Kablosuz Algılayıcı Ağları İçin TinyOS İle Uygulama Geliştirme

Kasım Sinan YILDIRIM[1], Aylin KANTARCI[2]

[1] Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, İzmir

[2] Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, İzmir

sinan.yildirim@ege.edu.tr,
aylin.kantarci@ege.edu.tr

Özet: Kablosuz iletişim yeteneğine ve kısıtlı kaynaklara sahip düğümleri içeren kablosuz algılayıcı ağları, günümüzde önemli bir uygulama alanıdır. Kablosuz algılayıcı ağları için uygulama geliştirme, düğümlerin kaynak kısıtları nedeniyle geleneksel yöntemlerden farklı bir şekilde yapılmalıdır. Bu bildiride kablosuz algılayıcı ağları için uygulama geliştirmeyi kolaylaştıran TinyOS işletim sistemi hakkında bilgi verilmektedir.

Anahtar Sözcükler: TinyOS, nesC, Kablosuz Algılayıcı Ağları

Application Development With TinyOS for Wireless Sensor Networks

Abstract: Wireless sensor networks is an important application domain which consists of nodes with resource constraints that have ability to communicate through wireless medium. Developing applications for wireless sensor networks must be done in a different way from classical techniques due to the resource constraints of sensor nodes. In this paper, we give information about TinyOS operating system which is designed to ease application development for sensor networks.

Keywords: TinyOS, nesC, Wireless Sensor Networks

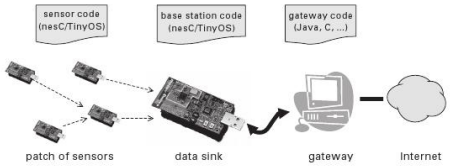
1. Giriş

Kablosuz algılayıcı ağları, kablosuz iletişim yeteneğine sahip ve kısıtlı kaynaklara sahip düğümlerden oluşmaktadırlar. Düğümler ortamı algılama özellikleri sayesinde orman yangını gibi çevresel felaketleri tespit eden, erişilemeyen coğrafi noktalardan veri toplamayan ya da askeri amaçlarla çevrenin kontrol altında tutulmasını sağlayan uygulamalarda kullanılabilirler.

Şekil 1. Genetlab [4] *Sensenode* düğümü

Şekil 1'de Genetlab firmasının *Sensenode* isimli düğümü görülmektedir. 10 KB ana belleğe sahip olan bu düğüm, 16 bit RISC mimarisine sahip Msp430 [5] mikrodenetleyicisine sahiptir. Görüldüğü gibi algılayıcı düğümleri kısıtlı sistem kaynaklarına ve işleme kapasitesine sahiptir.

Düğümlerin kaynak kısıtlarından dolayı, algılayıcı ağlarına yönelik uygulama geliştirilmesi, geleneksel sistemlerden değişiklikler göstermektedir. TinyOS [6], algılayıcı düğümlerinin kaynak kısıtları gözetilerek verimli bir şekilde programlanmasını amaçlar. TinyOS C diline benzer bir dil olan nesC [7] ile yazılmıştır ve uygulamaların da bu dille geliştirilmesine olanak kılar.



Şekil 2. Algılayıcı ağının programlanması [1]

Bir algılayıcı ağındaki düğümler ve merkezi düğüm, Şekil 2'de görüldüğü gibi TinyOS işletim sistemi ve nesC dili kullanılarak programlanırlar. Ağ ile bilgisayarlar arasındaki bağlantı ise daha yüksek seviyeli bir dil olan Java ile de yapılabilir. Ağ sisteminin kaynak kısıtına sahip olmayan bölümleri için TinyOS kullanmaya gerek yoktur.

Bu bildiride, algılayıcı düğümlerinin programlanması için kullanılan TinyOS işletim sistemi ve nesC dili hakkında bilgi verilmekte ve en temel programlama mantığı anlatılmaktadır.

2. TinyOS İşletim Sistemi

TinyOS kablosuz iletişim yapan gömülü sistemler için düşük güç kullanarak dışsal olayları yöneten uygulamaları daha kolay geliştirmeyi amaçlayan bir işletim sistemidir. TinyOS birçok *bileşen* içermektedir ve bu bileşenler uygulamaların ihtiyacına göre eklenip çıkarılabilmektedirler. TinyOS bu özelliği sayesinde gömülü sistemler için geliştirilmiş diğer işletim sistemlerinden ayrılmaktadır: Uygulamalar işletim sisteminin sunduğu sabit servislere göre şekilleneceğine, işletim sistemi uygulamanın ihtiyaçlarına göre eklenen ya da çıkarılan bileşenler ile yapılandırılır.

TinyOS işletim sistemi yeniden kullanılabilir birçok bileşen ve algılayıcı ağları uygulamaları için iyi tasarlanmış birçok programlama arayüzü içermektedir. Kablosuz algılayıcı ağları düşük kaynaklara (örneğin sadece 4KB ana bellek) ve çok kısıtlı enerji bütçelerine sahip sistemler oldukları için, uygulama geliştiriciler uygulamanın ihtiyaçlarına göre aynı servisin birçok sürümünü yazmak zorundadırlar. Bu yöntem geliştirilen uygulamanın sistem kaynakları en verimli şekilde kullanması sonucunu doğururken uygulamanın ihtiyaçlarını karşılayan servislerin yeniden kullanılabilir olmasını engellemektedir. TinyOS'un bileşen tabanlı mimarisi kablosuz duyarga ağlarının kaynak kısıtları göz önüne alınarak geliştirilmiştir.

TinyOS üzerinde uygulama geliştirirken kullanılan dil, TinyOS'un da geliştirildiği nesC dilidir. Bileşen tabanlı bir dil olan nesC olay tabanlı bir işletim mekanizmasına sahip olan kablosuz algılayıcı ağları için uygun bir programlama dilidir. *nesC* dilindeki bileşenler nesneye dayalı programlama paradigmasındaki nesnelere benzemektedirler: Bileşenler bir durum bilgisi ve o durum bilgisini işleyen işlevselliği barındırmaktadırlar. Bileşenler birbirleri ile arayüzler aracılığıyla etkileşim kurmaktadırlar. Kablosuz algılayıcı ağlarının kısıtlı bellek kaynağına sahip olmalarından ötürü, nesneye yönelik programlama dillerinin sunduğu dinamikliğin aksine, bileşenlerin sayısı ve bileşenler arası etkileşim uygulamanın derleme anında belirlidir.

TinyOS'un sunduğu özelliklerden belkide en önemlisi süreçler arası geçiş mekanizmasının bu işletim sisteminde ortadan kaldırılmış olmasıdır. Geleneksel işletim sistemlerinde süreçlerin her biri ayrı adres sahalarına ve ayrı bir çalışma içeriğine sahiptirler. İşletim sistemi, işlemci yönetimi kapsamında bir süreçten diğerine geçiş yaparak birçok sürecin aynı anda çalışmasını sağlamaktadır. Bu geçiş mekanizmasında işletim sistemi o an çalışmakta olan sürecin durumunu saklar. Ek olarak, hafif süreçler olarak nitelendirilebilen iş parçacıkları da aynı adres sahasında çalışmalarına rağmen kendi durumlarının işletim sistemi tarafından saklanabilmesi için ayrı yığıt çerçevelerine gereksinim duyarlar. Bir işparçacığından diğerine geçiş, o an çalışmakta olan işparçacığının durumunun kendi yığıt çerçevesinde saklanmasını gerektirir. İşletimi soyutlayan süreç ve işparçacığı kavramları, işletim sistemine ek yük getirmekte ve geçiş mekanizması kısıtlı kaynaklara sahip kablosuz algılayıcı düğümleri için uygun görünmemektedir. TinyOS, tek bir yığıt çerçevesi barındırmakta sistem içerisindeki tüm işletim bu yığıt çerçevesi üzerinden gerçekleştirilmektedir. Sistemdeki süreç kavramı aslında basit bir fonksiyondan çağırımından farklı birşey

değildir. Fonksiyonlar sonlanıncaya kadar bölünmeden çalıştırılırlar. Bu işletim, süreçler arası geçiş mekanizmasını ortadan kaldırmıştır.

3. Temel TinyOS Programlama

TinyOS bileşenleri sundukları ya da kullandıkları arayüzler üzerinden birbirlerine bağlanmaktadır. **Modül** olarak isimlendirilen bileşenler bir gerçekleştirim barındırırlar. Modüller sundukları arayüzlerin C dili ile gerçekleştirimlerini yine kullandıkları arayüzler ile sağlamaktadırlar. **Yapılandırıcı** olarak isimlendirilen bileşenler ise diğer bileşenleri bağlayan bileşenlerdir. **Arayüzler** bileşenlerin işlevselliğini belirlerler. TinyOS'ta tüm bileşenler ve arayüzlerin isimleri ile bunların gerçekleştirim dosyalarının isimleri aynı olmalıdır.

TinyOS'ta bileşenler arasındaki etkileşim çift yönlüdür. Bir bileşen kullandığı diğer bileşenin komutlarını çağırabilirken, kullanılan bileşen de olayları sinyalleyerek diğer bileşeni gerçekleşen olaylardan haberdar edebilir.

Şekil 3'de *PowerupC* modülü listelenmiştir. Bu bileşen Şekil 4'te listelenen *Boot* ve *Leds* arayüzlerini kullanmaktadır. Boot arayüzü sistem açıldığı anda *PowerupC* modülünün bu açılıştan haberdar olmasını sağlayan "booted" olayını içermektedir. Yani *PowerupC* modülü, sistem açılışında sinyallenecektir. Görüldüğü gibi modül, başka bir modül tarafından sinyallenerek aşağıdan yukarıya bir etkileşim sağlanmaktadır ve bu etkileşim "**event**" anahtar kelimesi ile belirtilen olaylar sayesinde olmaktadır. Leds arayüzü ise sistemdeki ledlerin yakılıp söndürülmesi işlevini yerine getirecek modüller tarafından gerçekleştirilecektir. Modülün "**implementation**" kısmındaki C kodu gerçekleştirimlerinde *PowerupC* modülü, bu arayüzün ledleri yakan komutunu çağırmaktadır. Yani yukarıdan aşağıya bir etkileşim söz konusudur. Komutlar "**command**" anahtar kelimesi ile belirtilirler ve çağırımları "**call**" anahtar kelimesi ile yapılmaktadır.

```
module PowerupC {
  uses interface Boot ;
  uses interface Leds ;
}
implementation {
  event void Boot.booted () {
    call Leds.led0On();
  }
}
```

Şekil 3. PowerupC modül bileşeni

```
interface Boot {
  event void booted ();
}

interface Leds {
  command void led0On();
  command void led0Off();
  command void led0Toggle();
  ...
}
```

Şekil 4. Boot ve Leds arayüzleri

PowerupC modülü hangi arayüzleri kullanacağını kendi gerçekleştiriminde belirtmişti. Ancak bu modülün belirttiği arayüzleri kullanabilmesi için, bu arayüzleri sağlayan modüllere bağlanması gerekmektedir. Modüllerin birbirleri ile etkileşime geçebilmesini sağlayan bağlama işlemi "**configuration**" anahtar kelimesi ile tanımlanan yapılandırma bileşenlerince gerçekleştirilirler. Şekil 5'te belirtilen *PoweupAppC* yapılandırıcı modülü, *PowerupC* bileşenin kullandığı arayüzleri sunan *MainC* ve *LedsC* isimli sistem bileşenlerini "->" operatörü ile *PowerupC* bileşenlerine bağlamaktadır. *MainC* ve *LedsC* yapılandırma bileşenleri Şekil 6'da listelenmiştir. Bu bileşenler "**provides**" anahtar kelimesi ile sundukları arayüzleri belirtmişlerdir. Örneğin *LedsC* bileşeni "**provides interface** Leds" ile *Leds* arayüzünün komutlarını gerçekleştirdiğini belirtmektedir. "PowerupC.Leds -> LedsC.Leds" kod parçacığı ile *PowerupC* bileşeni ile *LedsC* bileşeni birbirine bağlanmaktadır. Böylelikle *PowerupC* bileşeni, *LedsC* bileşeninin *Leds* arayüzü üzerinden sunduğu komutları

çağırabilmektedir. "MainC.Boot -> PowerupC.Boot" kod parçacığı ile sistemdeki *MainC* bileşenin açılışta çağıracağı "booted" olayının *PowerupC* bileşeni ilişkilendirilmesi sağlanır.

```
configuration PowerupAppC {

  implementation {
    components MainC , LedsC , PowerupC ;

    MainC.Boot -> PowerupC.Boot ;
    PowerupC.Leds -> LedsC.Leds ;
  }
}
```

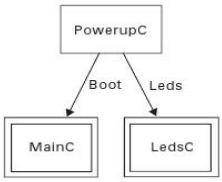
Şekil 5. PowerupAppC yapılandırıcı bileşeni

```
configuration LedsC {
  provides interface Leds;
}
implementation {
  …
}

configuration MainC {
  provides interface Boot;
  ...
}
implementation {
  ...
}
```

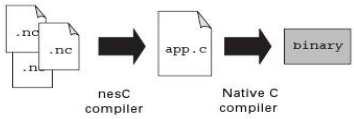
Şekil 6. LedsC ve Main C yapılandırıcı bileşenleri

Tüm bu gerçekleştirim ve bağlama işlemlerinden sonra Şekil 7'deki bileşen şeması elde edilir. Bu şemada çift dikdörtgenlerle gösterilmiş bileşenler TinyOS tarafından sağlanan sistem bileşenleridir. *Powerup* uygulaması, bu bileşenleri kullanarak sistem açılışında ledlerin yanmasını sağlamaktadır.



Şekil 7. Powerup uygulaması için bağlama şeması [1]

"PowerupC.nc" ve "PowerupAppC.nc" dosyalarından oluşan uygulama nesC derleyicisi tarafından derlendiğinde "app.c" isimli C dili dosyası oluşmaktadır. Bu dosya, daha sonra uygulama kodunun çalıştırılacağı platform için hedef ikili kod üreten bir C derleyecisi ile derlendikten sonra kablosuz algılayıcı düğümlerine yüklenebilir.



Şekil 8. Uygulama derleme aşamaları [1]

4. Tartışma

TinyOS işletim sistemi, tasarım amaçları açısından geleneksel işletim sistemlerinden ayrılmaktadır. Kullandığı nesC dili, TinyOS ile uygulama geliştirmeyi nesneye yönelik ortamlara benzer bir rahatlıkta yapmayı olanaklı kılar. Ancak TinyOS gereksiz bellek kullanımından ve nesneye yönelik ortamların getirdiği ek katmanların yarattığı verimsizlikten uygulamaları kurtarmayı hedeflemektedir. Bu nedenle TinyOS durağan bir bellek tahsisi mekanizmasını kullanır. Bileşen tabanlı mimarisi sayesinde daha derleme anında uygulamanın bellek gereksinimi belirlidir ve çalışma zamanında dinamik olarak değişmez. Nesneye yönelik bir programlama dili olan C++'taki gibi sanal fonksiyonların getirdiği ek yükler de nesC sayesinde TinyOS'ta yer almaz. TinyOS'ta herşey derleme anında belirlenir ve çalışma süresince sabittir.

REFERANSLAR

[1] Philip Levis, David Gay, TinyOS Programming, ISBN 0521896061

[2] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister, "System architecture directions for networked sensors", ACM SIGPLAN Notices, v.35 n.11, p.93-104, Nov. 2000

[3] David Gay, Phil Levis, David Culler, "Software design patterns for TinyOS", Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on

Languages, compilers, and tools for embedded systems, June 15-17, 2005

[4] Genetlab: http://www.genetlab.com/

[5] Texas Instruments: http://www.ti.com/

[6] TinyOS : http://www.tinyos.net/

[7] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, David Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems", In Proceedings of Programming Language Design and Implementation (PLDI), 2003