KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

GERÇEK ZAMANLI ÇOKLU GÖREV İŞLETİM SİSTEMİ TASARIMI

YÜKSEK LİSANS

Bilgisayar Müh. Tayfun Mehmet KARAN

Anabilim Dalı: Bilgisayar Mühendisliği

Danışman: Doç.Dr. Yaşar BECERİKLİ

KOCAELİ, 2008

KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

GERÇEK ZAMANLI ÇOKLU GÖREV İŞLETİM SİSTEMİ TASARIMI

YÜKSEK LİSANS TEZİ

Bilgisayar Müh. Tayfun Mehmet KARAN

Tezin Enstitüye Verildiği Tarih: 21 Mayıs 2008

Tezin Savunulduğu Tarih: 10 Temmuz 2008

Tez Danışmanı
Doç. Dr. Yaşar BECERİKLİ

Üye
Prof.Dr. A.Coşkun SÖNMEZ

Üye
Yrd.Doç.Dr. H.Engin DEMİRAY

KOCAELİ, 2008

ÖNSÖZ VE TEŞEKKÜR

Günümüzde hayatımızın vazgeçilmez birer parçası haline gelen gömülü sistemler, ucuzlayan fiyatları ve küçülen boyutları ile son döneme damgasını vurmuş yeni nesil bilgisayar sistemleri olarak anılırlar. Bu sistemlerin boyutlarındaki küçülme ve gelişen yetenekleri ile doğru orantılı olarak yazılım mimarilerinde de büyük gelişmeler yaşanmıştır. Bu tür sistemler geleneksel bilgisayar mimarilerinden farklılıklar gösterdiğinden, yazılım geliştirme süreçleri daha kompleks ve uzmanlık gerektirmektedir. Günümüzde çoğunlukla gerçek zamanlı sistemler olarak kullanılan bu gömülü sistemlerin yazılım yaklaşımları da gerçek zamanlı olmak zorundadır. Gerçek zamanlı yazılım yaklaşımları, uygulama yazılımları ve sistem yazılımları olmak üzere iki grupta incelenir. Ülkemizde gerçek zamanlı uygulama yazılımlarının yaygın bir şekilde gerçekleştirilmelerinin yanı sıra gerçek zamanlı sistem yazılımları neredeyse hiç geliştirilmemektedir.

Gerçek zamanlı sistemler için bir işletim sistemi oluşturma konusunda bana çalışma fırsatı veren BKS Ltd. Şti. yöneticilerine, son dönemde ailelerine katılma şansına sahip olduğum iDeal Teknoloji A.Ş. yöneticileri ve ekip arkadaşlarıma, proje ve tez aşamasında fikirleri ile beni yönlendiren ve her türlü desteğini benden esirgemeyen, uzun bir süredir birlikte çalışma fırsatına sahip olduğum değerli hocam Sn. Doç.Dr.Yaşar BECERİKLİ‘ ye teşekkür ederim. Ayrıca uykusuz gecelerimde bana göstermiş olduğu büyük destek ve özverinin karşılığını hiçbir zaman ödeyemeyeceğim sevgili eşim Sevil KARAN’ a sonsuz minnet duygularımı sunarım.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SEMBOLLER

Kısaltmalar

RTOS : Real Time Operating System
ANSI : American National Standards Institute
ROM : Read Only Memory
RAM : Random Access Memory
SRAM : Static Random Access Memory
CPU : Central Processing Unit
EEPROM : Electrically Erasable Programmable Read Only Memory
UART : Universal Asynchronous Receiver Transmitter
SPI : Serial Peripheral Interface
I2C : Inter-Integrated Circuit
ADC : Analog to Digital Converter
DAC : Digital to Analog Converter
PC : Personal computer
GPS : Global Positioning System
MMU : Memory Management Unit
PWM : Pulse Width Modulation
DMA : Direct Memory Access
DMAC : Direct Memory Access Controller
MII : Media Independent Interface
MAC : Media Access Controller
TCP : Transmission Control Protocol
IP : Internet Protocol
FAT : File Allocation Table
MMC : Multi Media Card
SD : Secure Digital
OPAMP : Operational Amplifier
HTML : HyperText Markup Language
CGI : Common Gateway Interface

GERÇEK ZAMANLI ÇOKLU GÖREV İŞLETİM SİSTEMİ TASARIMI

Tayfun Mehmet KARAN

Anahtar Kelimeler: Gömülü sistemler, işletim sistemleri, gerçek zaman.

Özet: Günlük yaşamda her alanda karşımıza çıkan hayatımızın vazgeçilmez bir parçası haline gelen gömülü sistemlerin, programlanabilir yapılar içeren bir çeşit küçültülmüş bilgisayar sistemleri olduğu çok fazla bilinmemektedir. Bu gerçek, beraberinde gömülü sistemler için oluşturulması gereken yazılım olgusunu karşımıza çıkartır. Gömülü sistemler için yazılım geliştirme, uzmanlık isteyen zorlu bir süreçtir. Geliştiriciler yazılım kavramının yanında donanım bileşenleri ve bu bileşenlerin nasıl kullanılacaklarını da bilmek zorundadırlar. Farklı donanım bileşenlerine sahip her bir gömülü sistem için, ayrı bir uzmanlığı zorunlu kılan bu engel, geçmişte bilgisayarların ilk ortaya çıktığı yıllarda da karşımıza çıkmıştır. Bu problem, programlanabilir yapılar için işletim sistemlerinin geliştirilmesine sebep olmuştur. Gömülü sistemler için oluşturulan işletim sistemleri, geleneksel işletim sistemlerinden farklılık göstererek temelde gerçek zamanlı sistemlerde kullanılabilmeleri amacını taşır. Ülkemizde gerçek zamanlı gömülü sistemler için işletim sistemi tasarımının çok fazla ele alınmamasından ve bu alandaki gerekliliğin göz ardı edilmesinden yola çıkarak bu tez kapsamında, kısıtlı donanım kaynaklarına sahip gömülü sistemler için genel amaçlı olarak kullanılabilecek, çoklu görev yeteneğine sahip, taşınabilir, ölçeklenebilir, esnek ve standart yapıları destekleyen bir işletim sistemi oluşturulması hedeflenmiştir. Çalışma süresince geçmişte gerçekleştirilmiş olan araştırma ve projelerin sağladığı bilgi ve tecrübelerden faydalanılarak, gerçek bir uygulama örneği oluşturulmuştur. Tez boyunca yapılan tüm araştırma ve incelemeler doğrultusunda gerçekleştirilen bu uygulama ile, gerçek zamanlı bir işletim sisteminin ne denli gerekli ve kullanışlı olduğu vurgulanmıştır. Amaçlandığı gibi gömülü sistem uygulamalarında, ürün geliştirme süreci kısaltılarak, daha güvenli ve verimli uygulamaların ortaya çıkartılması sağlanmıştır.

REAL TIME MULTITASKING OPERATING SYSTEM

DESIGN

Tayfun Mehmet KARAN

Keywords: Embedded systems, operating systems, real time.

Abstract: Usually, it is not known that embedded systems which we confront all around in daily living are some kind of small computer systems. This truth brings us the software fact for embedded systems. Software development of embedded systems is hard and expertise requiring phase. Developers are needed to know how to use the hardware components besides this software concept. Including different types of hardware components, every embedded system requires a different specialism and this problem is being seen since first computers are come into existence in the history. This problem caused the development of operating systems for programmable structures. Operating systems which are developed for embedded systems are intent to be used on real-time systems. In this thesis, multi-task capable, portable, scalable, flexible and standard structure supporting operating system development is aimed in order to compensate absence of operating system design for real-time embedded system in our country. During this study, a real application sample is built by using past study and project's information and knowledge. In parallel with all researches and observations during thesis, this application highlights the necessity and usability of real-time operating systems. In embedded system applications, development of more secure and effective applications are provided by shortening product development phase.

1. GİRİŞ

Basit bir tanım ile "gömülü sistemler", günümüzde yaşamımızın vazgeçilmez birer parçası haline gelmiş bilgisayar tabanlı elektronik aygıtlardır (Cheng, 2002). Bununla beraber uzun yıllardır bizlerin arasında olan fakat bir türlü bizler tarafından fark edilemeyen bu olgu çağımızdaki yenilikler ve ilerlemeler ile zirve noktasına ulaşmıştır.

Peki nedir bu gömülü sistemler? Günlük hayatta nerelerde karşımıza çıkarlar? Şöyle bir çevremize baktığımızda, aslında onların bulunmadığı tek bir sahneyle bile karşılaşmamız oldukça güçtür. Sürekli olarak yanımızda taşıdığımız cep telefonumuz, kıyafetlerimizin temizliğinden sorumlu çamaşır makinemiz, ev hanımların mutfaktaki müthiş yardımcıları olan bulaşık makinesi, mutfak robotu, fırın, evimizde keyifle karşında iyi vakit geçirmemizi sağlayan televizyonumuz, işte bunların hepsi birer gömülü sistemdir.

Günümüz teknolojisinin geldiği noktadan dolayı günlük hayatta kullandığımız her aygıtın yetenekleri artarken boyutları da küçülmektedir. Tabi bunun yanında yapıları da karmaşıklaşmaktadır. Bu kaçınılmaz bir durumdur. Her geçen gün karmaşıklaşan bu gelişmiş gömülü sistemlerden bahsederken iki olgu karşımıza çıkar.

- Sistem donanımı
- Sistemin işlevini gerçekleştirebilmesi için gerekli olan yazılımı

Şüphesiz ki elektronik temelli her aygıt, belli bir görevi yerine getirebilmesi için uygun donanım bileşenlerine ihtiyaç duyar. Örneğin cep telefonuna bir bilgi girebilmek için tuş takımına ihtiyacımız vardır. Yapılan işlemlerin sonuçlarını görmek için de bir ekrana.

Gömülü sistem denildiğinde akla gelen ilk donanım bileşeni mikro denetleyicilerdir (ARM Hodings, 2007), (Renesas Technology Corp., 2004). Gelişen teknoloji sayesinde özellikleri artarken, boyutları ve fiyatları hızla düşen bu mucizevi aletler hayatımızın ayrılmaz bir parçası haline gelmişlerdir.

Bununla beraber mikro denetleyiciler programlanabilir aygıtlar olduğundan, ortaya yazılım olgusunun eksikliği çıkar. Tezimizin ana konusu bu yazılım olgusuyla bire bir ilişkilidir.

Bu tez kapsamında "albatROS" adını verdiğimiz, 2004 yılının ortalarında temelleri atılmış, son iki yıldır üzerinde yoğun bir şekilde çalışılarak eksikleri tamamlanmış, gömülü sistemler için bir "gerçek zamanlı işletim sistemi" oluşturulmuştur. Bu çalışmada ortaya çıkan platform, sürekli denizlerde yaşayan, en uzun ömürlü kuş türüyle aynı adı paylaşır. Bu kuş türleri gibi oluşturulan işletim sisteminin de, zor ve ağır koşullarda yaşamını ve varlığını çok uzun süreler koruyabilmesi hedeflenmiştir.

Çıkış noktası, kısıtlı donanım kaynaklarına sahip gömülü sistemler için kendi öz kaynaklarımız ve mühendislik gücümüz ile bir "işletim sistemi ortaya çıkarmak" olan bu proje, günümüzde ulaştığı yetenek ve özellikler sayesinde, akademik çalışmaların yanı sıra ticari birçok uygulamada da başarıyla uygulanmıştır.

Başarısının ardında, projede çalışan ekibin uzun yıllar gömülü sistemlerin donanım ve yazılım tasarımlarında sahip oldukları tecrübelerini doğru bir şekilde bu projede kullanabilmeleri yatmaktadır. Çünkü bu proje saf bir yazılım tasarım örneği değildir. Donanım ile dinamik bir şekilde etkileşimde bulunmayı gerektirir. Her gömülü sistemde, donanım ve yazılım ayrılmaz bir bütündür. Tez kapsamında incelenen ve açıklanan konular belli bölümlere ayrılmıştır.

Bölüm 1, tez çalışmasına giriş niteliğinde olup, çalışmanın amacını, hedeflerini ve olabilirliğini belirtmektedir.

Bölüm 2'de gömülü sistemlere genel bir bakış yapılarak günlük hayatta nerelerde kullanıldığına değinilmiştir.

Bölüm 3'de gerçek zamanlı işletim sistemlerine giriş yapılarak, işletim sistemlerinin tarihçesi, tanımı, özellikleri, tasarım metotları ve bileşenleri hakkında bilgi verilmiştir. Bu bölüm tasarım aşamasında işletim sistemi için bir yol haritası teşkil eder.

Bölüm 4'de işletim sistemi nesnelerinden görevler ayrıntılı bir şekilde incelenmiştir. Görevlerin tipleri, amaçları ve özellikleri bu bölümde anlatılmıştır.

Bölüm 5, işletim sistemleri için diğer önemli bir nesne olan semaforlara ayrıntılı bir bakış sunmaktadır. Semaforların özellikleri, nasıl ve niçin kullanıldıkları bu bölümün konusudur.

Bölüm 6'da kablosuz sensör ağlarına giriş yapılarak, nerelerde kullanıldığı ve ne tür ağ topolojilerini desteklediklerinden bahsedilmiştir. Tez kapsamında ana konu, sensör ağlarında yönlendirme protokolleri olmadığı için bu konuya değinilmemiştir. Bu bölümde sensör ağlarında kullanılan düğümlerde , işletim sistemlerinin nasıl kullanılabileceği vurgulanmıştır.

Bölüm 7, tez kapsamında gerçekleştirilen uygulamanın ayrıntılarını açıklayarak, geliştirme aşamaları, kullanılan ekipman ve yöntem, elde edilen başarı ile ilgili bir değerlendirme sunmaktadır. Bu bölümde gerçekleştirilen yazılım modülleri parçalar halinde ele alınarak incelenmiştir.

Tez kapsamında, gerçek zamanlı gömülü sistemler için, çoklu görev yeteneği olan standart bir işletim sistemi tasarlanmak istenmektedir (Ready, 1986), (Stankovic, 1987), (Barello, 2005). Bu doğrultuda literatürde geniş çaplı bir tarama işleminden sonra, ülkemizde bu konunun yeteri kadar önemsenmediğinin farkına varılması, çalışmayı destekleyici önemli bir unsurdur. Gerçek zamanlı sistemler için bir işletim sisteminin ihtiyacı ve önemi doğrudan hedeflenen uygulamanın türüne bağlıdır. Basit mantıksal işlemlerin yapıldığı giriş çıkış denetleyicilerinde, bir işletim sistemi kullanmak gereksiz ve maliyetli olmakla birlikte, karmaşık işlemler dizisine sahip, ağ katmanı desteği gereken, dosya giriş çıkış işlemlerinin zorunlu olduğu sistemlerde ise işletim sistemi kullanımı, tasarım süreci açısından büyük zaman kazandırır.

Bunun yanında iyi tasarlanmış bir işletim sistemi kullanılarak gerçekleştirilen uygulamalarda hata riski daha az olur.

Bu amaç için tez kapsamında, bir çalışma platformu seçilerek, Olimex Ltd. (2007), bu platform üzerinde, çoklu görev, dosya sistemi ve ağ protokolleri desteği olan bir işletim sistemi hazırlanarak, örnek teşkil edebilecek bir uygulamanın bu işletim sistemi üzerinde koşturulması düşünülmektedir.

2. GÖMÜLÜ SİSTEMLER

Genel olarak önceden belirlenmiş işlemleri yerine getirmesi için tasarlanmış, belli bir takım donanım ve yazılım bileşenlerine sahip hesaplayıcı ve yorumlayıcı üniteler olarak tanımlanabilir. Gömülü kelimesi bu sistemlerin çok daha büyük sistemlerin küçük bir alt parçası olduğu gerçeğini yansıtır.

Günümüzde gömülü sistemler artık evlerimize kadar girmiş bulunuyor. Evimizdeki televizyon, çamaşır makinesi, bulaşık makinesi, fırın, buzdolabı, ısıtma-soğutma sistemi, günlük hayatta kullandığımız cep telefonu, cep bilgisayarı ve bunun gibi bir çok elektronik temelli programlanabilme özelliğine sahip cihazlar gömülü sistemler olarak adlandırılır. Gömülü sistemler, çevre birimler ile etkileşimde bulunan giriş çıkış ünitelerine sahip, bunun yanında çeşitli mantıksal ve matematiksel işlemleri yapabilme yeteneğinde, programlanabilen yapılar olarak da düşünülebilir. Gelişen teknoloji bu gömülü sistemlerin hızla ucuzlayıp, yaşamımızın vazgeçilmez bir parçası haline gelmesine neden olmuştur.

Bununla birlikte gömülü sistemler, donanım ve yazılım tasarımı açısından uzmanlık gerektiren yapılardır. Donanım tasarımları gerçekleştikten sonra bu donanım için uygun yazılım yine tasarımcının sorumluluğundadır. Bu durum bu teknolojilerin herkes tarafından kullanılmalarının önünü tıkayan bir engel olarak karşımıza çıkar. Kişisel bilgisayarların gelişimlerinin en önemli etkenlerinden biri standart yazılım mimarilerinin tüm bu donanımlar üzerinde çalışabilme yeteneklerindendir. Böylece her donanım üreticisi standartlar doğrultusunda istedikleri ürünleri yazılım kısmını düşünmeden geliştirebilmektedir.

Yazılım mimarisinde iki önemli nokta karşımıza çıkar. Sistem yazılımları ve uygulama programları. Bu ikili çoğunlukla ayrılmaz bir bütündür. İşletim sistemine ihtiyaç duyulmayan basit işlemlerin gerçekleştirildiği gömülü sistemlerde, uygulama yazılımı aynı zamanda sistem servislerini (donanım erişimleri gibi) de içerir.

Bu tür uygulamalarda işletim sistemi görünürde olmasa bile, uygulama programı gerçekte işletim sistemini de kapsar.

Günümüzde kişisel bilgisayarlarda kullanılan Windows, Unix, Linux gibi işletim sistemleri son kullanıcıların donanım bileşenleri ile uğraşmadan doğrudan uygulama programlarını geliştirmelerine olanak sağlamıştır.

Bu işletim sistemleri temel olarak kişisel masaüstü bilgisayarlar için geliştirildiklerinden gömülü sitemlerde kullanılma olanakları yoktur. Fakat bunların küçültülmüş alt versiyonları geliştirilerek gömülü sistemler için kullanılabilir yapıya getirilmişlerdir. Böylece kullanıcılar donanım ile uğraşmadan uygulama programlarını geliştirme imkanına sahip olmuşlardır. Fakat bu işletim sistemleri de bazı özel donanımlar için tasarlanmıştır. Bunun anlamı tasarladığımız her donanım için bu işletim sistemlerini kullanma olasılığımızın olmaması veya çok uzun ve zahmetli bir işlemin bizi beklediğidir.

Bu durum tez konusunun çıkış noktası olmuştur. Kendi tasarladığız donanımlar üzerinde yine kendi tasarladığımız işletim sistemlerinin çalışabilmesi, büyük bir güç ve avantaj kaynağıdır. İstenildiği anda sistemin hem donanım hem de sistem seviyesindeki yazılım modüllerine müdahale edilerek, sistemler ihtiyaca göre uyarlanabilirler.

Örneğin; bir gömülü sistem bazı durumlarda son kullanıcılar için bir havalandırma denetleyicisi olurken bazı durumlarda da bir füze kumanda sistemine dönüşebilir. Burada dikkat edilmesi gereken husus, her gömülü sistem için mutlaka bir yazılıma ihtiyaç vardır. Fakat bir işletim sistemi olma zorunluluğu yoktur.

İşletim sistemleri donanımın, son kullanıcılar tarafından kolay, zahmetsiz ve hızlı bir şekilde kullanılabilmelerine olanak sağlayan sistem yazılımlarıdır. İşletim sistemine sahip gömülü sistemlerde, kullanıcılar sadece uygulama ile ilgilenirler. Disk giriş/çıkış işlemleri, ağ protokolleri, haberleşme sistemleri gibi daha fazla uzmanlık ve bilgi birikimi isteyen alt katman işlevleri ile uğraşmak zorunda olmadıklarından, çok hızlı sonuca giderek hem zaman hem de maliyet tasarrufu sağlarlar.

Bu bakış açısı ile kendi donanımlarımız için uyarlanabilir bir işletim sistemine sahip olmamızın getireceği avantajlar göz önüne alınarak, ve daha önce bu konuyla ilgili ülkemizde çok fazla çalışma yapılmadığından güç alarak, çalışma konumuzun gömülü sistemler için bir işletim sistemi gerçekleştirmek olmasına karar verilmiştir.

3. GERÇEK ZAMANLI İŞLETİM SİSTEMLERİNE GİRİŞ

3.1. Giriş

Gerçek zamanlı bir işletim sistemi, günümüzde çoğu gömülü sistem için anahtar rol oynamakta ve uygulamaların inşa edilip çalıştırılabilmesi için bir yazılım platformu sunmaktadır (Li ve Yao, 2003), (Jerraya ve diğ., 2003). Tüm gömülü sistemlerde kullanılmasa da günümüzde artık bir çoğu, gerçek zamanlı işletim sistemi ile birlikte tasarlanmaktadırlar. Nispeten daha basit donanım bileşenlerine sahip bazı gömülü sistemler yada çok küçük uygulama kodu içeren gömülü sistemlerde, genellikle işletim sistemi kullanımı tercih edilmez. Bununla beraber bir çok gömülü sistem, özellikle de uygulama programı açısından çok daha karmaşık olanları ve bir takım düzenlemeleri gerektirenleri, bir gerçek zamanlı işletim sistemine ihtiyaç duyarlar (Joseph, 1996).

Aşağıda bu bölümde incelenecek olan konu başlıkları verilmiştir. Bu bölümlerde çoğu gerçek zamanlı işletim sisteminin dayandığı anahtar konular açıklanmaya çalışılmıştır.

- İşletim sistemlerinin kısa tarihçesi
- Bir gerçek zamanlı işletim sisteminin tanımı
- Görev planlayıcısının tanımı
- Nesneler
- Servisler
- Gerçek zamanlı işletim sisteminin anahtar karakteristikleri

3.2. İşletim Sistemlerinin Kısa Tarihçesi

Bilgisayar çağının ilk yıllarında, geliştiriciler sistem donanımına erişmek ve kullanmak için düşük seviyeli kod içeren uygulama programları oluşturuyorlardı. Bu yazılım ve donanım arasındaki dar etkileşim, taşınamaz uygulamaların ortaya çıkmasına neden oldu.

Donanımdaki küçük değişiklikler uygulamanın kendisinin büyük bir kısmını yeniden yazmayı gerektiriyordu. Açıkça bu sistemlerin gerçekleştirilmeleri oldukça zor ve varlıklarını sürdürmeleri çok maliyetliydi.

Yazılım endüstrisindeki ilerlemeler sonucunda, mikroişlemci tabanlı sistemler için temel yazılım kütüphaneleri sağlayan işletim sistemleri yavaş yavaş gelişti ve donanım kavramını uygulama yazılımından soyutlamayı kolaylaştıran kavramlar ortaya çıktı.

Yıllar sonra işletim sistemleri ve bunların birçok versiyonları gelişti ve ilerledi. Bunlar genel amaçlı işletim sistemlerinden, (UNIX, Windows gibi) daha küçük daha tümleşik gerçek zamanlı sistemlere kadar uzanır (VxWorks gibi). Her biri kısaca aşağıda açıklanmıştır.

1960'lar ve 1970'lerde, orta ölçekli ve ana bilgisayarların (mainframe) revaçta olduğu bu dönemde, pahalı ve kısıtlı bulunan bilgisayar sistemlerine çoklu kullanıcı desteği sağlamak için UNIX geliştirildi. UNIX bu pahalı ve kısıtlı bilgisayarların kaynaklarının, bir çok kullanıcı tarafından paylaşılmasına izin veren bir işletim sistemi idi. Çoklu kullanıcı erişimi için oldukça efektif idi. Bir kullanıcı dosyalarını yazdırırken, diğer bir kullanıcı dosyalarına veriler ekleyebiliyordu. Nihayetinde UNIX masa üstü bilgisayarlardan süper bilgisayarlara kadar tüm bilgisayar sistemlerine geçirildi.

1980'lerde Microsoft, kişisel bilgisayarlara yönelik olan Windows işletim sistemini lanse etti. Hedeflenen, ev ve ofis kullanıcılarının kolayca adapte olabileceği bir grafik ara yüzü sunmaktı. Windows işletim sistemi kişisel bilgisayarların gelişmesine itici bir güç etkisi sağlamıştır.

Yüzyılın sonlarına doğru, zaman artık yeni nesil bilgisayar sistemleri olan gömülü sistemlerindi. Gömülü sistemlerin ihtiyaçlarını karşılayabilmek için VxWorks, Digital Equipment Corp., (1997), gibi ticari gerçek zamanlı işletim sistemleri geliştirildi. Bazı gerçek zamanlı işletim sistemleri ve genel amaçlı işletim sistemlerinin, bazı fonksiyonel benzerliklerinin olmasının yanı sıra, gerçekte oldukça önemli farkları vardı. Bu farklar, neden gerçek zamanlı işletim sistemlerinin gerçek zamanlı gömülü sistemler için daha uygun olduklarını açıklamaya yardımcı oldu.

Gerçek zamanlı ve genel amaçlı işletim sistemlerinde bazı çekirdek işlevsellikler benzerlik gösterir. Bunlar;

- Bazı seviyedeki çoklu görev işletimleri
- Yazılım ve donanım kaynak yönetimi
- İşletim sistemi servislerinin uygulamaya sunulması
- Donanımı yazılımdan soyutlamak

Diğer taraftan, bir gerçek zamanlı işletim sistemi bazı anahtar farklılıklar içerir. Bu farklar gerçek zamanlı işletim sistemlerinin;

- Gömülü uygulamalarda daha güvenilir olması
- Uygulamanın ihtiyacına göre sistem bileşenlerinin arttırılıp azaltılabilmesi
- Daha yüksek hız performansına sahip olması
- Daha az bellek gereksinimine ihtiyaç duyması
- Gerçek zamanlı sistemler için işletim zamanlama poliçelerini içermesi
- ROM veya RAM den açılıp çalışmaya izin veren disksiz gömülü sistem uygulamalarını desteklemesi
- Farklı donanım platformlarına daha kolay taşınabilmesidir.

Günümüzde, genel amaçlı işletim sistemleri kişisel bilgisayarlar, iş istasyonları ve sunucu bilgisayarları gibi sistemlerde kullanılmaktadırlar. Bazı durumlarda genel amaçlı işletim sistemleri bol bol yetecek kadar belleği olan, gerçek zamanlama ile ilgili kriterlere çok az ihtiyaç duyan gömülü sistemlerde kullanılmaktadır. Genel amaçlı işletim sistemleri tipik olarak, yüksek performans ihtiyacı olan ve kısıtlı bellekleri bulunan gömülü sistemler için uygun değildir.

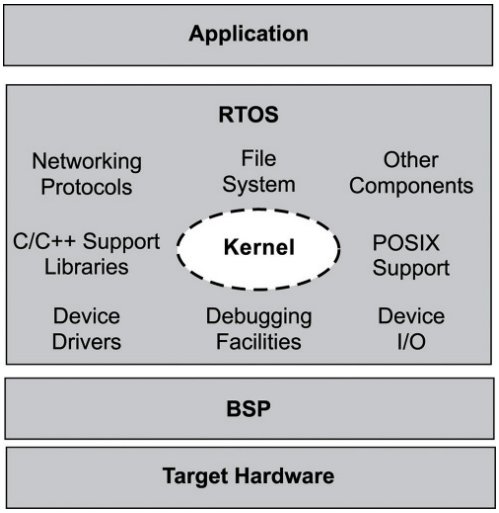
Gerçek zamanlı işletim sistemleri ise, bu ihtiyaçları karşılayabilecek yetenektedirler. Bu sistemler güvenilir, tümleşik ve genişleyip küçültülebilir özelliklere sahip olup gerçek zamanlı gömülü sistemlerde kullanılabilir niteliktedir. Buna ek olarak gerçek zamanlı işletim sistemleri sadece istenilen bu gereksinimleri özel bir uygulama için karşılayabilecek şekilde kolaylıkla uyarlanabilir.

Tekrar hatırlarsak, günümüzde çok daha küçük gömülü aygıtlar hala gerçek zamanlı işletim sistemlerine ihtiyaç duymadan kullanılmaktadır. Bu basit cihazlar tipik olarak düzenlemesi ve yazılması küçük uygulama kodları içerirler. Buradaki amacımız gerçek zamanlı işletim sistemleri kullanımına ihtiyaç duyacak sistemler için bir sistem yazılımı tasarlamaktır.

3.3. Gerçek Zamanlı İşletim Sisteminin Tanımı

Gerçek zamanlı bir işletim sistemi, uygulama kodu geliştirmek için tutarlı bir temel sağlayan, sistem kaynaklarını ve yürütme zamanlarını yöneten bir program yada programlar bütünüdür (Furr, 2002). Gerçek zamanlı işletim sistemi üzerinde geliştirilen uygulama kodu çok basit bir uygulamadan çok daha karmaşık uygulamalara kadar genişleyebilir. İyi tasarlanmış gerçek zamanlı işletim sistemleri, farklı uygulamaların ihtiyaçlarını karşılayabilecek farklı konfigürasyonlara ölçeklendirilebilir olmalıdır.

Örneğin; bazı uygulamalarda bir gerçek zamanlı işletim sistemi; kaynak yönetim algoritmaları, organizasyon ve minimum mantıksal işlemlerin yapılabilmesine olanak sağlayan çekirdek yönetim yazılım parçalarından oluşur (Barry, 2006). Her gerçek zamanlı işletim sistemi bir çekirdek (kernel) içermek zorundadır (Liedthe, 1994). Diğer taraftan bir gerçek zamanlı işletim sistemi, çekirdek işlevinin yanı sıra, dosya sistemi, ağ protokolleri veya Şekil 3.1'de gösterildiği gibi özel bir uygulamanın ihtiyaç duyduğu bileşenleri de içerebilir (Ysboodt ve Nil, 2006), (Dunkels ve diğ, 2006).

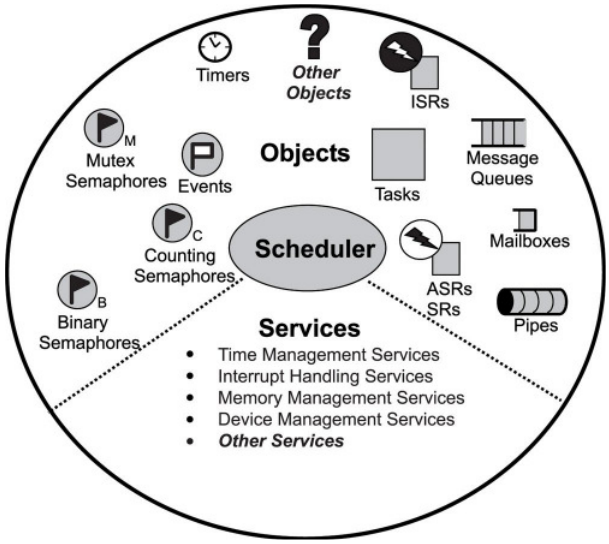


Şekil 3.1: Bir gömülü sistemde olan bileşenler ile gerçek zamanlı işletim sistemine genel bakış

Her ne kadar gerçek zamanlı işletim sistemleri, uygulamaların ihtiyaçlarını karşılayabilecek şekilde ölçeklendirilebilseler de bu bölümde bu işletim sistemlerinin çekirdeklerinin kalbini oluşturan ortak kavramlar üzerinde duracağız. Çoğu gerçek zamanlı işletim sistemi aşağıdaki bileşenleri içerir.

- Görev planlama ünitesi (Scheduler): Her çekirdek de yer alır ve hangi görevin (task) ne zaman yürütüleceğine karar veren bir takım algoritmalar kümesini barındırır. En çok bilinen ve kullanılan planlama algoritmaları round-robin ve rekabetçi (preemptive) planlamadır.
- Nesneler (Objects): Geliştiricilere gerçek zamanlı gömülü uygulamalar tasarlama esnasında yardımcı olan özel sistem servisleridir. Temel çekirdek nesneleri, görevler, semaforlar ve mesaj kuyruklarıdır.
- Servisler (Services): Çekirdeğin, zamanlama, kesme ve kaynak yönetimi gibi, nesneler yada genel operasyonlar üzerinde uyguladığı işlemlerdir.

Şekil 3.2 daha sonra açıklanacak olan bu bileşenleri göstermektedir.



Şekil 3.2: Ortak gerçek zamanlı işletim sistemi bileşenleri

Şekil 3.2 oldukça basitleştirilmiş bir gösterimdir. Tüm gerçek zamanlı işletim sistemi çekirdeklerinin, bu nesneleri, planlama algoritmalarını ve servisleri içermediği unutulmamalıdır.

3.4. Görev Planlama Birimi (Scheduler)

Görev planlayıcısı her işletim sistemi çekirdeğinin kalbidir. Bir planlayıcı hangi görevin ne zaman yürütüleceğine karar vermek için ihtiyaç duyulan algoritmalar kümesini bünyesinde barındırır. Görev planlayıcısının nasıl çalıştığını anlamak için aşağıdaki konular açıklanmaya çalışılacaktır.

- Planlanabilir varlık (schedulable entities)
- Çoklu görev (multitasking)
- İçerik anahtarlama (context switching)
- Görev sevkedici (dispatcher)
- Planlama algoritmaları (scheduling algorithms)

3.4.1. Planlanabilir varlıklar (schedulable entities)

Planlanabilir varlık, önceden belirlenmiş bir planlama algoritmasına bağlı olarak, bir sistemdeki yürütme zamanı için çekişen bir çekirdek nesnesidir. Görevler ve prosesler çoğu çekirdekte bulunan planlanabilir varlıklara örnektir.

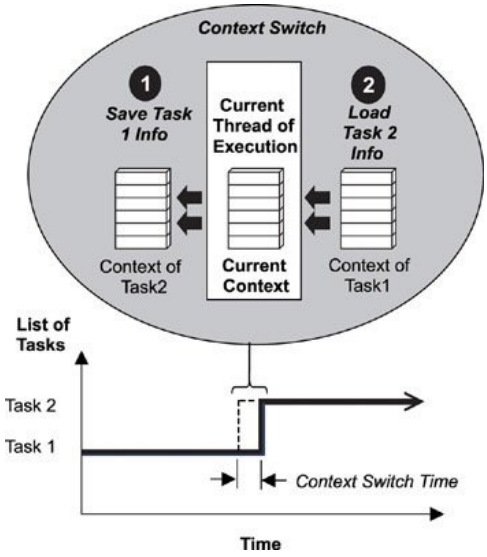
Bir görev; bağımsız, planlanabilir komutlar dizisini içeren bir icrayı gerçekleştiren bağımsız iş parçacığıdır. Bazı çekirdekler proses diye adlandırılan, diğer bir planlanabilir nesne tipini de sağlarlar. Prosesler görevlere benzemelerinin yanı sıra işlemcinin yürütme zamanını kullanmak için bu görevlerle çekişirler. Prosesler daha iyi bir bellek koruma özelliği sağlamaları hususunda görevlerden farklılık gösterirler. Tabi bunun bedelini performansta düşüklük ve daha fazla bellek tüketimi ile öderler. Bu farklılıklara rağmen, anlatım açısından basitlik için burada görev kelimesini ikisini de ifade etmek için kullanacağız.

Mesaj kuyrukları ve semaforlar (semaphore) planlanabilir varlıklar değildir. Bu öğeler görevler arasındaki haberleşme ve senkronizasyon için kullanılan nesnelerdir.

Burada önemli bir soru karşımıza çıkar. Birden fazla çalışma isteği olan planlanabilir varlığın aynı zamanda işletilebilmeleri nasıl mümkün olacak? Cevap; çoklu görev desteği. Çoklu görev özelliği, tek işlemcili sistemlerin içeriğinde tartışılan bir konudur. Çok işlemcili sistemlerde bu durum paralel işlemeye girer ki bu da bizim konumuzun dışındadır.

3.4.2. Çoklu görev (multitasking)

Çoklu görev, bir çalışma zaman aralığı içinde, çoklu aktivitelerin işletim sistemi tarafından ele alınıp işletilmesi yeteneğidir. Gerçek zamanlı bir çekirdek, çalışmak için işletime sokmak zorunda olduğu birçok göreve sahip olabilir. Örnek bir çoklu görev senaryosu Şekil 3.3'de gösterilmiştir.



Şekil 3.3: İçerik anahtarlama gerçekleştiren bir çoklu görev örneği

Bu senaryoda, çekirdeğin, aynı zamanda yürütmesi gereken iş parçacıklarını nasıl işletime alacağı görülmektedir. Bununla beraber, çekirdek aslında önceden belirlenmiş olan planlama algoritmasına göre, zaman paylaşımlı olarak görevleri sırayla işletime almaktadır. Planlayıcı, doğru görevin doğru zamanda işletime alınmasını sağlamak zorundadır.

Burada dikkat edilmesi gereken bir başka önemli nokta, daha önceden atanmış olan önceliklerine göre çalışmaya devam eden donanım kesme servisleri, görevlerin işleyişlerini bozmamaktadır. Her görev kesintiden sonra çalışmasına kaldığı yerden devam etmektedir. Bu çoklu görev planlayıcısının özelliğidir.

İşletilecek görev sayısı arttıkça, mikro işlemci daha fazla performansa ihtiyaç duyar. Bu durum, yürütülecek olan farklı iş parçacıkları arasında gerçekleştirilecek olan içerik anahtarlama işleminin, işlemcinin iş gücünü tüketmesinden kaynaklanır.

3.4.3. İçerik anahtarlama (context switching)

Her bir görev işletime alınacağı sırada, ihtiyaç duyduğu işlemci kayıtçılarından ve bir takım özel değerlerden oluşan bir içeriğe (context) sahiptir. Planlayıcı bir görevden bir başka göreve anahtarlama yapacağı sırada bir içerik anahtarlama işlemi meydana gelir. Bu durumu daha iyi anlayabilmek için tipik bir çekirdekte bu senaryonun nasıl işlediğini ele alalım.

Her yeni görev yaratıldığında, çekirdek aynı zamanda bu görevle ilişkili bilgileri barındıran görev kontrol bloğunu da (task control block - TCB) yaratır. Görev kontrol bloğu, çekirdeğin o görev ile ilgili özel bilgileri saklamasını ve kullanmasını sağlayan bir veri yapısıdır. Görev kontrol bloğu, bir çekirdeğin bir görev ile ilgili ihtiyaç duyduğu tüm bilgileri barındırır. Bir görev çalıştığı sırada onun içeriği oldukça dinamiktir. Her işlem sonucunda bu içerik, değişen veriler barındırır. Bu dinamik veriler görev kontrol bloğu içerisinde saklanır. Bir görevin çalışmama durumda tüm bu dinamik bilgileri, tekrar işletime alınacağı ana kadar kendi görev kontrol bloğunda saklanır. Tipik bir içerik anahtarlama işlemi Şekil 3.3'de gösterilmektedir.

Şekil 3.3'de görüldüğü gibi çekirdekte bulunan planlayıcı Görev-1'i durdurup Görev-2'yi çalıştırmaya karar verdiğinde aşağıdaki adımlar gerçekleşir.

i. Çekirdek Görev-1'in içerik bilgisini, onun kendi görev kontrol bloğuna saklar.
ii. Yürütülecek olan iş parçacığı Görev-2'nin içerik bilgileri, kendi görev kontrol bloğundan alınır.
iii. Görev-2 yürütülüyorken Görev-1'in içeriği korunur. Fakat planlayıcı tekrar Görev-1'i yürütmek isterse Görev-1 en son işletildiği andan itibaren işlemine devam eder.

Planlayıcının bir görevden diğer göreve işletimi verme anı, içerik anahtarlamanın yapılacağı andır. Bu durum işletimde olan görevin yaptığı işle çok ilintili değildir. Planlayıcı görevin işi ne olursa olsun istek geldiği anda, yürütmeyi bir başka göreve verebilir.

Uygulamanın bir sistem çağrısı yaptığı her durumda, planlayıcı gerekiyorsa içeriği anahtarlamak için fırsata sahiptir. Planlayıcı bir içerik anahtarlamanın yapılması gerektiğine karar verirse, bir anahtarlama işlemi sırasında ne yapılacağını bilen görev sevkedicisi (dispatcher) diye adlandırılan sistem modülüne güvenir.

3.4.4. Görev sevkedicisi (dispatcher)

Görev sevkedicisi, içerik anahtarlamayı gerçekleştiren ve işletim akışını değiştiren planlayıcının bir parçasıdır. İşletim sisteminin çalıştığı herhangi bir anda, kontrol akışı olarak da bilinen işleyiş akışı aşağıdaki üç yöntem vasıtasıyla değişir.

- Bir uygulama görevi vasıtasıyla,
- Bir kesme servis işlevi vasıtasıyla,
- Doğrudan çekirdek vasıtasıyla.

Bir görev veya kesme hizmeti bir sistem çağrısı yaptığında kontrol, çekirdek tarafından sağlanan sistem işlevlerini yürütebilmesi için çekirdeğin kendisine geçer. Bu aşamadan sonra işletim, kontrolü bir görevden uygulamanın diğer bir görevine verecek olan sevkediciye geçer. Burada işletime alınacak görevin hangisinin olacağı doğrudan planlama algoritmasına bağlıdır. Sevkedici sadece içerik anahtarlama işlemini yaparak kontrolü belirlenen göreve verir.

Çekirdeğin ilk olarak nasıl başlatıldığına bağlı olarak sevk işlemi farklılık gösterebilir. İşlev çağrımı şeklinde kullanılan sevkedicide, bir görev sistem çağrısı yaptığında, sevkedici her sistem çağrısı tamamlandıktan sonra çekirdekten çıkmak için kullanılır. Bu yüzden sevkedici herhangi bir sistem çağrısına neden olunduğunda, görev durumlarını düzenleyebilir. Örneğin; bir yada daha fazla görev çalışmak için hazır durumda olabilir.

Diğer taraftan, eğer bir kesme servis programı bir sistem çağrısı yaparsa, sevkedici kesme hizmeti tamamlanana kadar durdurulur. Görevler arasındaki içerik anahtarlamayı tetikleyen bazı kaynakların serbest bırakılması durumunda bile bu her zaman doğru bir harekettir. Bu içerik anahtarlama işlemi gerçekleşmeyecek demektir.

Çünkü kesme servis hizmet işleyişi bir görev tarafından kesintiye uğratılmadan tamamlanmak zorundadır. Kesme servis programı işleyişini tamamladıktan sonra, çekirdek doğru görevi işleyişe alan sevkedici vasıtası ile kontrolü bırakır.

3.4.5. Planlama algoritmaları (scheduling algorithms)

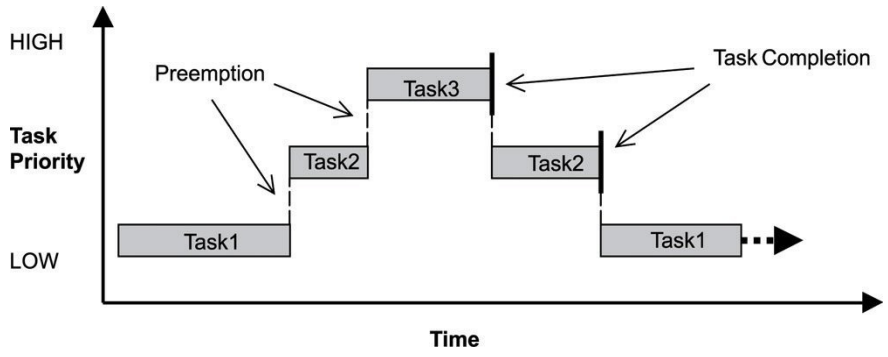
Planlayıcı, planlama kuralları olarak da bilinen algoritmalara göre hangi görevin işletime alınacağına karar verir (Chen ve Wei, 2003), (Zhang ve Bhuyan, 2003). Çoğu çekirdek günümüzde iki ana algoritmayı destekler.

- Rekabetçi öncelik tabanlı planlama (preemptive priority-based scheduling)
- Round-Robin planlama

Gerçek zamanlı işletim sistemi üreticileri, tipik olarak bu algoritmaları önceden tanımlarlar. Bununla beraber bazı durumlarda geliştiriciler kendi algoritmalarını tanımlar ve oluştururlar. Bu iki algoritma kısaca açıklanmaya çalışılacaktır.

3.4.5.1. Rekabetçi öncelik tabanlı algoritma

Bu iki planlama algoritmasından burada bahsedilecek olan, çoğu gerçek zamanlı çekirdek de varsayılan olarak kullanılan rekabetçi öncelik tabanlı planlama algoritmasıdır. Şekil 3.4'de gösterildiği gibi bu tip planlamada, herhangi bir noktada çalışacak olan görev, sistemdeki tüm çalışmaya hazır görevler arasında en yüksek önceliğe sahip olandır.



Şekil 3.4: Rekabetçi öncelik tabanlı algoritma

Gerçek zamanlı işletim sistemi çekirdekleri genellikle 256 öncelik seviyesini destekler. Burada 0 en yüksek önceliği, 255 de en düşük önceliği ifade eder. Bazı çekirdeklerde ise bu durum tam tersidir. Gerçekte bu durumla ilgili bir sınırlandırma yoktur.

Tasarımcı kendi isteği doğrultusunda kendi öncelik numaralandırmasını yapabilir. Yine de düşünce tarzı temel olarak aynıdır.

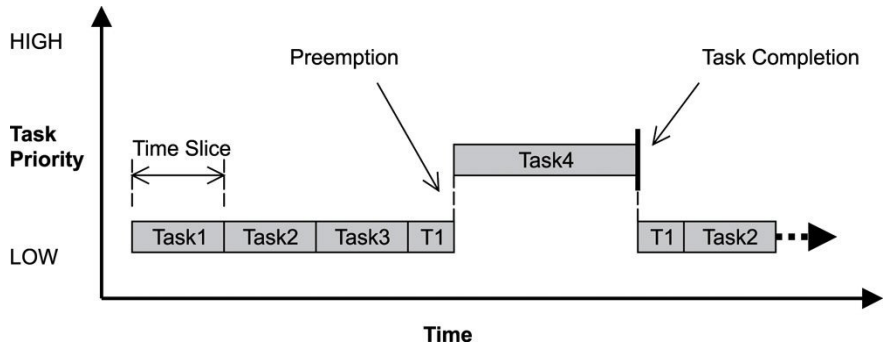
Bu tip planlayıcılar da her bir görev bir önceliğe sahiptir ve ilk olarak en yüksek öncelikli görev işletime alınır. Eğer sistemde o anda çalışmakta olan görevden daha yüksek önceliğe sahip çalışmaya hazır bir görev varsa, çalışan görevin tüm içeriği kendi görev kontrol bloğuna saklanır ve yüksek önceliğe sahip olan görev anahtarlanır.

Şekil 3.4'de görüldüğü gibi Görev-1 daha yüksek önceliğe sahip ve Görev-3 tarafından önlenecek olan Görev-2 tarafından önlenir. Görev-3 tamlandığında, Görev-2 işletimine kaldığı yerden devam eder. Benzer şekilde Görev-2 tamamlandığında da Görev-1 işletimine devam eder.

Her ne kadar görevler yaratıldıkları anda bir öncelikle ilişkilendirilseler de, bir görevin önceliği dinamik olarak çekirdek tarafından sağlanan sistem çağrıları yoluyla da değiştirilebilir. Görev önceliklerinin dinamik olarak değiştirilebilmesi yeteneğinin olması, oluşturulacak olan gömülü uygulamalara, harici olarak oluşan olaylarda nasıl davranacağının belirlenmesini sağlayacak bir esneklik kazandırır. Bu durum iyi tasarlanmış gerçek zamanlı sistemlerde olması gereken bir özelliktir. Bununla beraber bu yeteneğin doğru kullanılmaması, önceliklerin karıştırılmasına, ölümcül kilitlenmeye ve sonunda bir sistem hatasına neden olur.

3.4.5.2. Round-robin algoritması

Round-robin planlama, her bir görevin işlemci işletim zamanını eşit olarak paylaşmasını sağlar. Saf round-robin planlama algoritması, gerçek zamanlı sistem ihtiyaçlarını karşılayamaz. Çünkü gerçek zamanlı sistemlerde görevler önem derecelerine göre değişkenlik gösteren bir çalışmaya ihtiyaç duyarlar. Bu yüzden çekişmeli öncelik tabanlı planlama round robin ile birlikte kullanılarak, aynı önceliğe sahip görevlerin eşit zaman aralıklarında çalışmaları sağlanırken, daha yüksek bir önceliğe sahip olan görevin ise hemen işletime alınması garanti altına alınır. Bu durum Şekil 3.5'de gösterilmektedir.



Şekil 3.5: Round-robin ve öncelik tabanlı planlama

Zaman dilimleme ile her bir görev önceden tanımlanmış bir zaman aralığında işletilir. Bir çalışma zamanı sayıcısı her bir göreve ait olan ve belirlenen saat çeviriminde bir artan zaman dilimini izler. Bir görevin zaman dilimi tamamlandığında sayıcı sıfırlanır ve önceliği dikkate alınarak sıranın en sonuna atılır.

Eğer bir görevin round-robin işletimi, daha yüksek önceliğe sahip başka bir görev tarafından engellenirse, bu görevin çalışma zamanı değeri saklanır ve yüksek öncelikli görevin işletimi sona erdiğinde, çalışması için ayrılan süreden kalanı kadar işletimde tutulur. Bu durum Şekil 3.5'de gösterilmiştir. Burada Görev-1 daha yüksek önceliğe sahip Görev-4 tarafından engellenir fakat Görev-4 tamamlandıktan sonra işletimine, kalan süresi kadar kaldığı yerden devam eder.

3.5. Nesneler

Çekirdek nesneleri, gerçek zamanlı gömülü sistemler için uygulama geliştirme aşamalarını inşa etmede kullanılabilecek özel veri yapıları ve işlevleridir.

- Görevler (tasks): Aynı anda bir arada olan ve işlemci çalışma zamanı için birbirleriyle çekişen yürütülebilir bağımsız iş parçacıklarıdır.
- Semafor (semaphore): Senkronizasyon ve karşılıklı çekişme durumları için kullanılan, görevler tarafından arttırılıp azaltılabilen jeton benzeri nesnelerdir.
- Mesaj kuyrukları (message queune): Görevler arasındaki senkronizasyon, karşılıklı çekişme ve mesajlar vasıtasıyla veri değişimi için kullanılabilen tampon benzeri veri yapılarıdır.

Geliştiriciler, art arda çalışma, faaliyet senkronizasyonu ve veri haberleşmesi gibi ortak gerçek zaman tasarım problemlerini çözebilmek için burada bahsedilen ve bahsedilmeyen temel çekirdek nesnelerini bir arada kullanmak suretiyle gerçek zamanlı gömülü uygulamalar yaratırlar. Bu tasarım problemleri ve çekirdek nesneleri, sonraki bölümlerde daha ayrıntılı incelenecek olan bu sorunları çözmek için kullanılır.

3.6. Servisler

Nesnelerin yanı sıra çoğu çekirdek, gerçek zamanlı gömülü sistemler için uygulamalar oluşturmaya yardımcı olan servisler de sağlar. Bu servisler, çekirdek nesneleri üzerinde işlem yapılmasına yada zamanlayıcı yönetiminde kullanılabilen, kesme işleme, aygıt giriş/çıkışı ve bellek yönetimi gibi operasyonlara olanak sağlayan uygulama programlama arabirimi (application programming interface - API) çağrıları kümesinden meydana gelir. Burada bahsedilenlerin dışında da işletim sistemleri birtakım özel servisler sunabilmektedir.

3.7. Bir Gerçek Zamanlı İşletim Sisteminin Anahtar Karakteristikleri

Bir uygulamanın gereksinimleri, bu uygulamanın çalıştığı işletim sisteminin gereksinimlerini belirler. Ortak özelliklerin bazıları aşağıda gibidir.

- Güvenilirlik
- Öngörülebilirlik
- Performans
- Tümleşiklik
- Ölçeklendirilebilirlik

3.7.1. Güvenilirlik

Gömülü sistemler mutlak suretle güvenilir olmalıdır. Uygulamaya bağlı olarak, sistem insan müdahalesi olmaksızın uzun zaman aralıklarında işleyişine devam edebilir olmalıdır.

Sistemden sisteme farklı güvenilirlik derecelerine ihtiyaç duyulabilir. Örneğin; bir sayısal güneş enerjili hesap makinesi yeterli ışık alamadığı durumda kapanabilir. Bu kabul edilebilir bir unsurdur. Diğer taraftan bir telefon santrali kesik olduğu süre boyunca bir maliyete sebep olmadan operasyon sırasında kapatılamaz veya sıfırlanamaz (reset). Bu uygulamalardaki gerçek zamanlı işletim sistemleri farklı güvenilirlik derecelerine sahiptir.

Her ne kadar farklı güvenilirlik dereceleri kabul edilebilir olsalar da, genel olarak sistem güvenilir olmalı ve hata oluşmamalıdır. Geliştiricilerin sistemlerinin güvenilirlik düzeylerini saptayabilmek için kullandıkları ortak bir yol olan, örnek sistemlerin bir yıl içerisindeki servis dışı kalma sayıları Tablo 3.1.'de gösterilmektedir. Buradaki 9'ların sayısı ve bu kolonun altındaki yüzde ifadeleri, o sistemin toplam çalışma zamanında servis vermek zorunda olduğu yüzdeyi vermektedir.

Tablo 3.1: İzin verilen hata oranlarına göre örnek sistemler

9'ların sayısı	Bir yıldaki hata zamanı	Tipik uygulamalar
3 dokuz (99.9%)	~9 saat	Masa üstü bilgisayar
4 dokuz (99.99%)	~1 saat	Enterprise Server
5 dokuz (99.999%)	~5 dakika	Carrier-Class Server
6 dokuz (99.9999%)	~31 saniye	Carrier Switch Equipment

Gerçek zamanlı işletim sistemleri güvenilir olmak zorundayken, işletim sisteminin çalıştığı gömülü sistemin ne kadar güvenilir olduğunun belirlemesi söz konusu değildir. Bu durum donanım, aygıt sürücüleri, işletim sistemi ve çalışan uygulamanın bir kombinasyonu ile ilişkilidir.

3.7.2. Öngörülebilirlik

Bir çok gömülü sistem aynı zamanda gerçek zamanlı bir sistem olduğu için, gerekli olan zaman ihtiyacını karşılamak, uygun operasyondan emin olmak için anahtar bir rol oynar. Bu durumda gerçek zamanlı işletim sistemi kesin bir derecede tahmin edilebilirlik ihtiyacı için kullanılır. Deterministik terimi gerçek zamanlı işletim sistemlerinin, bilinen zaman periyotlarında meydana gelen işletim sistemi çağrılarının, öngörülebilir bir davranış ile tamamlanabilmesi durumunu açıklar.

Geliştiriciler bir gerçek zamanlı işletim sisteminin determenistikliğinin geçerliliğini onaylamak için, basit kıyaslama (benchmark) programları yazabilirler. Sonuç, spesifik sistem çağrılarına verilen cevapların işletim zamanlarına dayanır. İyi bir determenistiğe sahip işletim sistemlerinde, her bir tip sistem çağrısına verilen cevap süresindeki sapma payı çok küçük olur.

3.7.3. Performans

Bu gereksinim bir gömülü sistemin zaman ihtiyaçlarını yerine getirmek için yeterli hızda işleyebilmek zorunda olması gerektiğini dikte eder. Tipik olarak her bir görevin tamamlanması için verilen süre sistem işlemcisinin izin verdiği çalışma süresinden kısa olmalıdır. Her ne kadar sistem işlem gücü donanım ile ilişkili olsa da bu donanım üzerinde çalışan yazılım da sistem performansında etki eden bir unsurdur. Genel olarak bir işlemcinin performansı bir saniyede işleyebildiği milyon komut ile ifade edilir (million instructions per second - MIPS).

Üretilen işe de (throughput) donanım ve yazılım bileşenlerini hesaba katan, tüm sistem performansını ölçmede kullanılan bir başka ifadedir. Üretilen iş sistemin girişlerine bağlı olarak üretebildiği çıkışın oranı olarak tanımlanabilir. Aynı zamanda üretilen iş, veri transferi için belirlenen zaman aralıklarında yapılan toplam transfer miktarı olarak da görülebilir. Veri transferinde üretilen iş, bir saniyede gönderilen bitlerin sayısı olarak ölçülür.

Bazı durumlarda ise geliştiriciler işletim sistemlerinin performanslarını çağrı tabanlı olarak ölçerler. Kıyaslama işlemleri bir sistem çağrısının başlama ve tamamlanma aralığının zaman olarak ölçülmesi ile gerçekleştirilir. Bu yöntemler her ne kadar tasarım safhalarını analiz etmemize yardımcı olsa da, gerçek performans testi sadece sistemin bir bütün olarak ele alındığı durumda gerçekleştirilebilir.

3.7.4. Ölçeklenebilirlik

Gerçek zamanlı işletim sistemleri birbirinden çok farklılıklar gösterebilen gömülü sistemlerde çalışabileceklerinden ötürü, uygulamaya yönelik ihtiyaçları karşılayabilmek için genişleyip küçülebilir yapıda olmalıdır. Ne kadar fazla fonksiyonelliğe ihtiyaç olunduğuna bağlı olarak, işletim sistemi modüler birtakım bileşenleri sisteme ekleyip çıkartabilir yapıda olmalıdır. Örneğin; bazı uygulamalar basit mantıksal operasyonlar yaparken bazıları dosya sistemi, ağ katman protokolleri gibi daha karmaşık modüllere ihtiyaç duyabilir.

Bir işletim sisteminin uygun ve yeterli bir şekilde genişleyememesi durumunda, geliştirme takımı, ihtiyaç duyulan eksik parçaları satın alabilir yada yeniden oluşturabilir. Örnek olarak bir geliştirme takımının, gerçek zamanlı işletim sistemi kullanarak bir cep telefonu ve baz istasyonu projesi yapmak istediğini varsayalım. Eğer işletim sistemi yeterli bir şekilde ölçeklenebilirse her iki proje için de kullanılabilir olur. Bu durum iki farklı işletim sistemi kullanmanın getirdiği zaman ve para kaybını önler.

3.8. Hatırlanması Gereken Noktalar

Bazı önemli noktaları tekrar hatırlatacak olursak;

- Gerçek zamanlı işletim sistemleri, gerçek zamanlı uygulamaya yönelik gömülü sistemler için en uygun seçimdir. Genel amaçlı işletim sistemleri ise tipik olarak genel amaçlı sistemler için kullanılır.
- Çekirdek her bir işletim sisteminin ana modülüdür ve tipik olarak çekirdek nesneleri, servisler ve planlayıcıyı içerir.
- Çekirdek, görev yönetimi ve planlaması için farklı algoritmalar kullanabilir. En çok kullanılan algoritmalar, engelleyici öncelik tabanlı ve round-robin algoritmalarıdır.
- Gerçek zamanlı gömülü sistemler için tasarlanan işletim sistemleri; güvenilir, öngörülebilir, yüksek performanslı, tümleşik ve ölçeklendirilebilir olmalıdır.

4. İŞLETİM SİSTEMLERİNDE GÖREVLER

4.1. Giriş

Basit yazılım uygulamaları tipik olarak ardışık ve sırasal çalışmak üzere tasarlanırlar. Yani işlemci sırayla komut veya önceden tanımlanmış komutlar dizisini işler. Bununla beraber bu işleyiş yapısı, gerçek zamanlı gömülü sistemler için uygun değildir. Bu tür sistemlerde genel olarak dar bir zaman aralığında bir çok giriş ve çıkış işleminin gerçekleştirilmesi istenir. Bu sebepten gerçek zamanlı gömülü yazılım uygulamaları, birçok işlemi eş zamanlı olarak gerçekleştirecek şekilde tasarlanmak zorundadır.

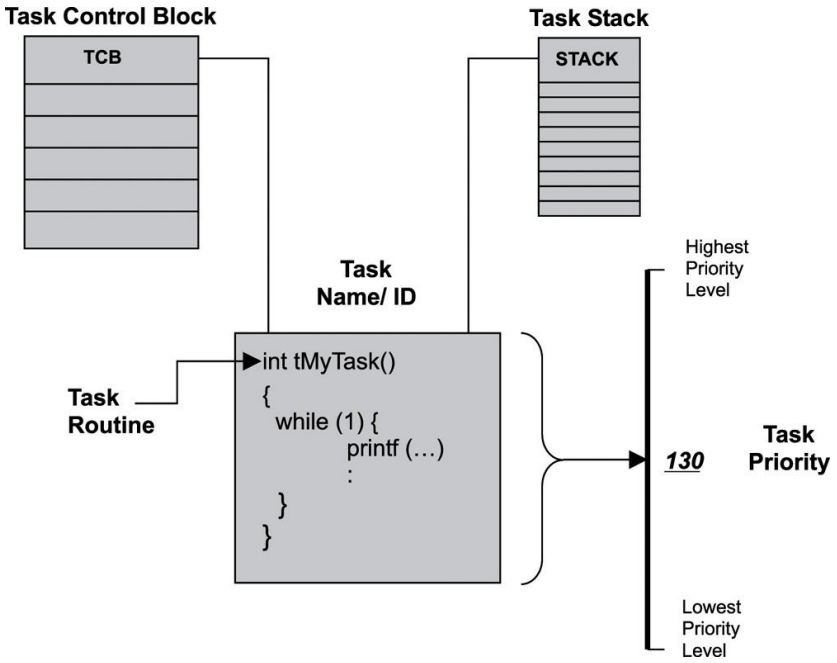
Eş zamanlı tasarım geliştiricilerin bir uygulamayı, küçük, planlanabilir ve sırasal parçalara bölmelerini gerekli kılar. Bu işlem doğru bir şekilde yapıldığında, eş zamanlı tasarım gerçek zamanlı sistemlerin zamanlama ve performans ihtiyaçlarını karşılayabilmek için çoklu görev kullanımına izin verir. Çoğu gerçek zamanlı işletim sistemi çekirdeği, bir uygulama geliştirirken kullanılabilecek görev nesneleri ve görev yönetim servislerini de sağlar. Bu bölümde aşağıdaki konular incelenecektir.

- Görev tanımı
- Görev durumları ve planlama
- Tipik görev operasyonları
- Tipik görev yapısı
- Görev koordinasyonu ve eş zamanlılık

4.2. Görev Tanımı

Bir görev, işlemci yürütme zamanı için diğer eş zamanlı görevler ile sürekli çekişen bağımsız bir iş parçacığıdır. Geliştiriciler bir uygulamayı, belirlenen zaman kısıtlaması içerisinde gerçekleştirebilmek için bir çok eş zamanlı yürütülen göreve bölerler ve dağıtırlar.

Bir görev planlanabilirdir. Bölüm 3.4'de bahsedildiği gibi, bir görev sistemde önceden belirlenen algoritmalara göre yürütme zamanını kullanabilmek için çekişme kabiliyetine sahiptir. Her bir görev farklı parametre kümesi ile tanımlanır ve özel bir veri yapısını kullanır. Özel olarak oluşturulma anında, her bir görev kendisi ile ilişkilendirilmiş bir isim, bir tanıtıcı, bir öncelik (eğer öncelik, planlama algoritmasında kullanılacak ise), bir görev kontrol bloğu, bir veri yığını ve bir görev işlevinden oluşur. Bu bileşenlerin tümü birlikte görev nesnesi olarak bilinen sistem nesnesini oluşturur. Şekil 4.1'de bir görev ve ona ait veri yapıları gösterilmektedir.



Şekil 4.1: Bir görev ve onunla ilişkili parametre ve veri yapıları

Çekirdek çalışmaya başladığında, kendine ait sistem görevlerini oluşturur ve bu görevler için ayrılmış olan öncelik kümesinden uygun olanlarını seçer ve ilişkilendirir. Ayrılmış öncelik seviyeleri işletim sisteminin dahili kendi görevleri için önceden belirlemiş olduğu önceliklerdir. Bir uygulama bu öncelik seviyelerini kendi görevleri için kullanmaktan kaçınmalıdır. Çünkü bu ayrılmış seviyeleri kullanan bir uygulama görevi, tüm sistem performansını ve davranışını etkiler. Çoğu işletim sistemi için, bu ayrılmış öncelikler kullanıma sunulmazlar. Çekirdek kendi sistem görevleri ve bunların ayrılmış önceliklerini sürekli işletmek ister. Bu öncelikler değiştirilmemesi gereken önemli unsurlardır.

Sistem görevleri tipik olarak .aşağıda ifade edilenler gibidir.

- Başlangıç yada açılış görevi: tüm sistemi başlangıç durumuna getirir ve sistem görevlerini yaratır ve başlatır
- Boş görev: hiçbir aktivitenin olmadığı işlemcinin boşta kaldığı zamanlarda kullanılır
- Logging görevi: sistem mesajlarını kaydeder
- İstisnai durum görevi: istisnai durumlarda çağırılan görevdir
- Veri ayıklama ajan görevi: bir hükmedici sistem vasıtası ile hata ayıklama işleminin yapılabilmesine olanak tanır.

Çekirdek içerisine dahil edilmiş olan diğer bileşenleri kullanan başka sistem görevlerinin de açılış esnasında başlatılacağı unutulmamalıdır.

Çekirdeğin başlaması esnasında yaratılan boş sistem görevi göz ardı edilmemesi gereken bir sistem görevidir. Bu görev sistemdeki en düşük öncelik seviyesine ayarlanır ve tipik olarak sonsuz bir döngü şeklinde uygulanır. Bu görev sistemde çalışan veya çalışmaya hazır hiçbir görevin bulunmaması durumunda işlemcinin boş zamanını kullanmak amacıyla işletime alınır. Boş sistem görevi işlemcinin çalışma prensibi itibariyle sürekli olarak bir komut işlemesi gerekmesinden dolayı gereklidir. İşlemcinin bekleme veya uyku durumuna geçmediği, sistemde hiçbir görev olmadığı veya çalışan bir görevin bulunmadığı durumda, işlemcinin program sayacı mutlak suretle geçerli bir komut göstermek zorunda olmasından dolayı boş görev gereklidir. Böylece boş görev işlemcinin program sayacının sürekli olarak geçerli bir değer göstermesini garanti eder. Aksi halde sistemde ölümcül bir hata olabilir ve güvenilirlik sarsılır.

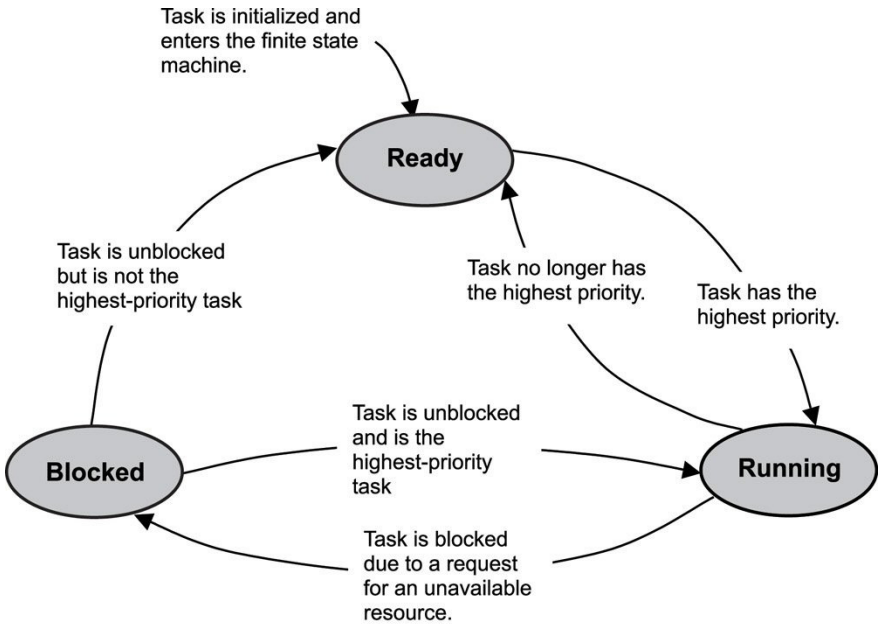
Bazı gerçek zamanlı işletim sistemi çekirdekleri bazı durumlarda, özel bir uygulamanın spesifik ihtiyaçlarını karşılamada kullanmak için, boş görev yerine kullanıcı tanımlı işlevlere izin verebilmektedir. Bu durum için en iyi örnek güç tasarrufu olabilir. Sistemde çalışan veya çalışacak hiçbir görevin olmadığı durumda çekirdek, görevi bu kullanıcı tanımlı işleve verir. Böylece kullanıcı boş görev içerisinde sistemde güç tasarrufu elde etmek için kendi yordamlarını kullanabilir.

Çekirdek başladığında ve tüm gerekli olan görevler yaratıldıktan sonra, uygulamanın başlaması için, çekirdek önceden tanımlanmış olan giriş noktasına (önceden tanımlı bir işlev gibi) dallanır. Bu giriş noktasında geliştirici kendi uygulaması ile ilgili diğer uygulama görevlerini ve kullanacağı sistem nesnelerini yaratır.

Uygulama geliştiricisi yeni bir görev yarattığında, bu görev ile ilgili isim, öncelik, yığın boyutu ve görev yordamını tanımlamak zorundadır. Geri kalan bu görevle ilişkili tanıtıcı numara, görev kontrol bloğunun yaratılması ve atanması, yığın için bellek tahsisi ve yönetilmesi işlemi çekirdeğin sorumluluğundadır.

4.3. Görev Durumları ve Planlama

İster sistem görevi isterse bir uygulama görevi olsun, bir görevin mevcut olduğu herhangi bir anda bu görevler hazır, çalışıyor yada engellenmiş gibi birkaç durumdan birinde olabilir. Bir gerçek zamanlı gömülü sistem çalışıyorken her bir görev sonlu durum makinesine göre bir durumdan diğerine geçer. Şekil 4.2, tipik bir sonlu durum makinesinin görev durumları üzerinde nasıl etki ettiğini göstermektedir.



Şekil 4.2: Görev işletim durumları için tipik bir sonlu durum makinesi

Her ne kadar çekirdek, görev durumlarını farklı tanımlayabilse de genel olarak çoğu tipik engelleyici planlama çekirdeklerinde üç ana durum kullanılır. Bunlar;

- Hazır: Görev çalışmaya hazırdır fakat daha yüksek önceliğe sahip bir görev işletildiği için çalışamamaktadır.
- Engelli: Görev mevcut olmayan bir kaynak isteğinde bulunduğu için engellenmiştir. Kendisi için belirlenen zaman süresince veya birtakım sistem olayları oluşuncaya dek bu durumda bekler.
- Çalışıyor: Görev sistemdeki en yüksek önceliğe sahiptir ve işletiliyordur.

VxWorks gibi bazı ticari gerçek zamanlı işletim sistemi çekirdekleri askıda (suspended), beklemede (pended) ve gecikmede (delayed) gibi daha fazla seçenekli durumlar da tanımlar. Bu durumlardan, beklemede ve gecikmede, engelleme durumun alt durumları olarak düşünülebilir. Bekleme durumu, serbest bırakılması gereken bir kaynağı bekleme anı; gecikme durumu ise devam edebilmek için belli bir zaman periyodunun dolmasını bekleme durumudur. Askıya alma, hata ayıklama amacıyla kullanılabilen bir durumdur. Bu görev durumları çoğunlukla işletim sistemi tasarımcılarına göre değişkenlik gösterir.

Çekirdek, görevin sonlu durum makinesinin nasıl gerçekleştirildiğine bakmaksızın sistemdeki tüm görevlerin mevcut durumlarını tutmak zorundadır. Çekirdek içerisinden yapılan çağrılar, ilk olarak çekirdeğin planlayıcısının hangi görevin durumunu değiştirmesi gerektiğine karar vermesini ve o görevin durumunu değiştirmesini sağlar.

Bazı durumlarda çekirdek bazı görevlerin durumlarını değiştirdiğinde en yüksek öncelikli görevin durumuna bağlı olarak içerik anahtarlama gerçekleşmeyebilir. Diğer taraftan bazı durumlarda da durum değiştirme işlemi içerik anahtarlamanın anında gerçekleşmesine neden olabilir. Böyle bir durum ortaya çıktığında, o anda çalışan görev ya engellenir yada hazır duruma getirilerek kuyruğa atılır. Yerine en yüksek öncelikli diğer bir görev getirilir.

Sonraki bölümde hazır, çalışıyor ve engelli durumları hakkında biraz daha bilgi edineceğiz. Bu açıklamalarda, tek işlemcili sistemlerin kullanıldığı ve çekirdeğin öncelik tabanlı engelleyici planlama algoritması kullanıldığı varsayılacaktır.

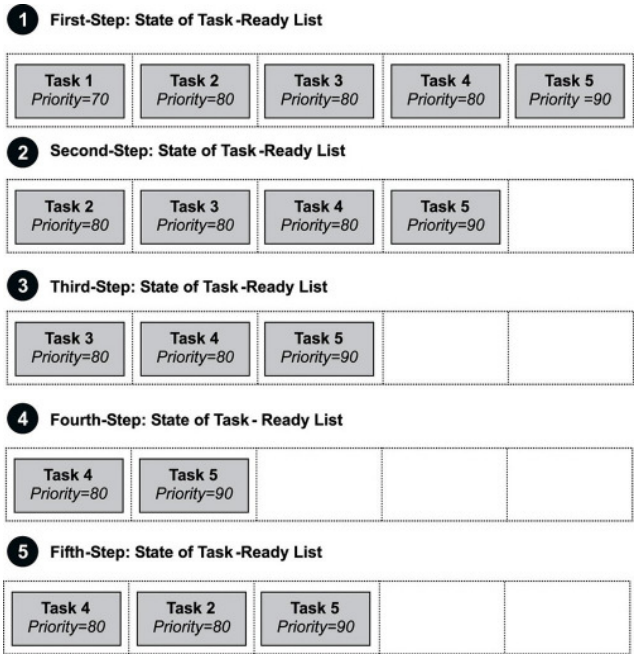
4.3.1. Hazır durumu

Bir görev yaratıldığında ve çalışmak için hazır duruma getirildiğinde, çekirdek bu görevi hazır duruma sokar. Bu durumda görev, sistemdeki diğer hazır görevler ile işlemci çalışma zamanını almak için çekişir. Şekil 4.2'de gösterildiği gibi hazır durumdaki görev doğrudan engellenmiş duruma getirilmez. Bir görev ilk önce çalışma isteğini bildirmek için bir engelleme çağrısı (blocking call) yapar. Bu çağrı, bir görevin tamamlanması için değil de o görevi doğrudan engelli duruma getirmek için kullanılır. Bu yüzden hazır durumdaki görevler, sadece çalışma durumlarına getirilebilirler. Çünkü çoğu görev hazır durumda olabileceğinden dolayı, çekirdek planlayıcısı hangi görevin çalışma durumuna getirileceğine karar vermek için her bir görevin kendine ait olan önceliği kullanır.

Her bir öncelik derecesi için bir tane göreve izin veren çekirdekler için, planlama algoritmasının (round-robin kullanılmadığı) bir sonraki seferde en yüksek önceliğe sahip görevi çalıştıracağı açıktır. Bu tür gerçeklemelerde, çekirdek bir uygulama içerisinde kullanılabilecek görevlerin sayısını, uygulamadaki toplam öncelik seviyeleri sayısına göre limitler.

Bununla beraber, uygulamalarda çok fazla görevin kullanılmasına izin veren çoğu çekirdek, her bir öncelik seviyesi için birden fazla görevin atanmasına da izin verir. Bu durumda planlama algoritması hazır görevler listesini tutmaktan çok daha fazla ve karmaşık işlemler gerektirir. Bazı çekirdekler her bir öncelik seviyesi için ayrı bir hazır görev listesi tutarken, bazıları da tümleştirilmiş tek bir liste tutabilir.

Şekil 4.3, görevleri hazır durumdan çalışıyor durumuna geçirmek için bir hazır listesi kullanan çekirdek planlayıcısının, bu işlemi nasıl gerçekleştirdiğini beş adımda göstermektedir. Bu örnek tek işlemcili bir sistemde, öncelik tabanlı çekişmeli planlama algoritmasının kullanıldığı ve 255'in en düşük, 0'ın da en yüksek önceliği gösterdiği varsayımı ile verilmektedir. Gösterimin basitliği için burada hiçbir sistem görevinin gösterilmediğine dikkat edilmelidir.



Şekil 4.3: Görev hazır listesinin nasıl çalıştığının beş adımda gösterimi

Bu örnekte, Görev 1,2,3,4 ve 5 çalışmaya hazır durumda ve çekirdek içerisinde önceliklerine göre düzenlenen bir hazır listesinde tutulmaktadırlar. Görev-1 en yüksek önceliğe sahip (70); Görev-2,3 ve 4 ise bir sonraki yüksek öncelikli görevlerdir. Görev-5 ise en düşük önceliğe sahip görevdir. Aşağıdaki adımlar, hazır görev listesini kullanarak, bir görevi hazır durumdan çalışıyor durumuna geçirecek olan çekirdeğin, bu işi nasıl gerçekleştirdiği açıklamaktadır.

i. Görev-1,2,3,4 ve 5 çalışmaya hazır durumda olup hazır listesinde beklemektedir.
ii. Görev-1 en yüksek önceliğe sahip olduğu için (70), çalışır duruma getirilecek ilk hazır görevdir. Eğer bundan daha yüksek önceliğe sahip bir görev çalışmıyorsa, çekirdek Görev-1'i hazır listesinden alır ve onu çalışıyor duruma getirir.
iii. Çalışma esnasında, Görev-1 bir engelleme çağrısı yapar. Bunun sonucunda çekirdek Görev-1'i engelli duruma geçirir; hazır listesinde bulunan bir sonraki en yüksek önceliğe sahip görev olan Görev-2 yi listeden alır ve çalışır duruma getirir.
iv. Daha sonra Görev-2 bir engelleme çağrısı yapar. Çekirdek Görev-2 yi engelli duruma geçirir; listedeki 80 önceliğine sahip diğer bir görev olan Görev-3 hazır listesinden alır ve çalışıyor durumuna geçirir.

v. Görev-3 çalışıyorken Görev-2'nin istekte bulunduğu kaynağı serbest bırakır. Bu durumda çekirdek Görev-2'yi, hazır durumuna alır ve onu hazır görev listesinde 80 numaralı öncelik seviyesinin en sonuna ekler ve Görev-3 çalışmaya devam eder

Her ne kadar burada gösterilmese de, bu senaryoda eğer Görev-1'in engelli durumu sona ererse, çekirdek Görev-1'i, öncelik seviyesinin o an çalışan (Görev-3) görevden daha yüksek olmasından dolayı çalışıyor duruma getirir. Daha önce Görev-2 gibi Görev-3'de bu noktada hazır duruma getirilir ve listede aynı önceliğe sahip Görev-2'nin arkasına ve Görev-5'in önüne yerleştirilir.

4.3.2. Çalışıyor durumu

Tek işlemcili bir sistemde bir anda sadece tekbir görev çalışabilir. Bu durumda bir görev çalışıyor durumuna taşındığında, işlemci bu görevin veri yapısında bulunan uygun yazmaç değerlerini yükler. İşlemci daha sonra görevin komutlarını işletir ve ilişkilendirilmiş yığını maniple eder.

Bir önceki bölümde açıklandığı gibi, bir görev çalışırken bile hazır duruma tekrar geri döndürülebilir. Bir görevin, çalışıyor durumdan hazır duruma, daha yüksek önceliğe sahip bir görev tarafından taşınması sağlanabilir. Bu durumda kontrolü kaybeden görev kendisi için uygun olan öncelik tabanlı hazır listesine yerleştirilir ve daha yüksek önceliğe sahip olan görev, hazır durumdan çalışıyor durumuna geçer.

Hazır görevden farklı olarak, çalışan bir görev engelli bir duruma aşağıdaki yöntemlerden birini kullanarak getirilebilir.

- Mevcut olmayan bir kaynak isteğinde bulunduğunda
- Bir olayın meydana gelmesini bekleyen bir istekte bulunduğunda
- Görevi belli bir süre geciktirecek bir çağrı oluştuğunda

Bu olaylardan her hangi birinde, görev çalışıyor durumundan engellenmiş duruma geçer.

4.3.3. Engellenmiş durum

Engellenmiş durumunun bir işletim sisteminde bulunması, gerçek zamanlı sistemlerde son derece önemlidir. Aksi halde daha düşük önceliğe sahip görevlerin çalışması söz konusu değildir. Eğer daha yüksek önceliğe sahip bir görev, engellenecek bir şekilde tasarlanmaz ise, işlemci kaynaklarının tümünü tüketebilir.

İşlemci kaynaklarının tükenmesi, sistemdeki daha yüksek önceliğe sahip görevin tüm işlemci çalışma süresini kullanarak daha düşük öncelikli görevlere izin vermemesi durumudur.

Bir görev sadece bir engelleme çağrısı veya birtakım engelleme kondisyonları oluştuğunda engelli duruma geçer. Bir görev engelleme kondisyonu devam ettiği sürece bu durumda kalır. Aşağıda engelleme kondisyonlarının nasıl oluşabileceği listelenmiştir.

- Bir görev tarafından daha önceden tutulmuş olan semafor nesnesine erişim yapılması durumunda
- Belli bir mesajın gelmesinin beklenmesi durumunda
- Görevin beklemesi için dayatılan belli bir zaman gecikmesi durumunda

Bir görevin engelli durumu ortadan kalktığında, bu görev eğer sistemdeki en yüksek önceliğe sahip görev değilse hazır görev listesine eklenir.

Bununla beraber, eğer engelli durumu ortadan kalkmış olan bir görev en yüksek önceliğe sahip ise, bu görev doğrudan çalışıyor konumuna taşınır. İşletimden alınan görev uygun öncelik seviyesindeki hazır listesine eklenir.

4.4. Tipik Görev Operasyonları

Çekirdek, görev nesnelerini barındırmasının yanı sıra aynı zamanda görev yönetim servislerini (task management services) de sağlar. Görev yönetim servisleri, çekirdeğin bir görevin işleyişiyle ilgili senaryoları uygulayabileceği eylemler kümesini içerir. Örneğin; bir görevin oluşturulması, görev kontrol bloğunun ve yığın alanının yönetilmesi gibi.

Çekirdek aynı zamanda geliştiricilere, görevleri maniple edebilmeleri için sistem çağrıları sağlar. Geliştiricilerin uygulayabilecekleri bazı ortak operasyonlar aşağıda listelenmiştir.

- Bir görevin yaratılması ve silinmesi
- Görev planlamalarının kontrol edilmesi
- Görev ile ilgili bilgilerin elde edilmesi

Uygulama geliştiricileri, çalışılacak proje için belirlenen çekirdeğin, bu operasyonlar için nasıl kullanılacağını bilmek zorundadırlar.

4.4.1. Görevlerin oluşturulması ve silinmesi

Geliştiricilerin uygulama tasarımı esnasında bilmek zorunda oldukları en temel operasyonlar oluşturma (creating) ve silmedir (deleting). Geliştiriciler kullanılan çekirdeğe bağlı olarak bir görevi oluşturmak için bir yada iki sistem çağrısı kullanırlar. Bazı çekirdeklerde, geliştiricilerin önce görevi oluşturmalarına daha sonra da başlatmalarına izin verilir. Bu durumda görev ilk önce yaratılır ve askıya alınır. Daha sonra görev başlatıldığında hazır durumuna getirilir.

Bu kullanım şekli görevin çalışmaya başlamadan önce ve oluşturulduktan sonra yapılması gereken özel kurma işlemlerini oluşturmada oldukça elverişlidir. Bununla beraber çoğu durumda tek bir sistem çağrısı içerisinde gerçekleştirilen oluşturma ve başlatma işlemi yeterlidir.

Askıya alma durumu, ne çalışıyor nede hazır durum olmamasından dolayı, engellenmiş duruma benzer. Bununla beraber bir görev, engelli duruma girmesini ve çıkmasını sağlayan operasyonlar vasıtası ile askıda olma durumuna girip çıkmaz. Gerçekte askıda olma durumu, işletim sistemlerine göre farklılıklar gösterir. Buradaki kullanımda henüz çalışmaya hazır olmayan görevler için bu durumun kullanıldığını bilmek yeterlidir.

Bir görevin başlatılması, onun hemen işletime alınması anlamına gelmez. Başlatma işlemi görevi hazır listesine ekler.

Çoğu çekirdek bunların yanı sıra kullanıcı tanımlı hook adı verilen servisler sunar. Hook spesifik çekirdek olaylarında, programcı tarafından tanımlanan fonksiyonların yürütülmesini sağlayan bir mekanizmadır. Programcı kendi özel fonksiyonunu, çekirdek tarafından sağlanan sistem çağrıları ile kayıt eder ve çekirdek bu fonksiyonu kendisi ile ilişkilendirilmiş bir olay tetiklendiğinde işletir. Bu olaylara örnek olarak;

- Bir görevin ilk yaratılma anı
- Herhangi bir nedenden dolayı bir görevin askıya alınması ve içerik anahtarlamanın oluşması
- Bir görevin silinmesi verilebilir.

Hook'lar, bir görevin yaratılması, silinmesi veya içerik anahtarlama anındaki olayların incelenmesi açısından oldukça faydalı hizmetlerdir.

Gömülü sistemlerde bir görevin silinmesi esnasında çok dikkatli olunmalıdır. Çoğu çekirdek, bir görev içerisinden bir başka görevin silinebilmesine olanak tanır. Silme işlemi esnasında çekirdek, görevi sonlandırır ve ona ait kontrol bloğunu ve yığını silerek o görevin kullandığı bellek alanını serbest bırakır.

Bununla beraber bir görev işletiliyorken, bir bellek isteğinde bulunmuş yada diğer sistem nesnelerini kullanarak bir kaynağa erişiyor olabilir. Eğer bir görev doğru olarak silinmez ise, bu görev kullandığı ve istekte bulunduğu kaynakları bırakamaz.

Örneğin; bir görevin paylaşılan bir veri yapısına erişirken sistemden bir semafor aldığını düşünelim. Görevin, bu veri yapısı üzerinde çalışıyorken silindiğini varsayalım. Eğer düzgün bir şekilde silme işlemi işletilmez ise sonuç aşağıdakiler gibi olabilir.

- Tamamlanmamış bir yazma operasyonundan dolayı veri yapısı bozulabilir.
- Bırakılmamış olan semafor nesnesini bekleyen diğer görevler, hiçbir zaman veri yapısına erişemeyebilir. Bu duruma ölümcül kilitlenme (deadlock) denir.

Sonuç olarak, bir görevin zamansız olarak silinmesi bellek yada kaynak kaçaklarına neden olabilir.

Bir bellek kaçağı, sistemden alınan belleğin tekrar geri verilmemesi durumunda oluşur ve er geç sistemde tüm belleğin tükenmesine neden olur. Bir kaynak kaçağı da sistemden alınan kaynağın tekrar geri verilmemesiyle meydana gelir. Bu durum da belleğin tükenmesine neden olur. Çünkü her bir kaynak isteği o kaynakla ilişki olan bellek isteğini de getirir. Çoğu işletim sistemi çekirdeği görev silinmelerine karşılık bir kilit mekanizması sunar. Bu mekanizma bir takım sistem çağrıları ile devreye sokulur ve kritik bir kod yürütme esnasında görevin silinmesini engeller.

Bu aşamada dikkat edilmesi gereken bir başka husus, bir görev silinirken o görevin tükettiği kaynak ve belleğin serbest bırakılması için yeterli temizleme süresinin de verilmesi gerektiğidir.

4.4.2. Görevlerin planlanması

Bir görevin oluşturulmasından silinmesine kadar olan sürede, görev yürütülmekte olan uygulamaya ve çekirdeğin durumuna göre çok çeşitli surumlar arasında geçiş yapabilir. Bu durumların çoğu otomatik olarak seçilse de çoğu işletim sistemi çekirdeği, görevleri bu farklı durumlara istedikleri anda geçirebilmelerini sağlayan sistem çağrıları da sunarlar. Bu durum manuel planlama (manual scheduling) olarak da adlandırılır.

Manuel planlamayı kullanarak geliştiriciler, bir uygulama içerisindeki görevleri durdurup tekrar kaldıkları yerden devam ettirebilirler. Bu durum hata yakalama amaçlı yada daha önceden bahsedildiği gibi düşük öncelikli bir görevin çalışması için yüksek öncelikli görevi durdurmak için kullanılabilir.

Bir uygulama geliştiricisi, bir görevi herhangi bir anda engellemek isteyebilir. Bir görevi belli bir süre durdurmak (delay), işlemcinin çalışma zamanının o göreve atanmasına ve görevin yürütülmesine neden olur. Süre dolduktan sonra görev aynı önceliğe sahip olan hazır listesinin sonuna eklenir.

Geliştirici, bir görevi yeniden başlatmak isteyebilir. Bu durum askıya almadan sonraki devam ettirme (resume) durumundan farklıdır. Yeniden başlatma, görevi işletilmeden önceki başlangıç kondisyonuna getirir. Görevin sahip olduğu içsel durumlar ve değişkenler yeniden başlatmadan sonra kaybolur.

Görevin önceliğinin okunup değiştirilebilmesi, geliştiricilerin yürütme esnasında, planlamanın işleyişine müdahale etmelerine izin verir.

Son olarak çekirdek, görevlerin çekişmesine müdahale eden bir mekanizma sunar. Bu mekanizma, uygulama içerisinden aktif veya pasif yapılarak görevlerin anahtarlanma işlemini durdurur. Bu özellik bir görevin kesinlikle anahtarlanmaması gereken kritik bir kodu yürütmesi esnasında kullanılır.

4.4.3. Görev bilgilerinin elde edilmesi

Çekirdek, geliştiricilere uygulamalarında görev ile ilgili bir takım bilgilere ulaşabilmelerini sağlayan sistem çağrıları da sunar. Bu çağrılar vasıtası ile, görevin tanıtıcı numarası, kontrol bloğu ve yığın kullanım miktarı gibi bir takım bilgilere ulaşılabilir.

4.5. Tipik Görev Yapısı

Bir görev için yürütme kodu yazılırken, görev yapısı iki şekilde olabilir;

- Tamamlanmak için çalışan
- Sonsuz döngü biçiminde olan

İki görev yapısı da oldukça basittir. Tamamlanmak için çalışan görevler, kurulum ve başlangıç işlemleri için oldukça kullanışlıdır. Bunlar tipik olarak sisteme enerji verildiğinde bir kere işletilirler. Sonsuz döngü şeklinde uygulanan görev yapıları ise, giriş/çıkış işlemleri ve uygulamanın ana işlevlerini içeren kod yapıları için uygundur. Bunlar sisteme enerji verildikten sonra sürekli çalışan görevlerdir.

4.5.1. Tamamlanmak için çalışan görevler

Bu yapıdaki görevlere örnek olarak uygulama seviyesindeki başlatma ve kurma görevleri verilebilir. Kurma görevi, uygulamanın ihtiyaç duyduğu ek servis ve çekirdek nesnelerini oluşturarak kullanılmalarını hazırlayan görevlerdir.

Tipik olarak uygulama başlatma görevi uygulama görevinden daha yüksek bir önceliğe sahiptir ve anahtarlanma işlemi gerçekleşmeden, tüm kurma ve başlatma prosedürlerini tamamlar. En basit durumda uygulamadaki diğer görevler daha düşük önceliğe sahip sonsuz döngü biçiminde gerçeklenirler. Uygulama başlatma görevi, çoğu durumda işleyişini bitirdikten sonra kendisini ya askıya alır yada siler.

4.5.2. Sonsuz döngü biçimindeki görevler

Uygulama başlatma görev yapısında olduğu gibi, sonsuz döngü biçimindeki görevler de başlatma fonksiyonlarını içerebilirler. Bu başlangıç kod parçacığının, görev başlatıldığında bir defa yürütülmesi gerekir.

Sonsuz döngü yapısındaki görevlerin tasarımı esnasında kritik nokta, döngü içerisinde bir yada daha fazla engelleme çağrısı yapılması gerektiğidir. Aksi halde sistemdeki daha düşük seviyeli görevlerin yürütülmesi söz konusu değildir.

4.6. Hatırlanması Gereken Noktalar

Bazı noktaları tekrar hatırlatacak olursak;

- Çoğu gerçek zamanlı işletim sistemi çekirdeği, geliştiricilerin gerçek zaman uygulamalarında ihtiyaç duydukları görev nesnelerini ve görev yönetim servislerini sağlar.
- Uygulamalar, her biri isme, tanıtıcı numaraya, öncelik seviyesine, görev kontrol bloğuna, yığına ve görev fonksiyonuna sahip sistem veya kullanıcı görevleri içerebilirler.
- Bir gerçek zamanlı uygulama, her biri işlemcinin yürütme zamanı için çekişen, yürütülebilir bağımsız birçok görevden oluşur.
- Görevler yaşamları boyunca üç ana durumdan birinde olurlar; hazır, çalışıyor ve engellenmiş.
- Öncelik tabanlı rekabetçi planlama algoritmasını kullanan çekirdekler, birden fazla görevin aynı önceliğe sahip olmasını, hazır görev listesinin yardımıyla izin verirler.
- Görevler sonsuz döngü biçiminde veya tamamlanacak biçimde oluşturulabilir. Sonsuz döngü biçiminde uygulanan görevlerde daha düşük önceliğe sahip görevlere izin vermek için çalışan görevin engellenmesi gerekir.
- Çekirdeğin uygulama geliştirmek için sunduğu temel operasyonlar; oluşturma ve silme, manuel planlama ve görev bilgilerinin dinamik olarak elde etme çağrılarını içerir.

5. İŞLETİM SİSTEMİ SEMAFORLARI

5.1. Giriş

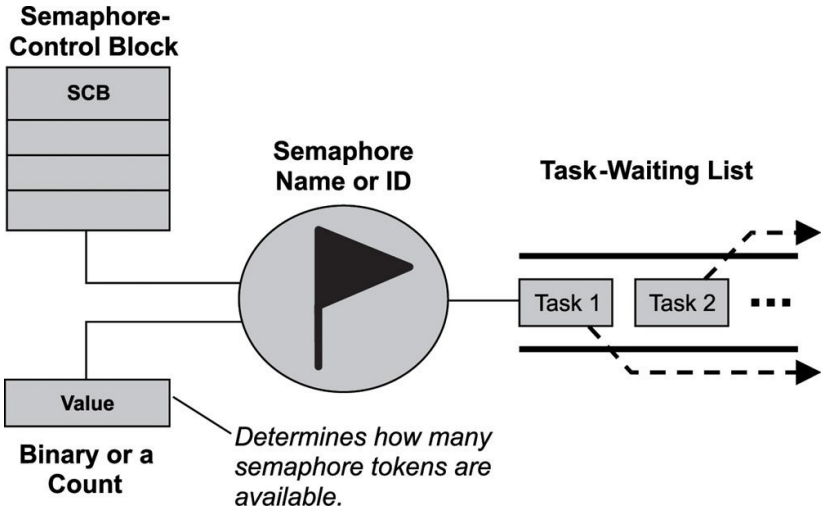
Bir uygulamada aynı anda yürütülmek için bulunan birden çok iş parçacığı paylaşılan bir kaynağa erişmek isterken, birbirleriyle senkronize olmak zorundadırlar. Bu ihtiyacı karşılamak için gerçek zamanlı işletim sistemi çekirdekleri semafor nesneleri ve bunlarla ilişkilendirilmiş semafor yönetim servisleri sunar. Bu bölümde aşağıdaki konuları incelemeye çalışacağız.

- Semaforun tanımı
- Tipik semafor operasyonları
- Kullanılan ortak semaforlar

5.2. Semaforun Tanımı

Bir semafor (bazen semafor jetonu olarak da adlandırılır) bir yada daha fazla iş parçacığının senkronizasyon amacı için, sistemden istekte bulunduğu veya bıraktığı bir işletim sistemi çekirdek nesnesidir.

Bir semafor ilk olarak yaratıldığında Şekil 5.1'deki gibi, çekirdek bu semafora bir semafor kontrol bloğu, tanıtıcı bir numara, bir değer (ikili veya bir sayaç) ve bekleyen görevler listesi atar.



Şekil 5.1: Bir semafor, ilişkili parametreleri ve veri yapısı

Bir semafor, bir göreve bazı operasyonları gerçekleştirmesine veya bir kaynağa erişmesine izin veren anahtar benzeri bir nesnedir. Tek bir semafor belli bir sayıda elde edilebilir. Örnek ile açıklanacak olursa; sistemden bir semaforu almak, bir apartman yöneticisinden apartmanın anahtarını almaya benzetilebilir. Anahtar isteğinde yönetici, her anahtarı isteyene anahtarı verirse, kopya anahtarlar oluşur. Eğer yönetici birden fazla kopyası olan anahtar istemiyorsa, bir başkasına anahtarı vermek için daha önce alan tarafından anahtarın geri getirilmesini bekler. Semaforda aynı bu anahtar örneğindeki gibidir.

Çekirdek semaforun yaratılması ile birlikte kurulan ve başlangıç durumuna getirilen bir sayaç değerini kullanarak, semaforun kaç kere alınıp bırakıldığını izler. Bir görev semaforu alırsa sayaç azaltılır, bıraktığı durumda da arttırılır.

Eğer sayaç değeri sıfıra ulaşırsa semafor hiçbir görev tarafından artık istenemez. Bu durumda bir görev semafordan istekte bulunursa, semaforu alamaz ve eğer semafor mevcut olana kadar bekleme seçeneği aktif ise engelli duruma geçer.

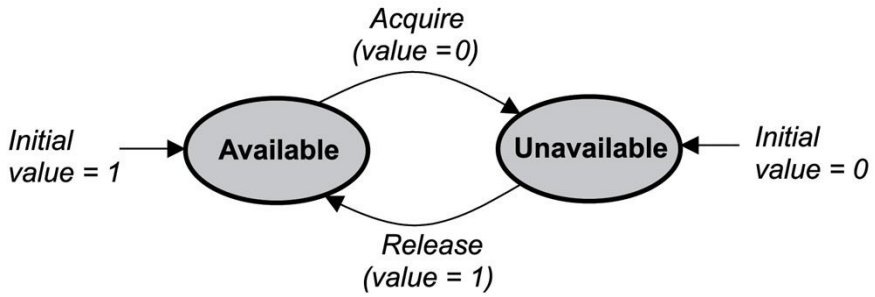
Semafordaki bekleyen görevler listesi, mevcut olmayan bir semafor için istekte bulunmuş ve engelli duruma geçirilmiş görevlerin listesini tutar. Bu liste içerisindeki bekleyen görevler için ya ilk giren ilk çıkar mantığı uygulanır yada ilk giren en yüksek önceliğe sahip görevin seçilmesine izin verilir.

Mevcut olmayan bir semafor, mevcut duruma geçtiğinde, çekirdek bekleyen görevler listesindeki ilk göreve semaforu alması için izin verir. Çekirdek, bu engelli durumu ortadan kaldırılan görevi, eğer en yüksek önceliğe sahipse çalışır duruma, aksi takdirde de hazır durumuna geçirir. Bekleyen görev listesinin gerçeklenmesi bir çekirdekten diğerine farklılık gösterir.

Bir çekirdek, ikili (binary), sayaç (counting) ve mutex (mutual exclusion) gibi farklı tipteki semaforları destekleyebilir.

5.2.1. İkili semafor

İkili semafor, bir yada sıfır değerine sahip olabilir. Bir ikili semaforun değeri sıfır ise semafor boş olarak varsayılır, değer bir ise bu ikili semafor dolu olarak kabul edilir. Bir ikili semafor oluşturulduğunda, değeri sıfır yada bir olarak ayarlanabilir. İkili semaforun durum diyagramı Şekil 5.2'de gösterilmektedir.

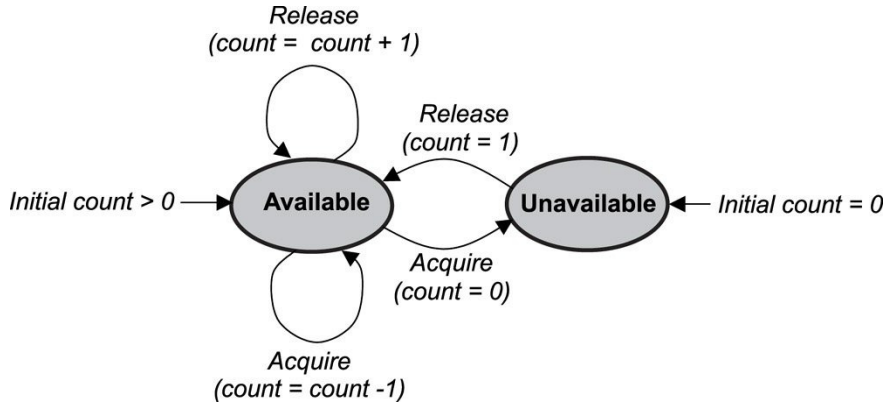


Şekil 5.2: İkili semaforun durum diyagramı

İkili semaforlar genel kaynaklar olarak kullanılır. Bunun anlamı bu semaforlara, onlara ihtiyaç duyan tüm görevler tarafından erişilebilir.

5.2.2. Sayaç semaforu

Sayaç semaforu birden fazla elde etme ve bırakmaya izin veren bir sayaç kullanır. Bir sayaç semaforu yaratıldığında bu sayaç değerine toplam erişim miktarı ayarlanır. Eğer oluşturulma anında sayaç sıfır değerine ayarlanırsa semafor mevcut değil demektir. Yaratıldığında sayaç sıfırdan büyükse semafor mevcuttur ve sayaç sıfır olana kadar kullanılabilir.



Şekil 5.3: Sayaç semaforunun durum diyagramı

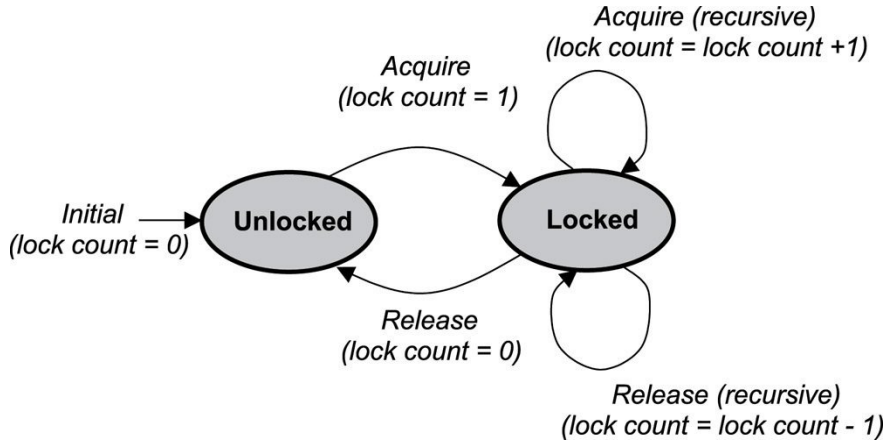
Bir yada daha fazla görev, semaforda alınacak hiçbir jeton kalmayana dek sayaç semaforundan elde etme isteğine devam eder. Tüm jetonlar alındığında yani sayaç sıfıra ulaştığında, sayaç semaforu mevcut durumdan mevcut olmayan duruma geçer. Semaforun tekrar mevcut duruma gelebilmesi için herhangi bir görev, almış olduğu jetonu bırakmak zorundadır.

İkili semafor gibi sayaç semaforu da, tüm kullanım alanında global olarak erişilir ve ihtiyaç duyan tüm görevler tarafından paylaşılır. Bu, özelikle herhangi bir görevin semafor jetonunu bırakabilmesine olanak tanır. Yani bir görev, herhangi bir semafordan bir jeton almamış olsa bile, bu görev tarafından gerçekleştirilen her bırakma işlemi, semaforun sayacını bir arttıracağından dolayı semaforda bir jeton daha bırakılmış olur.

Bazı sayaç semafor gerçeklemelerinde, sayaç değerinin sınırlandırılmasına izin verilir. Sınırlandırılmış sayaç, o semaforun maksimum jeton sayısını veren sayaç değeridir. Sınırlandırılmamış semaforda ise, semafor sayaç tipinin alabileceği maksimum sayı kadar jeton vardır (örneğin; unsigned integer'in alabileceği maksimum değer kadar jeton alınabilir).

5.2.3. Mutex semaforu

Mutex semaforu, sahipliğe, yinelemeli (rekürsif) erişime, güvenli bir biçimde görevlerin silinmesine izin veren özel bir ikili semafordur. Şekil 5.4, mutex semaforunun durum diyagramını göstermektedir.



Şekil 5.4: Mutex semaforunun durum diyagramı

İkili semafordaki dolu ve boş durumlarının tersi olarak, mutex semaforunun durumları kilitli (locked-1) ve kilitli değildir (unlocked-0). Bir mutex başlangıçta kilitli değildir ve bir görev tarafından alınabilir. Alındıktan sonra mutex kilitli durumuna geçer. Bunu tersi olarak bir görev mutex'i bıraktığında kilitli olmayan duruma geri döner.

Gerçeklemeye bağlı olarak bir mutex, ikili ve sayaç semaforlarında olmayan ek bazı özellikler içerebilir. Bu anahtar farklılıklar, sahiplik, yinelemeli kilitleme, güvenli görev silinmesi ve öncelik değiştirmeyi engelleyen protokoller gibi bazı özelikleri içerir.

5.2.3.1. Mutex sahipliği

Bir mutex'in sahipliği, bir görevin ilk o mutex'i almasıyla kazanılır. Bunun tersi olarak bir görev mutex'i bırakmak suretiyle onu kilitsiz duruma geçirir ve sahipliğini kaybeder. Bir görev bir mutex'e sahip olursa bir başka görevin o mutex'i kilitlemesi yada çözmesi mümkün değildir. Bu özellik, ikili semaforda olduğu gibi semaforu elde etmeyen bir görevin, o semaforu bırakabilmesi özelliğinin tam tersidir.

5.2.3.2. Yinelemeli kilitleme

Çoğu mutex gerçeklemeleri aynı zamanda, mutex'in sahibi olan görevin, mutex kilitli olmasına karşın defalarca o mutex'i tekrar kilitleyerek elde etmesine izin veren yinelemeli mutex yapısını destekler. Gerçeklemeye bağlı olarak, mutex'lerde yineleme işlemleri ya açılışta varsayılan olarak aktif yapılır yada her bir mutex için aktif/pasif yapılmasına izin verilir.

Yinelemeli kilitleme işlemine izin veren mutex'lere "yinelemeli mutex" adı verilir. Bu tip mutex'ler, bir kaynağa erişmek isteyen görev içerisinden çağrılan işlevlerin, görev ile aynı kaynağa erişmek istedikleri durumlarda oldukça kullanışlıdır.

Şekil 5.4'de görüldüğü gibi bir mutex ilk olarak kilitlendiğinde, çekirdek o görevi mutex'in sahibi olarak kayıt eder. Ardışık alma isteklerinde, çekirdek mutex'in sahibi olan görevin, kilitli olan mutex'ten kaç tane istek yaptığını izleyebilmek için o mutex ile ilişkilendirilmiş bir kilit sayacı tutar. Doğru bir şekilde mutex serbest bırakılmak istenirse elde edildiği sayı kadar bırakma işlemi yapılmalıdır.

Bu örnekte bir kilit sayacı, kilitlenmiş olan bir mutex'in yinelemeli olarak kaç defa daha durum değiştirdiğini izler. Burada mutex'in sahip olduğu sayma yeteneği, sayaç semaforundaki ile karıştırılmamalıdır. Mutex'te kullanılan sayaç, mutex'e sahip olan görevin kaç kez daha mutex'i alıp bıraktığını tutar. Semafordaki sayaç ise kaç tane görevin o semaforu alıp bıraktığını belirtir. Ayrıca, mutex'in sayacı, çoklu yinelemeli çağrıya izin verebilmek için sınırlandırılmamıştır.

5.2.3.3. Görevlerin güvenli bir şekilde silinmesi

Bazı mutex gerçeklemeleri, dahili olarak güvenli görev silinmesi için mekanizmalar içerir. Zamanından evvel olan görev silme işlemleri, o görevin mutex'i kilitleyip bırakması esnasında, görev silme kilidini kullanarak engellenir. Bu kabiliyetin aktif yapılması ile bir mutex kendisinin sahibi olan görevin silinmesini engeller. Tipik olarak, mutex'in oluşturulması anında uygun opsiyonların kurulması ile görevlerin zamanından önce silinmeleri engellenir.

5.2.3.4. Öncelik değiştirmenin engellenmesi

Çalışan görevlerin önceliklerinin değiştirilmesi genel olarak kötü tasarlanmış gerçek zamanlı gömülü sistemlerde yaşanır. Bu işleme, yüksek öncelikli bir görevin engellendikten sonra, daha düşük öncelikli bir görevin yürütülmesi esnasında engellenmiş olan görevin, yürütülmekte olan görevden daha düşük önceliğe sahip bir görev tarafından kullanılan kaynağı beklemesi durumunda ihtiyaç duyulur. Bu durumda engellenmiş olan en yüksek öncelikli görev sistemdeki en düşük öncelikli görev haline getirilir.

Bu öncelik değiştirme ihtiyacı tipik olarak mutex içerisine gömülen birtakım protokollerin aktif yapılması ile engellenir. Öncelik değiştirmeye engel olmak için genel olarak iki tip protokol kullanılır;

- Öncelik miras protokolü: mutex'i almış olan düşük öncelikli görevin öncelik seviyesi, mutex'den istekte bulunan yüksek öncelikli görevin öncelik derecesine yükseltilir. Önceliği yükseltilen görevin öncelik derecesi, görevin mutex'i bırakması ile birlikte eski haline getirilir.

- Önceliği yukarı yuvarlama protokolü: mutex'i alan görevin öncelik seviyesi, sistemde muhtemel olan en yüksek öncelik derecesine getirilir. Mutex bırakılınca görevin önceliği orijinal haline geri getirilir.

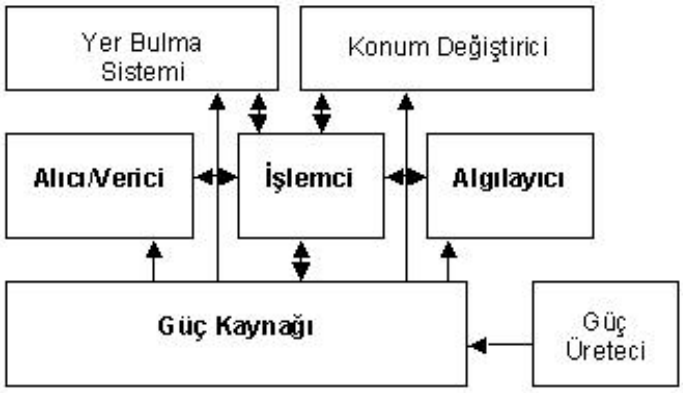
İşletim sistemi nesneleri tez kapsamında buraya kadar anlatılanlar ile sınırlandırılmıştır. Burada bahsedilmeyen ve her çekirdekte farklı kullanılabilen çok daha fazla işletim sistemi nesneleri olabilir. Tez kapsamındaki uygulama gerçeklemesinde, esas olarak görev ve semafor nesneleri kullanıldığı için diğer öğelere değinilmemiştir.

6. KABLOSUZ SENSÖR AĞLARINA GİRİŞ

6.1. Sensör Ağların Tanımı

Günümüze kadar farklı tiplerde ve büyüklüklerdeki sensörler, tetikleyici rolüyle elektronik sistemlerin bir parçası olarak kullanılmaktaydı. Mikro elektro-mekanik sistem (MEMS) ve telsiz iletişimi alanlarındaki teknolojik gelişmeler sonucu sensörler için farklı uygulama alanları doğdu; sensör ağlar. Askeri imkan ve kabiliyetlerin arttırılması ve muharebe alanında üstünlük sağlaması için halen üzerinde çalışmaların sürdürüldüğü sensör ağlar, geniş uygulama alanı olması sebebiyle sivil projelerde de kullanılmaktadır (Dunkels ve diğ., 2004a), (Sheth ve diğ., 2003).

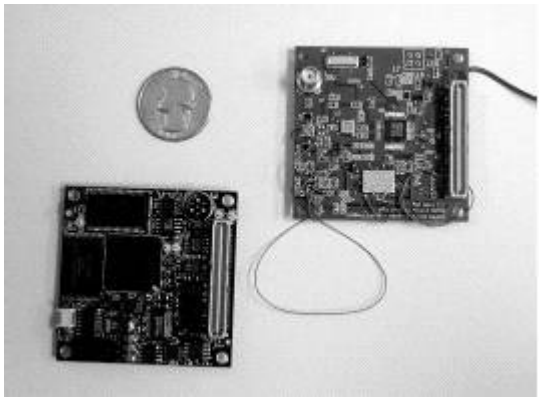
Sensör ağlarda görev yapan bir sensörün algılayıcı, işlemci, alıcı/verici ve güç birimleri olmak üzere dört ana elemanı vardır. Bunlara ilave olarak kullanım amacına göre bir sensör, yer bulma sistemi, güç üretim birimi, konum değiştirici bulundurabilir. Ana birimler başta olmak üzere tüm bu birimler bir kibrit kutusu büyüklüğünden, bozuk para boyutlarına düşürülmüş olup, bilgi sistemlerinin boyutları da donanım teknolojisindeki ilerlemelere paralel olarak daha da küçülmüştür.



Şekil 6.1: Sensör birimleri

Sensörlerin ana birimlerinden olan ve uygulamalara temel teşkil edecek çok çeşitli algılayıcı tipleri vardır. Bunlar;

- Sıcaklık ölçümü,
- Nem ölçümü,
- Hareket algılama,
- Aydınlık tespiti,
- Basınç ölçümü,
- Sismik değer ölçümü,
- Görüntü tespiti,
- Gürültü algılama/ölçümü,
- Canlı/cansız varlık tespiti,
- Mekanik gerginlik algılama/ölçümü,
- Hız, yön, miktar tespiti/ölçümü.



Şekil 6.2: Çeşitli boyuttaki sensörler

Sensör ağlar, uygulamaya bağlı olarak uygulama sahasında konumlandırılmasına (elle konumlarına yerleştirilmesi, uçaktan atılması gibi) müteakiben, sensörlerin birbiri ile iletişim kurması ile oluşmaya başlar. Donanım ve iletişim gücü itibariyle güçlendirilmiş sensörler, koordinatör (sink) etrafında dizayn aşamasında belirlenen protokoller çerçevesinde tamamen kendi kendilerine kısa sürede organize olurlar. Algılayıcıları vasıtasıyla tespit ettikleri veriyi koordinatöre birbirleri üzerinden ulaştırırlar. Koordinatör kendisine ulaşan veriyi kullanıcıya erişim noktalarından (uydu, sabit/hareketli aktarıcı) ya da direk olarak ulaştırır. Verinin iletimi sırasında internet, intranet gibi ağ erişimleri de kullanılabilir.

Sensör ağları geleneksel telsiz ağlardan ayıran genel özellikler şunlardır;

- Sensör ağlarındaki sensör sayısı geleneksel telsiz ağlardaki bilgisayar sayısından çok daha fazla olabilmektedir,
- Sensör uygulama sahasında sensörlerin yoğunluğu fazladır,
- Gerek donanımlarının minyatüre edilmiş olduğundan gerekse de atıldıkları saha özelliğinden, bazılarının çalışmama/çalışamama ihtimalleri vardır,
- Donanım özellikleri kısıtlıdır (sınırlı batarya, işlemci, bellek),
- Adrese dayanan statik bir topolojileri yoktur,
- Her birinin başında kullanıcısı yoktur, uygulama sahasına bırakıldıktan sonra kendi kendilerine organize olmak zorundadırlar.

Sensör ağların dizaynını etkileyen unsurlar şunlardır: Hata toleransı, ölçeklenebilirlik, maliyet, uygulama sahası, ağ topolojisi, donanım kısıtlamaları, iletişim ortamı kuralları ve güç tüketimi. Bunlardan araştırmalara konu olan en önemli unsur güç tüketimidir. Depolanabilen güç miktarı sensörün hayatta kalabilme süresini, dolayısıyla ağ ömrünü belirlediğinden, tüm ağ katmanlarında efektif güç tüketimini temel alan çalışmalar halen sürmektedir.

6.2. Sensör Ağların Uygulama Alanları

Sensör ağların uygulama alanları, algılayıcı tiplerinin genişliği oranında çeşitlendirilebilmekle beraber, uygulamalar aşağıdaki gibi başlıklar altında toplanabilir.

Çevresel uygulamalar;

- Orman yangını, sel, deprem, gibi doğal afetlerin ölçümlendirilmiş olarak hızlı bir şekilde ihbar edilmesinde,
- Hava kirliliği tespiti ve ayrıntılı rapor alınmasında,
- Doğal yaşamın gözlenmesinde.

Sağlık uygulamaları;

- İnsanların fizyolojik verilerinin uzaktan izlenmesi,
- Hastanede bulunan doktorların yerinin ve hastaların durumunun (kalp atışı, kan basıncı vb.) izlenmesi,
- Hastanedeki ilaç dağıtımının yönetimi.

Ticari uygulamalar;

- Küçük çocukların konumlarının aileleri tarafından takip edilmesi,
- Güvenlik ihtiyaçları, hırsızlarının tespiti,
- Envanter yönetim yardımcı aracı,
- Araçların izlenmesi ve tespit edilmesi.

Askeri uygulamalar;

- Dost kuvvetlerin teçhizat ve cephanesinin izlenmesi,
- Savaş alanının gözlenmesi,
- Arazi hakkında keşifte bulunma,
- Hedefin konumu, sürati gibi hedef bilgilerinin tespiti,
- Düşmana verilen hasar miktarının tespit edilmesi,
- Nükleer, biyolojik ve kimyasal (NBC) saldırıları ihbarının alınması ya da keşfi.

Sensör ağ kavramı, mikro elektro-mekanik alanlarındaki ilerlemelere paralel olarak geliştirilmektedir. Geleneksel ağlardan farklı olarak sensör sayısının çok fazla olabilmesi, iletişim kısıtlamaları, sensör ağların kendi kendine organize olması, uzaktan erişime olanak tanıması, mimarisinin dinamiklerinden kaynaklanan (sınırlı enerjisi, kısıtlı belleği, farklı işlemci yapısı) nedenler ile sıcaklığını sürdüren araştırma konularından biri olmuştur.

Sensör ağların insan hayatına getireceği kolaylıklar nedeniyle, hayatımızın bir parçası olan Internet gibi yakın gelecekte etkileri yaşamımızda hissedilmeye başlanacaktır. Bu açıdan akademik çalışmalar desteklenmeli, askeri ve ticari alanlarda da hazırlıklar yapılmalıdır.

6.3. Sensör Ağlarında Yeni Bir Standart - ZigBee

ZigBee, genel bir açık kaynaklı standarda dayalı, güvenilir, ekonomik, düşük güç tüketimine sahip, kablosuz ağ bağlantısına olanak sağlayan izleme ve kontrol ürünlerinin geliştirilmesi için oluşturulmuş, birlikte çalışan bir şirketler topluluğu olan "ZigBee Ortaklığı" tarafından geliştirilmiş bir standarttır (ZigBee Alliance, 2006).

ZigBee Ortaklığının amacı üstün bir esnekliğe, hareketliliğe, kablosuz bir zeka ve yeteneğe sahip binaların ve sistemlerin günlük hayatta kolaylıkla kullanıma sunulmasına hizmet sağlamaktır.

ZigBee teknolojisi kısa bir süre içerisinde, tüketici ürünlerinden ticari ve endüstriyel ürünlere kadar çok geniş bir alanda kullanılmaya başlanacaktır. İlk olarak şirketler kolay, güvenilir, düşük maliyetli ve düşük güç tüketimine sahip kontrol ve uzaktan izleme ihtiyacını karşılayabilecek standart temelli kablosuz bir platforma sahip olacaklardır. ZigBee haberleşme standardı, birbirleriyle haberleşme ihtiyacına sahip çok çeşitli kablosuz ev ve bina otomasyon uygulamaları için esas teşkil edecektir.

6.3.1. Neden ZigBee?

Bluetooth ve Wi-Fi gibi diğer kısa mesafeli kablosuz haberleşme standartları, düşük güçlü izleme ve kontrol uygulamaları için tasarlanmamıştır. ZigBee ile bu durum değişmiştir. Kablosuz kontrol ve izleme uygulamaları için temel ihtiyacı karşılayabilecek özelliğe sahip ZigBee'nin kullanılması ile;

- Büyük hacimli sensör noktalarının gerçekleştirilmesi
- Çok düşük sistem/nokta maliyeti
- Ucuz bataryalar ile yıllar süren çalışma imkanı
- Sensör ağları arasında güvenilir ve sağlam bağlantı imkanı
- Kolay konfigürasyon ve dağıtım
- Genel amaçlı çözümler sağlanabilir.

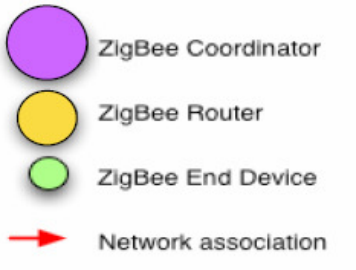
Son kullanıcı ürünleri sağlayan firmalar için standart ZigBee teknolojisi kullanarak uygulamalar geliştirmek, yeniden büyük çaba harcayarak bir tasarım yapmaktan çok daha ucuza mal olur. Aynı zamanda ZigBee sayesinde bu firmalar radyo çip üreticilerine bağımlı olmadan genel amaçlı çözümlerini kolaylıkla üretebilir olurlar. Kablolu alan ağı teknolojilerinin ZigBee'ye geçirilmesi, kurulum maliyetinin müşterilere aksettirilmesine neden olmaz ve bunun yanında ürünlerde kullanılan eleman sayılarını azaltarak ZigBee uyumlu çözümlerin daha ekonomik olmasına sebep olur.

6.3.2. ZigBee ağ topolojileri

Genel olarak ZigBee ağ topolojisinde üç tip cihaz bulunur;

- Tam fonksiyonlu aygıt (Full Function Device)
- Azaltılmış fonksiyonlu aygıt (Reduced Function Device)
- Koordinatör

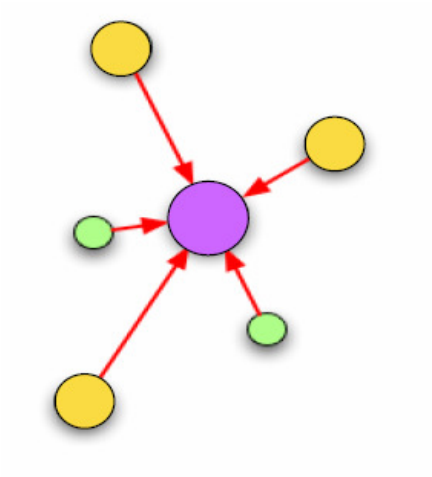
ZigBee genel olarak Star, Clustered Tree ve Mesh ağ topolojilerini destekler. Bu topolojiler ZigBee standartlarında belirtilirler. Şekil 6.3'de, belirtilen ağ topolojilerinde kullanılan ZigBee aygıtlarının sembolik olarak işlevleri gösterilmektedir.



Şekil 6.3: ZigBee ağ bileşenleri

6.3.2.1. Star ağ topolojisi

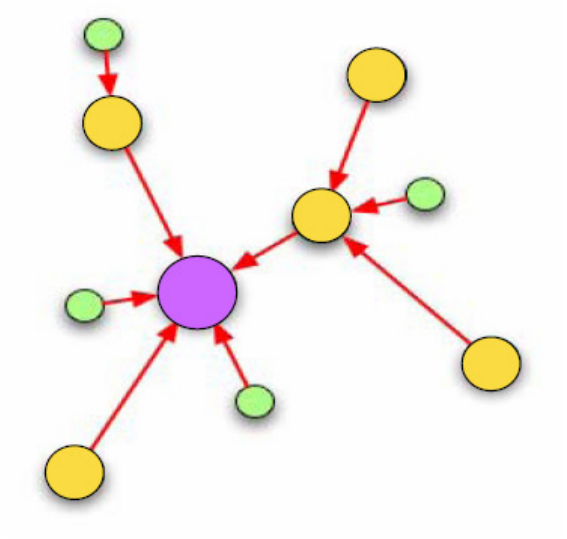
ZigBee Star ağı, bir ZigBee koordinatör ve bir veya birden fazla son aygıttan meydana gelir. Son nokta aygıtları birbirleriyle ZigBee koordinatör vasıtası ile haberleşirler. Burada koordinatör mesajların kimlere gönderileceğine karar veren yapı olarak düşünülebilir.



Şekil 6.4: Star ağ yapısı

6.3.2.2. Clustered-tree ağ topolojisi

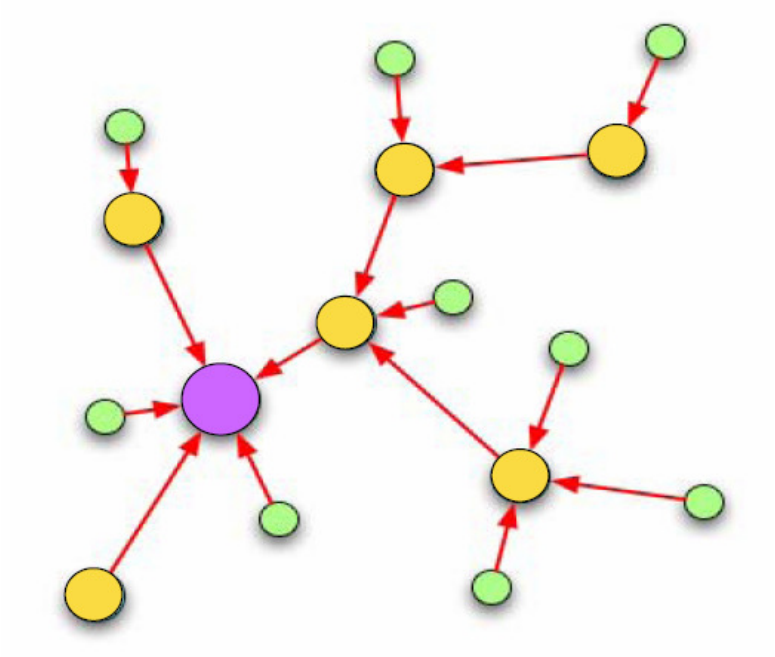
Bu ağda, bir ZigBee koordinatör ve birbirlerine bağlı bir ve birden fazla Star ağ konfigürasyonu bulunur. ZigBee yönlendiriciler (routers) ağ kapasitesini (range) son nokta aygıtlarının, merkezi koordinatöre bağlanmalarına gerek kalmadan birbirlerine bağlanmalarına izin verecek şekilde genişletir. Her bir aygıt kendi sahibi (parent) ve üyesi (child) ile doğrudan bağlantı kurabilir. Aygıtlar arasındaki mesajlar ağ vasıtası ile bir ağaç yapısında yönlendirilir.



Şekil 6.5: Clustered Tree ağ yapısı

6.3.2.3. Mesh ağ topolojisi

Mesh ağı Clustered Tree ağına çok benzer. Farklı olarak ZigBee yönlendiriciler, mesajları ağaç yapısını izleyerek koordinatör üzerinden iletmek yerine doğrudan diğer yönlendiricilere iletir. Bu topolojinin avantajı, mesaj gecikmesinin azalıp güvenilirliğin artmasıdır. Dezavantajı ise gerçekleştirme esnasında çok daha fazla program ve veri belleğine ihtiyaç duymasıdır.



Şekil 6.6: Mesh ağ yapısı

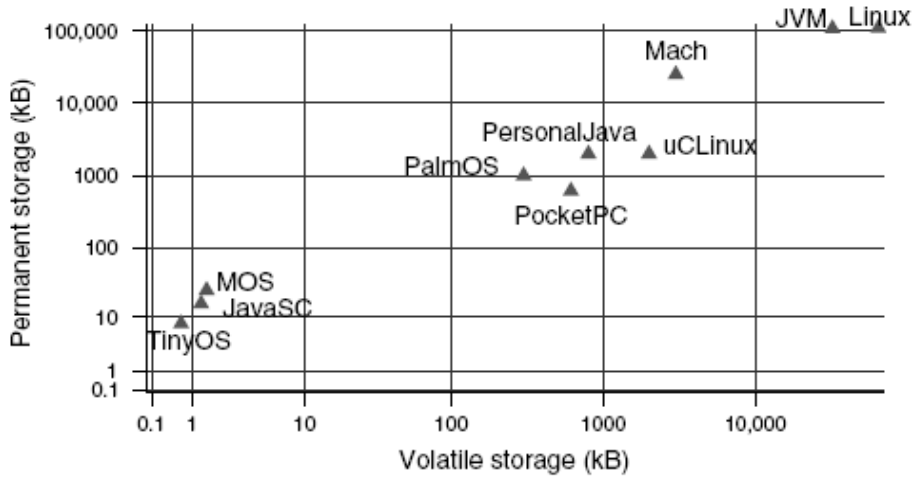
6.4. Sensör Ağları İçin Gerçek Zamanlı İşletim Sistemi

6.4.1. Giriş

Sensör ağları için bir işletim sistemi konusunda akla gelebilecek ilk soru "neden bir sensör düğümü bir işletim sistemine ihtiyaç duysun?" olabilir. Şüphesiz ki, bir işletim sistemine ihtiyaç duyulmadan kullanılan çok fazla gömülü sistem cihazı ile karşılaşabiliriz. Dijital kamera yada mikrodalga gibi özel bir uygulama için uyarlanan aygıtlarda, uygulama kodunu doğrudan mikro denetleyici üzerinde çalıştırmak çok daha kolay olabilir. Diğer taraftan PDA (Personal Digital Assistant)'ler gibi daha genel amaçlı aygıtlarda tam özellikli bir işletim sistemini kullanmak (örneğin;MS Mobile PocketPC OS, Embedded Linux gibi), çoklu uygulamalar için kullanılabilecek ortak temel servisler sağlar.

Sensör düğümleri, bir işletim sistemine ihtiyaç duymayan tekil bir uygulamada veya daha kapsamlı geleneksel bir işletim sistemini kullanan genel amaçlı bir uygulamada kullanılabilmektedir. Bu durum, uygulama geliştiricileri için sınırlı ortak servis sağlayan, geleneksel anlamdaki işletim sisteminden farklılıklar gösteren sensör ağ işletim sisteminin tasarlanmasını ortaya çıkarmıştır. Bir sensör ağ işletim sistemi, uygulama geliştiricileri için kısıtlı sayıda ortak servisler sağlar. Bu ortak servisler tipik olarak sensörlerin donanım yönetimi, radyo haberleşme kısmını, G/Ç yolunu ve harici çevre elemanlarının yönetimini içerir. Uygulama tarafından ihtiyaç duyulan diğer servisler, görev koordinasyonu (task coordination), güç yönetimi (power management), kaynak kısıtlamalarının adaptasyonu ve ağ haberleşmesidir.

Şekil 6.7'de hedeflenen sensör ağ işletim sistemleri için tasarım uzayı gösterilmektedir. Bir işletim sisteminin ihtiyaç duyduğu minimum gereksinimler çalışma zamanı yürütme fonksiyonları ve kalıcı saklama için RAM ve ROM dur. Bu ihtiyaç Şekil 6.7'de gösterilmektedir.



Şekil 6.7: İşletim sistemleri için kalıcı ve kalıcı olamayan bellek ihtiyaçları

6.4.2. Sensör düğüm donanımları

Bir gömülü mikro sensör düğümü donanımı için işletim sistemi geliştirirken, seçilen donanımın sistem tasarımının çoğu safhasına doğrudan etkisi vardır. Kablosuz mikro sensör düğümünün donanımı tipik olarak beş ana alt sistemden oluşur.

- Mikro denetleyici
- Sensörler
- Radyo haberleşme kısmı
- Güç ünitesi
- Kalıcı hafıza

Burada işletim sistemine en büyük etkiyi sağlayan mikro denetleyici ve diğer aygıtları inceleyeceğiz.

6.4.2.1. FLASH, SRAM ve EEPROM

Günümüzde çoğu düşük güçlü mikro denetleyici, işlem ünitesinin veri yolu ve program hafızasına erişme yolunun birbirinden ayrı olduğu Harvard mimarisini kullanılır.

Zaman ve enerji bakımından, flaş bellek okuma esnasında ucuz, yazma esnasında ise pahalı bir maliyete sahiptir. Bu bakımdan program belleği olarak uygundur. Diğer taraftan mikro denetleyicinin kendi flaş belliğine uygulama programı tarafından yazma işlemi mümkün olabilir de olmayabilir de. Bir başka hususta, flaş bellek blok tabanlı yazmaya uygundur. Fakat rasgele olarak okunabilir. Belirlenen bloğa yazma sayısı da sınırlıdır. Flaş bellek uçucu olmayan bir hafıza elemanı olduğu için enerji gitse dahi içeriği değişmez.

SRAM, rasgele erişimli ve genellikle okuma ve yazma işlemi için çok hızlı fakat uçucu, enerji tüketimi flaş bellekten daha az ve daha pahalı bir malzemedir. SRAM'in flaş bellek gibi bir yazma/okuma ömrü yoktur. Bu yüzden SRAM flaşa göre daha küçük tutulur ve veri hafızası olarak kullanılır. Oldukça kısıtlı olan veri hafızası, sensör ağ programlamasındaki en önemli kısıtlamalardan bir tanesidir.

Ek uçucu olmayan bellek ihtiyacı, harici bir flaş eleman kullanılarak veya diğer bellek teknolojileri (EEPROM) kullanılarak giderilebilir. Bir işletim sistemi perspektifinden EEPROM, flaş gibi davranır. Uçucu değildir, sınırlı yazma ömrüne sahiptir, ve yazma süresi yavaştır. Flaş ve EEPROM seri veya paralel arabirimli olarak yüksek kapasitede olabilirler. Seri olanları sensör düğümlerinde disk gibi kullanılabilirler.

6.4.2.2. Çevresel aygıtlar

Modern mikro denetleyiciler bir takım çevresel aygıtlar ile gelirler ve tüm bunlar aynı çip içerisinde, bellek ve CPU ile birliktedir. Radyo ve sensörler gibi çevresel aygıtlar bu arabirimlerin bir veya daha fazlası aracılığı ile mikro denetleyiciye bağlanırlar. Bellek-haritalı çevre birimleri nadiren kullanılır.

Ortak arabirimler UART ve I2C, ADC, DAC ve bunun gibi çevre bileşenleri olabilir. Her bir çevre biriminin; tipi, maksimum hızı, yol başına kullanılan aygıt sayısı ve sinyal protokolleri gibi tekil karakteristiklere sahiptir. Bununla beraber çoğu ayrıntı, arabirimler donanım kısmında gerçeklendiği için işletim sistemi tasarımcısı tarafından gizlenir.

6.4.2.3. Radyo kısmı

Radyo frekans aygıtlar sensör işletim sistemi performansına derin bir etki edebilirde etmeyebilirde. Bu durum RF protokollerinden kaç tanesinin donanım tarafından ele alınacağına bağlıdır. Örneğin; en çok bilinen Mica2 sensör düğümü, ham bit düzeyli arabirim sunan CC1000 radyo çipini kullanır. Bunun yanında karmaşık yazılımlarım işletilmesi esnasında, CC1000'in çalışma anında oluşan gürültü bitleri sürekli olarak sisteme kesme isteğinde bulunur. Bu durum radyo kısmının enerji tüketimi arttırmakla beraber mevcut CPU bant genişliğini azaltır. Bunun tersi olarak en son ortaya çıkartılan MicaZ düğümü paket tabanlı, donanım tarafından gönderilen tüm paketi işleyen yapıya sahip CC2420 radyo çipini kullanır. Bu durum yazılım karmaşıklığını oldukça azaltmakla birlikte radyo kısmının gönderme ve alım anında da CPU' nun düşük güç modunda olmasını sağlar.

6.4.2.4. Sensörler

Sensörlerin bir grup olarak karakterize edilmeleri zordur. Çünkü çok geniş bir yelpazeye sahiptirler. Sensörler dijital yada analog olabilirler. CPU tarafından kontrol edilmeye ihtiyaç duyabilir veya duymaya bilirler. Geniş bir sensör sınıfını destekleyebilmek için bir işletim sistemi modüler ve esnek tasarlanmak zorundadır.

6.4.2.5. Güç sistemleri

Sensör düğümleri tipik olarak pille çalışırlar. Bu durum işletim sistemi tasarımı esnasında efektif güç yönetimi için dikkat edilmesi gereken önemli bir noktadır. Dış ortamlarda sürekli olarak enerjinin sağlandığı yöntemler kullanıldığında dahi örneğin; güneş panelleri gibi, işletim sistemi için güç tüketimini minimum yapmak göz ardı edilmemesi gereken bir durumdur.

6.4.3. Bilgisayar sistemleri ile farkı

Ortama dağıtılan sensör düğümleri tipik olarak ekranlar gibi çıkış aygıtlarını desteklemezler. Kullanıcı etkileşimi, PC ve PDA deki gibi başlıca bir ihtiyaç değildir. Düğümler yaşamlarının neredeyse tamamı boyunca fiziksel olarak bir daha dikkate alınmayacakları ortamlara bırakılırlar. Benzer olarak bu sensör düğümleri klavye ve fare gibi tipik giriş aygıtlarını da desteklemezler.

6.4.4. Sensör işletim sisteminin tasarım prensipleri

Modern sensör işletim sistemleri geleneksel PC işletim sistemlerinden farklılıklar gösterir. Bu farklar donanım ve enerji kısıtlamalarının tipik olarak sensör ağlarında dikkate alınmasından kaynaklanır.

6.4.4.1. Donanım yönetimi

Herhangi bir işletim sisteminin ilk görevi, aygıt üzerindeki mevcut donanım kaynaklarının yönetilmesidir. Sensör işletim sistemleri de bu durumu göz ardı etmezler. İşletim sistemi sensörü okumak, radyo üzerinden veri göndermek ve almak, zamanlayıcıları kullanmak gibi soyut servisler sağlar.

Tipik mikro denetleyicilerde ana donanım kısıtlamalarından bir tanesi bellek yönetim ünitesinin zayıflığıdır. Ek olarak çoğu sensör düğümü denetleyicisi tekil bir işletim sistemi moduna sahiptir. Oysaki tipik işlemciler kullanıcı ve denetleyici modlarına sahiptir. Bu durum yürütme çekirdek kodu ve yürütme uygulama kodu arasındaki farkı elimine eder. Bu yüzden donanım yönetimi kütüphane fonksiyonu şeklinde gerçeklenebilir. Bu fonksiyonlar donanıma açık ve soyut bir arabirim sunarken, kazara veya kötü niyetli girişimler sonucunda donanıma doğrudan erişimi koruyamazlar.

6.4.4.2. Görev koordinasyonu

İşletim sistemi tarafından çözülmesi gereken diğer bir ana problem, çoklu görevlerin koordinasyonudur. Bu durum iki ana alt problemden oluşur; görev listesi yönetimi ve senkronizasyon. İşletim sistemi işlemciyi hangi göreve ne zaman atayacağını belirlemek ve kullanıcı programlarının güvenilir bir şekilde yürütülmesi için gerekli olan mekanizmaların sağlanmasını garanti etmek zorundadır.

Bazı sistemler bu iki problemi, kullanıcıyı tek bir görevde kısıtlayarak çözer. Bu durum uygulama kodunun doğrudan mikro denetleyici üzerinde çalışmasıdır. Bazı modellerde çoklu mantıksal görevler bulunabilir. Fakat bu görevlerin koordinasyonu işletim sisteminin standart yöntemleri ile yönetilmekten ziyade, uygulama programcısı tarafından uygulama programı ile ele alınır. TinyOS gibi bazı sensör işletim sistemleri donanım yönetimi problemini çözerken görev koordinasyon problemi ile ilgilenmez.

Görev yönetimi ile ilgili ilişkilendirilmiş iki maliyet vardır; CPU bant genişliğinin kısıtlı olması ve belleğin önemli miktarda az olması. Bellek maliyeti yüksektir çünkü çoklu görevlerin her biri kendi statik bellek ihtiyacına ve yürütme yığınına (stack) aynı anda ihtiyaç duyar. Bu duruma çözüm olarak, durdurulmuş görevlerin belleğinin flaş bellek üzerinden, çalışacak olan göreve atanması (swap) ön görülebilir. Fakat bu durum içerik anahtarlama (context switching) süresini oldukça arttırarak CPU'nun görev yönetim sistemi için çok fazla yorulmasına neden olur.

Görev koordinasyonu uygulama programından işletim sistemine oldukça önemli karmaşıklığa neden olur. Bu durum uygulamanın çok basit olduğu hallerde bir dezavantaja dönüşebilir.

6.4.4.3. Kaynak kısıtlamaları

Sensör düğümleri PC'lerde yada diğer küçük gömülü aygıtlardan olan PDA'lerde göz ardı edilmeyen kaynak kısıtlamalarını irdelemek zorundadır. Örneğin; Mica2 düğümü sadece 4 KB veri belleğine 128 KB program belleğine sahip, 8 bitlik, 7.3 MHz. kristal frekansında çalışan sahip bir CPU 'ya sahiptir. Bu sınırlamaların hepsi sensör işletim sisteminin tasarımına etki eder.

6.4.4.4. Veri belleği

Veri belleği oldukça kıt bir kaynaktır. Özellikle çoklu göreve sahip sistemlerde, her bir göreve ait yığın aynı anda sistemde tutulmak zorundadır. Fiziksel belleğin küçük olmasının yanı sıra, sensör düğümleri donanım bellek yönetim ünitesini zayıflığı nedeni ile kısıtlanır. Burada sıfır kopya ağ yığını (zero-copy network stack), düşük özellikli görev yönetimi (lightware schduling) ve derleme zamanı optimizasyonu (compile-time optimization) gibi bellek tüketimini azaltan yöntemler vardır. Bu yöntemlerin bazıları işletim sisteminin sağlamış olduğu servisleri kısıtlar.

Donanım bellek yönetiminin zayıflığı yazılım mühendisliğinin, sensör düğümlerinde kritik derecede önem arz etmesine neden olur. Normalde bir görevin diğer bir görevin bellek alanının üzerine yazmayı engel olan bir yol yoktur. Çoklu görevin olmadığı durumlarda bir uygulama işletim sisteminin durum bilgilerinin tutulduğu yada bellek haritalı kayıtçıların bulunduğu alana yazma isteğinde de bulunabilir.

6.4.4.5. Program belleği

Program belleği flaş belleğin daha ucuz olması nedeniyle veri belleği kadar ciddi bir kısıtlamaya sahip değildir. Tüm program kodunu burada saklamak genellikle bir problem değildir. Flaş belleğe istenmeyen bir yazma olayının gerçekleşmesi çok zor yada hemen hemen imkansızdır. Program belleği ile ilgili istenmeyen tek kısıtlama flaş belleğin limitli bir yazma-silme ömrüne sahip olmasıdır. Program belleği çok sıklıkla değişmediği için bu durum göz önüne alınacak bir problem değildir.

6.4.4.6. CPU bant genişliği

Sensör düğümlerinde görülen önemli görevlerden ilk bakışta dikkati çeken, giriş çıkış sınırlamasıdır. İkincisi ise sensör olayları için bekleme ve veri alma/gönderme olayıdır. Bununla beraber çoğu araştırma konusu CPU bağımlı, veri işleme ve transformasyon, ağ işleme ve kriptografi konuları olarak karşımıza çıkar. 8-bitlik mikro denetleyicilerde bazı şifreleme yöntemleri çok uzun zaman alır. Bölüm 6.4.1 de belirtildiği gibi sensör işletim sistemi, görevler arasındaki CPU işlem zamanının atamasının yönetimini gerçekleştirebilir de gerçekleştiremeyebilir de.

6.4.4.7. Ağ yönetimi

"Sensör ağ" teriminde belirtildiği gibi, bir sensör düğümünde ağ yapısı çalışan ana uygulamadır. Aktüel uygulama yazılımı, ağ yapısına göre daha basit ve küçültülmüştür. Ağ yapısının bellek tüketimi önemli bir konudur. Tipik sensör ağ paketi sadece 32 byte olabilir. 32 byte mevcut bellek yanında önemi olmayan bir miktardır. Bellek oldukça limitli ve de ağ ana bellek tüketicisi olduğu için, tampon kopyalamaktan sakınmak için çapraz katman arabirimi tanımlamak önemli bir noktadır.

6.4.4.8. Algılama

"Sensör ağ" isminden de anlaşılacağı gibi algılama, sensör işletim sistemi tarafından desteklenmesi gereken anahtar ihtiyaçtır. Basit analog-sayısal çeviricilerden karmaşık GPS alıcılarına kadar çok değişken olabilen giriş sensörleri desteklenmelidir. Ortamda sensörlerin kalibrasyonları en zor çözülebilen problemlerden biridir. Verilen bir sensör düğümünde çoklu algılamayı sağlamak için çoklu sensörler bulundurabilir. Örneğin; sıcaklık, basınç ve rölatif nem gibi. Bazı algılama davranışları periyodik olabilir. Örneğin; düğüm her T saniyede bir uyanarak istenilen ölçümü yapabilir. Sensör işletim sistemi mümkün olan temel tüm periyodik algılama işlemlerini desteklemelidir. Diğer tip algılama davranışları, daha uyarlanabilir yapıda olabilir; örneğin hedef takip gibi.

7. ÖRNEK UYGULAMA

7.1. Gerçek Zamanlı İşletim Sistemi Çekirdeği

Bu tezin kapsamında, önceki bölümlerde ifade edilmeye çalışılan bilgiler ışığında bir gerçek zamanlı işletim sistemi çekirdeği oluşturulmuştur. Bu çekirdek çoklu görev desteğine sahip olup, mümkün olduğunca standart özellikleri destekleyecek şekilde tasarlanmıştır.

Hedef olarak seçilen işlemci ilesi ARM7 ve ARM9 olmakla birlikte, sensör düğümlerinde de kullanabilmesi için burada anlatılan özelliklerin sınırlandırılması koşulu ile Renesas M16C ailesinde de başarıyla uygulanmıştır. Tasarlanan işletim sisteminin test ve geliştirme aşamalarında, STMicro firmasının STR912FAW44 kodlu ARM9 ailesine ait, tek çip (single chip) çözümlerde kullanılan mikro işlemcinin geliştirme seti kullanılmıştır. Sensör düğümü olarak ise BKS Ltd.Şti. tarafından geliştirilen, radyo çipi olarak Texas Ins. TRF6901, mikro işlemci olarak ta Renesas 30624FGFP kodlu M16C ailesinden bir üyenin kullanıldığı donanım üniteleri kullanılmıştır.

STR912FAW44'nin temel özellikleri;

- 16/32-bit 96 MHz ARM9E tabanlı Mikro denetleyici
  ARM966E-S RISC koru: Harvard mimarisi
- 32-bit genişliğinde Burst Flaş bellek
  512KB Ana Flaş, 32KB ikincil Flaş
- 32-bit genişliğinde SRAM
  96K pil desteği opsiyonlu
- 9 kanal programlanabilir DMA
  Bir tanesi Eternet, geri kalan 8 tanesi genel amaçlı
- Vektör kesme yöneticisi (VIC)
  32 IRQ vektör, 30 tane kesme isteğinden biri FIQ olabilir

- 8-kanal, 10-bit A/D çevirici (ADC)

  0 - 3.6V aralığında, 0.7 usec çevirim süresi
- 11 Haberleşme arabirimi

  10/100 Ethernet MAC, DMA desteği ve MII portu ile

  USB Full-speed (12 Mbps) ikincil aygıt

  CAN arabirimi (2.0B Aktif)

  3 16550-tarzı IrDA protokolü destekli UART

  2 hızlı I2C™, 400 kHz

  2 kanal SPI™ veya Microwire™ için SSI™

  8/16-bit EMI bus
- 80 I/O pini (arabirimler ile paylaşılır)

  5 V toleranslı, 16'sı yüksek sink akımına sahip(8 mA)
- 16-bit standart zamanlayıcı (TIM)

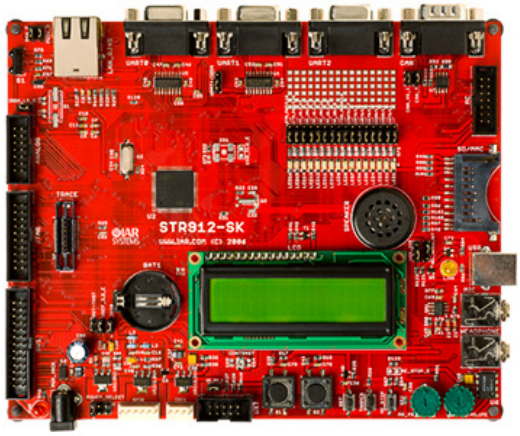
  4 adet Capture/Compare/PWM özelliklerine sahip zamanlayıcı
- 3-Faz motor denetleyicisi (IMC)
- Boundary scan desteği olan JTAG arabirimi

  Hata ayıklama için ARM EmbeddedICE® RT

  Flaşı programlamak için In-System Programming (ISP)
- Embedded trace modül (ARM ETM9)

  9 pinlik arabirimi ile yüksek hızlı trace yeteneği

Geliştirme seti Şekil 7.1'de gösterilmektedir.. Olimex firmasının tasarlamış olduğu bu geliştirme seti ile birlikte Şekil 7.2'de gösterilen, IAR firmasının JTAG arabirimi kullanılmıştır (IAR Systems AB, 2005).



Şekil 7.1: Kullanılan geliştirme kartı

Şekil 7.2: Kullanılan geliştirme ve hata ayıklama birimi

İşletim sistemi ANSI C uyumlu olarak geliştirilmiş olup, düşük seviyeli kodlamalar assembler ile gerçekleştirilmiştir. Geliştirme ortamı olarak da IAR Embedded Workbench kullanılmıştır.

İşletim sistemi, tasarım aşamasında farklı modüllere bölünmüştür. Bu modüllerin her biri ayrı tasarlanmış ve kodlanmıştır. İşletim sistemi modülleri şu şekildedir;

- HAL – Donanım soyut katmanı
  - ARCH – işlemci mimarisi ile ilgili sistem çağrıları ve kodlamalarını içeren modüldür.
  - ASM – Düşük seviyeli kodlamaları içeren modüldür.
  - LIB – Fiziksel katmana erişimde kullanılan yardımcı sistem çağrılarını içeren modüldür.
- KERNEL – İşletim sistemi çekirdeği
  - XKERNEL – Tüm işletim sistemi çağrı arabirimlerini, görev zamanlayıcı, planlayıcı ve sevkedicisini içeren ana modüldür.
  - XLIST – Yazılımsal kuyruk ve bağlantılı liste gerçeklemesini içeren modüldür. Hazır görevlerin listesinin düzenlenmesi ve işletilmesinde kullanılır.
  - XSEMA – Semafor gerçeklemesini içeren modüldür.
  - XMUTEX – Mutex semaforunun gerçeklemesini içeren modüldür.
  - XTIMER – Tüm sistem ve kullanıcı tanımlı zamanlayıcı işlevlerini ve veri yapılarını içeren modüldür.

- MEMORY – Sistem seviyesinde kullanılan bellek yönetimi
  - Bu modül dinamik bellek yönetimi işlevlerini ve bunların kullandığı yardımcı kütüphane çağrılarını içerir.
- NETWORK – Ağ katmanı protokol kümeleri ve uygulamaları
  - XNET – TCP/IP ve UDP protokollerinin gerçeklemelerini içeren modüldür.
  - APPS – Ağ katmanı ile ilgili uygulama örneklerinin gerçeklemelerini içeren modüldür.
- FILESYS – Dosya sistemi
  - FATFS – FAT12/FAT16/FAT32 dosya sistemlerini destekleyen çekirdek dosya sistemi modülüdür.
- DRIVERS – Sistemde kullanılan aygıt sürüleri
  - LCD – Karakter ekran için aygıt sürücü modülü.
  - MMC – SD/MMC kartları için aygıt sürücü modülü.
  - NET – Ağ katmanı fiziksel aygıt sürücü modülü.
  - RTC – Gerçek zamanlı saat aygıt sürücü modülü.
  - UART – Asenkron haberleşme kanalları için aygıt sürücü modülü.
- AUDIO – Mutimedya özellikleri
  - WAVE – WAV dosyalarını çalabilen Player modülü

Tüm bahsi geçen modüller işletim sistemi kapsamında gerçekleştirilmiş ve geliştirme kartı üzerinde denenmiştir.

7.1.1. Hal

Bu modül donanım katmanına erişimlerde kullanılan kütüphane çağrılarını içerir. Bu kütüphane çağrıları mikro işlemcinin giriş/çıkış portları, sistem zamanlayıcısı, bellek yönetim bloğu, doğrudan bellek erişim ünitesi, görev yöneticisi zamanlayıcıları gibi özel donanım bileşenlerine erişim sağlar. Üç alt gruba ayrılmıştır.

7.1.1.1. arch modülü

Kullanıcıya açık olan sistem çağrılarını içerir. Bu çağrılar işletim sistemi çekirdeği içerisinden de kullanılır. Kullanıcıyı ve sistemin geri kalanını, donanım katmanından soyutlar. Aynı zamanda çekirdeğe hizmet veren, görev yaratılması ile ilgili ana işlevlere sahiptir. Bu modülde yer alan sistem çağrıları aşağıdaki gibidir;

HAL_cpu_init: İşlemci saat frekansı ayarlamaları ve gerekli yazmaç kurulumlarını gerçekleştirir.

HAL_cpu_deinit: Kurulmuş olan işlemci bileşenlerini varsayılan durumuna getirir ve işlemciyi en yavaş saat frekansına getirir.

HAL_mmu_init: Bellek yönetim ünitesi ile ilgili başlatma işlemlerini gerçekleştirir. Gerekli olan tüm bellek havuzlarını kurar.

HAL_mmu_deinit: Kurulmuş olan bellek yönetim birimini eski haline getirir ve ayrılmış bellek havuzlarını serbest bırakır.

HAL_dma_init: Doğrudan bellek erişim denetleyicini kurar. Bu denetleyici sistemde bir çok yerde kullanılır.

HAL_dma_deinit: Kurulmuş olan bellek erişim denetleyicisini kapatır.

HAL_tmr_init: Sistemde kullanılacak tüm zamanlayıcı bileşenlerini kurar.

HAL_tmr_deinit: Kurulmuş olan sistem zamanlayıcılarını durdurup kapatır.

HAL_fs_init: Sistemdeki dosya sistemini kurar ve başlatır. SD hafıza kartının sabit disk sürücü gibi kullanılmasına olanak sağlar.

HAL_fs_deinit: Kurulmuş olan dosya sistemini durdurur ve gerekli bellek temizleme işlerini yerine getirir.

HAL_fs_isvalid: Dosya sisteminin doğru bir şekilde başlatılıp başlatılmadığını anlamak için kullanılan yardımcı fonksiyondur.

HAL_tmr_start: Kurulmuş olan tüm sistem zamanlayıcılarını yeniden başlatmak için kullanılır.

HAL_tmr_stop: Başlatılmış olan tüm sistem zamanlayıcılarını durdurmak için kullanılır.

HAL_tsktmr_start: Kurulmuş olan görev yönetici zamanlayıcısını yeniden başlatır.

HAL_tsktmr_stop: Başlatılmış olan görev yönetici zamanlayıcısını durdurur.

HAL_tsktmr_status: Görev yönetici zamanlayıcısının çalışma durumunu anlamak için kullanılır.

HAL_systmr_start: Kurulmuş olan ana zamanlayıcıyı yeniden başlatır.

HAL_systmr_stop: Başlatılmış olan ana zamanlayıcıyı durdurur.

HAL_systmr_status: Ana zamanlayıcının çalışma durumunu anlamak için kullanılır.

HAL_task_context_init: Çoklu görev uygulamalarında herhangi bir görevin başlatılmasını sağlar.

HAL_task_idle_context_init: Sistemdeki boş görevin başlatılmasını sağlar.

HAL_io_outb: Donanım üzerindeki çıkış kanallarına 8 bitlik veri yolu üzerinden erişmeyi sağlar.

HAL_io_outbit: Donanım üzerindeki çıkış kanallarına 1 bitlik veri yolu üzerinden erişmeyi sağlar.

Bu sistem çağrıları kullanıcıya, sistem seviyesinde erişimler için izin vererek büyük bir güç ve esneklik sağlar. Çekirdek için gerekli olan tüm donanım ile ilgili kodlamalar bu modülde gerçeklenmiştir. Mikro işlemcinin ayarları, sistem ve görev zamanlayıcılarının yönetimleri, dosya sistemi, bellek yönetimi bloklarının kurulması ve başlatılması, giriş/çıkış işlemleri için arabirimler bu modülde yer alır.

7.1.1.2. asm modülü

Bu modül çoklu görev yeteneğini sağlayan ana kısımdır. Düşük seviyeli içerik anahtarlama bu modül içerisindedir. Çoklu görev yeteneğine sahip işletim sistemlerinde içerik anahtarlama düşük seviyeli (assembler) kod ile gerçekleştirilir. Bunun nedeni o an çalışılmakta olan görevin tüm işlemci kayıtçıları, kayıpsız olarak saklanmak zorundadır.

Tez kapsamında gerçekleştirilen işletim sistemi çekirdeğinde içerik anahtarlama durumunda, o anda çalışan göreve ait işlemci kayıtçıları, bayrak durumu ve geri dönüş adresi, görevin kullandığı yığın bloğunun en sonuna yazılır. Böylece görev için ayrı bir kayıtçı bloğu tahsis edilmek zorunda kalınmaz.

asm modülü aynı zamanda düşük seviyeli kesme işleyicilerinin yönetimini de içerir. İşlemcide oluşan kesme isteğine bağlı olarak hangi yordama yönlendirileceği bu modül içerisinde belirlenir.

İşletim sistemi çekirdeğinde aynı önceliğe sahip görevlerin zaman paylaşım olarak işletilebilmeleri için (round-robin planlama algoritması) belirli zaman periyotlarında görev planlayıcısının dürtülmesi gerekir. Bu dürtme işlemi sistem içerisindeki bir zamanlayıcı tarafından gerçekleştirilir. Bu zamanlayıcı görev planlayıcısına atanarak, sistemde başka bir çağrı tarafından kullanılması engellenir.

Bu modül, görevlerin engellenmesi, askıya alınması veya devam ettirilmek istenmesi durumlarında, planlama zamanlayıcısından bağımsız olarak görev değiştirilmesini sağlayan Task_yield gibi sistem çağrılarının düşük seviyeli kodlarını da içerir.

İstisnai durum yöneticisi ile işletim sistemi başlatma çağrısı da, bu modülde yer alır.

7.1.1.3. lib modülü

Bu modül işletim sistemi çekirdeği tarafından kullanılır ve mikro işlemci üzerindeki tüm donanım bileşenlerine erişmek için yardımcı kütüphane çağrılarını içerir. Bu modül bileşenleri genellikle işlemcinin üreticisi tarafından sağlanır.

Burada bahsi geçen bu üç grup modül işlemcinin donanım bileşenleri ile etkileşimli bir şekilde haberleşme işlemlerini sağlayıp tamamıyla platform bağımlıdır. Tüm platform bağımlı sistem çağrıları tek bir modül içerisine toplanarak, taşınabilirlik maksimum düzeye çıkartılmaya çalışılmıştır. Bu sayede işletim sistemini bir başka mikro işlemci için uyarlanmak istendiği zaman, sadece HAL modülünde uygun değişiklikleri yapmak yeterli olacaktır.

7.1.2. Kernel

Bu modül işletim sisteminin tüm çekirdek bileşenlerini kapsar. Alt modülleri ile beraber standart bir çok işletim sistemi nesnesini ve çağrısını destekleyecek yapıdadır. Tüm sistem kurulum, görev ve nesne yönetim çağrıları, mesajlaşma üniteleri, zamanlayıcı ve kuyruk gerçeklemeleri bu modül içerisinde oluşturulmuştur. Beş alt gruba ayrılmıştır.

7.1.2.1. xkernel modülü

İş planlayıcısı, görev yöneticisi ve zamanlayıcısı, görev planlama algoritmaları, görev yönetim fonksiyonları, işletim sistemi başlatma ve durdurma çağrıları bu modülde gerçekleştirilmiştir. Çekirdeğin kalbidir. İki temel planlama algoritmasını destekler.

- Rekabetçi öncelik tabanlı
- Round-robin

Görev yöneticisi iki durumda görev değişimi isteğinde bulunur.

- Görev zamanlayıcı dürtüsü
- Herhangi bir görev içerisinden yapılan sistem çağrısı

Görev yöneticisine, belirli zaman periyotlarında, görev zamanlayıcısı tarafından anahtarlama isteğinde bulunulur. Görev yöneticisi oluşan bu dürtü ile hal modülündeki zamanlayıcı kesme rutinine gider. Burada rtosTASKSWITCH isimli özel hal-asm yordamı, o anda çalışan görevin tüm işlemci kayıtçılarını, bayrak durumunu ve geri dönüş adresini görevin yığın alanına kopyalar.

Bu kopyalama işleminden sonra xkernel'daki görev planlayıcısını (scheduler) çağırır.

Görev planlayıcısı çalışmakta olan ve anahtarlama isteği gelen görevin çalışma zamanını kontrol eder. Eğer anahtarlanması için gereken süre dolmadıysa işlev sonlanır ve görev kaldığı yerden çalışmasına devam eder.

Bu özellik her bir göreve belli bir çalışma süresi atanmasına yarar. Anahtarlama zamanı geldiyse çekirdek, görev sevkedicisini çağırır.

Görev sevkedicisi, o an çalışmakta olan görevi aynı önceliğe sahip görevlerin tutulduğu hazır görev listesinin sonuna ekler ve durumunu hazır yapar. Daha sonra sistemdeki en yüksek önceliğe sahip hazır görev listesinin en başında bulunan görevi alır durumunu çalışıyor yapar ve hizmeti hal-asm deki rtosTASKSWITCH'e geri bırakır.

Değiştirilmiş olan mevcut görevin tüm kayıtçıları, bayrakların durumu ve geri dönüş adresi işlemcinin kayıtçıları ile değiştirilir. Kesme hizmeti sona erdiğinde, seçilen görevin en son kaldığı yerden itibaren uygulama çalışmaya devam eder.

Aynı şekilde bir sistem çağrısı olan TASK_yield fonksiyonu kullanılarak da görev anahtarlama isteğinde bulunulabilir. Bu durumda hal-asm deki rtosTASKYIELD yordamı çağrılır ve rtosTASKSWITCH deki işlem adımları gerçekleştirilir.

Gerçekleştirilen işletim sistemi çekirdeği özel olarak bir de istisnai durum işleyicisi içerir. Yürütülmekte olan bir görev herhangi bir anahtarlama isteği gelmeden kullanıcı tarafından sonlandırılırsa (return işlevi ile) bir istisnai durum işleyicisi otomatik olarak devreye girerek, görevi listeden çıkartır ve sistemden siler. Çekirdeği yeni bir anahtarlama isteğinde bulunarak durumdan haberdar eder. Çekirdek bir sonraki yüksek önceliğe sahip görevi hazır listesinden alarak işletime koyar.

İstisnai durum işleyicisi gerçek zamanlı işletim sistemlerinde çok sık karşılan bir unsur değildir. Genelde gerçek zamanlı işletim sistemlerinde görev yordamı sonsuz döngü biçiminde uygulanır. Bizim geliştirdiğimiz sistemde ise tamamlanmak için çalışan yapıda da görevlere izin verilir. Bu görevler bir kere işletildikten sonra sonlandıklarında istisnai durum işleyicisi tarafından işlenir ve sistemden silinirler.

Temel çekirdek çağrıları aşağıdaki gibidir;

CS_lock: Kritik bölgede çalışıldığını çekirdeğe bildirmek için kullanılır. Bu bölge işletilirken görev yöneticisinin, işletimi bir başka göreve ataması engellenir.

CS_unlock: Kritik çalışma bölgesini sonlandırır.

CS_waitlock: Kritik bölgede çalışıldığından emin olmak için kullanılır.

TASK_create: Uygulama içerisinde yeni bir görev yaratmak için kullanılır.

TASK_destroy: Daha önceden oluşturulmuş bir görevi sonlandırmak için kullanılır.

TASK_block: İşletilmekte olan bir görevi engelli duruma getirmek için kullanılır.

TASK_suspend: İşletilmekte olan bir görevi bekleme durumuna getirmek için kullanılır.

TASK_resume: Bekleme veya engelli durumdaki görevleri tekrar işletime almak için kullanılır.

TASK_resumeall: Sistemdeki tüm beklemedeki ve engelli durumdaki görevleri işletime alınması için kullanılır.

TASK_getstate: Bir görevin çalışma durumunu verir.

TASK_setstate: Bir görevin çalışma durumunu değiştirmek için kullanılır.

TASK_getpriority: Bir görevin öncelik seviyesini verir.

TASK_setpriority: Bir görevin öncelik seviyesini değiştirmek için kullanılır.

TASK_setevent: Bir görevin belirlenen bir olayını tetiklemek için kullanılır.

TASK_resetevent: Bir görevin tetiklenen bir olayını temizlemek için kullanılır.

TASK_waitevent: Bir görevin belirlenen görevinin oluşmasını bekler.

TASK_checkstack: Bir görevin yığın bellek kullanımını hesaplar.

TASK_yield: İşletilmekte olan bir görevin işletimini hazır kuyruğunda bekleyen bir başka göreve vermesini sağlar.

OS_init: Tüm işletim sistemi servislerini kurar.

OS_run: İşletim sistemini başlatır ve işletimi boş göreve verir.

OS_task_create: Kullanıcı düzeyinde yeni bir görev oluşturmak için kullanılır.

OS_msg_send: Bir göreve istenilen bir mesajı yollar.

OS_msg_get: Bir görevin mesaj kuyruğundaki ilk mesajı verir.

TMR_wait: Kullanıcı düzeyinde döngü tabanlı, sistemi engelleyen bekleme yordamı.

TMR_delay: Kullanıcı düzeyinde kesme tabanlı, sistemi engelleyen bekleme yordamı.

7.1.2.2. xlist modülü

xkernel modülünde gerçeklenmiş olan görev yöneticisinin kullanmış olduğu hazır görevler listesinin düzenlenmesinde kullanılır. Her farklı öncelik seviyesi için ayrı bir hazır görev listesi tutulur. Böylece görevlerin birbirleri ile çekişmelerinde hangi görevin işletileceğine karar verilen mekanizmanın mümkün olan en kolay ve basit şekilde gerçeklenmeleri sağlanır. xlist modülü bir çeşit kuyruk yapısını gerçekler. Listeye eklenen yeni bir görev kuyruğun en sonuna eklenir. Listeden çekilen eleman ise kuyruğun en başındaki elemandır ve listeden çıkartılır. Sistem çağrıları aşağıdaki gibidir.

LIST_init: Bir bağlantılı liste yapısını kurar.

LIST_head: Bir listenin ilk öğesini verir.

LIST_add: Listenin sonuna yeni bir öğe ekler.

LIST_get: Listedeki istenilen öğeyi arar ve bulunca bulduğu öğeyi verir. Fakat bulduğu öğeyi listeden çıkartmaz.

LIST_remove: Listenin en başındaki öğeyi verir ve bu öğeyi listeden çıkartır.

7.1.2.3. xsema modülü

İşletim sistemi nesnelerinden semaforun gerçeklendiği modüldür. Bölüm 5'de bahsedildiği gibi semafor görevler arasında paylaşılan bellek veya kaynaklar için senkronizasyon nesnesi olarak kullanılır. Gerçek zamanlı işletim sistemi çekirdeklerinde önemli bir yer tutar. Gerçekleştirilen işlevleri aşağıdaki gibidir.

SEMA_init: Bir semafor nesnesini kurar.

SEMA_obtain: Belirlenen bir semafor nesnesini elde etmek için kullanılır. Eğer nesne bir başkası tarafından daha önceden alınmış ise bu fonksiyonun çağırıldığı görev bekleme durumuna getirilir.

SEMA_release: Elde edilen semafor nesnesini geri bırakmak için kullanılır.

SEMA_tryobtain: Bir semafor nesnesini elde etmeye çalışır. Eğer nesne bir başkası tarafından daha önceden alınmış ise fonksiyon sonlanır.

7.1.2.4. xmutex modülü

İşletim sistemi nesnelerinden mutex semaforunun gerçeklendiği modüldür. Gerçekleştirilen işlevleri aşağıdaki gibidir.

MTX_init: Muteks semaforunu kurar.

MTX_lock: Belirlenen bir muteks semaforunu kilitler. Eğer muteks daha önceden kilitlenmiş ise, fonksiyonun çağrıldığı görev bekleme durumuna getirilir.

MTX_unlock: Kilitlenmiş olan bir muteks semaforunu serbest bırakır.

MTX_trylock: Bir muteks semaforunu elde etmeye çalışır. Eğer muteks daha önceden kilitlenmiş ise fonksiyon derhal sonlandırılır.

7.1.2.5. xtimer modülü

Tüm sistem ve kullanıcı tanımlı zamanlayıcılar için veri yapılarının ve işlevlerinin gerçeklendiği modüldür. Zamanlayıcı, temel tik sayacı olarak 1 ms'lik sistem zamanlayıcısını kullanır. Bu zamanlayıcı görev yöneticisini dürten zamanlayıcıdan farklıdır. Bunun nedeni kritik bölgedeki kodları işletirken, görev anahtarlamasına engel olmak için görev zamanlayıcısının durdurulmasının, sistemin ana zamanlayıcısını etkilememesi içindir. Bu yapıda tek bir zamanlayıcı kesme rutini içerisinde çoklu zamanlayıcı maniplasyonu uygulanır.

Uygulamada, sistem zamanlayıcı kesme rutini içerisinde tek bir sistem değişkeninin değeri sürekli olarak arttırılır. xtimer içerisindeki işlevler yardımıyla bu sayaç değeri referans alınarak tüm zamanlayıcı ihtiyaçlarını karşılayacak çağrılar kümesi tanımlanmıştır. Bunlar aşağıdaki gibidir.

timer_set: Belirlenen bir zamanlayıcıyı istenilen zamana kurar.

timer_reset: Kurulmuş bir zamanlayıcıyı temizler.

timer_restart: Kurulmuş bir zamanlayıcıyı yeniden başlatır.

timer_expired: Kurulmuş bir zamanlayıcının, zamanının dolup dolmadığını anlamak için kullanılır.

timer_free: Kurulmuş olan bir zamanlayıcıyı durdurur.

7.1.3. Memory

Bu modül dinamik bellek yönetimi ile sistem kütüphanelerini içerir. Tasarlanan işletim sistemi, ilk aşamada donanımsal bellek kontrol üniteleri olmayan (MMU) mikro işlemcileri hedef aldığından, bellek yönetimi yazılımsal olarak gerçekleştirilmiştir.

Amaç, bir blok olarak tanımlanmış bellek havuzunun dinamik olarak paylaşılmasına izin veren bir sistem kütüphanesi gerçekleştirmektir. Bu sayede uygulama içerisinde heap alanı kullanılmadan dinamik bellek yönetimi yapılarak, olası bellek parçalanma risklerinin ortadan kaldırılması hedeflenmiştir. Bu modül, derleyicilerin bellek ayarları ile ilgilenmeden doğru çalışan kod yığınlarının oluşturulmasına da yardımcı olur. Derleme anında belirlenen bir maksimum alan kadar ayrılan doğrusal bellek havuzu (tek boyutlu bir dizi şeklinde), çalışma anında herhangi bir görev içerisinden çağrılarak kullanılabilir. Üç adet basit çağrısı mevcuttur.

mmem_init: İstenilen bir bellek havuzunun kurulmasını sağlar.

mmem_alloc: Daha önceden sistem tarafından tanımlanmış olan büyük bellek havuzu içerisinden, istenilen miktarda boş alan ayırır.

mmem_free: Ayrılmış olan alanı temizler ve sisteme geri verir.

7.1.4. Network

İşletim sisteminin ağ haberleşmesi protokol ve uygulamalarının gerçekleştirildiği modüldür. Günümüz gömülü cihazların bir çoğu artık ağ desteği ile birlikte gelmektedir. Bu aşamada ağ desteği olmayan bir işletim sistemini düşünmek söz konusu olamaz. İşletim sistemi içerisinde network modülü iki alt gruba ayrılarak gerçekleştirilmiştir.

7.1.4.1. xnet modülü

Bu modül Data Link Layer'ın üzerinde yer alan ağ katmanındaki protokol kümelerini içerir. İşletim sisteminin ağ desteği için RFC uyumlu protokoller yazılmak yerine literatürdeki araştırmalar sonucunda açık kaynaklı olan bir yazılıp kütüphanesinin kullanılmasına karar verilmiştir. Bir çok açık kaynaklı ağ protokolleri kütüphanesi olmasına rağmen yapılan incelemelerde uIP TCP/IP kütüphanesi, Dunkels, (2006), kullanım için uygun bulunmuştur. En önemli özelliği 8/16/32 bitlik işlemcilerde kolaylıkla kullanılabilmesi, çok düşük bellek gereksinimi olması ve tabi en önemlisi anlaşılabilir ve açıklamalı olan kaynak kodlarının, serbestçe kullanılabilmesidir. Açık kaynaklı olmasının tercih edilmesinin başlıca nedeni, ilerideki çalışmalarda güvenlik gerektiren haberleşme kanallarında, kripto algoritmalarının bu protokol kümeleri üzerinde uygulanmak istenmesidir. Dünyada bir çok kişi tarafından bilinen ve kullanılan bu protokol kümesi güncelliğini korumakta, geliştirme ve iyileştirme çalışmaları halen devam etmektedir.

Geleneksel TCP/IP gerçeklemeleri birkaç yüz kilobayt RAM ile yüz kilobayt civarında ROM gereksinimi duyar. Bu gerçekleme kısıtlı kaynaklı mikro işlemciler için pek mümkün gözükmemektedir. Bunun yanında uIP birkaç kilobaytlık ROM'a ve birkaç yüz baytlık RAM'e ihtiyaç duyar. Kesin bellek gereksinimleri derleme anında belirlenen özelliklere bağlı olarak değişkenlik gösterir (Dunkels ve diğ., 2003, 2004b,2005a).Genel özellikleri aşağıdaki gibidir.

- İyi dokümante edilmiş, hemen hemen her kod satırında açıklama bulunan kaynak kodu
- Çok düşük kod büyüklüğü
- Derleme anında konfigüre edilebilen düşük RAM kullanımı
- ARP, SLIP, IP, UDP, ICMP (ping) ve TCP protokolleri
- Aynı anda kullanılabilen ve maksimum sayısı derleme anında belirlenen TCP bağlantı desteği
- Ticari ve ticari olmayan her türlü uygulamada serbestçe kullanım izni
- Akış kontrolü, parçalanmış paketleri tekrar bir araya getirme ve zaman aşımında tekrar gönderim desteği olan, RFC uyumlu TCP ve IP protokol gerçeklemesi

Bu kütüphanenin verimli bir şekilde kullanılabilmesi için, tüm kod yapısı ve dokümanları incelenmiş ve bir takım eklentiler ve değişikler gerçekleştirilmiştir. Özellikle aynı anda birden fazla farklı protokol kullanan uygulama desteği için sistem çağrıları ve yordamları eklenmiştir.

7.1.4.2. apps modülü

Bu modül TCP/IP protokol kümesini kullanan uygulamaların gerçeklemelerini içerir. Bu tez kapsamında WEB server uygulaması eklenmiş ve kullanılmıştır. WEB server uygulaması dinamik bellek yönetimi ve dosya sistemi gerçeklemesi kullanılarak, SD kart içerisindeki HTML sayfalarını hizmet verecek şekilde düzenlenmiş ve kodlanmıştır. İlerideki çalışmalarda DHCP istemci, FTP, SMTP, POP3 gibi uygulama seviyesi protokollerinin de bu modüle eklenmesi hedeflenmektedir.

7.1.5. Filesys

İşletim sistemi kapsamında standart dosya sistemi yapısının getireceği avantajlar göz önüne alınarak bu modülde FAT12/FAT16/FAT32 dosya sistemleri gerçeklenmiştir. Gömülü cihazlarda saklama birimi olarak kullanılan en yaygın birimler MMC ve SD kartlardır. Bu donanım bileşenleri seri haberleşme kanalını kullanırlar. Bu modülde, dosya sistemi gerçeklemesi için açık kaynaklı bir kütüphane kullanılmıştır. Çok çeşitli kütüphanelerin incelenmesinin ardından FatFs kütüphanesinin kullanımına karar verilmiştir (ChaN, 2008).

7.1.5.1. FatFs modülü

Bu modül standart FAT12/FAT16/FAT32 dosya sistemlerini destekleyen ve gömülü sistemler için kullanıma elverişli, açık kaynaklı bir kütüphanedir. Tüm dosya sistemi işlevlerinin yanında, kullanılacak platforma uygun hale getirmek için kodlamalar ve sürücü gerçeklemeleri tez kapsamında gerçekleştirilmiştir. Uygulama seviyesi çağrıları aşağıdaki gibidir.

f_mount: Çalışılacak olan donanım bileşeni ile ilişkilendirmenin yapılmasında kullanılır.

f_open: Bir dosyayı açmak veya oluşturmak için kullanılır.

f_read: Açılan bir dosyayı okumak için kullanılır.

f_write: Açılan bir dosyaya veri yazmak için kullanılır.

f_lseek: Açılan bir dosyanın istenilen pozisyonuna konumlanmak için kullanılır.

f_close: Açılan bir dosyayı kapatmak için kullanılır.

f_opendir: Bir klasörü açmak için kullanılır.

f_readdir: Klasör bilgilerini okumak için kullanılır.

f_stat: Dosya durumunu öğrenmek için kullanılır.

f_getfree: Kullanılan fiziksel aygıttaki toplam boş küme (cluster) miktarını bulur.

f_truncate: açılmış olan bir dosyayı kırpar.

f_sync: Ön bellekte duran tüm verilerin anında fiziksel aygıta yazılmasını sağlar.

f_unlink: bir klasör veya dosyayı fiziksel aygıttan kaldırmayı sağlar.

f_mkdir: bir klasör oluşturulmasını sağlar.

f_chmod: bir dosyanın erişim özelliklerini değiştirmede kullanılır.

f_utime: Dosyanın zaman damgasını değiştirmek için kullanılır.

f_rename: Bir klasör veya dosyanın ismini değiştirmek için kullanılır.

f_mkfs: bir sürücü üzeride dosya sistemi oluşturmak için kullanılır.

7.1.6. Drivers

İşletim sistemi kapsamında bazı aygıt sürücülerinin gerçeklemeleri de yapılmıştır. Bu aygıt sürücüleri, örnek uygulamada kullanılan donanım bileşenleri için geliştirilmiş olup ihtiyaca göre genişletilebilir. Aygıt sürücülerinin gerçeklemesi HAL ve LIB modülleri kullanarak yapılmıştır. Bu kullanım, taşınabilirliği kolaylaştıran önemli bir yaklaşımdır.

7.1.6.1. lcd modülü

Geliştirme seti üzerinde bulunan 2 x 16 karakterlik LCD'yi kullanabilmek için bu aygıt sürücüsü gerçeklenmiştir. Kullanılan LCD HD44780 uyumlu standart bir karakter LCD sürücü denetleyicisine sahiptir.

Bu sebepten aygıt sürücü kısmında bu denetleyicinin komut kümesi kullanılmıştır (Ouwehand, 2006).

7.1.6.2. mmc modülü

Dosya sistemi gerçeklemesinde, SD/MMC kartına erişim için kullanılan sürücüdür. Bu sürücü hızlı seri haberleşme kanalını kullandığı ve çevre aygıt (SD kart), sisteme göre daha yavaş kaldığı için efektif bir erişim ancak DMA ile gerçekleştirilebilir. Bu sayede SD kartına büyük bloklar halinde erişim yapılırken işlemcinin yürütme zamanı kullanılmaz. SD/MMC kart arabirim sürücüsü gerçekleştirilirken, daha önceden bu konuyla ilgili olarak yapılmış çalışmalardan faydalanılmasının yanı sıra tüm sürücü katmanı yeniden oluşturulmuştur (Thomas, 2006).

7.1.6.3. net modülü

Ağ katmanı protokolleri fiziksel katmana erişmek için bu modülü kullanılır. Bu modül hızlı veri haberleşmesini desteklemek için DMA kullanılarak gerçeklenmiştir. Ağ katmanı gerçekleştirilen işletim sisteminde, fiziksel ara yüz olarak ethernet'i kullanmaktadır. 100 MBit'e kadar veri transfer hızını destekleyen ethernet çipi ile mikro işlemci arasındaki haberleşme MII denilen aygıt bağımsız arabirimi vasıtasıyla gerçekleştirilir. Kullanılan işlemci üzerinde bulunan MAC katmanı, ethernet çipini kullanarak dış dünya ile haberleşir.

7.1.6.4. rtc modülü

Gerçek zamanlı saat desteği için kullanılan bu modül dosya sistemine ve genel amaçlı kullanıma hizmet etmektedir. Dosya sistemi içerisinde, klasör veya dosyaların yaratılması ve değiştirilmesi işlemlerinde, uygun zaman damgasını basmak için kullanılan bu modül, istenildiği anda kullanıcı tarafından sistem çağrıları vasıtası ile zamana erişmeyi de sağlar.

7.1.6.5. uart modülü

Asenkron seri haberleşme kanallarının kullanılabilmesi için gerçeklenmiştir. İşlemci üzerinde bulunan bu haberleşme kanalları dış dünya ile veri alış verişinde kullanılmaktadır. Gerçekleştirilen uygulamada dış dünyadaki bir sensör düğümünün gönderdiği sıcaklık bilgisini, işletim sistemi içerisinden kullanabilmemiz için gereklidir.

7.1.7. Audio

İşletim sistemi içerisine, çoklu görev desteğinin avantajlarını gösterebilmek için bir audio modülü eklenmiştir. Bu modülün ilerideki çalışmalarda değişik dosya formatlarını çalabilme yeteneğinin olması planlanmakla beraber bu tez kapsamında sadece wave modülü ile WAV dosyalarını çalabilme özelliği eklenmiştir.

7.1.7.1. wave modülü

audio modülü altında WAV dosyalarını çalabilmek için eklenmiş bir modüldür. Amaç, SD kart üzerindeki WAV dosyalarını açıp, ham veriyi sayısal analog dönüştürücü ve bir yükselteç vasıtasıyla hoparlöre iletmektir. Kullanılan işlemcinin DAC birimi yoktur. Fakat hemen hemen her işlemcinin sahip olduğu PWM birimine sahiptir. PWM birimi, periyodu ve darbe genişliği ayarlanabilen kare dalga işaretleri üretilmesini sağlar. Üretilen bu kare dalga işaret, işlemcinin bir çıkış birimi üzeriden dış dünyaya aktarılır. PWM işareti sayısal haberleşmeden bildiğimiz üzere eş bir analog işarete dönüştürülebilir. Bunun için işareti alçak geçiren süzgeçten geçirmek yeterlidir. Elde edilen analog işaretin genliği PWM işaretinin darbe genişlik oranı ile orantılıdır. Bu sayede işlemci üzerinde kullanabileceğimiz yapay bir DAC ünitesi oluşturmuş oluruz.

Bir başka önemli nokta, 44100 Hz lik örnekleme frekansına sahip bir WAV dosyası için DAC, yaklaşık 22,68 us de bir güncellenmek zorundadır. Bu işlemcinin neredeyse tüm gücünü tüketir. Özel bir yöntem ile wave modülü DMA kullanılarak gerçekleştirilmiştir. Bu yöntemde SD karttan bloklar halinde okunan WAV dosyası, DMA kullanılarak PWM birimine aktarılır. Bu sayede, bellek ile PWM ünitesi arasındaki veri transferi, işlemcinin yürütme zamanı kullanılmadan gerçekleştirilir. Bu, sistem üzerinde birçok görev çalışırken arka planda WAV dosyasının oynatılmasını sağlar. PWM ünitesi 8 bitlik kullanılmak zorunda olduğu için şimdilik 8-bitlik WAV dosyaları çalınabilmektedir. Aynı zamanda PWM'den elde edilen DAC, çok yüksek kaliteli olmayıp kayıpları olduğundan seste bazı bozulmalar da meydana gelmektedir. Bu durum dış dünyada ayrı bir CODEC kullanılması ile çözülebilir.

7.2. Uygulama Örneği

Tez kapsamında tüm bu gerçekleştirilenleri gösterebilmek amacıyla örnek bir uygulama oluşturulmuştur. Bu örnekte, STR912 Starter Kit'i üzerine albatROS işletim sistemi ve bunu kullanan çok görevli bir uygulama yazılmıştır. Uygulamada 5 farklı görevin aynı anda çalışmasına izin verilmiştir.

Görevlerden bir tanesi wave modülüne atanarak SD karttaki bir klasör altında bulunan tüm WAV dosyalarını sıra ile çalacak şekilde uyarlanmıştır.

Diğer bir görev xnet modülüne atanarak TCP/IP yi kullanan bir WEB server uygulamasının çalışmasına izin verilmiştir. Bu WEB server hosttan gelecek istekler doğrultusunda yine SD kart üzerinde bulunan HTML sayfalarını hosta göndermekle yükümlüdür. Bu istekler birden çok bilgisayardan gelebilir. Burada gerçekleştirilen en önemli özellik dinamik HTML sayfa desteğidir. Bunun anlamı host bilgisayara gönderilen HTML sayfaları işletim sistemi içerisindeki apps modülü tarafından değiştirilebilmektedir. Bu bir çeşit CGI scripti vasıtasıyla gerçekleştirilir. WEB server içerisinde çalışan script yorumlayıcı, SD kart üzerindeki HTML sayfası içerisinde bir script ifadesine rastlarsa, sistemde daha önceden kayıtlı olanlar ile karşılaştırarak uygun değerleri hosta yollar.

Bu özellik seri haberleşme kanalı vasıtasıyla dış dünyadan toplanan veri yığınlarını, grafik olarak host bilgisayara göndermek için kullanılır.

Gerçekleştirilen script yorumlayıcı, HTML sayfası içerisinde %! işaretleri ile başlayan bir ifadeyle karşılaşırsa, bu işaretlerin hemen ardından gelen string'i kendi listesinde arar. Bu string saha önceden tanımlanmış ise, string ile beraber tanımlanan işlev sayesinde, host bilgisayara nasıl bir bilgi gönderileceğine karar verilir. Örneğin;

%! temp-cnt ifadesi ile karşılaşan script yorumlayıcı, bu ifade yerine dış dünyadan altığı sıcaklık bilgileri toplamını, HTML kod içerisinde gönderir.

Dış dünyadan alınan veri yığını ise bir sensör düğümüne aittir. İki adet kablosuz sensör düğümü içerisine işletim isteminin küçük bir modeli yerleştirilmiştir. Bu düğümlerden bir tanesi kendi okuduğu sıcaklık değerini uzaktaki düğüme göndermeye çalışır. Bizim ana sistemimize bağlı olan düğüm ise uzaktaki düğümden gelen bu sıcaklık bilgilerini geliştirme setine, seri haberleşme kanalı üzerinden atmaktadır. Bu şekilde hazırlanan işletim sisteminin karmaşık bir uygulama örneğinde de başarıyla kullanılabildiği gösterilmiş olur.

8. SONUÇLAR VE ÖNERİLER

Tez kapsamında gerçekleştirilen uygulama ile, uzaktaki bir sensör düğümünden gelen sıcaklık bilgileri, grafik olarak WEB hizmeti ile, istekte bulunan tüm host cihazlara gönderilir. Gönderilecek HTML sayfalar, SD karttan bloklar halinde okunur. Bunun için dosya sistemi ve bellek yönetim ünitesi kullanılır. Bu sırada başka bir görev içerisinde, yine SD kart içerisindeki bir klasör taranarak tüm WAV dosyaları sırayla oynatılır. Ayrıca üç farklı görev, geliştirme seti üzerindeki LED'lere, mutex kullanarak sırayla erişir. Bu tez kapsamında, geliştirilen platform vasıtası ile, bir gömülü sistem üzerindeki gerçek zamanlı işletim sisteminin, karmaşık işlevlere sahip bir uygulamayı gerçekleştirme esnasında, ne kadar verimli ve kolay kullanıldığını göstermektedir.

Tezin ön hazırlık aşamasından sonra, uygulamanın planlanıp gerçekleştirilmesi aşaması, en zahmetli ve zaman alan aşaması olmuştur. Bu aşamada onlarca mikro işlemci incelenmiş, bunların CPU özelliklerinden donanım özelliklerine kadar tüm önemli noktaları üzerinde uzun görüşmeler ve tartışmalar yapılmıştır.

Bunun yanında, ticari ve açık kaynaklı onlarca gerçek zamanlı işletim sistemi ayrıntılı bir şekilde incelenerek, yapılması gerekenlerin doğru bir şekilde tasarlanması ve bir mantık üzerinde oturtulması sağlanmıştır.

Tüm bu aşamalardan sonra kodlamaya sıra geldiğinde, geçmişte edinilmiş bilgi ve tecrübelerin yanı sıra, eğitim hayatından elde ettiğimiz yazılım mühendisliği olgusunu da kullanarak, ortaya başarılı olduğuna inandığımız bir platform çıkartılmıştır.

Tez kapsamında oluşturulan işletim sistemi, amaçladığı tüm hedeflere ulaşmasının yanında, ilerideki çalışmalarda geliştirilmesi düşünülen alanları boş bırakmıştır.

Bu alanlardan en önemlisi güvenli bir haberleşme alt yapısı sunabilmek için, kripto destekli protokol kümeleridir. Bu sayede sivil uygulamaların yanında, askeri uygulamalarda da önemli bir güç kazanacağımızı düşündüğümüz bu özellikle birlikte kendi ağ katman protokol kümelerimizin de yazılması da planlanmaktadır.

Bunun yanı sıra orta uygulama katmanı olarak adlandırılan middleware katmanı için farklı geliştirme ve özellikler düşünülmektedir. Bunların başında, kullanıcıların uygulama programları oluşturmaları esnasında, Java kullanabilmeleri gelmektedir. Bu aşamada Java sanal makinesinin sistem içerisinde koşturulması gerekir. Bu konu ile ilgili çeşitli araştırma ve incelmelerin yanında çalışan bir örnekte oluşturulmuştur. Tezimizin kapsamı dışında olduğu için değinilmemiş ve ilerideki çalışmalar da ayrıntılı olarak incelenmesi planlanmıştır.

İnovatif bir düşünce ile, çekirdekte çoklu işlemci desteği için bir geliştirme öngörülmektedir. Bu konu daha çok paralel veri işleme ile ilintili olup ülkemiz için benzersiz bir çalışma olacağı düşünülmektedir. Bu aşamaya varılabilmesi için, geliştirilen işletim sisteminin çekirdek özelliklerinin de genişletilmesi, henüz uygulanmamış servis nesneleri desteklemesi gerekmektedir.

Bu tez konusu ile ilgili düşünülen en önemli geliştirme, oluşturulan işletim sistemini kullanarak, insansız hava taşıtları ile ilgili, çalışan bir model oluşturmaktır. Bunun için işletim sistemi içerisine özel yöntem ve algoritmaların eklenmesi, orta katman uygulamalarının, kontrol ve kumanda konusunu kapsayacak şekilde geliştirilmesi planlanmaktadır.

KAYNAKLAR

Akyildiz, F., Su, W., Sankarasubramaniam, Y., Cayirci, E., "A Survey on Sensor Networks", ***IEEE Communications Magazine***, (2002).

ARM Hodings, *Advanced Risc Machines* [online], http://www.arm.com, **(Ziyaret tarihi: 15 Aralık 2007)**.

Auslander, D.M., Ridgely, J.R., Ringgenberg, J.D., "Design and Implementation of Real Time Software for Control of Mechanical Systems", ***University of California Berkeley***, (2000).

Barello, L., *AVRX Real-Time Multitasking Kernel for the Atmel AVR series of micro controllers* [online], http://www.barello.net/avrx/index.htm, **(Ziyaret tarihi: 7 Eylül 2005)**.

Barr, M., "Programming Embedded systems in C and C++", ***O'Reilly & Associates***, (1999).

Barry, R., *The FreeRTOS Project* [online], http://www.freertos.org/, **(Ziyaret tarihi: 13 Mart 2006)**.

ChaN, *FAT File System Module* [online], Ohta city, Japan, http://elm-chan.org/fsw/ff/00index_e.html, **(Ziyaret tarihi: 10 Mart 2008)**.

Chen, M., Wei, G., "Scheduling Algorithm for Real-time VBR Video Streams Using Weighted Switch Deficit Round Robin", ***Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks***, (2003).

Cheng, A.M.K., "Real-Time Systems Scheduling, Analysis, and Verification", ***A John Wiley & Sons***, (2002).

Crossbow Technology, *Crossbow motes* [online], USA, http://www.xbow.com, **(Ziyaret tarihi: 29 Mayıs 2005)**.

Digital Equipment Corporation , "Introduction to VxWorks", ***Digital Equipment Corporation*** , (1997).

Dunkels, A., "Full TCP/IP for 8-Bit Architectures", ***Swedish Institute of Computer Science***, (2003).

Dunkels, A., Feeney, L.M., Grönvall, B., "An integrated approach to developing sensor network solutions", ***Swedish Institute of Computer Science***, (2004a).

Dunkels, A., Alonso, J., Voigt, T., "Making TCP/IP Viable for Wireless Sensor Networks", ***Swedish Institute of Computer Science***, (2004b).

Dunkels, A., "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors", ***Swedish Institute of Computer Science***, (2004c).

Dunkels, A., "Towards TCP/IP for Wireless Sensor Networks", Licentiate Thesis, ***Swedish Institute of Computer Science***, (2005a).

Dunkels, A., Schmidt, O., Voigt, T., "Using Protothreads for Sensor Node Programming", ***Swedish Institute of Computer Science***, (2005b).

Dunkels, A., Schmidt, O., Voigt, T., Ali, M., "Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems", ***Swedish Institute of Computer Science***, (2006a).

Dunkels, A., Finne, N., Eriksson, J., Voigt, T., "Run-Time Dynamic Linking for Reprogramming Wireless Sensor Networks", ***Swedish Institute of Computer Science***, (2006b).

Dunkels, A., Österlind, F., He, Z., "An Adaptive Communication Architecture for Wireless Sensor Networks", ***Swedish Institute of Computer Science***, (2007a).

Dunkels, A., Österlind, F., Tsiftes, N., He, Z., "Demo Abstract: Software based Sensor Node Energy Estimation", ***Swedish Institute of Computer Science***, (2007b).

Dunkels, A., "Programming Memory-Constrained Networked Embedded Systems", Doctoral Thesis, ***Swedish Institute of Computer Science***, (2007c).

Dunkels, A., "Poster Abstract: Rime - A Lightweight Layered Communication Stack for Sensor Networks", ***Swedish Institute of Computer Science***, (2007d).

Dunkels, A.., Österlind, F., Tsiftes, N., He, Z., "Software-based On-line Energy Estimation for Sensor Nodes", ***Swedish Institute of Computer Science***, (2007d).

Dunkels, A.., *uIP TCP/IP Stack for Embedded Microcontrollers* [online], Sweden, Swedish Institute of Computer Science, http://www.sics.se/~adam/uip/, **(Ziyaret tarihi: 15 Mayıs 2006)**.

Dunkels, A.., Woestenberg, L., Mansley, K., Monoses, J., *lwIP - A Lightweight TCP/IP stack* [online], Project Savane, http://savannah.nongnu.org/projects/lwip/, **(Ziyaret tarihi: 20 Mayıs 2006)**.

Eriksson, J., Dunkels, A., Finne, N., Österlind, F., Voigt, T., "Poster Abstract: MSPsim – an Extensible Simulator for MSP430-equipped Sensor Boards", ***Swedish Institute of Computer Science***, (2007).

Furr, S., "What Is Real Time And Why Do I Need It?", ***QNX Software Systems Ltd***., (2002).

Hoon, M., "Performance Analysis of Distributed Real-Time Embedded Systems", Master Thesis, ***Department of Electrical Engineering Information and Communication Systems/Electronic Systems***, Eindhoven, (2005).

IAR Systems AB, *IAR Systems* [online], Uppsala, Sweden, http://www.iar.com, **(Ziyaret tarihi: 1 Ekim 2005)**.

Jerraya, A.A., Yoo, S., Verkest, D., Wehn, N., "Embedded Software for SoC", ***Kluwer Academic Publishers***, (2003).

Joseph, M., "Real-Time Systems Specification, Verification and Analysis", ***Prentice Hall International***, (1996).

Karan, T.M., Becerikli, Y., Turalı, T.M., "Kontrol Sistemleri için Gerçek Zamanlı Çoklu Görev İşletim Sistemi", ***Türkiye Ulusal Otomatik Kontrol Kurultayı***, Ankara, (2006).

Karan, T.M., Becerikli, Y., Turalı, T.M., "Java Destekli Gerçek Zamanlı İşletim Sistemi ve Bir Uygulama", ***Türkiye Ulusal Otomatik Kontrol Kurultayı***, İstanbul, (2007).

Kim, K.H., "Middleware of Real-Time Object Based Fault Tolerant Distrubuted Computing Systems", ***IEEE 0-7695-1414-6/0***, (2001).

Labrosse, J., "MicroC/OS-II: The Real-Time Kernel", 2nd edition, ***CMP Books***, (1998).

Ledin, J., "Embedded Control Systems in C/C++: An Introduction for Software Developers Using MATLAB", ***CMP Books***, (2004).

Levis, P., Culler, D., "Mate: a Virtual Machine for Tiny Networked Sensors", ***Architectural Support for Programming Languages and Operating Systems***, (2002).

Levis, P., Lee, N., *Simulating Tinyos Networks* [online], http://www.cs.berkeley.edu/pal/research/tossim.html, **(Ziyaret tarihi: 8 Şubat 2007)**.
Li, Q., Yao, C., "Real-Time Concepts for Embedded Systems", ***CMP Books***, (2003).

Liedtke, J., "On μ-Kernel Construction", ***15th ACM Symposium on Operating System Principles***, Colorado, (1994).

Luo, J., Jha, N.K., "Battery-Aware Static Scheduling for Distributed Real Time Embedded Systems", ***Proc. 38th Design Automation Conference***, ACM Press, pp. 444-449, (2001).

MANTIS group, *Multimodal Networks of In-situ Sensors Project* [online], University of Colorado, http://mantis.cs.colorado.edu/index.php/tiki-index.php, **(Ziyaret tarihi: 25 Aralık 2005)**.

Olimex Ltd., *Olimex Limited* [online], Bulgaria, http://www.olimex.com, **(Ziyaret tarihi: 10 Temmuz 2007)**.

Ouwehand., P., *How to control a HD44780-based Character-LCD* [online], http://home.iae.nl/users/pouweha/lcd/lcd.shtml, **(Ziyaret tarihi: 8 Temmuz 2006)**.

Österlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T., "Demo Abstract: Cross-level Simulation in COOJA", ***Swedish Institute of Computer Science***, (2007a).

Österlind, F., Pramsten, E., Roberthson, D., Eriksson, J., Finne, N., Voigt, T., "Integrating Building Automation Systems and Wireless Sensor Networks", ***Swedish Institute of Computer Science***, (2007b).

Österlind, F., "A Sensor Network Simulator for the Contiki OS", ***Swedish Institute of Computer Science Technical Report***, (2006).

Pont, M.J., "Embedded C", ***Addison-Wesley***, (2002).

Ready, J.F., "VRTX: A Real-Time Operating System for embedded Microprocessor Applications," ***IEEE Micro. 6(4)***, pp.8-17, (1986).

Renesas Technology Corp, *Renesas Microcontrollers* [online], http://www.renesas.com, **(Ziyaret tarihi: 2 Haziran 2004)**.

Sandell, D., Ermedahl, A., Gustafsson, J., Lisper, B., "Static Timing Analysis of Real-Time Operating System Code", ***Dept. of Computer Science and Engineering Malardalen University***, (2003).

Sheth, A., Shucker, B., Han, R., "VLM2: A Very Lightweight Mobile Multicast System for Wireless Sensor Networks", ***IEEE Wireless Communications and Networking Conference***, New Orleans, Louisiana, (2003).

Simon, D.E., "An Embedded Software Primer. Reading", ***Addison-Wesley***, (1999).

Stankovic, J.A., Ramamritham, K., "The Design of the Spring Kernel", ***Proc. IEEE-Real-Time Systems Symposium***, pp.146-57, (1987).

Thomas, M., *Interfacing ARM controllers with Memory-Cards* [online], http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/efsl_arm/index.html, **(Ziyaret tarihi: 14 Ekim 2006)**.

UC Berkeley, *Open Source OS for the networked sensor regime Project* [online], Berkeley, University of California, http://www.tinyos.net/, **(Ziyaret tarihi: 4 Ocak 2006)**.

Ye, W., Heidemann, J., Estrin, D., "An Energy-Efficient MAC Protocol for Wireless Sensor Networks", ***In Proceedings INFOCOM***, New York, NY, USA, (2002).

Ysboodt, L., Nil, M.,D., *Embedded File System Library* [online], http://efsl.be/, **(Ziyaret tarihi: 5 Mart 2006)**.

Zhang, X., Bhuyan, L.N., ”Deficit Round-Robin Scheduling for Input-Queued Switches”, ***IEEE Journal on selected areas in communication***, vol. 21, no. 4, (2003).

ZigBee Alliance, *ZigBee Alliance* [online], http://www.zigbee.org, **(Ziyaret tarihi: 3 Kasım 2006)**.

KİŞİSEL YAYINLAR VE ESERLER

Becerikli.Y., Karan, T.M., "A New Fuzzy Approach for Edge Detection", ***International Work-Conference on Artificial Neural Networks***, LNCS 3512, pp. 943–951, (2005).

Becerikli, Y., Karan, T.M., "Bulanık Mantık Kontrollü DC Motor Sürücü", ***Türkiye Ulusal Otomatik Kontrol Kurultayı***, İstanbul, (2005).

Karan, T.M., Becerikli, Y., Turalı, T.M., "Kontrol Sistemleri için Gerçek Zamanlı Çoklu Görev İşletim Sistemi", ***Türkiye Ulusal Otomatik Kontrol Kurultayı***, Ankara, (2006).

Karan, T.M., Becerikli, Y., Turalı, T.M., "Java Destekli Gerçek Zamanlı İşletim Sistemi ve Bir Uygulama", ***Türkiye Ulusal Otomatik Kontrol Kurultayı***, İstanbul, (2007).

ÖZGEÇMİŞ

1979 yılında İstanbul'da doğdu. İlk, orta ve lise öğrenimini İstanbul'da tamamladı. 1997 yılında girdiği İstanbul Üniversitesi Kontrol Sistemleri Teknolojisinden 1999 yılında mezun oldu. Aynı yıl girdiği Kocaeli Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nden 2005 yılında mezun olduktan sonra, 2006 yılında Kocaeli Üniversitesi Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı'nda öğrenimine devam etti. 2004 yılında, Kocaeli Üniversitesi adına ilk kez, Microsoft Üniversiteler Arası .NET Proje yarışmasında Mansiyon ödülü kazandı. Uzun yıllar gömülü sistemler için donanım ve yazılım tasarımları yapan firmaların ARGE departmanlarında çalışmış olup, şu an iDeal Teknoloji A.Ş.'de ARGE bölümünde Sistem Yazılımları Uzmanı olarak çalışmaktadır.