COUNTER_2  0x42
#define PIT_1_CONTROL_REGISTER 0x43
#define PIT_2_COUNTER_0  0x48
#define PIT_2_COUNTER_1  0x49
#define PIT_2_COUNTER_2  0x4A
#define PIT_2_CONTROL_REGISTER 0x4B
//---------------------------------------------------------------------------
//8042 Klavye denetleyicisinin programlanabilmesi için gerekli olan portlar
//Bu port numaraları A20 adres bacağının aktif hale getirilmesi için kullanılacaktır.
#define KEYBOARD_DATA_REGISTER          0x60
#define KEYBOARD_COMMAND_REGISTER   0x64  //komut yazma
#define KEYBOARD_STATUS_REGISTER        0x64   //durum bilgisi alma

#endif
```

Process.h

```
//  Süreçler ile ilgili tanımlamaların yapıldığı kod bölgesidir.

#ifndef PROCESS
#define PROCESS

#include "Memory.h"

#define TASK_RUNNING            1
#define TASK_READY              2
#define TASK_WAITING            3
#define TASK_TERMINATED         4

//---------------------------------------------------------------------
//Intel 386 ve sonrası işlemcileri için donanımsal olarak tanımlanmış
// TSS (task state segment) yani süreç durum bilgilerini tutan yapı
struct task_state_segment
{
        long    Previous_Link;
        long    ESP0;
        long    SS0;
        long    ESP1;
        long    SS1;
        long    ESP2;
        long    SS2;
        long    CR3;
```

```
        long    EIP;
        long    EFlags;
        long    EAX;
        long    ECX;
        long    EDX;
        long    EBX;
        long    ESP;
        long    EBP;
        long    ESI;
        long    EDI;
        long    ES;
        long    CS;
        long    SS;
        long    DS;
        long    FS;
        long    GS;
        long    LDT_selector;
        long    IO_Bitmap_Base_Adress;     //31-16 I/O Bitmap, Bit 0->debug trap
};

// İşletim sistemimizde yer alacak olan bir sürecin tamamen iç yapısını
// tutan C yapısı
struct Task
{
        //Sürece ait hafıza tablolarınu tutan yapı---
        struct address_space *addr_space;
        //------------------------------------------

        //sürece ait özel veriler--------------------
        unsigned long ID;                  //süreç ID
        char Durum;                        //süreçin durumu
        //------------------------------------------

        //------------------------------------------
        //sürecin klavye tamponu
        char keyboard_buffer[256];
        //------------------------------------------

        // sürecin segment bilgileri...--------------
        //(fiziksel adres olarak)
        unsigned long code_segment_base;
        //------------------------------------------

        //sürecin durum bölgesi----------------------
        struct task_state_segment Tss; //süreçin durumu saklanacak
        //------------------------------------------

        //------------------------------------------
```

```
        // o sürecin sürec kuyruğundaki diğer elemanlar
        // ile bağlantısını sağlayan değişkenler
        struct Task *onceki_surec;
        struct Task *sonraki_surec;
};

//---------------------------------------------------------------------
// sistemdeki süreçleri tutan süreç listesi için bir yapı
struct Liste
{
        struct Task *liste_basi;
        struct Task *liste_sonu;
        unsigned long eleman_sayisi;
};

//--------------------------------------------------------------
// Bazı önemli makrolar...
#define FIRST_TSS_ENTRY 5

//verilen indexi GDT'deki indexe dönüştüren makrolar
//TSS= (ilkTSS + n) << 3
#define _TSS(n) (((unsigned long) n + FIRST_TSS_ENTRY) << 3)

//Task ve LDT yazmaçlarını yükleyen makrolar.
#define ltr(n) __asm__("ltr %%ax"::"a" (_TSS(n)))
#define lldt() __asm__("lldt %%ax"::"a" (sel_LDT))
//--------------------------------------------------------------

//--------------------------------------------------------------
// süreçler arası geçişi sağlayan makro
#define switch_to(n) {\
                                        struct {long a,b;} __tmp; \
                                        __asm__("movw %%dx,%1\n\t" \
                                       "ljmp %0" \
                                       ::"m" (*&__tmp.a),"m" (*&__tmp.b),"d"
_TSS(n));}
//--------------------------------------------------------------

void initTask();
void Exit();
void Exec(unsigned long code_segment_base);
void deleteProcess(struct Task *task);
void killProcess(unsigned long taskID);
#endif
```

Process.c

```
//  Süreçler ile ilgili tanımlamaların yapıldığı kod bölgesidir.

#include "Process.h"
#include "AsmDefines.h"
#include "Descriptor.h"
#include "Memory.h"
#include "Scheduler.h"
#include "Console.h"
#include "System.h"

//------------------------------------------------------------
// Global Descriptor Table tablosu (Start.asm'den)
extern struct i386_Descriptor Global_Descriptor_Table[256];
//------------------------------------------------------------

//------------------------------------------------------------
//tüm süreçlerin adres sahasına eklenecek kernel sayfa tabloları
extern unsigned long *KernelPageTable_High;
//------------------------------------------------------------;

//------------------------------------------------------------
//(Scheduler.c)
extern struct Liste Hazir_Surec_Listesi;
extern struct Liste Bitmis_Surec_Listesi;
extern struct Liste Bekleyen_Surec_Listesi;

//(Scheduler.c)
extern struct Task *aktif_surec;
//------------------------------------------------------------

//------------------------------------------------------------
static unsigned long Surec_Sayisi=0;// sistemde bulunan tüm süreclerin
                                    // toplam sayısını tutan değişken
//------------------------------------------------------------

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "LDT_Init"
// Açıklama:
//       Sistemde tüm süreçlerin kullanacağı bir LDT tablosu oluşturur.
// Parametreler :
//       YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
```

```
void initTask()
{

	//------------------------------------------------------------
	// ilk context-switch için gerekli TSS yaratılıyor...
	Descriptor_Doldur(&Global_Descriptor_Table[100],	//rastgele bir index
					0x68,		//104 byte=TSS uzunlugu
					(unsigned long)0x300000,
					PRESENT | TSS,
					AVAILABLE);
	//------------------------------------------------------------

	//------------------------------------------------------------
	// gerekli yazmaçları ilkle
	ltr(95);//TR'yi yükle.verilen 100. girdi 95. süreçin
			//TSS'sine karşılık geliyor
	//------------------------------------------------------------

	Println("<Zeugma> Surec yonetimi icin GDT ilklendi.");
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "fill_GDT"
// Açıklama:
//	Sistemde sürece ait TSS alanına ilişkin tanımlayıcıyı GDT'ye
//	ekler.
// Parametreler :
//	task -> GDT değerleri dolacak olan süreç yapısı
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void fill_GDT(struct Task *task)
{
	int GDT_index;
	struct i386_Descriptor *gdt;

	//------------------------------------------------------------
	//o sürecin LDT ve TSS bölgeleri için gerekli GDT selektör indexi
	//hesaplanıyor
	// 0->NULL desc.
	// 1->KernelCS desc.
	// 2->KernelDS desc.
	// 3->UserCS desc.
	// 4->UserDS desc.
	//	--------
	// 5-> TSS0 |-- 0.süreç için TSS
	//  --------
```

```
	// 6-> TSS1 |-- 1.süreç için TSS
	//  --------
	// |
	// |
	// V
	//-------------------------------------------------------------

	GDT_index=task->ID + 5;
	gdt=(struct i386_Descriptor *)phys_to_virt((unsigned
long)Global_Descriptor_Table);
	//-------------------------------------------------------------
	//sürecin TSS'sine ait GDT descriptorunu doldur
	Descriptor_Doldur(&gdt[GDT_index],
				0x68, //104 byte=TSS uzunlugu
				(unsigned long)&task->Tss,
				PRESENT | TSS,
				AVAILABLE);
	//-------------------------------------------------------------
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "fillTSS"
// Açıklama:
//	Sürecin gerekli durum bilgilerinin saklandığı TSS yapısını doldur
// Parametreler :
//  task ->	süreç yapısı
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void fill_TSS(struct Task *task)
{
	//-------------------------------------------------------------
	//segment selektörleri LDT değerlerini göstersin
	task->Tss.CS=sel_UserCS;
	task->Tss.DS=sel_UserDS;
	task->Tss.ES=sel_UserDS;
	task->Tss.FS=sel_UserDS;
	task->Tss.GS=sel_UserDS;
	task->Tss.SS=sel_UserDS;
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	//yığıt segmenti kullanıcı yığıt segmentini göstersin
	task->Tss.ESP=(unsigned long)allocUserPages(1,
					task->addr_space->user_ptable_1)
					 + 4095;
	//-------------------------------------------------------------
```

```
	//------------------------------------------------------------
	//SS0 ve ESP0 kernel yığıt segmentini göstersin
	task->Tss.SS0=sel_KernelDS;
	//task->Tss.ESP0=(long)(task->Kernel_Stack+1023);
	task->Tss.ESP0=(unsigned long)allocKernelPages(1)+4095;
	//------------------------------------------------------------

	//------------------------------------------------------------
	//sürecin tüm yazmaçlarını ilk başta sıfırla
	task->Tss.EAX=0;
	task->Tss.EBX=0;
	task->Tss.ECX=0;
	task->Tss.EDX=0;
	task->Tss.ESI=0;
	task->Tss.EDI=0;
	task->Tss.EBP=0;
	//------------------------------------------------------------

	//------------------------------------------------------------
	//flag değeri yazılıyor
	task->Tss.EFlags=EFLG_IF | EFLG_IOPL3; //interrupt enable ve IOPL
	task->Tss.EIP=0;
	task->Tss.IO_Bitmap_Base_Adress=0;
	task->Tss.LDT_selector=0;
	//------------------------------------------------------------
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "createPageTables"
// Açıklama:
//	verilen sürecin sayfa tablolarına yer ayırır, onları doldurur.
// Parametreler :
//  task ->	süreç yapısı
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void createPageTables(struct Task *task)
{
	void *physical;
	void *virt;
	int i;

	//------------------------------------------------------------
	//sürece ait sayfa dizin tablosu ve sayfa tabloları  doldurulmalı.
	//Bu nedenle sürece ait sayfa dizin tablosu ve sayfa tabloları için
	//gerekli yer, yine işletim sistemi tarafından atanmalı
```

```
	allocPages(3,&physical);
	mapPages(KernelPageTable_High,3,PTE_PRESENT|PTE_WRITE,
			VIRT_OFFSET,physical,&virt);
	task->addr_space=(struct address_space *)virt;
	//-----------------------------------------------------------

	//-----------------------------------------------------------
	// sayfa tabloları ilkleniyor...
	for(i=0;i<1024;i++)
	{
		task->addr_space->pdir[i]=0;
		task->addr_space->user_ptable_0[i]=i*4096|PTE_PRESENT|
							PTE_WRITE|PTE_USER;
		task->addr_space->user_ptable_1[i]=0;
	}

	// sayfa dizininin ilk elemanı, sürece ait ilk sayfa tablosunu göste-
	// riyor.
	task->addr_space->pdir[0]= (unsigned long)(physical+4096)|PDE_PRESENT
				|PDE_WRITE|PDE_USER;
	task->addr_space->pdir[1]= (unsigned long)(physical+4096*2)|
				PDE_PRESENT|PDE_WRITE|PDE_USER;
	// ikinci eleman ise kernel sayfa tablosunu gösteriyor(fiziksel adres)
	task->addr_space->pdir[2]= (unsigned long)KernelPageTable_High
				|PDE_PRESENT|PDE_WRITE;
	//-----------------------------------------------------------

	//-----------------------------------------------------------
	//sürecin sayfa tablosunu göster
	task->Tss.CR3=(unsigned long)physical;
	//-----------------------------------------------------------
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Create_Process"
// Açıklama:
//	Sistemde süreç oluşturmak için kullanılan temel fonksiyondur.
// Parametreler :
//  task ->	yaratılacak süreç için yapı
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
struct Task *createProcess()
{
	struct Task *yeni_surec;

	//-----------------------------------------------------------
```

```
        //Sürece ait bilgilerin işletim sistemi tarafından tutulması için
        //gerekli fiziksel bellek ayırma işlemleri...
        //boş bir sayfa al
        //sürece ait bilgiler bu sayfa içerisinde tutulacak
        yeni_surec=(struct Task *)allocKernelPages(1);
        //------------------------------------------------------------

        //------------------------------------------------------------
        //sürece ait sayfa tablolarını doldur
        createPageTables(yeni_surec);
        //------------------------------------------------------------

        //------------------------------------------------------------
        //süreçin ID'sini süreç listesindeki eleman sayısı olarak atanıyor
        yeni_surec->ID=Surec_Sayisi;
        //------------------------------------------------------------

        //------------------------------------------------------------
        //sürece ait sayfa durum bilgilerini doldur
        fill_TSS(yeni_surec);
        //------------------------------------------------------------

        //------------------------------------------------------------
        //o sürecin LDT ve TSS bölgeleri için gerekli GDT girdilerini
        //yarat
        fill_GDT(yeni_surec);
        //------------------------------------------------------------

        //------------------------------------------------------------
        //süreç çalıştırılmaya hazır...
        yeni_surec->Durum=TASK_READY;
        //------------------------------------------------------------

        //sistemdeki süreç sayısını 1 arttır
        Surec_Sayisi++;

        return yeni_surec;
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Exec"
// Açıklama:
//        Sistemde süreç oluşturmak için kullanılan temel fonksiyondur.
// Parametreler :
//  code_segment_base -> süreçin kod başlangıç adresi(fiziksel)
// Geri Dönüş Değeri:
//   YOK
```

```
//--------------------------------------------------------------------
void Exec(unsigned long code_segment_base)
{
	struct Task *yeni_surec;		//oluşturulacak süreç için yapı

	//-------------------------------------------------------------
	//yeni bir süreç oluştur
	yeni_surec=(struct Task *)createProcess();
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	//süreç yapısı dolduruluyor...(fiziksel adreslerle)
	yeni_surec->code_segment_base=code_segment_base;
	yeni_surec->Tss.EIP=code_segment_base;
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	Insert_Task((struct Liste *)phys_to_virt((unsigned
long)&Hazir_Surec_Listesi),
				yeni_surec);
	//süreç, süreç kuyruğuna yerleştirildi...Çalıştırılmaya hazır...
	//-------------------------------------------------------------
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "deleteProcess"
// Açıklama:
//	Sistemde bulunan bir süreçi silmek için kullanılır
// Parametreler :
//  task -> silinecek süreçe işaretçi
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void deleteProcess(struct Task *task)
{
	struct address_space *addr_spc;

	//-------------------------------------------------------------
	//süreç,o an zaten aktif süreç listesinden çıkartılmıştır ve ça-
	//lışan süreçtir.dolayısıyla, sadece sürece ait bellek bölgeleri-
	//ni geri vermemiz, yeterli olacaktir.
	//-------------------------------------------------------------

	addr_spc=task->addr_space;

	//-------------------------------------------------------------
	// sürece ait kullanıcı yığıtı sisteme veriliyor
```

```
        freePages(addr_spc,1,(void *)task->Tss.ESP);
        //------------------------------------------------------------

        //------------------------------------------------------------
        // sürece ait kernel yığıtı sisteme veriliyor
        freePages(addr_spc,1,(void *)task->Tss.ESP0);
        //------------------------------------------------------------

        //------------------------------------------------------------
        // sürece ait süreç yapısı sisteme veriliyor
        freePages(addr_spc,1,(void *)task);
        //------------------------------------------------------------

        //------------------------------------------------------------
        // sürece ait sayfa tabloları için ayırılmış bölgeler de tekrar
        // sisteme veriliyor.
        freePages(addr_spc,3,(void *)addr_spc);
        //------------------------------------------------------------

}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Exit"
// Açıklama:
//        Sistemde bulunan bir süreç,bu çağırım sayesinde çalışmasını sonlandırır.
// Parametreler :
//  YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void Exit()
{
        struct Task **surec;
        struct Task *terminated_task;

        //------------------------------------------------------------
        // Kesmeleri kapat
        cli();
        //------------------------------------------------------------

        //------------------------------------------------------------
        // o an aktif olan surecin sanal adresini al
        surec=(struct Task **)phys_to_virt((unsigned long)&aktif_surec);

        terminated_task=(*surec);
        //------------------------------------------------------------
```

```
	//------------------------------------------------------------
	// aktif sureci NULL yap
	(*surec)=NULL;
	//------------------------------------------------------------

	//------------------------------------------------------------
	// sureci bitmiş süreçler listesine koy
	terminated_task->Durum=TASK_TERMINATED;
	Insert_Task((struct Liste *)phys_to_virt((unsigned long)
			&Bitmis_Surec_Listesi),
				terminated_task);
	//------------------------------------------------------------

	//------------------------------------------------------------
	// Kesmeleri aç
	sti();
	//------------------------------------------------------------

	//------------------------------------------------------------
	// işlemci bölüştürücüsünü çağır ki o süreci çalıştırmayı kessin
	Scheduler();
	//------------------------------------------------------------
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "surecBul"
// Açıklama:
//         ID'si ve liste verilen süreci geri döndürür
// Parametreler :
//   ID -> sürecin ID'si
//   lst-> aranacak liste
// Geri Dönüş Değeri:
//   bulunan task
//--------------------------------------------------------------------
struct Task *surecBul(struct Liste *lst,unsigned long ID)
{
	unsigned long bulundu;
	struct Task *task;

	bulundu=0;

	//------------------------------------------------------------
	// elemanı listede ara
	if(lst->eleman_sayisi!=0)
	{
		task=lst->liste_basi;
		while(task!=NULL)
```

```
			{
				if(task->ID==ID)
				{
					bulundu=1;
					break;
				}
				else
					task=task->sonraki_surec;
			}
		}
		//-----------------------------------------------------------

		//-----------------------------------------------------------
		// eğer bulunduysa listeden çıkart ve döndür
		if(bulundu)
		{
			Remove_Task(lst,task);
			return task;
		}
		else
			return NULL;
		//-----------------------------------------------------------
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "endProcess"
// Açıklama:
//          verilen süreci bitmis süreçler listesine koyar
// Parametreler :
//   task -> sonlanacak süreç
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void endTask(struct Task *task)
{
		cli();
		//-----------------------------------------------------------
		// sureci bitmiş süreçler listesine koy
		task->Durum=TASK_TERMINATED;
		Insert_Task((struct Liste *)phys_to_virt((unsigned long)
					&Bitmis_Surec_Listesi),
					task);
		//-----------------------------------------------------------
		sti();

}
//--------------------------------------------------------------------
```

```
// Fonksiyon Adı:
//   "killProcess"
// Açıklama:
//         ID'si verilen süreci öldürür.
// Parametreler :
//   ID -> sürecin ID'si
// Geri Dönüş Değeri:
//   YOK
//----------------------------------------------------------------------
void killProcess(unsigned long taskID)
{
        struct Liste *lst;
        struct Task *task;

        Set_Color(5);
        //------------------------------------------------------------
        // eğer shell veya init öldürülmek istenirse buna izin verme
        if((taskID==0)||(taskID==1))
        {
                Println("<kill> init veya shell oldurulemez!");
                return ;
        }
        //------------------------------------------------------------

        //------------------------------------------------------------
        // çalışmaya hazir süreçlerin tutulduğu listeye ait sanal adres
        //alınıyor
        lst=(struct Liste *)phys_to_virt((unsigned long)&Hazir_Surec_Listesi);
        task=surecBul(lst,taskID);
        if(task!=NULL)
        {
                endTask(task);
                return;
        }
        //------------------------------------------------------------

        //------------------------------------------------------------
        // bekleyen süreçlerin tutulduğu listeye ait sanal adres
        //alınıyor
        lst=(struct Liste *)phys_to_virt((unsigned long)&Bekleyen_Surec_Listesi);
        task=surecBul(lst,taskID);
        if(task!=NULL)
        {
                endTask(task);
                return ;
        }
        //------------------------------------------------------------
        Println("<kill> surec yaratilmamis!");
```

```
}
```

Scheduler.h

```
//  İşletim sistemimiz birden fazla sürecin aynı anda çalışmasına izin
// verecek şekilde tasarlanmışıtr.Bu sebeple, timer donanım kesmesine
// yerleştirilecek olan ve sistemde süreçlere işlemciyi eşit zaman
// aralıklari ile dağıtacak olan fonksiyon, burada tanımlanmıştır.

#ifndef SCHEDULER
#define SCHEDULER

#include "Process.h"

void Remove_Task(struct Liste *lst,struct Task *task);
void Insert_Task(struct Liste *lst,struct Task *task);
void Init_Scheduler();
void surecBilgisi();

#endif
```

Scheduler.c

```
//  İşletim sistemimiz birden fazla sürecin aynı anda çalışmasına izin
// verecek şekilde tasarlanmışıtr.Bu sebeple, timer donanım kesmesine
// yerleştirilecek olan ve sistemde süreçlere işlemciyi eşit zaman
// aralıklari ile dağıtacak olan fonksiyon, burada tanımlanmıştır.

#include "Scheduler.h"
#include "Console.h"
#include "Ports.h"
#include "AsmDefines.h"
#include "Descriptor.h"
#include "System.h"
#include "Lib.h"

extern void Timer_Interrupt();


//------------------------------------------------------------
// Interrupt Descriptor Table tablosu (start.asm'den)
extern struct i386_Descriptor Interrupt_Descriptor_Table[256];
//------------------------------------------------------------
```

```
//------------------------------------------------------------
// Sistemdeki çalışmaya hazir tüm süreçlere ait süreç yapılarını
// tutan liste (Ready Queue)
struct Liste Hazir_Surec_Listesi;
// Sistemdeki IO bekleyen tüm tüm süreçlere ait süreç yapılarını
// tutan liste (Waiting Queue);
struct Liste Bekleyen_Surec_Listesi;
// Sistemdeki çalışmasını bitirmiş süreçleri tutan liste
// (Terminated Queue)
struct Liste Bitmis_Surec_Listesi;
//------------------------------------------------------------

//------------------------------------------------------------
//o an çalışan süreç
struct Task *aktif_surec;
//------------------------------------------------------------

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "freeTask"
// Açıklama:
//         Sistemde bitmiş süreç listesinde bulunan süreçlerden bir tanesini
//  alır ve o sürece ait adres sahasını işletim sistemine geri verir
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void freeTask()
{
        struct Task *task;
        struct Liste *lst;

        //------------------------------------------------------------
        // bitmiş süreç listesinin sanal adresini al
        lst=(struct Liste *)phys_to_virt((unsigned long)&Bitmis_Surec_Listesi);
        //------------------------------------------------------------

        //------------------------------------------------------------
        // eleman yok ise çık
        if(lst->eleman_sayisi==0)
                return;
        //------------------------------------------------------------

        //------------------------------------------------------------
        //eğer liste boş değilse
        task=lst->liste_basi;           //liste başını al
        Remove_Task(lst,task);          //listeden çıkart
```

```
        deleteProcess(task);            //sürece ait tüm bellek bölgesini işle-
                                        //tim sistemine ver
        //------------------------------------------------------------
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Scheduler"
// Açıklama:
//        Süreç organizasyonun sağlar, işlemciyi round-robin algoritmasına
//        göre süreçlere dağıtır.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
extern void Scheduler()
{
        struct Task *task;
        struct Liste *lst;

        //------------------------------------------------------------
        // eğer var ise bitmiş süreç listesinden bir süreci çıkart ve onu
        // sistemden tamamiyle sil
        freeTask();
        //------------------------------------------------------------

        //------------------------------------------------------------
        // İşlemci paylaşımını yapan kod bölgesi ---> Hafızada birden
        // fazla sürecin bulundurulması ve çalıştırılması işleminin
        // merkezi
        //
        // hazir süreç listesi, sistemde çalışmaya hazır süreçleri tutmaktadır.
        // algoritmada, hazir süreç listesinin liste başı elemanı alınır ve o
        // an çalışmakta olan süreç ise yine aynı listenin sonuna eklenir.
        // böylece FCFS algoritması ve ROUND_ROBIN algoritmaları beraberce iş-
        // lemektedir.

        //------------------------------------------------------------
        // hazir süreç listesinin sanal adresini al
        lst=(struct Liste *)phys_to_virt((unsigned long)&Hazir_Surec_Listesi);
        //------------------------------------------------------------

        //------------------------------------------------------------
        //eğer liste boş değilse
        if (lst->eleman_sayisi!=0)
        {
                //------------------------------------------------------------
```

```
                // hazir süreç listesinden, liste başı sürecini çıkart
                task=lst->liste_basi;
                Remove_Task(lst,task);
                //------------------------------------------------------------

                //------------------------------------------------------------
                //aktif süreci (eğer geçerli bir süreç ise) tekrar hazir sürec
                //listesinin sonuna ekle...
                if(aktif_surec!=NULL)
                {
                        //aktif süreç, artık çalışmıyor...
                        aktif_surec->Durum=TASK_READY;
                        //hazir süreç listesine ekle
                        Insert_Task(lst,aktif_surec);
                }
                //------------------------------------------------------------

                //------------------------------------------------------------
                //çalışacak olan sürecimiz artık yeni süreçtir
                aktif_surec=task;
                //------------------------------------------------------------

                //------------------------------------------------------------
                //yeni süreç çalışacağı için durumu "ÇALIŞIYOR" yapılıyor
                task->Durum=TASK_RUNNING;
                //------------------------------------------------------------

                //------------------------------------------------------------
                //Task switching işlemi yapılıyor...(çok basit...:))
                switch_to(task->ID);
                //------------------------------------------------------------
        }
        //------------------------------------------------------------
}

//------------------------------------------------------------------
// Fonksiyon Adı:
//   "Remove_Task"
// Açıklama:
//        Sistemde, süreç listesinden verilen süreç çıkartılır.
// Parametreler :
//   task -> çıkartılacak sürecin adresi
//   lst  -> işlem yapılacak liste
// Geri Dönüş Değeri:
//   YOK
//------------------------------------------------------------------
void Remove_Task(struct Liste *lst,struct Task *task)
{
```

```
	//------------------------------------------------------------
	//eğer listede sadece 1 süreç var ise,liste baş ve sonu
	//işlemleri ele alınmalı
	if(lst->eleman_sayisi==1)
	{
		lst->liste_basi=NULL;
		lst->liste_sonu=NULL;
	}
	else //eğer liste başı çıkartılıyorsa
		if(lst->liste_basi==task)
		{
			//liste başı ayarlamaları...
			lst->liste_basi=task->sonraki_surec;
			lst->liste_basi->onceki_surec=NULL;
		}
		else //eğer liste sonu çıkartılıyorsa
			if(lst->liste_sonu==task)
			{
				//liste basi ve sonu ayarlamaları
				lst->liste_sonu=task->onceki_surec;
				lst->liste_sonu->sonraki_surec=NULL;
			} //eğer normal bir eleman çıkartılıyorsa
			else
			{
				task->onceki_surec->sonraki_surec =
								task->sonraki_surec;
				task->sonraki_surec->onceki_surec=
								task->onceki_surec;
			}
	//------------------------------------------------------------

	(lst->eleman_sayisi)--;
}
//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Insert_Task"
// Açıklama:
//         Sistemde, süreç listesine verilen süreç eklenir.
// Parametreler :
//   task -> eklenecek sürecin adresi
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void Insert_Task(struct Liste *lst,struct Task *task)
{

	//------------------------------------------------------------
	//listede hiç eleman yoksa
```

```
	if(lst->eleman_sayisi==0)
	{
		//------------------------------------------------------------
		lst->liste_basi=task;
		lst->liste_sonu=task;
		task->onceki_surec=NULL;
		task->sonraki_surec=NULL;
		//------------------------------------------------------------
	}
	else
	{
		//------------------------------------------------------------
		//liste sonundan sonraki süreç, yeni eklenen süreç
		lst->liste_sonu->sonraki_surec=task;
		//eklenen süreçten önceki süreç liste sonu
		task->onceki_surec=lst->liste_sonu;
		//sonraki süreç yok
		task->sonraki_surec=NULL;
		//yeni liste sonu, yeni süreç
		lst->liste_sonu=task;
		//------------------------------------------------------------
	}
	//------------------------------------------------------------

	(lst->eleman_sayisi)++;
}

//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "surecBilgisi"
// Açıklama:
//	Sistemde bulunan surecler ve süreç listeleri hakkında bilgi veren
//  fonksiyondur.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void surecBilgisi()
{
	struct Liste *lst;
	struct Task *task;
	struct Task **aktif;

	Set_Color(13);
	Println("Sistemdeki ana surecler:");
	Set_Color(12);
	Println("	Init sureci icin ID 0");
```

```
	Println("	Shell sureci icin ID 1");

	//-------------------------------------------------------------
	// çalışmaya hazir süreçlerin tutulduğu listeye ait sanal adres
	//alınıyor
	lst=(struct Liste *)phys_to_virt((unsigned long)&Hazir_Surec_Listesi);
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	//listeyi dolas
	Set_Color(13);
	Println("TASK_READY surecler:");
	Set_Color(12);
	if(lst->eleman_sayisi==0)
		Println("	Bu listede eleman yok.");
	else
	{
		task=lst->liste_basi;
		while(task!=NULL)
		{
			Print("	Surec ID ->");Print_Sayi_Hex(task-
>ID);Println("");
			task=task->sonraki_surec;
		}
		Set_Color(10);
		Print("	Toplam->");Print_Sayi_Hex(lst->eleman_sayisi);
		Println("");
	}
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	// bekleyen süreçlerin tutulduğu listeye ait sanal adres
	//alınıyor
	lst=(struct Liste *)phys_to_virt((unsigned long)&Bekleyen_Surec_Listesi);
	//-------------------------------------------------------------

	//-------------------------------------------------------------
	//listeyi dolas
	Set_Color(13);
	Println("TASK_WAITING surecler:");
	Set_Color(12);
	if(lst->eleman_sayisi==0)
		Println("	Bu listede eleman yok.");
	else
	{
		task=lst->liste_basi;
		while(task!=NULL)
		{
```

```
				Print("	Surec ID ->");
				Print_Sayi_Hex(task->ID);Println("");
				task=task->sonraki_surec;
			}
			Set_Color(10);
			Print("	Toplam->");Print_Sayi_Hex(lst->eleman_sayisi);
			Println("");
		}
		//----------------------------------------------------------------

		//----------------------------------------------------------------
		// çalışması bitmiş süreçlerin tutulduğu listeye ait sanal adres
		//alınıyor
		lst=(struct Liste *)phys_to_virt((unsigned long)&Bitmis_Surec_Listesi);
		//----------------------------------------------------------------

		//----------------------------------------------------------------
		//listeyi dolas
		Set_Color(13);
		Println("TASK_TERMINATED surecler:");
		Set_Color(12);
		if(lst->eleman_sayisi==0)
			Println("	Bu listede eleman yok.");
		else
		{
			task=lst->liste_basi;
			while(task!=NULL)
			{
				Print("	Surec ID ->");
				Print_Sayi_Hex(task->ID);Println("");
				task=task->sonraki_surec;
			}
			Set_Color(10);
			Print("	Toplam->");
			Print_Sayi_Hex(lst->eleman_sayisi);Println("");
		}
		//----------------------------------------------------------------

		//----------------------------------------------------------------
		// o an çalışan süreç alınıyor
		aktif=(struct Task **)phys_to_virt((unsigned long)&aktif_surec);
		//----------------------------------------------------------------
		Set_Color(13);
		Println("TASK_RUNNING surec:");
		Set_Color(12);
		Print("	Surec ID ->");Print_Sayi_Hex((*aktif)->ID);Println("");

	}
```

```
//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "Init_Scheduler"
// Açıklama:
//        Sistemde bulunan timer interrupt'ı üzerine yerleştirilecek işlem-
//        ci planlayıcısının ilkleme ve aktivasyon işlemleri burada yapılıyor.
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void Init_Scheduler()
{
        char value;
        int i;

        //-----------------------------------------------------------
        //Timer interrupt'ını handle edeck kod parçası IDT'ye işlensin
        Gate_Doldur((struct i386_Gate *)&Interrupt_Descriptor_Table[0x70],
                        (unsigned long)Timer_Interrupt,      //handler
                        sel_KernelCS,                        //selector
                        0,                                   //parametre sayisi
                        INTERRUPT_GATE|PRESENT);   //access
        //-----------------------------------------------------------

        //-----------------------------------------------------------
        //IRQ 0 = Timer Interrupt aktif hale getiriliyor...
        //interrupt mask alınıyor
        value=in_port(PIC_MASTER_PORT_1);
        //IRQ 0 aktif hale gelsin
        out_port((value&0xfe),PIC_MASTER_PORT_1);
        //-----------------------------------------------------------

        //-----------------------------------------------------------
        //Hazir Süreç Listesini ilkle
        Hazir_Surec_Listesi.liste_basi=NULL;
        Hazir_Surec_Listesi.liste_sonu=NULL;
        Hazir_Surec_Listesi.eleman_sayisi=0;
        //-----------------------------------------------------------

        //-----------------------------------------------------------
        //Bekleyen Süreç Listesini ilkle
        Bekleyen_Surec_Listesi.liste_basi=NULL;
        Bekleyen_Surec_Listesi.liste_sonu=NULL;
        Bekleyen_Surec_Listesi.eleman_sayisi=0;
        //-----------------------------------------------------------
```

```
	//-----------------------------------------------------------
	//Bitmiş Süreç Listesini ilkle
	Bitmis_Surec_Listesi.liste_basi=NULL;
	Bitmis_Surec_Listesi.liste_sonu=NULL;
	Bitmis_Surec_Listesi.eleman_sayisi=0;
	//-----------------------------------------------------------

	//-----------------------------------------------------------
	//başlangıçta, çalışan süreç yok...
	aktif_surec=NULL;
	//-----------------------------------------------------------

	Println("<Zeugma> Scheduler ayarlari yapildi.");
}
```

System.h

```
//  Sistem genelinde kullanılan önemli tanımlamaları ve sabitleri içerir.

#ifndef SYSTEM
#define SYSTEM

#define NULL 0x00000000

#endif
```

Timer.h

```
//  Sistemde bulunan PIT programlanmasi için gerekli bilgileri içerir
#ifndef TIMER
#define TIMER

#define HZ 10

//fonksiyon prototipleri
void Init_Timer();

#endif
```

Z_Api.h

```
//  Sistemdeki API prototiplerini içerir

#ifndef Z_API
#define Z_API

void API_Cls();
void API_Set_Cursor(int koord_x,int koord_y);
void API_Print(char *Karakter_Dizisi);
void API_Println(char *Karakter_Dizisi);
void API_Exec(unsigned long code_segment_base);
void API_Exit();

#endif
```

API_Table.c

```
//  API çağırımlarını için gerekli olan fonksiyon adresleri tablosunu
// içerir...

#include "Console.h"
#include "Process.h"
#include "Scheduler.h"
#include "Keyboard.h"
#include "Lib.h"

unsigned long Zeugma_Api_Table[]={(unsigned long)Cls,                    //0
                                  (unsigned long)Set_Cursor,             //1
                                  (unsigned long)Print,                  //2
                                  (unsigned long)Println,                //3
                                  (unsigned long)Exec,                   //4
                                  (unsigned long)Exit,                   //5
                                  (unsigned long)Set_Color,              //6
                                  (unsigned long)Set_Background_Color,   //7
                                  (unsigned long)Scanf,                  //8
                                  (unsigned long)surecBilgisi,           //9
                                  (unsigned long)killProcess,            //10
                                  (unsigned long)Print_Sayi_Hex          //11
                                 };
```

Z_Api.c

```
//  Sistemdeki API çağırımlarını içerir

#define API_CLS_INDEX                  0
#define API_SET_CURSOR_INDEX           1
#define API_PRINT_INDEX                2
#define API_PRINTLN_INDEX              3
#define API_EXEC_INDEX                 4
#define API_EXIT_INDEX                 5
#define API_SET_COLOR_INDEX            6
#define API_SET_BG_COLOR_INDEX         7
#define API_SCANF_INDEX                8
#define API_SUREC_BILGISI_INDEX        9
#define API_KILL_PROCESS_INDEX         10
#define API_PRINT_SAYI_HEX_INDEX       11

extern unsigned long Zeugma_Api_Table[];

//---------------------------------------------------------------------
// Fonksiyon Adı:
//   "SystemCall"
// Açıklama:
//   Sistemden gerekli çağırımın yapılması işleminden sorumludur.
// Parametreler :
//   API_Index - >Çağırılacak API'nin indexi
//          param1    - >1. parametre
//          param2    - >2. parametre
//          param3    - >3. parametre
//          param4    - >4. parametre
// Geri Dönüş Değeri:
//   YOK
//---------------------------------------------------------------------
long SystemCall(int API_Index,       //çağırılan fonksiyon indexi
                        long param1,   //1. parametre
                        long param2,   //2. parametre
                        long param3,   //3. parametre
                        long param4)  //4. parametre
{
        long ret;


        //ilgili API'yi parametreleri göndererek çağır
        __asm__ __volatile__ ("int $0x47 "
                              :"=a"(ret)
                              :"a"(API_Index),
                               "b"(param1),
```

```
                                                "c"(param2),
                                                "d"(param3),
                                                "S"(param4));
        return ret;
}


//---------------------*********API Bölgesi*********-------------------------//
//------------------------------------------------------------
//Cls
void API_Cls()
{
        SystemCall(API_CLS_INDEX,0,0,0,0);
}
//------------------------------------------------------------

//------------------------------------------------------------
//Set Cursor
void API_Set_Cursor(int koord_x,int koord_y)
{
        SystemCall(API_SET_CURSOR_INDEX,koord_x,koord_y,0,0);
}
//------------------------------------------------------------

//------------------------------------------------------------
//Print
void API_Print(char *Karakter_Dizisi)
{
        SystemCall(API_PRINT_INDEX,(long)Karakter_Dizisi,0,0,0);
}
//------------------------------------------------------------

//------------------------------------------------------------
//Println
void API_Println(char *Karakter_Dizisi)
{
        SystemCall(API_PRINTLN_INDEX,(long)Karakter_Dizisi,0,0,0);
}
//------------------------------------------------------------

//------------------------------------------------------------
//Exec
void API_Exec(unsigned long code_segment_base)
{
        SystemCall(API_EXEC_INDEX,code_segment_base,0,0,0);
}
//------------------------------------------------------------
```

```
//------------------------------------------------------------
//Exit
void API_Exit()
{
	SystemCall(API_EXIT_INDEX,0,0,0,0);
}
//------------------------------------------------------------

//------------------------------------------------------------
//Set_Color
void API_Set_Color(char color)
{
	SystemCall(API_SET_COLOR_INDEX,color,0,0,0);
}
//------------------------------------------------------------

//------------------------------------------------------------
//Set_Background_Color
void API_Set_Background_Color(char color)
{
	SystemCall(API_SET_BG_COLOR_INDEX,color,0,0,0);
}
//------------------------------------------------------------

//------------------------------------------------------------
// Scanf
void API_Scanf(char *komut)
{
	SystemCall(API_SCANF_INDEX,(unsigned long)komut,0,0,0);
}
//------------------------------------------------------------

//------------------------------------------------------------
//surecBilgisi
void API_Surec_Bilgisi()
{
	SystemCall(API_SUREC_BILGISI_INDEX,0,0,0,0);
}
//------------------------------------------------------------

//------------------------------------------------------------
//killProcess
unsigned long API_Kill_Process(unsigned long taskID)
{
	SystemCall(API_KILL_PROCESS_INDEX,taskID,0,0,0);
}
//------------------------------------------------------------
```

```
//----------------------------------------------------------
//Print_Sayi_Hex
void  API_Print_Sayi_Hex(unsigned long sayi)
{
        SystemCall(API_PRINT_SAYI_HEX_INDEX,sayi,0,0,0);
}
//----------------------------------------------------------
```

Timer.c

```
//  İşletim sisteminin donanım kesmelerini yerleştirmesi ve bu kes-
//  meler yardımı ile sistemi denetlemesi görevlerinden sorumludur.

#include "Timer.h"
#include "AsmDefines.h"
#include "Console.h"
#include "Ports.h"

#define LATCH (1193180/HZ)

//-----------------------------------------------------------------
// Fonksiyon Adı:
//   "Init_Timer"
// Açıklama:
//         Sistemde bulunan PIT (Programmable Interval Timer)'ı programlar
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//-----------------------------------------------------------------
void Init_Timer()
{
        out_port(0x36,PIT_1_CONTROL_REGISTER); //binary ,
                                        //mode 3 (Square Wave Rate Generator)
                                        //0.kanal
                                        //LSB ve MSB

        //PIT 18.2 hertz sinyal üretmesi için programlanıyor.
        out_port(LATCH & 0xff,PIT_1_COUNTER_0) //LSB (Least significant
byte)
        out_port(LATCH >> 8 ,PIT_1_COUNTER_0)  //MSB (Most significant
byte )

        Println("<Zeugma> PIT(Programmable Interval Timer) cipi programlandi.");
}
```

Shell.c

```
//  İşletim sisteminin komut yorumlayıcısı

#include "Z_Api.h"
#include "Lib.h"


void komutBekle();
void logoCiz();
void surec1();
void surec2();
void surec3();
void surec4();

//-----------------------------------------------------------------------
// Fonksiyon Adı:
//   "init"
// Açıklama:
//   Sistemde başlangıçta yaratılan ve hiçbir suretle sistemden atılma-
//        yan surec
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//-----------------------------------------------------------------------
extern void Shell()
{
	logoCiz();
	API_Println("-----------------------------Zeugma
Shell------------------------------");
	API_Set_Color(5);
	API_Println("");
	API_Println("Sistem yardimi icin \"yardim\" yaziniz.");
	//-------------------------------------------------------------
	// Kullanıcıdan komut alma işlemini yapar
	komutBekle();
	//-------------------------------------------------------------
}

//-----------------------------------------------------------------------
// Fonksiyon Adı:
//   "logoCiz"
// Açıklama:
//   Isletim sistemi logosunu çizer
// Parametreler :
//   YOK
```

```
// Geri Dönüş Değeri:
//   YOK
//---------------------------------------------------------------------
void logoCiz()
{
	//-----------------------------------------------------------
	// işletim sistemi logosunu çizer
	API_Set_Color(1);

	API_Println("  zzzzzzzzzz");
	API_Println("        zzz eeee  u  u gggg mmmmm  aaa");
	API_Println("      zzz   eeee  u  u g  g m m m a  a     TURKCE ISLETIM
SISTEMI");
	API_Println("     zzz      e     u  u gggg m m m a  a           ver 0.1");
	API_Println("   zzz      eeee  uuuu    g m m m  aaaa");
	API_Println("
zzzzzzzzzzzzzzzzzzzzzzzzzgzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz");
	API_Println("                        g");
	API_Println("                     gggg");
	//-----------------------------------------------------------

}
//---------------------------------------------------------------------
// Fonksiyon Adı:
//   "version"
// Açıklama:
//   İşletim sistemi versiyon bilgisini yazar
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//---------------------------------------------------------------------
void version()
{
	//-----------------------------------------------------------
	// versiyon bilgilerini yaz
	logoCiz();
	API_Println("");
	API_Set_Color(14);
	API_Println("");
	API_Println("                              Kasim Sinan YILDIRIM        ");
	API_Println("                               Ege Universitesi         ");
	API_Println("                        Bilgisayar Muhendisligi Bolumu   ");
	API_Println("                                     2003              ");
	API_Set_Color(10);
	API_Println("                         ---------------------------    ");
	API_Println("                        | yildirim@bilmuh.ege.edu.tr |   ");
	API_Println("                         ---------------------------    ");
```

```
        API_Println("");
        API_Println("                                              ");
        //------------------------------------------------------------

}
//------------------------------------------------------------------
// Fonksiyon Adı:
//   "help"
// Açıklama:
//   Yardım komutlarini yazar
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//------------------------------------------------------------------
void help()
{
        //------------------------------------------------------------
        // yardim menüsü
        API_Set_Color(2);
        API_Println("Zeugma komutlari ------------------------------");
        API_Set_Color(3);
        API_Print("* yardim          ");
        API_Set_Color(5);
        API_Println("--> Komut yardimi");
        API_Set_Color(3);
        API_Print("* bilgi           ");
        API_Set_Color(5);
        API_Println("--> Isletim sistemi bilgisi");
        API_Set_Color(3);
        API_Print("* surec 1         ");
        API_Set_Color(5);
        API_Println("--> 1 numarali sureci cagirir.");
        API_Println("  =>(6 adet 10 satir 10 sutundan olusan matrisleri 150000 defa
carpar)");
        API_Set_Color(3);
        API_Print("* surec 2         ");
        API_Set_Color(5);
        API_Println("--> 2 numarali sureci cagirir.");
        API_Println("  =>(bos surectir.islem yapmaz)");
        API_Set_Color(3);
        API_Print("* surec 3         ");
        API_Set_Color(5);
        API_Println("--> 3 numarali sureci cagirir.");
        API_Println("  =>(kullanici girdisi bekler ve onu ekrana yazar)");

        API_Set_Color(3);
        API_Print("* surec 4         ");
```

```
        API_Set_Color(5);
        API_Println("--> 4 numarali sureci cagirir.");
        API_Println("  =>(0 ile bolme yazilim kesmesi cagirilir)");

        API_Set_Color(3);
        API_Print("* surec bilgisi    ");
        API_Set_Color(5);
        API_Println("--> Sistemdeki surec listelerine ait bilgi verir.");
        API_Set_Color(3);
        API_Print("* kill XXXXXXXX    ");
        API_Set_Color(5);
        API_Println("--> Sistemdeki ID'si (hex olarak) verilen sureci sonlandirir.");
        API_Set_Color(2);
        API_Println("------------------------------------------------");
        //-------------------------------------------------------------
}
//--------------------------------------------------------------------
// Fonksiyon Adı:
//   "komutIsle"
// Açıklama:
//   Kullanıcının girdiği komutu işler
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//--------------------------------------------------------------------
void komutIsle(char *komut)
{
        int i;
        unsigned long ID;
        unsigned char digit;

        //-------------------------------------------------------------
        // kullanıcının girdiği komutlari teker teker kontrol et
        if(strcmp(komut,"",strlen(komut)))
                return;
        if(strcmp(komut,"yardim",6))
                help();
        else if(strcmp(komut,"bilgi",5))
                version();
        else if(strcmp(komut,"surec 1",7))
                API_Exec((unsigned long)surec1);
        else if(strcmp(komut,"surec 2",7))
                API_Exec((unsigned long)surec2);
        else if(strcmp(komut,"surec 3",7))
                API_Exec((unsigned long)surec3);
        else if(strcmp(komut,"surec 4",7))
                API_Exec((unsigned long)surec4);
```

```
	else if(strcmp(komut,"surec bilgisi",13))
		API_Surec_Bilgisi();
	else if(strcmp(komut,"kill",4))
	{
		ID=0;
		if(strlen(komut)==4)
			return;
		for(i=0;i<strlen(komut)-5;i++)
		{
			digit=komut[i+5];

			if((digit>=0x30)&&(digit<=0x39))
				digit=digit-0x30;
			else
				digit=digit - 0x61 + 10 ;

			ID=(ID<<4)+digit;
		}
		API_Kill_Process(ID);
	}
	else
		API_Println("Bilinmeyen komut...");
	//-----------------------------------------------------------
}
//-------------------------------------------------------------------
// Fonksiyon Adı:
//   "komutBekle"
// Açıklama:
//   Kullanıcıdan komut bekler
// Parametreler :
//   YOK
// Geri Dönüş Değeri:
//   YOK
//-------------------------------------------------------------------
void komutBekle()
{
	char Komut_Tamponu[128];

	while(1)
	{
		API_Set_Color(3);
		API_Print("<Z-Shell>");
		//-----------------------------------------------------------
		// komutun girilmesini bekle
		API_Scanf(Komut_Tamponu);
		//-----------------------------------------------------------

		//-----------------------------------------------------------
```

```
                // komutu işle
                komutIsle(Komut_Tamponu);
                //------------------------------------------------------------
        }
}

//------------------------------------------------------------
// deneme süreçleri
void surec1()
{
        unsigned long matris1[10][10];
        unsigned long matris2[10][10];
        unsigned long matris3[10][10];
        unsigned long matris4[10][10];
        unsigned long matris5[10][10];
        unsigned long matris6[10][10];
        unsigned long sonuc[10][10];
        unsigned long toplam=0;
        unsigned long i,j,k;

        API_Set_Color(5);
        API_Print("<surec 1> Calismaya basliyorum...");
        for(k=0;k<1500;k++)
        {
                for(i=0;i<10;i++)
                        for(j=0;j<10;j++)
                        {
                                matris1[i][j]=i;
                                matris2[i][j]=i+100;
                                matris3[i][j]=i+200;
                                matris4[i][j]=i+300;
                                matris5[i][j]=i+400;
                                matris6[i][j]=i+500;
                        }

                for(i=0;i<10;i++)
                        for(j=0;j<10;j++)
                                sonuc[i][j]=matris1[i][j]*matris2[j][i];

                for(i=0;i<10;i++)
                        for(j=0;j<10;j++)
                                sonuc[i][j]=matris3[i][j]*sonuc[i][j];

                for(i=0;i<10;i++)
                        for(j=0;j<10;j++)
                                sonuc[i][j]=matris4[i][j]*sonuc[i][j];

                for(i=0;i<10;i++)
```

```
				for(j=0;j<10;j++)
					sonuc[i][j]=matris5[i][j]*sonuc[i][j];

			for(i=0;i<10;i++)
				for(j=0;j<10;j++)
					sonuc[i][j]=matris6[i][j]*sonuc[i][j];

			for(i=0;i<10;i++)
				for(j=0;j<10;j++)
				toplam+=sonuc[i][j];
		}

		API_Set_Color(5);
		API_Print("<surec 1> Matris islemleri bitti ---> toplam : ");
		API_Print_Sayi_Hex(toplam);
		API_Exit();
}
//---------------------------------------------------------------
// boş süreç
void surec2()
{
		int i;

		while(1);
}
//---------------------------------------------------------------
//kullanıcıdan tuş alıp ekrana basan süreç
void surec3()
{
		char deneme[100];

		API_Scanf(deneme);
		API_Set_Color(5);
		API_Print("<surec 3> girdiginiz string--> ");
		API_Print(deneme);

		while(1);
}
//---------------------------------------------------------------
void surec4()
{
		int i;

		i=0/0;
		while(1);
}
//---------------------------------------------------------------
```

KAYNAKÇA

80386 MICROPROCESSOR HANDBOOK
Chris H. Pappas and William H. Murray III
Osborne McGraw-Hill

PROGRAMMING THE 80286,80386,80486 AND PENTIUM-BASED PERSONAL COMPUTER
Barry B. Brey
Prentice Hall

THE INTEL MICROPROCESSORS - 8086/8088, 80186/80188, 80286, 80386, 80486, PENTIUM AND PENTIUM PRO PROCESSOR – ARCHITECTURE, PROGRAMMING AND INTERFACING
Barry B. Brey
Prentice Hall

THE PENTIUM MICROPROCESSOR
James L. Antonakos
Prentice Hall

PENTIUM PRO FAMILY DEVELOPER'S MANUAL VOLUME 3: OPERATING SYSTEM WRITER'S GUIDE
intel

OPERATING SYSTEM CONCEPTS
Abraham Silberschatz - Peter Baer Galvin
Addison Wesley

OPERATING SYSTEMS: DESIGN AND IMPLEMENTATION
Andrew S. Tanenbaum – Albert S. Woodhull
Prentice Hall

MMURTL V1.0
Richard A. Burgess
Sensory Publishing

UNDERSTANDING THE LINUX KERNEL FROM I/O PORTS TO PROCESS MANAGEMENT
Daniel P. Bovet and Marco Cesati
O'Reilly