YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GÖMÜLÜ SİSTEMLER İÇİN GERÇEK ZAMANLI İŞLETİM SİSTEMLERİ İLE KONTROL

Elektrik Müh. Erdem PAMUK

FBE Elektrik Mühendisliği Anabilim Dalı Kontrol ve Otomasyon Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı : Prof. Dr. Galip CANSEVER (YTÜ)

İSTANBUL,2007

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

E_i	Görev ‘i’ nin maksimum çalışma süresi
Ti	Görev ‘i’ nin çalışma peryodu

KISALTMA LİSTESİ

ADC	Analog sayısal dönüştürücü (Analog Digital Converter)
CPU	Merkezi işlem birim. (Central Processing Unit)
ECB	Durum kontrol bloğu (Event Control Block)
FIFO	İlk giren ilk çıkar (First In First Out)
GPO	Genel amaçlı işletim sistemi (General Purpose Operating System)
KHY	Kesme Hizmet Yordamı (Interrupt Service Routine)
KERNEL	Mikroişlemcinin zamanını kontrol ederek bütün kritik zamanlı işlemlerin mümkün olduğunca verimli bir şekilde yapılmasını sağlayan çekirdek yazılım parçasıdır
LC	Yük Hücresi (Load cell)
LCD	Likit kristal gösterge
MCU	Mikro denetleyici birimi (MicroController Unit)
PC	Kişisel bilgisayar (Personal Computer)
RAM	Rasgele erişilebilen bellek (Random Access Memory)
ROM	Sadece okunabilir bellek (Read Only Memory)
RTOS	Gerçek zamanlı işletim sistemi.
TCB	Görev Kontrol Bloğu (Task Control Block)

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Bu tezin hazırlanmasında bana destek olan aileme, ilgi ve önerilerini çalışma boyunca esirgemeyen Prof. Dr. Galip CANSEVER'e teşekkürü bir borç bilir saygılarımı sunarım.

ÖZET

Gömülü sistemler günümüzde hayatın her alanında geniş bir kullanım alanına sahiptir. Gömülü sistemler için çok basit sistemlerden çok karmaşık sistemlere kadar bir çok uygulama alanı vardır. Son zamanlarda teknolojideki gelişmeler ve mikroişlemcilerin yeteneklerinin de artmasıyla gerçek zamanlı işletim sistemleri gömülü sistemlerde de kullanılmaya başlandı.Gömülü sistemlerin bir çoğu da kritik zamanlama beklentilerine sahiptir. Bu sistemlerin en önemli problemleri kritik zamanlı işlemlerin doğru bir şekilde yapılabilmesi ve kısa zamanda tasarlanıp sorunsuz çalışmasıdır. Gerçek zamanlı işletim sistemleri de kritik zamanlama ve hızlı tasarım problemini çözmek için kullanılır.

Bu çalışmada öncelikle gömülü sistemler üzerinde durulmuştur ve günümüzde gömülü sistem tasarımında ortaya çıkan problemler belirtilmiştir. Sonraki bölümlerde işletim sistemleri, gerçek zamanlı sistemler ve gömülü sistem tasarımındaki problemin çözümü olarak düşünülen gerçek zamanlı işletim sistemleri incelenmiştir. Gerçek zamanlı işletim sisteminin detayları MicroC/OS-II işletim sistemi üzerinde anlatılmıştır. MicroC/OS-II gerçek zamanlı işletim sistemi kullanılarak bir gömülü sistem tasarımı uygulaması yapılmıştır. Sonuç bölümünde gömülü sistem tasarımında klasik yaklaşım yerine gerçek zamanlı işletim sistemi kullanmanın avantajları, dezavantajları ortaya konulmuş ve kritik zamanlı sistemlerde gerçek zamanlı işletim sistemlerinin kullanılmasının kaçınılmaz olduğu görülmüştür.

ABSTRACT

Embedded systems are being used in every part of our life and there are a lot of aplications for emnbedded systems from simple systems to complex systems. In recent days, by technological developments Real time system design by embedded systems are also increased Real time term must not be understood as very fast. Real time meaning must be thought that operation is done just on time. The main problems about this kind of systems are managing critical timings successfully, desining system as soon as possible and bug-freee. Real time operating systems are used to solve the critical timing and desing timing problems.

Embedded systems and problems on the embedded system design are discussed firstly. On the next chapters operating systems, real time systems, real time operating systems and the details of a real time operating system is explained on MicroC/OS-II. By using MicroC/OS-II design of an embedded system is realised. In conclusion chapter, the advantages and disadvantages of using real time operating system instead of backgorund/foreground system are explained and seen that using real time operating system in the time critical systems solves the timing problem.

1. GİRİŞ

Dünyanın ilk mikroişlemcisi 1971 yılında Intel tarafından üretilen 4004 adlı çiplerdi. Bu çipler bir Japon firması olan Busicom'un hesap makinelerinde kullanılmak üzere üretilmişti. Busicom her yeni hesap makinesi modelinde kullanmak üzere kendilerine özel bir dizi çip yapması için Intel e başvurdu. Intel'in cevabı 4004 adlı işlemci oldu. Her hesap makinesi için özel bir donanım tasarımı yerine, Intel bütün hesap makineleri serisinde kullanılabilecek genel amaçlı bir devre tasarladı. Bu genel amaçlı işlemci, harici bir bellekte kayıtlı komut listesini okuyup işletmek için tasarlanmıştı. Intel' in düşüncesine göre her hesap makinesine ona has özelliğini veren yazılım olacaktı (Barr, 1999).

Bu dönemden sonra mikroişlemciler birden bire patlama yaşadılari ve geçtiğimiz yıllar boyunca kullanımını sürekli arttırdılar. Trafik ışıklarından, uçuş kontrol sistemlerine, cep telefonlarından, uzaktan kumandalara kadar birçok uygulamada kullanılmaya başlandılar. 1980 lerden itibaren gömülü sistemler de mikroişlemci dalgasına katıldılar ve mikroişlemciler günlük hayattan endüstriyel hayatın her alanına kadar taşındı. Evlerde, işyerlerinde, fabrikalarda, birçok gömülü sistemin içinde onlarca mikroişlemci çalışmaktadır.

Sistemlerin basit, mikroişlemcilerin de yeteneksiz olduğu dönemde, gömülü sistem tasarımında programcı her şeyi kendisi yazmak zorunda ve bütün koda hakim olmak zorundaydı. Yazılımcılar makine dilinde programları satır satır yazıp aynı zamanda mikroişlemcinin donanımını da iyi bilmek zorundaydılar.

Silikon teknolojisindeki gelişmeler ile daha yetenekli işlemciler daha küçük boyutta, daha ucuza üretilmeye başlandı. Teknolojideki bu gelişme ile mikroişlemcilerin kullanımı daha da arttı. Artık hemen hemen her şeyin içinde bir mikroişlemci bulunur olmuştu. İşlemcilerin yeteneklerinin artması ile bu mikroişlemciler daha karmaşık sistemlerde kullanılmaya başlandı. Eskiden basit bazı giriş çıkış ve hesap işleri yaparken, sonraları birçok işi bir arada karmaşık hesaplamalarla yapmaya başladılar.

Yapılan sistemlerde karmaşık işlerle karşı karşıya kalınınca makine diliyle yazılım yapmak çok zorlaştı.ve tasarım süreci uzamaya başladı. Ayrıca büyük bir yazılımı makine diliyle yazmak sadece sahada çalışırken görülebilecek hataların çıkmasına neden olacak noktaların gözden kaçırılmasına da sebep olmaya başladı. Rekabetin inanılmaz boyutlarda olduğu bu dönemde tasarım sürecinin hızlı bir şekilde tamamlanması ve sorunsuz çalışacak cihazlar yapılması çok önem kazanmıştı. Büyük işlemcilerle yapılan sistemlerde zaten artık makine

dili kullanılmıyordu.

90’lı yılların sonlarına doğru küçük işlemciler içinde C derleyicilerinin çıkmasıyla artık yazılım yapmak kolaylaştı, tasarım süreci kısaldı ve yazılımın parçalara ayrılarak nesneler halinde yapılması okunabilirliği arttırdı. Böylece bir kişinin yaptığı yazılım başkaları tarafından da kolayca anlaşılabilir oldu. Yazılım makine dilinde yazılmadığı için kodun taşınabilirliği arttı.

C derleyicilerinin çıkıp yazılımların C ile yapılmasından sonra ve mikroişlemcilerdeki sürekli yenilikten sonra artık yazılımcılar donanıma ait bütün işleri kendileri yapmak istemez hale geldiler. İşletim sistemlerinin ilk çıkış amacı da donanım ile yazılım arasında bir ara yüz olmaktır. Ancak gömülü sistemlerde kullanılacak işletim sistemi biraz farklı olmalıdır.

Gömülü sistemlerin çoğunda zamanlama da önemli olduğundan gömülü sistemlerde kullanılan işletim sistemlerinde zamanlama önemli bir faktör olarak işin içine girdi. Gerçek zamanlı işletim sistemleri gömülü sistemlerde kullanılmaya başlanması ile sonsuz döngülü klasik bir yazılım mantığı yerine çoklu görevlerle uğraşabilen, sanki birden fazla işlemci varmış gibi çalışan yazılımlarla tasarım yapılmaya başlandı. Gerçek zamanlı işletim sistemleri artık 8 bitlik küçük işlemcilerle dahi kullanılabilmektedir. Bu tezin amacı da böyle bir işletim sistemi ve gömülü sistem tasarımı üzerine yöneliktir. Tezde öncelikle gömülü sistemler ve işletim sistemleriyle ilgili temel kavramlar, daha sonra gerçek zamanlı sistemler, gerçek zamanlı işletim sistemleri ve uygulama üzerinde durulmuştur. Tez boyunca gömülü sistem tasarımında karşılaşılan kritik zamanlamaların yakalanması ve tasarım sürecinin kısaltılması gibi problemlere çözüm aranmaya çalışılacaktır

.

2. GÖMÜLÜ SİSTEMLER

Gömülü sistemler donanım ve yazılımın sıkı bir birliktelik içinde çalıştığı ve ek başka çevre birimlerinin de olduğu belirli bir amaç için tasarlanmış sistemlerdir. Sistemin belirli bir amaç için tasarlanmış olması önemlidir, çünkü özel bir amaç için tasarlanmamış ise o sisteme gömülü sistem denemez. Evlerdeki kişisel bilgisayarlar bu durum için güzel bir örnektir. Bir kişisel bilgisayar da donanım, yazılım ve mekanik parçalara sahiptir (disk sürücüleri gibi) ancak bir kişisel bilgisayar belirli bir amaç için tasarlanmamıştır. Üretici kişisel bilgisayarın ne amaçla kullanılacağını bilmez. Bir kullanıcı network server olarak kullanırken bir diğeri sadece oyun oynamak için kullanabilir.

Gömülü (Embedded) sözcüğü, bu tip sistemlerin genelde daha büyük sistemlerin parçası olduğu için kullanılır. Büyük sistemin içine gömülmüş küçük bir sistem. Olabilecek durumların çoğunda gömülü sistemler tamamen gömülüdür yani sistemlerin içindeki sistemlerdir. Çoğunlukla kendi başlarına çalışamazlar (Barr, 1999).

Günümüzde evlerde bulunan dijital set-top box (DST) lar örnek alınırsa, DST'nin bir iç parçası olan A/V decoder olarak adlandırılan sayısal ses/görüntü çözümleme sistemi bir gömülü sistemdir. A/V decoder tek bir multimedya veri dizisi alır, ses ve görüntü olarak çıkışına verir. Uydudan DST'nin aldığı sinyalin içinde bir çok veri ve kanal mevcuttur. Bu yüzden A/V decoder transport veri dizisi decoder'i ile ilişkili çalışır ki bu da bir embedded sistemdir. Transport veri dizisi decoder'i gelen multimedya veri dizisini kanallara ayırır ve seçilen kanalı A/V decoder'e verir. Bir biriyle etkileşimli çalışan bir gömülü sisteme örnekte otomobillerdeki elektronik kontrol mekanizmalarıdır. Bu gelişmiş araçlar bir çok gömülü sisteme sahiptir. Bu sistemlerden biri frenlerin kilitlenmemesini kontrol ederken, diğeri hava yastığı ile ilgili kontrolü yapar, bir diğeri ise göstergede bilgiler gösterir (Qing ve Yao, 2003).

Bazı durumlarda gömülü sistemler tek başlarına da çalışabilirler. Bir network router'i tek başına çalışan bir gömülü sistemdir. Özel bir işlemci, bellek, network portlarından ve gelen paketleri göndermek için yazılmış algoritmalardan oluşur. Diğer bir deyişle network router'i tek başına çalışan programlanmış bir algoritmaya dayalı olarak bir portundan gelen veriyi diğer portuna veren bir gömülü sistemdir.

2.1 Gömülü İşlemciler ve Uygulamaya Özel Olma

Kişisel bilgisayarlarda bulunan işlemciler genel amaçlı işlemciler olarak tabir edilirler.

Tasarımları karmaşıktır, çünkü bu işlemciler geniş bir yelpazedeki özelliklere ve işlevselliğe sahiptir. Birçok uygulama için kullanılabilecek şekilde tasarlanmıştır. Örneğin modern işlemciler bellek koruması ve sanal bellek sağlaması için kendi içlerinde bellek yönetimine sahiptir. Bu evrensel işlemciler gelişmiş cache mantığına sahiptir. Ondalıklı matematiksel işlemleri kolay yapabilmek için kendi içlerinde matematik yardımcı-işlemcileri bulunur. Bu işlemciler birçok harici aygıt ile haberleşebilmek için çevre birimlerine sahiptir. Yüksek güç tüketirler, ısı yayarlar, boyutları büyüktür ve fiyatları daha pahalıdır.

Eski günlerde gömülü sistemler genel amaçlı işlemciler kullanılarak yapılmaktaydı Mikroişlemci teknolojisindeki gelişmeler sayesinde gömülü sistemler gömülü işlemcilerle yapılır oldu. Gömülü işlemciler belirgin uygulamalar için tasarlanmış özel amaçlı işlemcilerdir. Buradaki anahtar sözcük uygulamaya özel olma tanımıdır. Gömülü sistemler belirli bir amaca yönelik tasarlandıkları için bu sistemlerde bu amacı sağlamaya yönelik özelliklere sahip olan işlemciler (gömülü işlemciler) kullanılır.

Bazı gömülü işlemciler boyut, güç tüketimi ve fiyat üzerine yoğunlaşmıştır. Bununla birlikte bazı gömülü işlemciler fonksiyonel olarak kısıtlıdırlar, tasarlandığı uygulama için çok kullanışlı iken başka bir uygulama için tamamen yetersiz olabilirler. Bu yüzden bir çok gömülü işlemci yüksek CPU hızlarına sahip değildirler. 8-16 bitlik küçük işlemciler uygulamaya yönelik tasarlanmışlardır. Bazıları CAN BUS arabirimi, motor kontrol arabirimi, LCD gösterge sürme özelliklerine sahiptir. Kısıtlı özellik demek güç tüketiminin azaltılması ve pil ömrünün uzaması demektir. Küçük boyutlu oldukları için üretimi daha kolay ve fiyatları daha düşüktür.

Bazı gömülü işlemciler ise performans, boyut, güç tüketiminin hepsine birden yoğunlaşmıştır. Cep telefonlarında kullanılan gömülü sayısal işaret işleyiciler (DSP) örnek olarak alınabilir. Gerçek zamanlı ses haberleşmesi sayısal işaret işlemciyi gerektirir ve gecikme kabul etmez. Bir DSP aritmetik birimlerle özelleştirilmiş, bellek tasarımında optimize edilmiş ve kompleks hesaplamaları gerçek zamanda hızlı yapabilmek için bus yapısına ve çoklu işlem kapasitesine sahiptir. Bir DSP aynı hızda çalışan bir genel-amaçlı işlemciye göre sayısal işaret işlemede çok daha üstündür ve daha az güç tüketir.

2.2 Donanım ve Yazılımın Birlikte Tasarımı Modeli

Genellikle bir gömülü sistem için donanım ve yazılım paralel olarak geliştirilir. Yazılım ve donanım tasarım ekipleri arasında bir iletişim olur. Her iki taraf diğerinin ne yapabileceğini

bilme avantajına sahiptir. Yazılım takımı yüksek performansa ulaşabilmek için tasarlanmış donanımın özelliklerinden faydalanır. Donanım takımı yazılım tarafından başa çıkılabilecek bir özellikse donanımın karmaşıklaşmasına ve fiyatın artmasına engel olarak tasarım yapar. Böyle bir işbirliği ile tasarım yapıldığında uygulamaya özel bir sistem ortaya çıkar (Qing ve Yao, 2003, Tomiyama, vd., 2005).

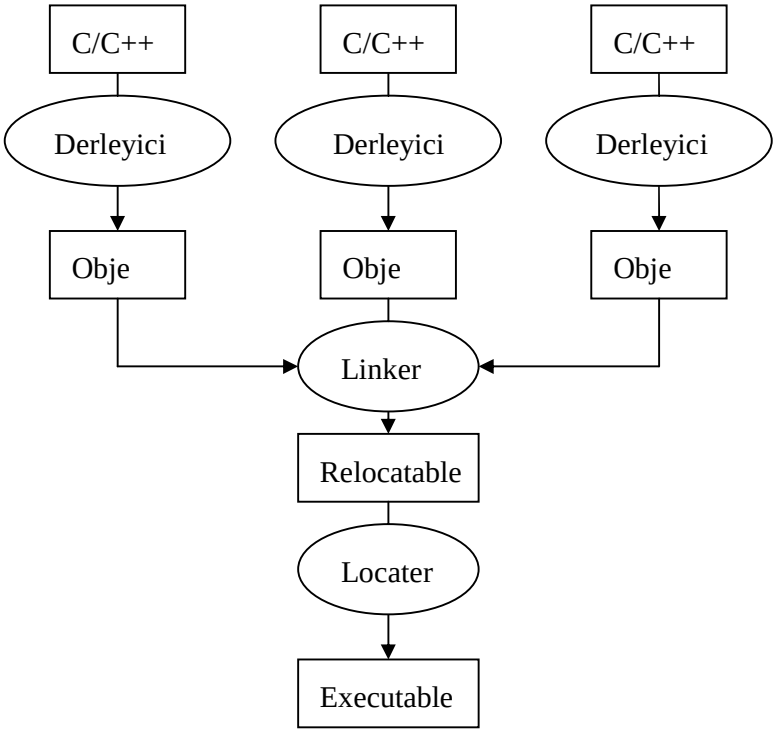
2.2.1 Yazılım Tasarımı

Muhakkak ki yazılım tasarımı gömülü sistemler için çok önemli bir parametredir. Gömülü sistemlerde kullanılan küçük işlemciler için dahi C derleyicilerin çıkmasıyla yazılım tasarım süreci giderek azalmıştır. Bir yandan da işlemcilerin giderek gelişmesiyle daha karmaşık yazılımlar yapmak gerekliliği ortaya çıkmıştır. Yazılım karmaşıklaştıkça tasarım süreci giderek uzar. Tasarım sürecini kısaltmak için yazılım tasarımında şu noktalara dikkat edilmelidir.

- Yazılımın sonunda oluşacak sistemin parametreleri iyi belirlenmelidir.
- Yapılacak işler parçalara bölünmelidir.
- Yazılan kod taşınabilir olmalıdır.
- Algoritma iyi tasarlanmalıdır.
- Kod açık ve anlaşılır olmalıdır.
- Mümkün olduğunca hatadan ayıklanmış olmalıdır.

2.2.2 Hedef Platforma Yükleme

Yazılım tasarımı yapıldıktan, gömülü yazılımı çalıştırılabilir binary image dosyasına dönüştürme işlemi üç adım içerir. Öncelikle her bir kaynak dosyası derlenmeli yada bir derleyici tarafından obje dosyasına çevrilmeli, ikinci olarak ilk adımdan gelen bütün obje dosyaları birleştirilerek tek bir obje dosyası oluşturulur. Son olarak fiziksel bellek adresleri de dosyanın içine eklenir. Böylelikle süreç tamamlanır. Süreç sonunda ortaya çıkan dosya işletilebilir ikilik düzendedir ve gömülü sistem üzerinde çalışabilir (Barr, 1999).



Şekil 2.1 Kodun oluşturulma aşamaları

Şekilde üç adım yukarıdan aşağıya görülmektedir. Elips şeklindeki kutular yazılım araçlarını göstermektedir. Bu araçların her biri bir yada daha fazla dosyayı alıp çıkışlarında tek bir dosya verirler.

Gömülü yazılım geliştirme sürecindeki bütün adımlar genel amaçlı bir bilgisayar üzerinde koşan yazılımın dönüştürme işidir.

2.2.2.1 Derleme

Bir derleyicinin amacı temel olarak insanlar tarafından okunabilen bir dille yazılmış olan programları belirgin bir işlemci için uygun olan opcode setlerine çevirmektir. Bu durumda assembler de bir derleyicidir, sadece insanlar tarafından okunabilen kodu bire bir eşdeğeri olan opcode lara çevirir. Bu araçlar gömülü yazılım oluşturmanın birinci adımıdır. Tabi ki, her işlemci kendine has bir makine diline sahiptir, bu yüzden işlemcinize uygun programları oluşturacak bir derleyici seçilmelidir. Gömülü sistemlerde bu derleyici çoğunlukla bilgisayarda koşar. Gömülü sistemin kendisinde derleyici işletmek pek mantıklı değildir. Bir bilgisayar platformu üzerinden koşarken başka bir platform için kod üretilen derleyicilere

cross-compiler denir

Giriş dili (C/C++,assembly yada başka) ne olursa olsun cross-compiler ın çıkışı bir obje dosyası olacaktır. Bu dosya özel olarak biçimlenmiş bir binary dosyadır, komut setlerini ve diğer dilden çevrilen verileri içerir. Bu dosyanın parçaları çalıştırılabilir kodlar içerse de, bu dosya tek başına çalıştırılamaz.

Birçok obje dosyası takip eden kısımları açıklayan bir başlık ile başlar. Bu kısımların her biri bir yada daha fazla kod blokları yada orijinal kaynak dosyadan alınan verileri içerir. Bununla birlikte derleyici bu grupları ilgili kısımlara göre yeniden gruplar Örneğin bütün kod bloklarını text olarak adlandırılan kısma, ilk değer verilmiş global değişkenleri ve ilk değerlerini data kısmına ve ilk değer verilmemiş global değişkenleri bss diye adlandırılan kısımda toplar.

Kaynak dosyanın içinden alınan fonksiyon ve değişkenlerin isimlerinin ve yerlerinin olduğu sembol tabloları olabilir. Bu tablonun bazı kısımları tamamlanmamış olabilir çünkü bütün değişkenler ve fonksiyonlar her zaman aynı dosyada tanımlanmaz. Bunlar başka kaynak dosyalarda tanımlanmış değişkenler ve fonksiyonların sembolleridir. Bu gibi çözümlenmemiş kaynakları bulmak linker'ın görevidir.

2.2.2.2 Linker

Birinci adımdan elde edilen bütün obje dosyalarının çalıştırılabilmesi için özel bir adımdan geçirilmelidir. Linker'in görevi bu obje dosyalarını birleştirmek, işlemek ve çözümlenmemiş sembolleri çözümlemektir.

Linker'ın çıkışı yine bir obje dosyasıdır. Kodu ve veriyi girişindeki obje dosyalarından almış ve yine aynı biçimde bir obje dosyası oluşturmuştur.Giriş dosyasının text, data ve bss kısımları birleştirilir. Linker çalışmasını bitirdiğinde giriş dosyalarındaki bütün makine dili kodları yeni dosyanın text kısmında, bütün ilk değer verilmiş ve verilmemiş değişkenleri yeni dosyanın dat ve bss kısımlarında olacaktır.

Linker kısımların içeriğini birleştirirken çözümlenmemiş sembollere de bakar. Örneğin bir obje dosyasında o obje dosyasında tanımlanmamış olan bir değişken (foo) kullanılmış ise ve başka bir obje dosyasında da bu değişken tanımlanmış ise linker bu iki değişkeni eşleştirecektir. Eğer bir sembol birden fazla obje dosyasında tanımlanmış ise linker hata mesajı verecektir.

Bütün kodları ve veri kısımlarını birleştirdikten , bütün sembolleri çözümledikten sonra linker programın bir “relocatable” bir kopyasını oluşturur. Artık program bir şey hariç tamamlanmıştır; bellek adresleri ve kodlara ve verilere uygulanmamıştır.

2.2.2.3 Locater

Relocatable programı çalıştırılabilir binary image dosyasına çeviren araç’a Locater denir. Locater kodlar ile bu kodların bellekte tutulacağı adresleri eşleştirir. Locater sonunda elde edilen program doğrudan gömülü sistemin ROM’una yüklenir.

3. GERÇEK ZAMANLI GÖMÜLÜ İŞLETİM SİSTEMLERİ

3.1 İşletim Sistemlerinin Amacı ve Genel Amaçlı İşletim Sistemleri

Bilgisayar sistemlerinin ilk günlerinde uygulama geliştiriciler işlemcinin ve diğer donanımların durumundan ve kontrolünden tamamen sorumluydu. İlk işletim sistemlerinin amacı programcıların işini kolaylaştıracak sanal bir donanım platformu olmak idi. İşletim sistemleri uygulama yazılımları ile donanım arasında arayüz sağlar, kullanıcının anlayacağı yüksek seviye ile donanımın anlayacağı düşük seviye arasında etkili bir haberleşme sağlar. Kullanıcı için donanımın erişimini kolaylaştıran bir API gibidir.

Bir işletim sistemi kaynakları yönetir (CPU, bellek, …), tuş takımı girişlerini algılar, diskteki klasörlerin ve dosyaların izlerini takip eder. Çevre birimleri ile haberleşir. Gömülü işletim sistemleri de uygulamayı ve işletim sistemini tek parça halinde birleştirir ve aygıtın ROM'unda tutar.

Gömülü sistemlerin ihtiyaçlarını karşılamak için birçok ticari ve açık-kodlu Gerçek Zamanlı İşletim Sistemi (RTOS) geliştirilmiştir. RTOS lar ile Genel Amaçlı İşletim Sistemi (GPO) arasında benzerlik ve farklar bulunmaktadır. Bu farklar RTOS ların gerçek zamanlı gömülü sistemler için neden uygun olduğunu gösterir (Qing ve Yao, 2003).

RTOS lar ile GPO lar arasındaki temel benzerlikler;

- Belli oranda multitasking
- Yazılım ve donanım kaynaklarının yönetimi
- Donanımı yazılımdan soyutlamak

Farklar ise ;

- Gömülü uygulamalar için daha güvenilirlerdir.
- Uygulamanın ihtiyaçlarına göre boyutlandırılabilirler
- Performansları daha iyidir.
- Bellek ihtiyaçlarını azaltırlar.
- Zaman planlama (Scheduling) algoritmaları gerçek zamanlı gömülü sistemler için

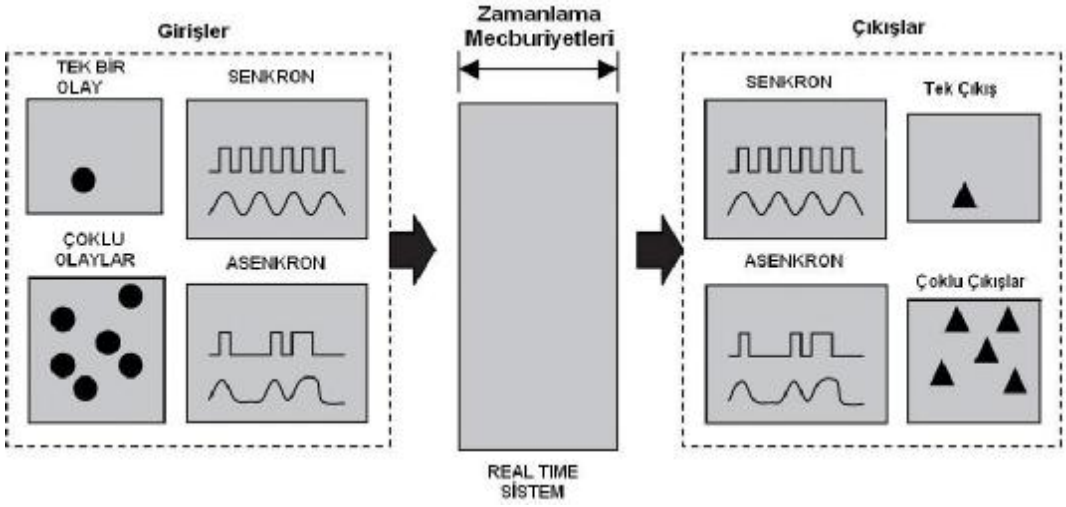
uyarlanmıştır.

- Disksiz gömülü sistemler için ROM yada RAM den doğrudan çalıştırılabilirler.
- Farklı donanım platformlarında kullanılabilmek üzere daha iyi taşınabilirliğe sahiptir.

Günümüzde GPO lar genel amaçlı bilgisayarlarda, kişisel bilgisayarlarda ve büyük bilgisayarlarda kullanılırlar. Diğer yandan RTOS lar güvenilir, kompakt, ölçeklenebilir ve gömülü sistemlerde daha iyi performans gösterirler. Ayrıca sadece uygulama için gereken kısımlar kullanılarak bellek kazanımı da sağlarlar. Ancak birçok küçük gömülü aygıt hala RTOS olmadan tasarlanmaktadır. Bu tezde ağırlıklı olarak küçük gömülü sistemler için uygun olan Gerçek Zamanlı İşletim Sistemlerinden bahsedilecektir (Qing ve Yao, 2003).

3.2 Gerçek Zamanlı Sistem Nedir ?

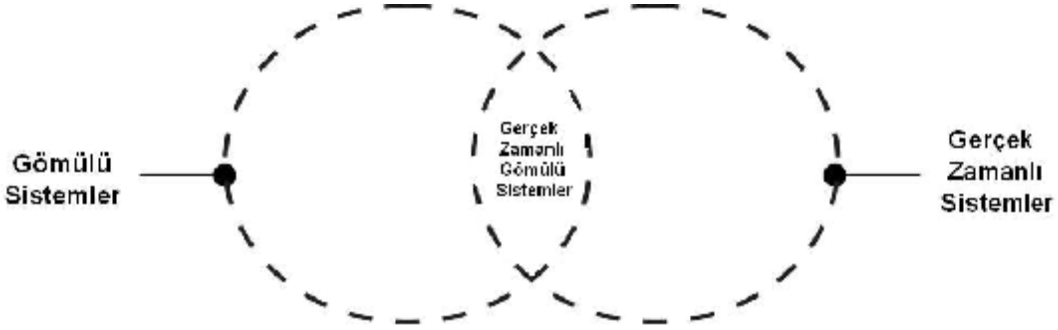
Genel bir tanım ile Gerçek Zamanlı Sistemler zamanlama kısıtları yada zorunlulukları olan sistemlerdir. Bu sistemler harici olaylara zamana bağlılık ekseninde cevap verirler. Cevap verme süreleri garantidir.



Şekil 3.1 Gerçek zamanlı sistemlerin basit bir blok diyagramı

Harici olaylar senkron yada asenkron karakteristiklere sahip olabilirler. Bu harici olaylara cevap verme; olayın algılanmasını, bu olayla ilgili işlemlerin yapılmasını ve gerekli sonuçların doğru zamanlarda çıkışa verilmesini kapsar. Gerçek zamanlı sistemler ile gömülü sistemler arasındaki ilişki Şekil3.2 den görüldüğü üzere gömülü sistemlerle gerçek zamanlı sistemlerin kesişim bölgesi olmasına rağmen ne bütün gömülü sistemler gerçek zamanlıdır ne

de bütün gerçek zamanlı sistemler gömülü sistemdir.



Şekil 3.2 Gerçek zamanlı gömülü sistem

Tahmin edilebilir sistemler elde edilen sonuçların alınmasına ve doğruluğuna bağlıdır. Gerçek zamanlı sistemlerin doğruluğu zamanlamalara uyulmasıyla ölçülür. Bu tip sistemler zamana bağlı olarak hesaplamalar ve kesin kararlar verebilme yeteneğine sahiptirler.Bu önemli hesaplamalar kritik bitiş zamanlarına (deadline) sahiptirler. İşler bu zaman zarfında yapılmak zorundadır. Zamanında yapılmayan bir hesaplama yanlış hesaplama kadar kötüdür. Gerçek zamanlı sistemler için sistemin tümünün doğruluğu işlevsel doğruluk kadar önemlidir. Bir gerçek zaman mimarisi kesin zamanlamalara ulaşmaya engel olmamalıdır.

Bir sistemi mimarilendirirken yapılan seçimler kaynak paylaşımında ve yapılan zaman planlaması sonunda istenen değerlerin elde edilmesinde etkin rol oynar. Bir çok mimari ve metodoloji, cevap verme süresini ekonomik olarak düzeltmek güç olduğu için ihmal eder.

3.2.1 Gerçek Zamanlı Sistemlerde Zaman Beklentilerinin Seviyeleri

Zamanlama tahminlerinin beş kategorisi vardır. Her kategori farklı seviyelerde beklentilere sahiptir. Daha da önemlisi her kategori farklı altyapı ve haberleşme teknikleri kullanır. Bir kategori için kullanılan mimari yapılar ve teknikler diğer bir kategori için uygun olmayabilir. Aşağıda listelenen son iki teknik kesin cevap verme sürelerini garanti eder fakat çok farklı maliyetlere ve hata karakteristiklerine sahiptir (Gerhardt ve Locke, 2005).

Nicel ölçümler : Nicel ölçümler tekrar edilebilir de edilmeyebilir de. Kısa süreli anlık ölçümler yanlış yönlendirici olabilir. Örneğin 3 gün boyunca yapılan sıcaklık ölçümleri yarının sıcaklığı için yeterli hazırlığı sağlamayacaktır.

Tekrarlı ölçümler : Ölçümlerin alındığı ortamdaki bilgi kullanılabilir, bu bilgilerle tahmin

yürütülebilir. Bölgenin bilinmesi, günün hangi saatinde alındığının bilinmesi, tarihin bilinmesi, üç günlük ölçüm değerine bakarak dördüncü günün ölçüm değerinin tahmin edilmesinde çok faydalı olur. Bu ölçüm bir yazılımın çalışma süresini ölçmek ise, arka planda gecikme yapabilecek yada zaman alacak şeyleri de hesaba katarak tahmin yapmak daha kolaydır.

İstatistiki olarak tahmin edilebilir mimari : İstatistiki karakteristikler ortalama cevap süresini ve ilgili standart sapmaları verirler. İstatistiki tahmin sistemlerine uygulanan mimarilerde, kuyruk (FIFO) yapıları, asenkron mesajlaşma, olaylara tepki veren yapılar bulunur.

Analitik olarak garanti edilmiş gecikme süresi :

Gecikme süresi bir olayın olmasıyla sistemin bu olaya cevabının arasındaki beklemedir. Garanti edilen sınırlandırılmış gecikme sürelerinin içinde kaynakların paylaşımı da vardır. Kaynaklar istemciler tarafından kullanılan ve kullanılırken başkasının kullanımına karşı kilitlenen yapılardır. Gerçek zamanlı işletim sistemleri rate monotonic analiz, önceliklere bakarak kaynaklara erişim gibi analitik yapıları kullanırlar (Li, vd. 1997).

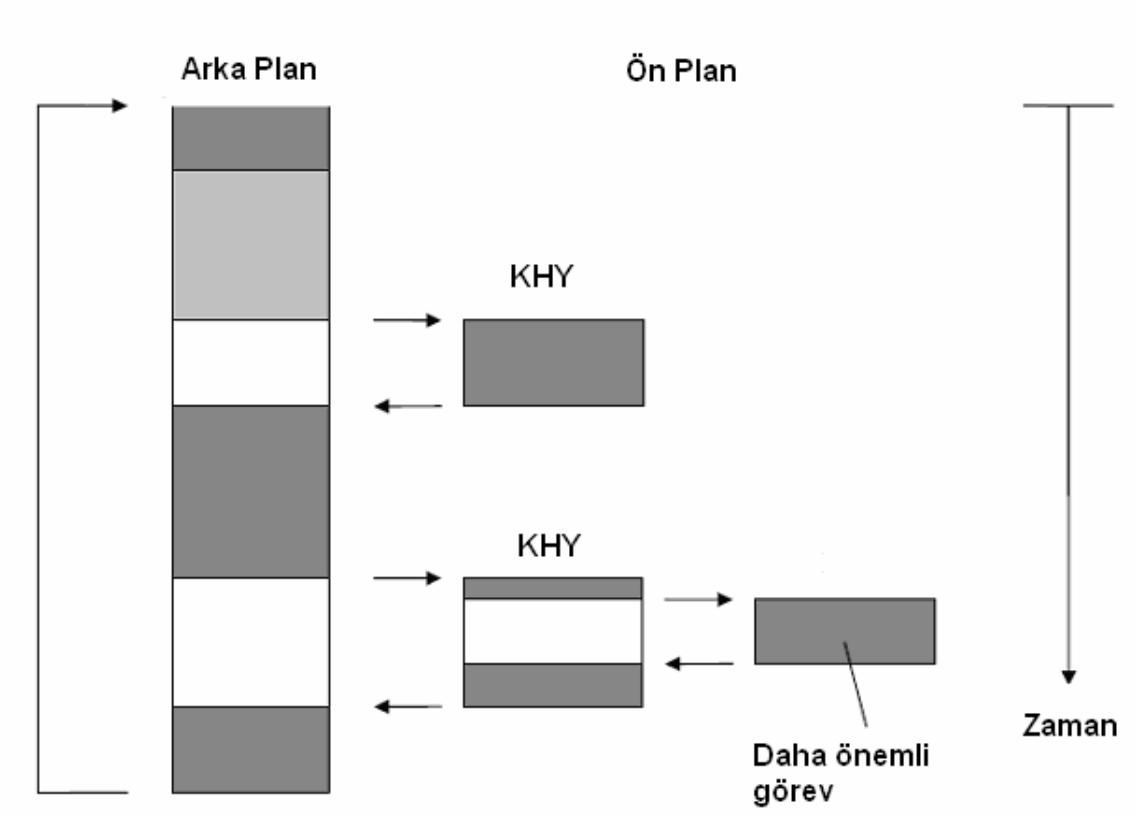
Belirginlik : Belirgin bir mimari sistemin zaman içinde geçebileceği her durum ve olaya vereceği cevaplar hakkında bilgi sağlar.

3.3 Gerçek Zamanlı İşletim Sistemlerinin Gömülü Sistemlerde Kullanılması

Bir .çok küçük gömülü uygulama Ön plan / Arka plan (foreground/backgorund) olarak adlandırılan bir yazılım mantığı ile tasarlanır. Küçük gömülü sistemlerin çoğu motor kontrolü, akıllı cihazlar, tüketici elektroniği, robotlar, haberleşme cihazları gerçek zamanlı multitasking işletim sistemlerinden faydalanabilir. Birçok sitem bir işletim sisteminin dosyalama sistemini, I/O yönetimini ve networking gibi tüm özelliklerinin ne karmaşıklığını ne de maliyetini kaldıramaz. Bu durumda asgari özelliklere sahip öncelikliğe dayalı multitasking yapısı olan gerçek zamanlı bir işletim sistemi kullanılabilir.

Mikroişlemciler bir birim zamanda tek bir işlem yapabilirler fakat bu işi çok hızlı yapabilirler. Bir kontrol ve izleme sistemi tasarlarken geleneksel yaklaşım ön plan/arka plan denen tekniktir. Şekil 3.3 de gösterilen sistem bir sonsuz döngüye sahiptir (arka plan). Bu kısım modülleri çağırarak gereken işlemlerin yapılmasını sağlar. Kesme hizmet yordamında sıralı işletilen modüllerle de asenkron olaylarla ilgilenilir. Kritik işlemlerin zamanında

yapıldığından emin olmak için bu işlemler KHY ler içinde yapılır. Bu yüzden KHY ler olmaları gerektiğinden daha uzun olmaya meyilli olurlar; bir KHY basitçe kesme oluşturan kaynaktan veriyi ve durumunu alıp bununla ilgili işlemi daha sonra yapılmak üzere bir göreve bırakır (Labrosse, 2005)



Şekil 3.3 Ön plan /arka plan sistem

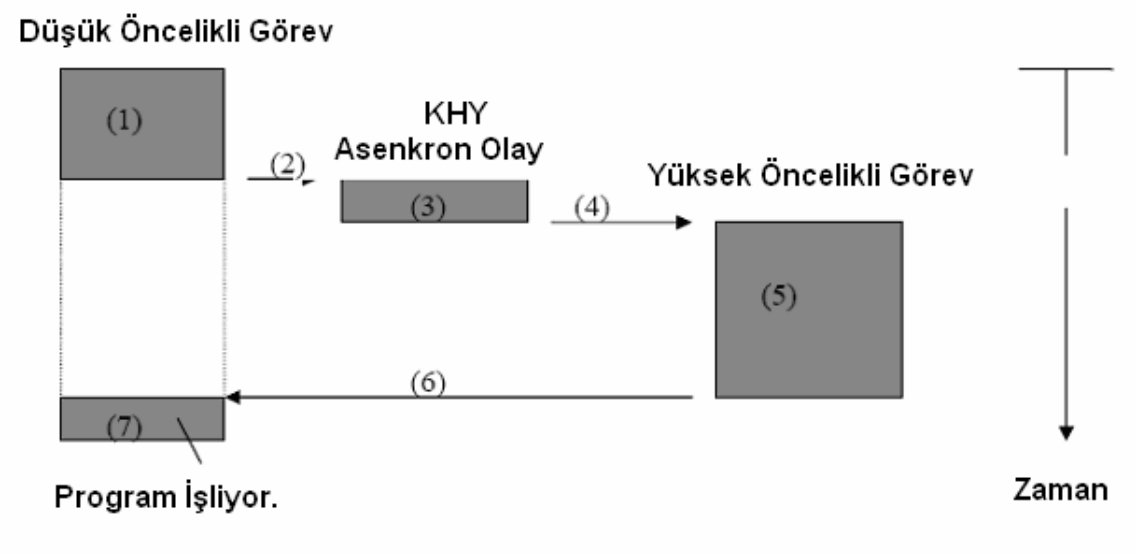
Bir KHY tarafından arka plan için elde edilen bilgi, arka plan döngüsünde sırası gelene kadar işletilmez. Buna task level response denir. Oluşabilecek en kötü gecikme ana döngünün bir tam işletilme süresi kadardır. Tipik kodun işletilme süresi sabit olmadığından, bir döngünün ardışıl döngü süreleri de farklı olabilir.

Ön plan /arka plan sistemler bütün kritik işlemlerin arka plan döngüsünün en kötü zamanından daha kısa bir sürede yapıldığı sistemlerde uygundur. Ne yazık ki en kötü durumdaki döngü süresini tahmin etmek çok zordur. Ön plan/arka plan sistemleri yeni fonksiyonların eklenmesi açısından da kötüdür. Klasik yaklaşımla tasarlanan mikroişlemcili sistemler ön plan/arka plan sistemlerin tipik bir örneğidir. Haberleşmeler ve zamanlamalar ön plan da yapılırken kullanıcının kodu da arka planda koşar (Labrosse, 2005).

Gerçek zamanlı kontrol ve izleme sistemleri için Şekil 3.3 de gösterilen den daha iyi bir yazılım yaklaşımı Şekil 3.4 ve Şekil 3.11 de gösterildiği gibi görevlerin birbirinin içine girebildiği gerçek zamanlı multitasking bir çekirdek, diğer bir deyişle gerçek zamanlı işletim sistemi kullanmaktır. Çekirdek; mikroişlemcinin zamanını kontrol ederek bütün kritik zamanlı işlemlerin mümkün olduğunca verimli bir şekilde yapılmasını sağlayan bir yazılım parçasıdır ve işletim sisteminin çekirdeğidir.

Çekirdek kullanılması sistemin tasarım sürecini basitleştirir çünkü çekirdek sistemin birbirinden bağımsız çoklu görevlere bölünmesine müsaade eder. Bir görev basit bir programdır ve mikroişlemcinin tamamen kendisine ait olduğunu düşünür. Her bir görev önemine göre önceliklendirilir. Böylelikle gerçek zamanlı sistemin tasarımı artık problemin parçalarından sorumlu olan görevleri oluşturmak olur. Mikroişlemci hala aynı hesap gücüne sahiptir fakat işler önceliklendirilebilir. Bir çekirdek ayrıca uygulama için zaman gecikmeleri, sistem zamanı, mesajlaşma, senkronizasyon ve ortak kaynakların kullanımı gibi önemli hizmetler sağlar.

Şekil 3.4 de gösterilen sistemde görevin çalışmasını durdurup diğer görevin çalışmasını sağlayabilen bir çekirdek (preemptive – müdahale edilebilir yapı) kullanılmıştır. Önce düşük öncelikli görev işler durumdadır (1). Asenkron bir olay mikroişlemciyi kesmeye uğratır (2). Mikroişlemci kesme hizmet yordamına gider (3) ve bu kesme içinde yüksek öncelikli bir görev çalışmaya hazır hale getirilir. Kesme sonunda çekirdek yüksek öncelikli görevin yürütülmesine karar verir (4). Yüksek öncelikli görev, bitene yada kesmeye uğrayana kadar işletilmeye devam eder (5). Görevin sonunda çekirdek düşük öncelikli görevi tekrar çalıştırır (6). Düşük öncelikli görev çalışmaya devam eder (7) (Labrosse, 2005).



Şekil 3.4 Preemptive çekirdek örneği

Bir çekirdek kritik zamanlı görevlerin önce işletileceğini garanti eder. Başka bir deyişle bir çekirdek olmadan kritik zamanlı işin önce yapılması söz konusu değildir, çünkü o anda kritik zamanlı olmayan bir işlem gerçekleştiriliyor olabilir ve bu işlem bölünemez. Üstelik kritik zamanlı görevlerin yürütülmesi belirgindir ve nerdeyse yazılım değişikliklerine duyarsızdır.

3.3.1 Yazılımın Görevler Halinde Yapılarak Sistemin Tasarlanması

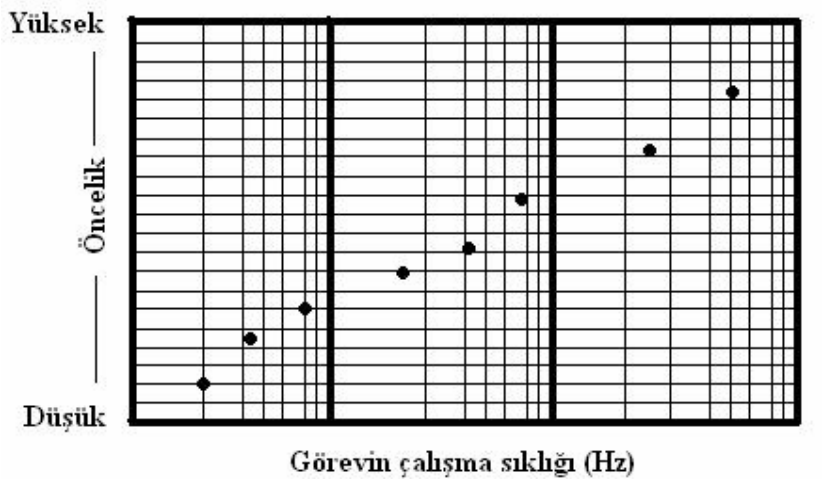
Çekirdek işlerin biririnden bağımsız çoklu görevlere dağıtılıp yapılmasını sağlamak ile tasarımı basitleştirir. Aşağıdaki görevlerin hepsi bir yada daha fazla göreve bölünebilir.

- Tuş tarama
- Operatör arayüzleri
- Gösterge (LED, LCD,…)
- Seri haberleşme
- Analog girişlerin okunup kayıt edilmesi ve gösterimi
- Kontrol döngüleri (PID, Fuzzy lojik …)

Her bir görev kendi yığınını gerektirir. Çekirdek'e bağlı olarak her bir görev için aynı boyut ayrılmış olabilir yada kullanıcıya boyutları tanımlama fırsatı verilmiş olabilir. Genellikle ikinci yöntem daha çok tercih edilir, çünkü her bir görev farklı boyutta yığın ihtiyacı duyabilir

(Qing, ve Yao, 2003).

Ayrıca her bir göreve öncelik verilmelidir. Görevlerin önceliklerini belirlemek gerçek zamanlı sistemlerin karmaşık yapısından dolayı önemsiz bir iş olarak görülmemelidir. Birçok sistemde bütün görevler kritik değildir. Kritik olmayan görevlere düşük öncelikler verilmelidir.Birçok gerçek zamanlı sistem yumuşak ve katı (Hard / Soft)gereksinimlerin kombinasyonuna ihtiyaç duyar. Soft gerçek zamanlı sistemlerde görevler sistem tarafından mümkün olduğunca hızlı bir şekilde yapılmaya çalışılır fakat kesin bir zamanda bitmesi gerekmez. Hard gerçek zamanlı sistemlerde görevler tam zamanında ve doğru yapılmış olmalıdır. İlginç bir teknik olan Rate Monotonic Scheduling, görevlere kullanılma sıklığına göre öncelik vermek için oluşturulmuştur (Li, vd., 1997). Basitçe en çok kullanılan görev en yüksek önceliğe sahiptir.



Şekil 3.5 Görev çalışma sıklığı – öncelik grafiği

RMS bazı varsayımlar yapar.

- Bütün görevler periyodiktir (düzenli aralıklarla oluşur)
- Görevler diğerleriyle senkronize olmaz, kaynak paylaşmaz yada veri alıp vermez
- CPU her zaman çalışmaya hazır en yüksek öncelikli görevi işletir. Diğer bir değişle bir görevi keserek diğer görevi çalıştırabilen bir çekirdek kullanıyordur.

N adet görevin RMS ile önceliklendirileceğini düşünelim, temel RMS teoremi eğer aşağıdaki eşitlik kullanılırsa bütün hard bitiş zamanlarının her zaman yakalanacağını belirtir.

$$\frac{E_1}{T_1}+\frac{E_2}{T_2}+\frac{E_3}{T_3}+...+\frac{E_n}{T_n}\leq n\left(2^{\frac{1}{n}}-1\right) \quad (3.1)$$

Ei görev ‘i’ nin maksimum çalışma süresini belirtir. Ti görev ‘i’ nin çalışma periyodunu belirtir. Diğer bir değişle Ei/Ti görev ‘i’ nin işletilmesi için gereken CPU zamanını verir. Çizelge 3.1 görevlerin numaralarına göre $n(2^{1/n}-1)$ tabanında değerlerini verir. Sonsuz sayıdaki görevlerin üst limiti ln(2) yada 0,693 ile verilmiştir. Bunun anlamı RMS’ye göre bütün hard gerçek zamanlı kritik bitiş sürelerini yakalamak için bütün kritik zamanlı görevlerin CPU kullanımı %70 den az olmak zorundadır. Şu da göz önünde bulundurulmalıdır ki, sistemde hala kritik zamanlı olmayıp CPU’nun zamanının %100’ünü kullanan görevler olabilir. CPU’nun zamanının %100’ünü kullanmak, istenmeyen bir davranıştır, çünkü bu durum kod değişikliklerine ve eklerine izin vermez. Kabul edilen temel bir kural olarak CPU’yu %60 ve %70 ten fazla kullanan sistemler tasarlanmamalıdır (Labrosse, 2005).

Çizelge 3.1 Görev sayısına göre CPU’nun işletilme sıklığı

Görev sayısı	$n(2^{1/n}-1)$
1	1.000
2	0.828
3	0.779
4	0.756
5	0.743
-	-
-	-
-	-
∞	0.693

Bir görev temel olarak Şekil 3.6’da görüldüğü gibi bir sonsuz döngüdür. Çekirdek in diğer görevlerin çalışmasına da izin verebilmesi için, çekirdek tarafından belirli bir olayın olmasını bekleyen bir hizmet olmalıdır. Bu olay bir zamanın, başka bir görevin yada KHY’den bir işaretin beklenmesi olabilir. Bazı çekirdekler görev ilk çalışmaya başladığında göreve bir bilgi vermeyi de mümkün kılar. Bu C deki “argc” ve “argv” argumanlarının main()’e gönderilmesi gibidir. Bu özellik bir verinin belirgin özelliklerini alan genel bir görev yazılmasına imkan

sağlar. Örneğin kendisinin kontrol ettiği port hakkında bilgiler (port adresi, Kesme numarası, baud rate,…) alan genel bir seri haberleşme yönetim görevi yazılabilir.

```
void Task (void *arg)
{

    /* görev argumanı ile bişeyler yapar    */
    while (1) {
    /* uygulamaya özel bazı kodlar          */
    /* bir olay için bekler                 */
    /* kernel tarafından                    */
    /*      1) n sistem tiki kadar bekle    */
    /*      2)bir semaforda bekle           */
    /*      3)bir görevden yada ISR den mesaj bekle*/
    /*      4) diğer işler                  */
    /*uygulamaya özel bazı kodlar           */
    }

}
```

Şekil 3.6 Bir görevin pseudo kodlarla gösterimi

Birçok çekirdek görev yaratılmasına imkan sağlar. Bir görev statik (yazılımın kontrolü çekirdek'e verilmeden önce) yada dinamik (uygulama çalışırken) olarak yaratılabilir. Bir görevin yaratılması demek basit anlamda çekirdek' e o görevi yürütmesi gerektiğini söylemektir. Gömülü sistemlerde çoğu görevler statik olarak yaratılır. Çekirdekler ayrıca görevleri dinamik olarak yok etmeye de imkan sağlayabilir. Artık ihtiyaç duyulmayan bir görev bu özellik ile boşa çıkarılabilir. Örneğin sistem bir acil stop'a ihtiyaç duymuş olabilir. Bu durumda kontrol döngüleri (görevleri) durdurulup yok edilir ve bir "kapanma görevi" çıkışları güvenli konuma getirir.

3.3.2 Gecikmelerin Kullanılması

Hemen her çekirdek tarafından sağlanan en temel hizmet gecikmelerdir. Çekirdekler yazılımcının onlara sistem tiki (sistemin ana saat işareti) denen bir periyodik Kesme sağlamasını ister. Bu saat işareti genel olarak her 10 ile 100ms arasında CPU'yu kesmeye uğratır. Belli bir sistem saat işareti kadar zaman sonra bir görevin kendini askıya almasını çekirdekler bu şekilde sağlar. Bu durum gerçekleştiğinde çekirdek'in listesinde çalıştırılmak için bekleyen en önemli görev çağrılır. Bir görevin durdurulup CPU'nun diğer görevi çalıştırmaya başlamasına "görev değişimi" denir. Görev değişiminden sonra durdurulan görev zamanının gelmesini beklerken nerdeyse hiç CPU zamanı harcamaz. Bu noktada eğer bu görev en yüksek öncelikli görev ise çalışmaya kaldığı yerden devam edecektir. Basit bir gecikmeiçin sözde kod (Şekil 3.7)aşağıdadır.

```
/* Görev kodu                                                    */
OStimeDly(10);   /* görevi 10 sistem tiki süresi boyunca beklet  */
/* Görev kodu 10 sistem tiki süresi bekledikten sonr tekrar çalışır */
```

Şekil 3.7 Gecikme fonksiyonu

Bu basit mekanizma bir tuş tarama işinde kullanılabilir (Şekil 3.8). Eğer sistem tiki her 20ms de bir geliyorsa saniyede 25 kere tuş taraması yapılabilir.

```
void KeyScanTask (void *arg)
{
    while (1) {
        OSTimeDly(2);   /* her 2 sistem tiki süresinde tuş taraması yap */
        /* tuş taraması ile ilgili kod                                   */
    }
}
```

Şekil 3.8 Tuş tarama işinde gecikme kullanılması

Bu yöntem ile kontrol döngülerinin işletilmesinin planlanması, belirgin aralıklarda analog girişlerin okunması ve belirgin aralıklarda analog çıkışların güncellenmesi yapılabilir (www.micrium.com).

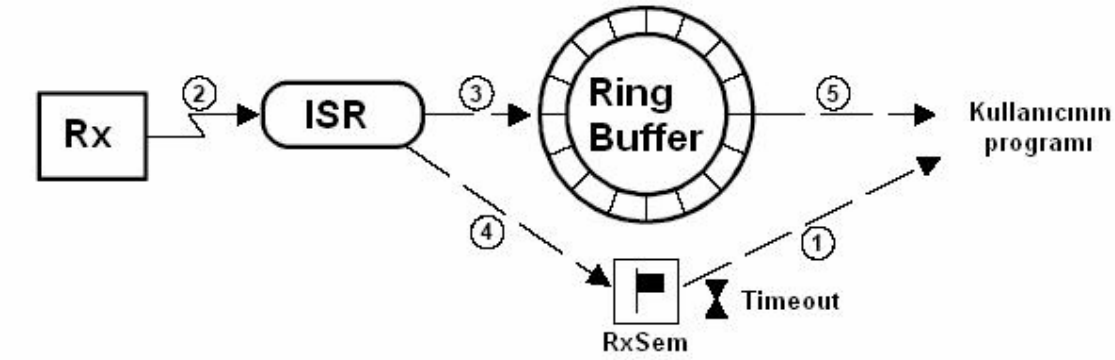
3.3.3 Semaforlar Nedir Ne Amaçla Kullanılır

Semafor birçok çekirdek tarafından sağlanan bir kontrol mekanizmasıdır ve şu amaçlarla kullanılır.

- Ortak bir kaynağa erişimin kontrolü
- Bir olayın gerçekleştiğine dair işaret
- İki görevin aktivitelerini senkronize etmesine olanak sağlamak için

Semafor kodun çalışmasını sürdürebilmesi için bir anahtardır. Eğer semafor başkası tarafından kullanımda ise; istekte bulunan görev, karşı taraf semaforu serbest bırakana kadar askıya alınır. Başka bir değişle istekte bulunan görev “bana anahtarı ver eğer başkası kullanıyor ise beklemeye razıyım” der.(*) Bir görev başkaları tarafından kullanılma ihtimali olan bir kaynağa erişmeye kalktığında da semafor kullanılabilir. Örneğin bir görev yazıcıya veri göndermeden önce, yazıcının uygun olup olmadığını öğrenmeyi gerektirebilir. Bunun gibi, yazıcının uygun olup olmadığının anlaşılması için genellikle binary semaforlar kullanılır (Labrosse, 2002).

Semaforlar aynı zamanda bir olayın gerçekleştiğine dair bir işaret olarak ta kullanılabilirler. Sayaç tipindeki semaforlar bu amaçla kullanılabilir. Örneğin seri porttan gelen bir bilgiyi işlediğimizi ve bunu KHY içinde yapamadığımızı düşünelim. Bunu başarabilmek için Şekil 3.9 daki ring buffer kullanılmıştır. (1) durumunda uygulama semaforda bekler konumdadır. Bir karakter alındığında, KHY bu karakteri seri porttan alır (2) ve ring buffer'a atar (3). Daha sonra KHY semafora, bekleyen göreve bir karakter alındığını söylemesi için, işaret verir (4).Semafora işaret vermek bekleyen görevi çalışmaya hazır hale getirir. KHY tamamlandığında çekirdek bekleyen görevin en yüksek öncelikli görev olup olmadığına bakar. Eğer öyle ise, KHY karakteri bekleyen görevin yeniden çalışmasını sağlar . Uygulama daha sonra ring buffer' dan karakteri alır ve gereken işlemi yapar. Bir çok gerçek zamanlı çekirdek bu gibi durumlarda en çok ne kadar bekleneceğine dair süreler koymuştur. Bu da uygulamaya haberleşme hattında bazı problemler olabileceği bilgisini vermeyi sağlar.



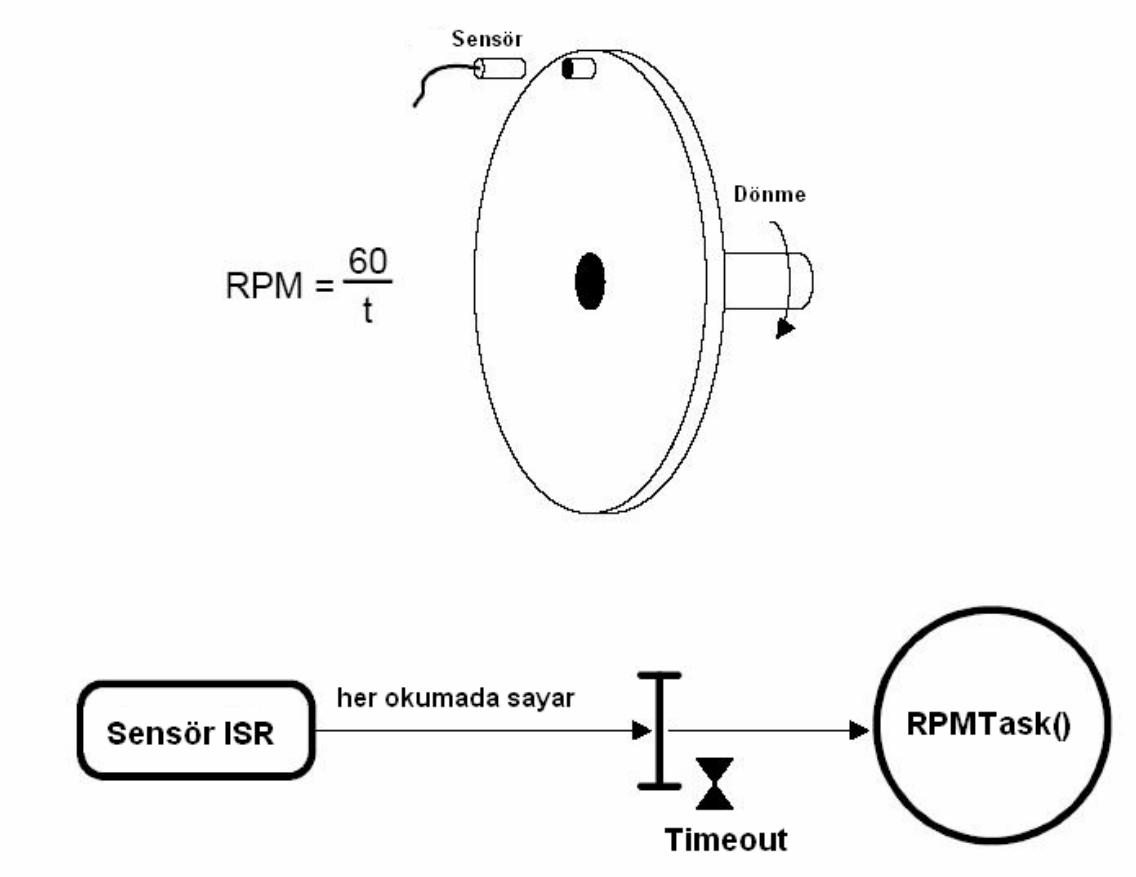
Şekil 3.9. Semafor kullanımı

Böyle bir durumda semafor kullanmak CPU'ya oldukça oldukça yük getirir, çünkü karakteri bekleyen görev ile düşük öncelikli görev arasında sürekli sabit bir görev değişimi olacaktır. Daha iyi bir kullanım ise belirgin bir karakter alındığında semafora işaret verilmesi olabilir. Böylece yüksek öncelikli görev sadece komut tam seri porttan alındığında uyarılır (Labrosse, 2005).

3.3.4 Görevler Arası Haberleşme Nasıl Yapılır? Mesajlaşma Nedir?

Birçok çekirdek bir görevden başka bir göreve yada bir KHY' den bir göreve bilgi aktarılmasını destekler. Böyle bir veri alışverişi mesaj posta kutuları ve mesaj kuyrukları ile gerçekleştirilebilir. Bir posta kutusu genelde tek bir gösterici boyutunda değişkenden ve posta kutusuna mesaj gelmesini bekleyen görevlerin listesinden oluşur. Çekirdek tarafından

sağlanan bir özellik sayesinde görev posta kutusuna mesaj gelmesini bekleme özelliğine sahip olur. Eğer bir mesaj zaten erişilebilir konumda ise, görev posta kutusundan bu mesajı alır ve işleme devam eder. Eğer posta kutusunda hiçbir mesaj yoksa, görev mesaj alınana kadar bekleme listesine alınır. Mesaj kuyruğu da aynı şekilde çalışır, sadece kuyruk birden fazla mesaj depolayabilir. Şekil 3.10' da manyetik alıcı dönen tekerlek üzerindeki pimi her algıladığında bir Kesme oluşur. KHY bu olayın olma sürelerini ölçer. Bu dönüşün RPM'sinin hesaplanması işlem ve zaman gerektirdiği için bir görev içinde yapılır. Çekirdekler semaforlar gibi mesajların bekleneceği maksimum sürelerin de belirlenmesine olanak sağlar. Bu özellik tekerleğin durmasının algılanmasında kullanılabilir. Eğer hiçbir mesaj gelmiyorsa RPM sıfır dır.



Şekil 3.10 Mesaj kullanımı

```
void RPMTask (void *arg)
{
    unsigned char err;
    void  *msg;

    while (1) {
        msg = OSMboxPend(&mailbox,100,&err);    /* mesaj için bekle */
        if (err==OS_NO_ERR) {
            /*ISR den alınan sayac bilgisini saniyeye çevir     */
            RPM=60.0 / time;
            /*ek işlemleer                                      */
        }
        else if (err==OS_TIMEOUT) {
            RPM = 0;
        }
    }
}
```

Şekil 3.11 Mesaj kullanımı ile ilgili kodlar

KHY' nin, zamanı doğru bir şekilde ölçmek için donanımsal bir timer kullandığı düşünülür. Timer zamanı doğrudan saniyeler şeklinde ölçemez fakat çok kolay bir şekilde saniyeye dönüştürülebilecek sayılar halinde tutar. Görevde yapılan ilk şey bir mesajın (sayacın göstericisi) KHY' den posta kutusuna ulaşmasını beklemektir. Eğer mesaj 100 sistem tiki süresinde ulaşamazsa bir hata durumu oluşur ve tekerleğin dönmediği belirtilir. Eğer bir hata oluşmazsa timer sayacı gerçek saniyelere çevrilerek RPM hesaplanır. Görev ayrıca diğer ek özellikleri de içerir.

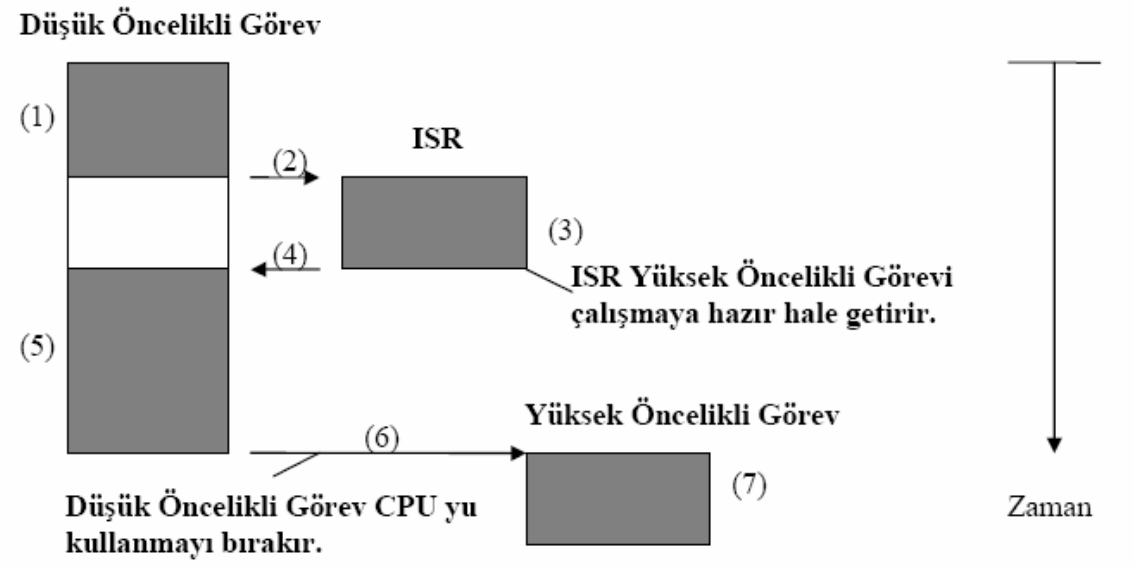
- RPM nin eşik değerinden yüksek olup olmadığını kontrol eder (aşırı hız)
- Maksimum RPM'nin takibini yapar
- Tekerleğin ne kadar zamandır döndüğünün takibini yapar.

Çekirdekler sadece kritik zamanlı sistemler için değildir. Çekirdek kullanımı gerçekte sistemi basitleştirir çünkü problemin küçük parçalara bölünmesini sağlar. Bu küçük parçalara da öncelikler verilerek en önemli parçanın sırası geldiğinde en önce işletilmesi sağlanır. Düşük öncelikli görevlerin eklenmesi yüksek öncelikli görevlere cevap verme süresini etkilemez.

3.3.5 Gerçek Zamanlı Çekirdek'in Detayları

Çekirdekler preemptive ve non-preemptive olmak üzere iki çeşittir Non-preemptive çekirdek de bir görev işletilirken KHY daha yüksek öncelikli bir görevi çalışmaya hazır hale getirir ancak KHY daima kesilen göreve geri döner. Yeni yüksek öncelikli görev mevcut görev bitip CPU'nun kontrolünü bıraktığı zaman çalışır. Non-preemptive çekirdekler gerçek zamanlı

uygulamalarda nadiren kullanılırlar. Sistem cevabının önemli olduğu uygulamalarda preemptive çekirdek kullanılır. Preemptive çekirdek kullanılarak çalışmaya hazır en yüksek öncelikli görevin CPU'nun kontrolünü alacağı garanti edilmiş olur. Bir görev daha yüksek öncelikli bir görevi çalışmaya hazır hale getirdiğinde kendisini askıya alır ve CPU'nun kontrolünü bu göreve verir. Eğer bir KHY önceliği yüksek bir görevi çalışmaya hazır hale getirirse, KHY bittiğinde, kesmeye uğrayan görev durdurulur ve önceliği yüksek yeni görev tekrar çalışmaya başlar. Preemptive çekirdek kullanılarak yapılan bir sistem örneği Şekil 3.4'te verilmişti.



Şekil 3.12 Non-preemptive çekirdek

Non-preemptive çekirdekte düşük öncelikli bir görev işletilirken, KHY yüksek öncelikli görevi çalışmaya hazır hale getirdiğinde yüksek öncelikli görev Kesme çıkışında hemen çalışmaya başlamaz. Kesmenin kesmeye uğrattığı görevin çalışması bittikten sonra yüksek öncelikli görev çalışmaya başlar (Labrosse, 2002).

3.3.6 Görevler

Bir görev herhangi bir zamanda şu dört durumdan birinde olabilir, yaratılmamış (Dormant), hazır (Ready), çalışıyor (Running), zaman bekler yada bir olayın olmasını bekler (waiting) konumda. Yaratılmamış; görevin bellekte olduğunu ancak henüz gerçek zamanlı çekirdek tarafından multitasking için erişilebilir konumda olmadığını gösterir. Hazır; görevin işletilmeye hazır olduğunu belirtir ancak şu anda önceliği daha yüksek bir görev işletiliyordur.

Çalışıyor, CPU'nun kullanımının bu görevde olduğunu gösterir. Zaman bekler konum; görevin bir zamanın dolmasını beklediğini gösterir. Bekler konum ise görevin başka bir görevi, bir I/O yu yada ortak bir kaynağın uygunluğunu bekler konumda olduğunu gösterir. Ayrıca görev bir Kesme ile bölünmüş ve Kesme hizmet yordamı işletiyor ise görev kesmeye uğramış konumda olur.

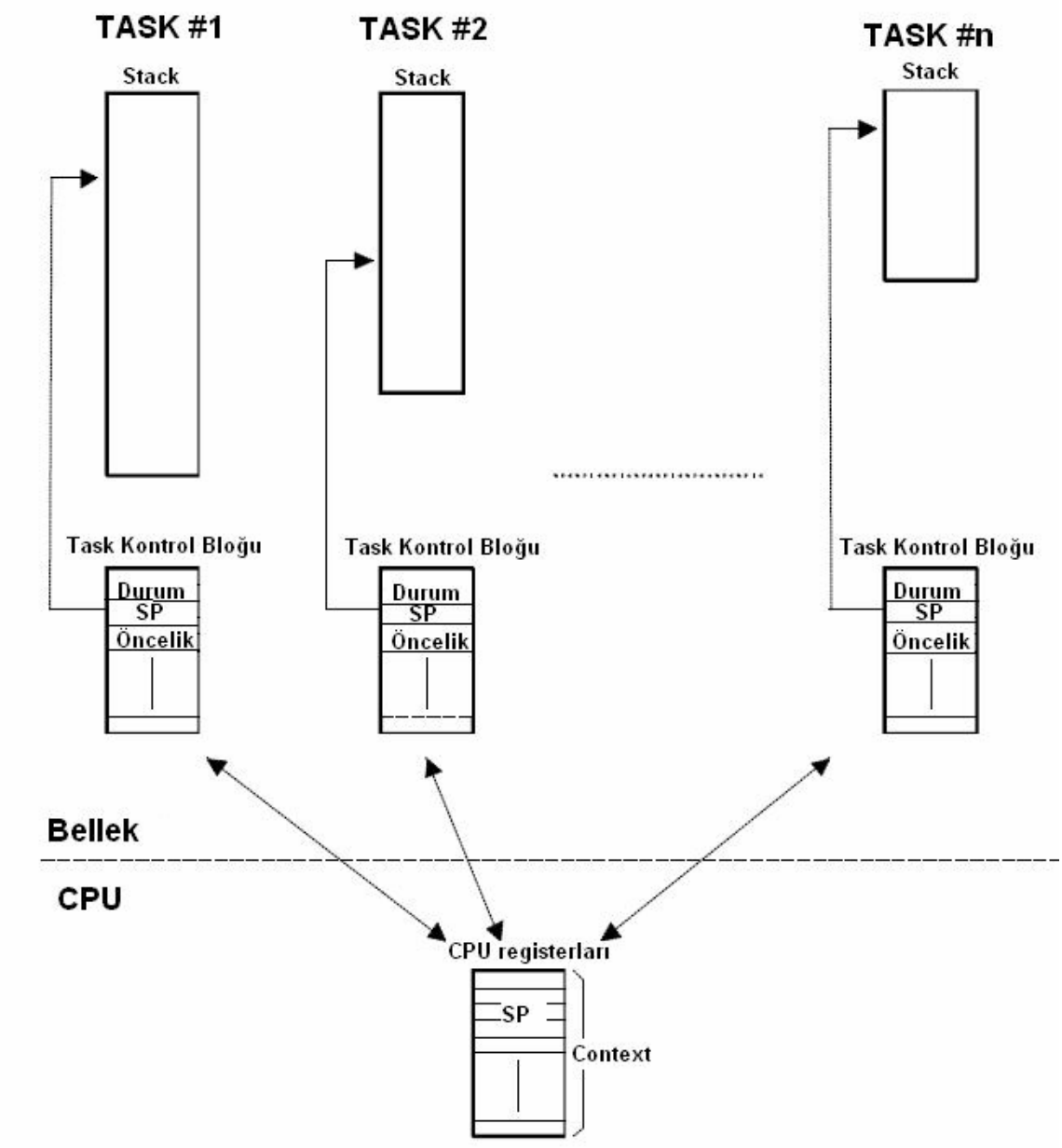
Her bir görevin yönetimini sağlayacak hizmet çekirdek tarafından sağlanmalıdır. Bu hizmet, görevi yaratılmamış konumdan alıp hazır hale getirir. Her bir görevin kendi yığını vardır ve bütün CPU kaydedicilerine erişebilir. Bir görevin yaratılmasıyla ilgili fonksiyon Şekil 3.13'de verilmiştir.

```
void OSTaskCreate (void (*task)(void),void *stack, unsigned char priority)
```

Şekil 3.13 Görevin yaratılmasıyla ilgili fonksiyon

"task" çekirdek tarafından yönetiminin yapılması gereken görevin fonksiyonunun ismidir. "stack" bu görev tarafından kullanılan yığın'ın en üstü için bir göstericidir. "priority" görevin önceliğidir.

Çekirdek her bir görevin durumunu Task Kontrol Bloku olarak (Task Control Block –TCB) adlandırılan bir veri yapısı ile takip eder. TCB görevin durumunu (Hazır, bekler,…), görevin önceliğini, yığın göstericisi ve çekirdek ile ilgili diğer bilgileri tutar. Şekil 3.14' de görevyığınları, TCB ler ve CPU kaydedicileri arasındaki ilişki gösterilir. Olaylara göre çekirdek görevler arasında değişim yapacaktır. Bu işlem genel olarak CPU kaydedicilerinin içeriğinin ilgili görevin yığınına kaydedilmesi, yığın göstericisinin ilgili görevin TCB sine kaydedilmesi, yeni görevin yığın poniter ının bu görevin TCB sinden yüklenmesi, CPU kaydedicilerinin yüklenmesi biçiminde olur. Bu işleme context switch – görev değişimi denir.

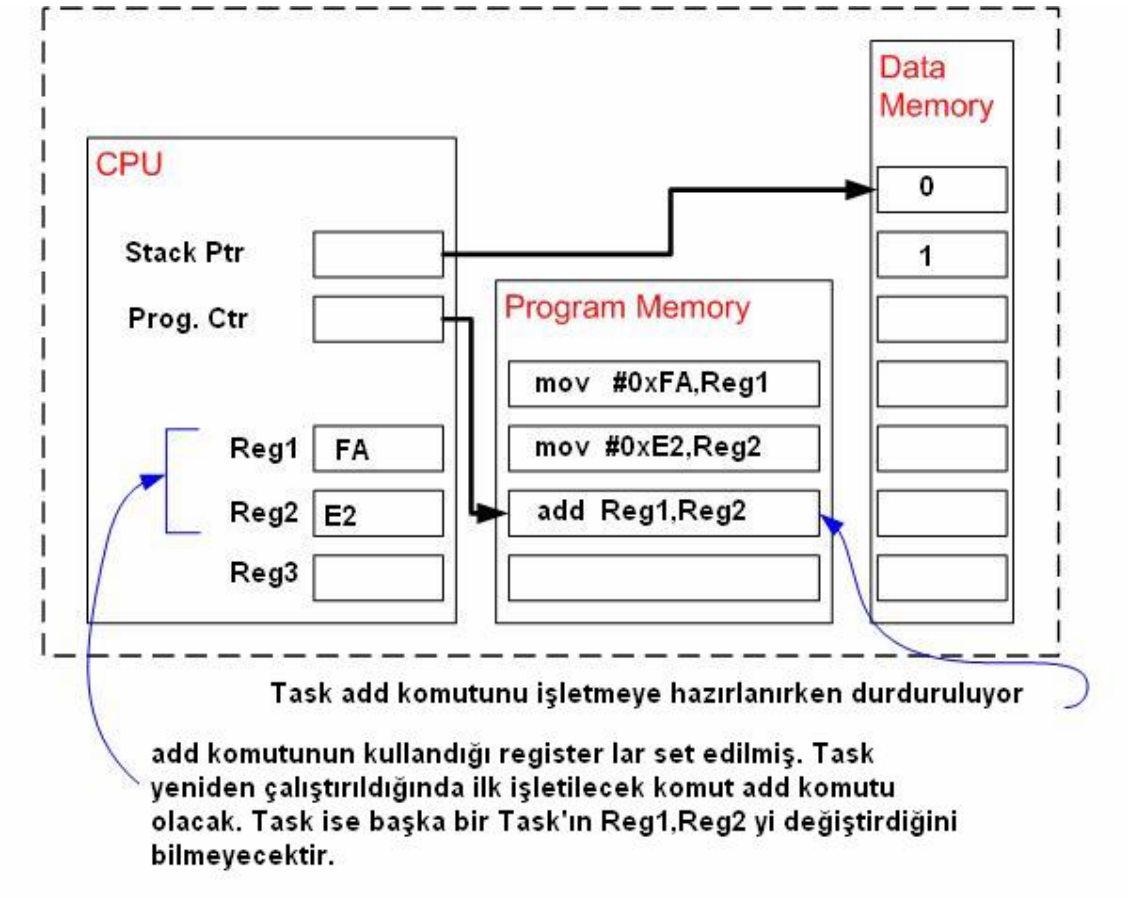


Şekil 3.14 Çoklu görevler

3.3.7 Görevler Arası Geçiş Nasıl Yapılır?

Görev değişimi gerçek zamanlı işletim sistemleri için önemli bir konudur bu yüzden bu konuyu biraz daha detaylı incelemek gerekir. Bir görev işletilirken diğer herhangi bir program gibi işlemcinin kaydedicilerini değiştirir, RAM ve ROM' a erişir. Bu kaynaklar, CPUkaydedicileri, yığın, program counter ve donanıma özgü başka şeyler olabilir. Bir görev ardışıl kodlardan oluştuğuna göre, çekirdek tarafından ne zaman durdurulacağını yada

bekletileceğini, hatta bunun ne zaman olduğunu bilemez. Şekil 3.15'teki örnekte görev işlemcinin iki kaydedicinin içeriğini toplayan komutu işletmeden hemen önce durduruluyor (www.freertos.org).

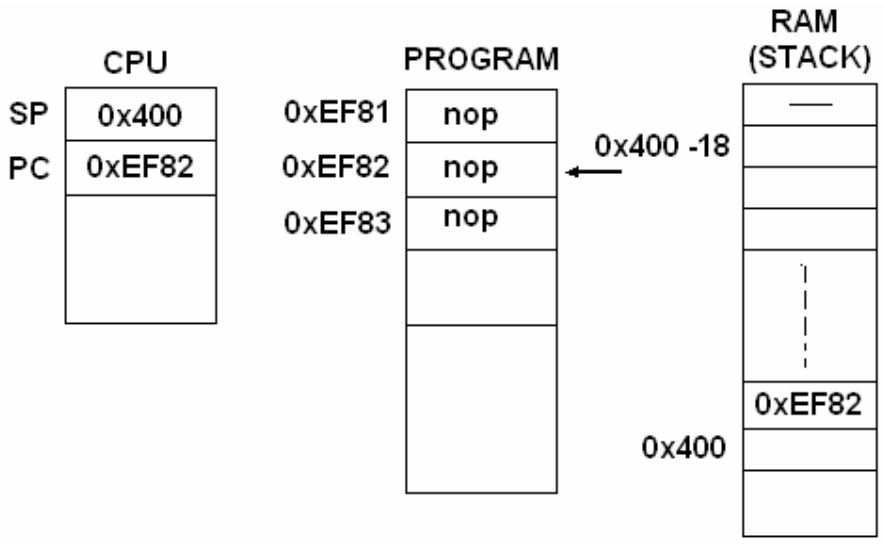


Şekil 3.15 Görevin içeriğinin kaydedilmesi

Görev durdurulduğunda diğer görevler işlemci kaydedicilerinin içeriğini değiştirmiş olabilirler. Görev geri geldiğinde bu değerlerin değiştirildiğini bilemez ve hatalı bir işlem yapmış olabilir. Bu tip hataları gidermek için görev bir içeriğe (context) sahiptir. Görev bu içeriği tekrar çalışmaya başladıktan sonra kullanacaktır. İşletim sistemi bu içeriği saklamaktan sorumludur. Görev tekrar çağrıldığında bu içerik çekirdek tarafından geri yüklenir. Şekil 3.15'te bir komut işletilecekken görevin durdurulması gerekiyor. Bu durumda işletim sistemi kaydedicileri görevin yığınına kaydediyor (www.freertos.org).

Görev değişiminde yapılan iş aslında bir programın interuptta gidip geri dönmesi gibidir. Buradaki tek fark kesmeden döndüğü yer kesmeye dallandığı yer değildir. Mikroişlemcilerdeki Kesme yapısını görev değişimini daha iyi anlamak için biraz daha detaylı

inceleyelim. Mikroişlemcilerde Kesme isteği geldiği zaman program counter, CPU kaydedicileri ve donanıma göre değişebilen diğer kaynakları, kesmeden döndüğünde kaldığı yerden devam edebilmek için, yığında saklar.



Şekil 3.16 Kesme yapısı

Şekil 3.16'da görüldüğü gibi mikroişlemci 0xEF81 adresindeki kodu işletiyor (kodun NOP olduğu varsayılmıştır) ve 0xEF82 adresindeki kodu işletecekken bir kesme geliyor. Bu esnada yığın göstericisi 0x400 ve program counter değeri de 0xEF82 dir. Kesme geldiğinde mikroişlemci donanımının 18 bytle'lık özel kaynaklarını otomatik olarak yığına attığını düşünelim. Bu 18 byte içinde program counter da 2 byte lık bir değerdir. Donanım bu değerleri yığına yazar ve yığın göstericisini 18 byte aşağı çeker. Kesme dönüşü ise donanım "rti" – "return from interrupt" komutu ile yığına attığı değerleri geri yükler. Program counter değerini de yığından alır (0xEF82). Bu değeri CPU'nun içindeki program counter'a yazar ve "rti" komutunun bitişinde program counter değerinde 0xEF82 olduğu için yazılım doğrudan oxEF82 adresine gider ve dolayısıyla kaldığı yerden devam etmiş olur.

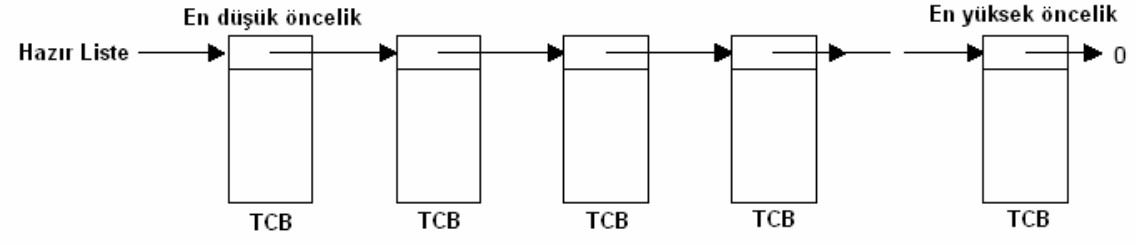
Eğer bir RTOS kullanılırsa yukarda anlatılan mekanizmadan görev sayısı kadar olduğunu düşünürsek. Yani 3 görev varsa 3 adet yığın vardır.Yeni bir görevin işlemcinin kontrolünü almasına karar verildikten sonra. Çalışmaya başlanacak görevin yığını aktif yığın olarak alınır. "rti" komutu işletildikten sonra bu yığında kayıtlı olan program counter değeri ne ise yazılım oraya gider ve ilgili görev kaldığı yerden çalışmaya devam etmiş olur. Şekil 3.17'de görev 3 ün o an için çalışmasına karar verilen görev olduğunu düşünelim. Aktif yığın olarak

görev 3'ün yığını ayarlanır yani yığın göstericisi değerine 0x211-18=0x1FF değeri yazılır. Bu durumda kesme dönüşünde yazılım 0x8147 adresine gidecektir.

Şekil 3.17 Görev değişiminde yığınlar

3.3.8 Hazır Liste Kullanımı

Çekirdek'in görevlerinden biri işletilmeye hazır görevleri önceliklerine göre sıralamaktır. Bu listeye hazır liste denir. Çekirdek daha önemli bir görevi işletmeye karar verdiğinde basitçe yapacağı şey hazır listenin en başındaki TCB'yi almak olur. Bu işlem scheduling olarak adlandırılır.

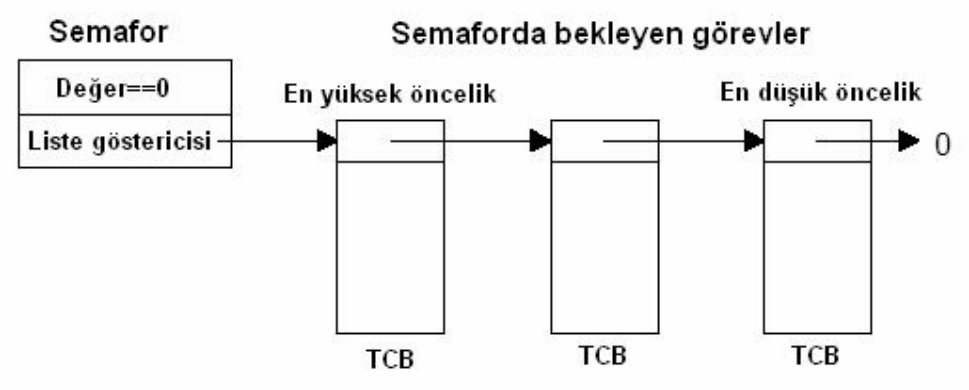


Şekil 3.18 Hazır Liste

3.3.9 Olay Yönetimi

Çekirdek bir olay gerçekleşene kadar görevin askıya alınmasına olanak sağlayan hizmetler

sunar. Böyle bir bekleme için kullanılan en genel mekanizma semaforlardır. Semaforların ortak bir kaynağa erişiminde, bir olayın geçekleştiğine dair işaret olarak yada iki görevin aktivitelerini senkronize etmek için kullanılabileceği önceki kısımlarda anlatılmıştı. Bir semafor genellikle bir değer ve bekleyen listesinden oluşur.



Şekil 3.19 Semafor içeriği

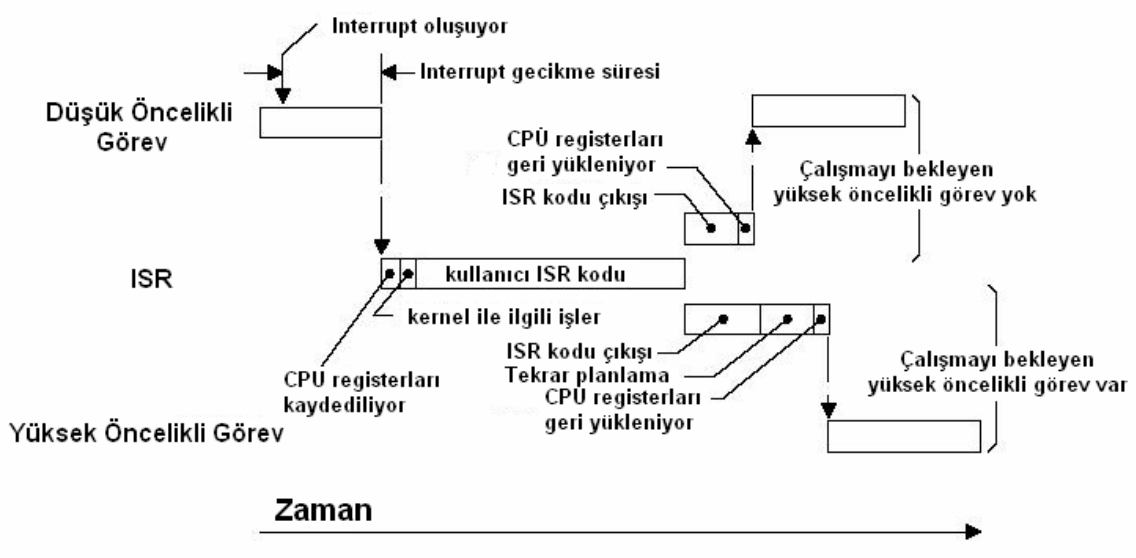
Çekirdek tarafından kullanılmadan önce semafora önce bir ilk değer verilmelidir. Çekirdek multitasking bir yapıyı desteklediği için bir kaynağa aynı anda iki görev tarafından erişim engellenmelidir. Semafora 1 ilk değerinin verildiğini düşünelim. Bir yazıcıya erişmeye çalışan görev semafor için bekler konuma gelir. Eğer semafor 1 ise, bu değeri azaltarak 0 yapar ve çalışmaya devam eder. Bu noktada yazıcı göreve aittir. Eğer başka bir görev yazıcıya erişmeye çalışırsa o da yine bekleme konumuyla karşılaşır. Ancak bu sefer semafor değeri 0 olduğu için kaynağa erişemez, hazır listeden çıkarılır ve semaforun bekleyen listesine alınır. İlk görev işini tamamladığında bir işaret verir ve yazıcı serbest bırakılır. Bu noktada bekleyen listesindeki görev hazır listeye alınır. Eğer yazıcı için bekleyen görev mevcut görevden daha önemli ise CPU'nun kontrolü bu göreve verilir.

3.3.10 Sistemin Ana Saat Frekansı

Zamanın takibi için her çekirdek bir mekanizma sağlar. Bu mekanizma CPU'yu periyodik olarak kesmeye uğratan donanımsal bir timer dır. Çekirdek tarafından bu timer ın oluşturduğu KHYde zamana bağlı değişkenler güncellenir. Özellikle timeout lar önemlidir. Bu KHY genelde saat tiki yada sistem tiki olarak adlandırılır. Bu kesme genelde 10-100ms arasına ayarlanır. Bir görev kendini belli bir saat tiki sayısı süresi kadar askıya alabilir. Sürenin dolmasını bekleyen görevlerin tutulduğu özel bir liste mevcuttur. Bu listeye gecikmeli görev listesi denir.

3.3.11 Kesmelerin işletilmesi

Çekirdekler KHY ler tarafından kullanılan görevlerin olaylar hakkında bilgilendirilmesiyle ilgili hizmetler sunar. Bu hizmetlerin kullanılabilmesi için KHY lerin bütün CPU kaydedicilerini kaydetmesi, çekirdek' e KHY 'ye girildiğini belirtebiliyor olması gerekir. Böyle bir durumda çekirdek bir sayacı arttırır. Bu sayaç iç içe kaç kesme geldiğini algılamak için kullanılır. KHY kodunun tamamlanmasıyla çekirdek tarafından sağlanan başka bir hizmet KHY' nin bittiğinin belirtilmesidir. Bu genelde sayacın azaltılmasıyla olur. Bütün KHY ler bitip, birinci kesme seviyesine geri döndüğünde, çekirdek bu KHY lerden biri tarafından daha yüksek öncelikli bir görevin hazır hale getirilip getirilmediğine bakar. Eğer kesmeye uğrayan görev hala en yüksek öncelikli görev ise CPU registeları geri yüklenir ve görev kaldığı yerden devam eder. Eğer daha yüksek öncelikli bir görev çalışmaya hazır hale geldiyse çekirdek kesmeye uğrayan görevin CPU kaydedicilerini onun TCB sine kaydeder, yeni görevin yığın göstericisini ve CPU kaydedicilerini alarak bu görevi işletmeye devam eder (Labrose, 2002, Li, ve Yao, 2003).



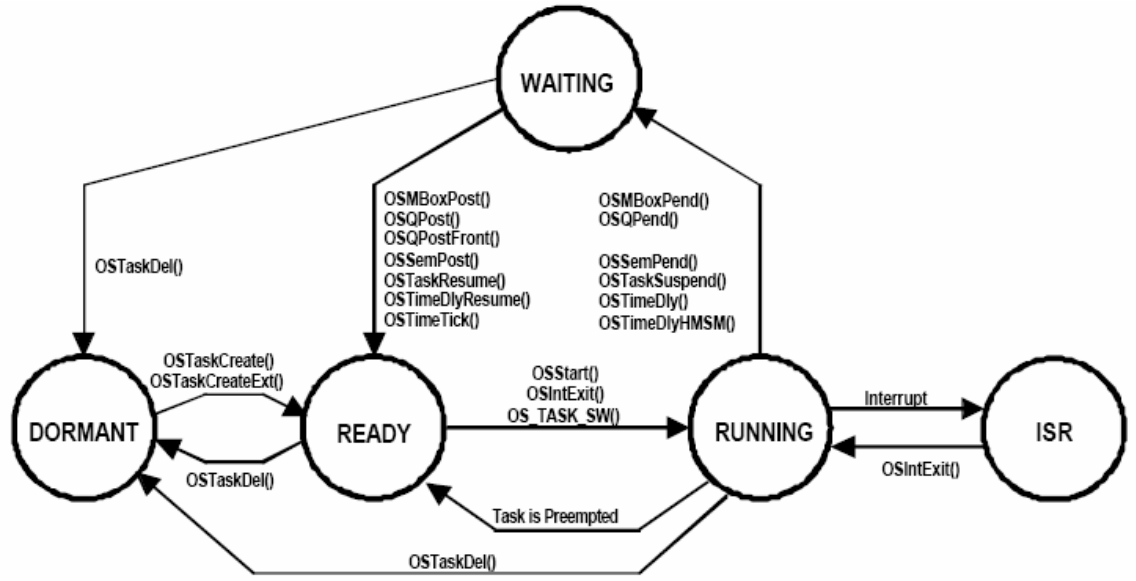
Şekil 3.20 Kesmenin işletilmesi

4. UYGULAMADA KULLANILAN İŞLETİM SİSTEMİ MICROC/OS-II'NİN ÖZELLİKLERİ

Giriş kısmında gömülü sistemlerin giderek karmaşıklaştığını, daha kompleks problemlerle ve kritik zamanlamalı işlerle uğraşmaya başladığını söylemiştik. Yapılan sistemler bir yandan karmaşıklaşırken tasarım sürecinin de kısalması gerekmektedir. Yazılımları artık asembler kodlarıyla uzun uzun düşünerek yazma devre de kapanmıştır. Yapılması gereken şey problemi küçük parçalara bölerek yazılım yapmak, mümkün olduğu kadar donanımla uğraşmamaktır. Bunun yanında mühendisler kritik zamanlamaları yakalayabilmek için de hazır çözüm arayışına girdiler. Bir yazılımdaki en zor işlerden biri de zamanlamaları yakalayabilmektir, çünkü yaptığınız her değişiklik, her yeni ek zamanlamayı etkileyebilir. Sizin yerine bu işleri halleden bir mekanizmanın olması tasarımcıyı çok rahatlatır. Böylece tasarımcı bütün iş gücünü sistemin fonksiyonlarını yapmaya verebilir. İşte bu yüzden bir Real Time Operating System (RTOS – Gerçek Zamanlı İşletim Sistemi) kullanma gerekliliği ortaya çıkmıştır.

Tezin sonunda yapılan uygulamada Micrium firması tarafından yazılmış olan MicorC/OS-II adlı işletim sistemi kullanılmıştır. Bu bölümde bu işletim sisteminin çalışması kısaca anlatılmıştır. MicroC/OS-II açık kaynak kodlu bir işletim sistemidir. İşletim sisteminin büyük bir kısmı ANSI C uyumludur. Mikroişlemciye özgü kodlar makine dilinde yazılmıştır ve minimum tutulmaya çalışılmıştır. Bu özelliği ile hayli taşınabilir bir kod özelliğine sahiptir. İstenmeyen özellikler derlenmeyerek bellek boyutu ayarlanabilir. Preemptive bir yapıya sahiptir. 64 adet görev yönetebilir. 8 adet görev sistem için ayrılmıştır ve kullanıcıya 56 adet görev bırakılmıştır. Her bir görev ayrı önceliğe sahiptir ki bu da round robin algoritması kullanmadığını gösterir. Her bir görev kendi yığınını gerektirir ve MicroC/OS-II her bir görevin farklı boyutta yığın kullanmasına imkan sağlar. Böylece RAM kazancı sağlanır. Yığın kontrol fonksiyonları ile görevin yığın'ın ne kadarını kullandığı öğrenilebilir. Mesaj kutuları, kuyruk yapıları, semaforlar, bellek yönetimi ve zaman ile ilgili fonksiyonlar sağlar.

MicroC/OS-II' de görevler beş durumda olabilir. (Şekil 4.1).

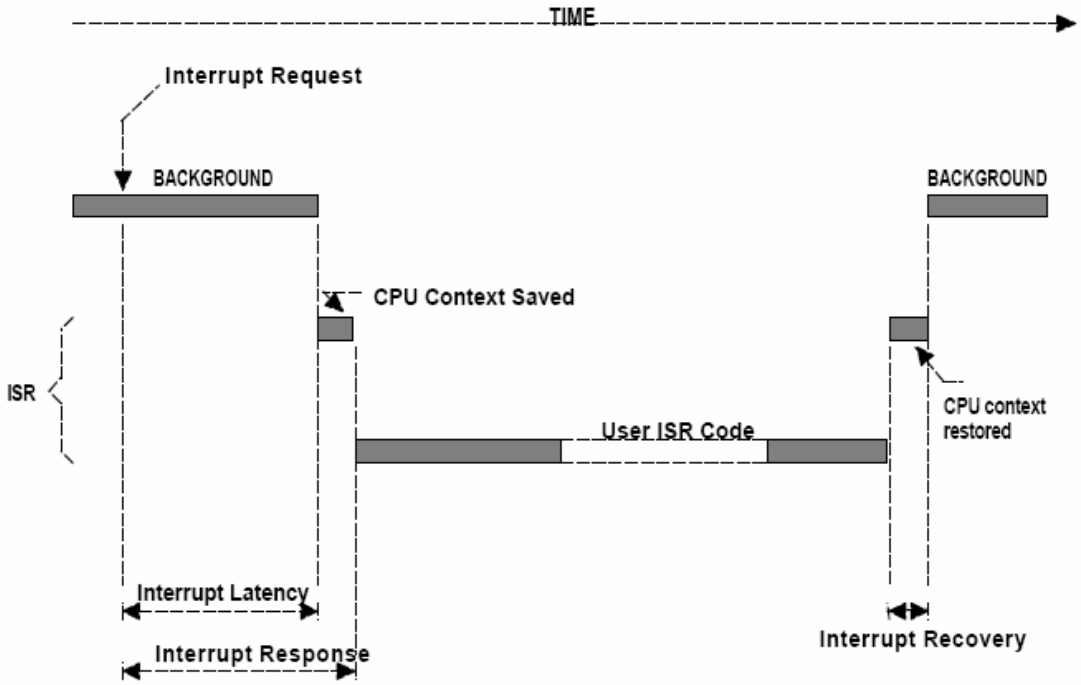


Şekil 4.1 MicroC/OS-II görev durumları. (CMP Books - MicroC-OS-II The Real-Time Kernel 2nd Ed -2002)

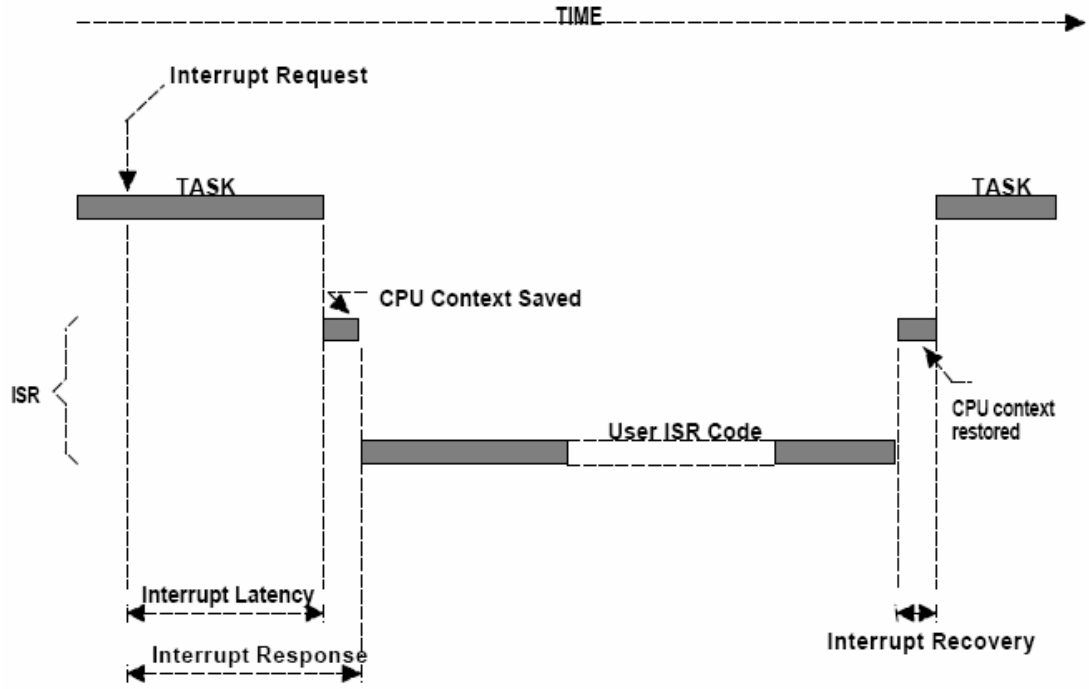
4.1 Gömülü Sistemlerde Kesmelerin Etkisi Nedir ? RTOS Kullanıldığında Bu Etki Ne olur

Kesmeler CPU yu asenkron olaylardan haberdar etmek için kullanılır. Program bir kesme ile kesilirse , kesildiği koda yada koşmaya hazır göreve geri döner. İşlemciler genel olarak iç içe kesmelerin gelmesine imkan sağlar.

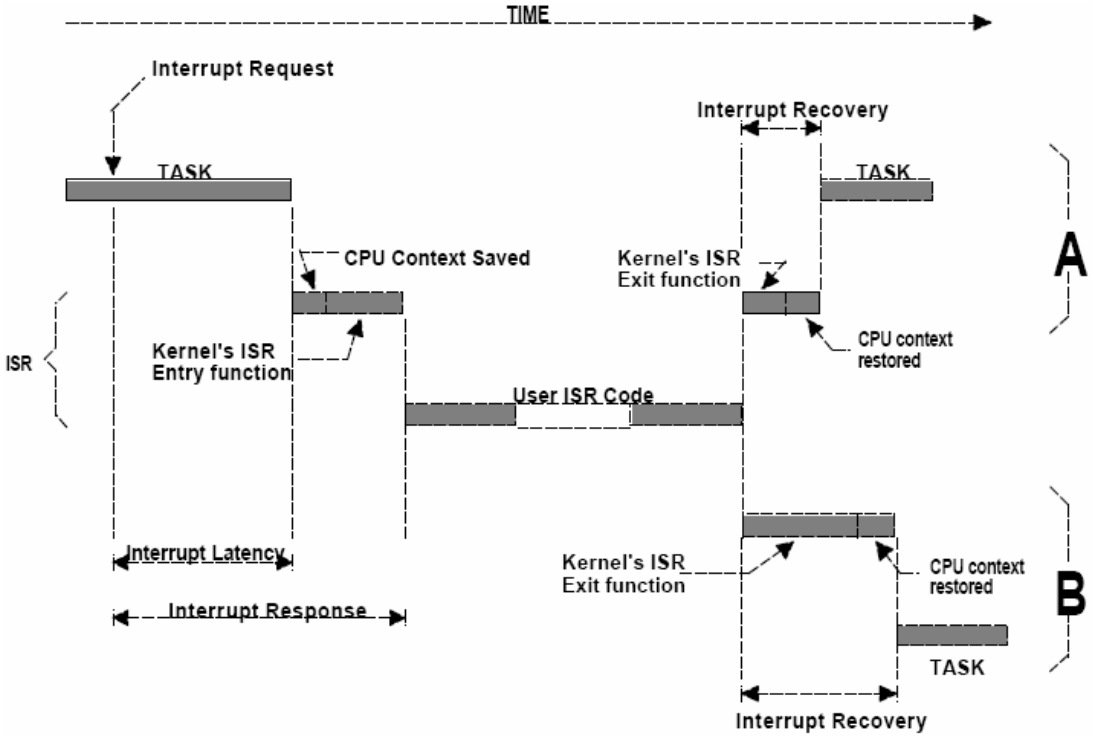
Gerçek zamanlı çekirdeklerin çoğunda kesmeleri durdurma ve yeniden açma yapılır. Kesmelerin durdurulduğu süre ne kadar uzarsa kesme gecikmeleride o kadar artar. Kesme gecikmesi; kesmelerin kapalı kaldığı maksimum süre ile KHY deki ilk kodun işlemeye başlama sürelerinin toplamı kadardır. Kesme cevabı; kesme isteğinin oluşması ile kullanıcı tarafından yazılmış kodun çalışmaya başlaması arasındaki süredir. Ön plan/Arka plan bir sistem için kesme cevabı; kesme gecikmesi ile CPU'nun kaydedicilerinin kaydedilme süreleri toplamı kadardır. Non-preemptive bir çekirdek'de kesme cevabı da yine aynıdır. Preemptive bir çekirdekte kesmeye girildiğini algılamak, bunun takibini yapmak ve içi içe kaç kesme geldiğni algılamak için bir fonksiyon çalıştırılır. MicroC/OS-II de OSIntEnter() fonksiyonu bu iş için kullanılır. Bu durumda kesme cevabına bir de bu fonksiyonun işletilme süresi eklenir (Labrosse 2002).



Şekil 4.2 Ön plan/Arka plan bir sistem için kesme gecikmesi, cevabı ve dönüş süresi (CMP Books - MicroC-OS-II The Real-Time Kernel 2nd Ed-2002)



Şekil 4.3 Non-Preeemptive bir sistem için kesme gecikmesi, cevabı ve dönüş süresi (CMP Books - MicroC-OS-II The Real-Time Kernel 2nd Ed-2002



Şekil 4.4 Preemptive bir sistem için kesme gecikmesi, cevabı ve dönüş süresi (CMP Books - MicroC-OS-II The Real-Time Kernel 2nd Ed-2002)

Dönüş süresi işlemcinin kesilen koda geri dönmek için harcadığı süre olarak tanımlanabilir. Ön plan / Arka plan bir çekirdek için bu süre CPU kaydedicilerinin geri yüklenmesi ve kesmeden dönüş komutunun işlenmesi kadardır. Non-preemptive bir çekirdek için de bu süre aynıdır. Preemptive bir çekirdekte bu süre yüksek öncelikli bir görev var mı kontrolü süresi, yüksek öncelikli görevin CPU kaydedicilerini CPU ya yükleme ve kesmeden dönüş komutunun işlenmesi kadar süre alır.

4.2 Uygulamada Kullanılan Görev Yönetiminin Gerçekleştirilmesi

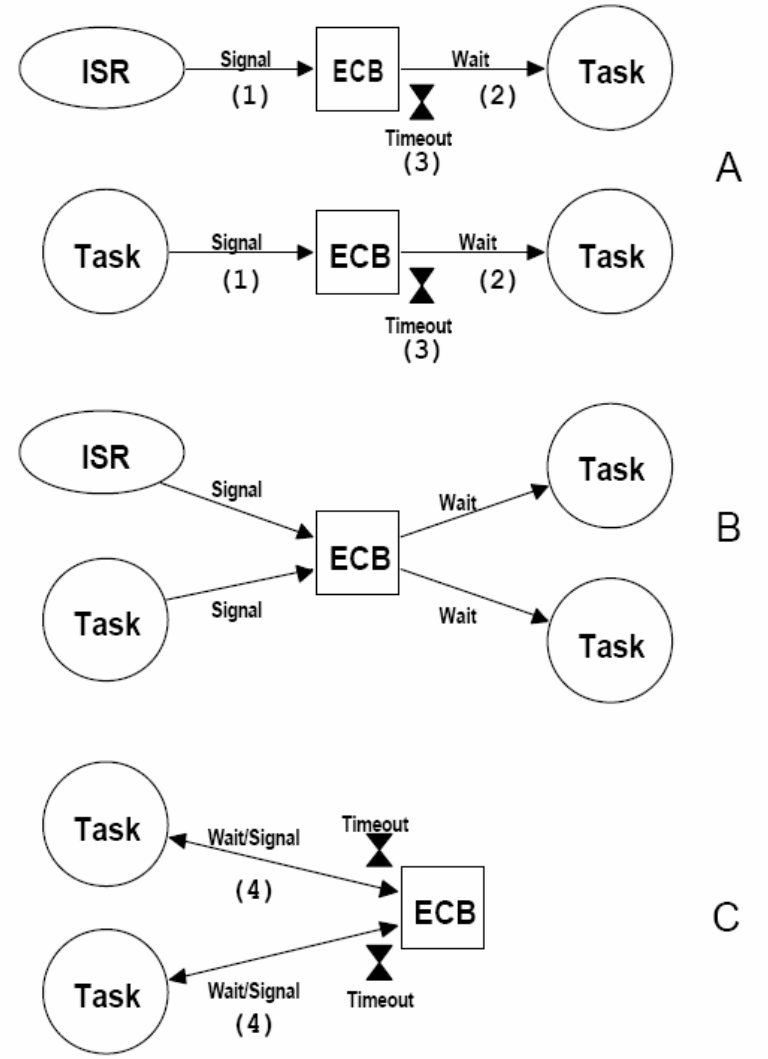
Uygulamada ki görevler aşağıda anlatılan fonksiyonlarla oluşturulmuş ve yönetimi yapılmıştır. Bir görev oluşturulduğunda o göreve ait bir de görev kontrol bloğu oluşturulur (Task Control Block –TCB). Bir veri yapısı olan bu blok işletim sistemi tarafından görev durdurulduğunda görevin durumunu öğrenmek için kullanılır. Bu veri yapısından görev için tanımlanmış yığının tepe noktası, en alt noktası, yığının boyutu, bir mesaj yada süre için bekleyip beklemediği, çalışıyor mu, duruyor mu, bekliyor mu ile ilgili bilgi ve görevin önceliği öğrenilir.

```
INT8U OSTaskCreate (void (*task)(void *pd), void *pdata, OS_STK *ptos, INT8U prio)
{
    void   *psp;
    INT8U   err;


    if (prio > OS_LOWEST_PRIO) {                                          (1)
        return (OS_PRIO_INVALID);
    }
    OS_ENTER_CRITICAL();
    if (OSTCBPrioTbl[prio] == (OS_TCB *)0) {                              (2)
        OSTCBPrioTbl[prio] = (OS_TCB *)1;                                 (3)
        OS_EXIT_CRITICAL();                                               (4)
        psp = (void *)OSTaskStkInit(task, pdata, ptos, 0);                (5)
        err = OSTCBInit(prio, psp, (void *)0, 0, 0, (void *)0, 0);        (6)
        if (err == OS_NO_ERR) {                                           (7)
            OS_ENTER_CRITICAL();
            OSTaskCtr++;                                                  (8)
            OSTaskCreateHook(OSTCBPrioTbl[prio]);                         (9)
            OS_EXIT_CRITICAL();
            if (OSRunning) {                                             (10)
                OSSched();                                               (11)
            }
        } else {
            OSTCBPrioTbl[prio] = (OS_TCB *)0;                            (12)
        }
        return (err);
    } else {
        OS_EXIT_CRITICAL();
        return (OS_PRIO_EXIST);
    }
}
```

Şekil 4.5 Görev oluşturulması (CMP Books - MicroC-OS-II The Real-Time Kernel 2nd Ed-2002)

MicroC/OS-II nin görev yönetimi için kullandığı fonksiyonlar şöyle sıralanabilir. OSTaskCreate() fonksiyonu ile görev oluşturulabilir(Şekil 7.4). Bu fonksiyon çağrılırken görevin yani kullanıcı tarafından yazılan C fonksiyonun adresi, kullanıcının verebileceği bir arguman, yığının tepesini gösterir bir gösterici, ki bu da kullanıcı tarafından belirlenen yığının adresidir, ve görevin önceliği verilir. Bu fonksiyon görevin çalışması için gerekli ilklendirmeleri yapar.Bu fonksiyon ile uygulamada ki görevler oluşturulmuştur. OSTaskStkChk() fonksiyonu görevin gerçekte ne kadar yığın kullandığını öğrenmek için kullanılır. OSTackDel() fonksiyonu görevi çekirdek'in yönetiminden çıkarmak için kullanılır. OSTaskChangePrio() görevin önceliğini değiştirmek için kullanılır. OSTaskSuspend() görevi askıya almak ve bekletmek için kullanılır. Buna örnek olarak uygulamada gerçekleştirilen LCD görevinin kendini gecikme fonksiyonu içinde durdurması verilebilir. Gecikme fonksiyonu içinde OSTaskSuspend() fonksiyonu ile LCD görevi durdurulmuştur. OSTaskResume() askıya alınmış görevi tekrar çalıştırmak için kullanılır (Labrosse, 2002).



Şekil 4.6 Durum kontrol bloklarının kullanımı (CMP Books - MicroC-OS-II The Real-Time Kernel 2nd Ed-2002)

4.3 Uygulamadaki Görevler Arası Haberleşme ve Senkronizasyon

Ortak kaynakların kullanımı ve görevler arası haberleşme için MicroC/OS-II birçok mekanizma sunar. Bunlar kesmelerin enable disable edilmesi, semaforlar, mesaj kutuları, mesaj kuyrukları dır. Şekil 4.6' da görevlerin kesmelerle nasıl ilişkili çalıştığı gösterilmiştir. Durum kontrol bloğu denen yapılarla görevler ve KHY ler görevlere işaret verir (A). Bir görev başka bir görevin yada KHY' nin işaret vermesini bekleyebilir (B). Beklenen işaretin gelmemesine karşı opsiyonel olarak bir zaman aşımı belirlenebilir. Bir çok görev bir başka görevin yada KHY' nin bir işaret vermesini bekliyor olabilir. Durum Kontrol Bloğuna işaret verildiğinde bekleyen en yüksek öncelikli görev bu işareti alır ve çalışmaya hazır hale gelir.

4.3.1 Durum Kontrol Blokları

MicroC/OS-II OS_EVENT (Event Control Block) olarak adlandırdığı veri yapılarını senkronizasyon ve ortak kaynaklara erişimin yönetimi için kullanır. Bir durum kontrol bloğu kendisi (semafor için bir sayaç, mesaj kutusu için bir gösterici, mesaj kuyruğu için göstericiler dizisi), bu olayın olmasını bekleyen görevler için bir bekleme listesinden oluşur. ECB nin yapısı Şekil 4.7'de gösterilmiştir.

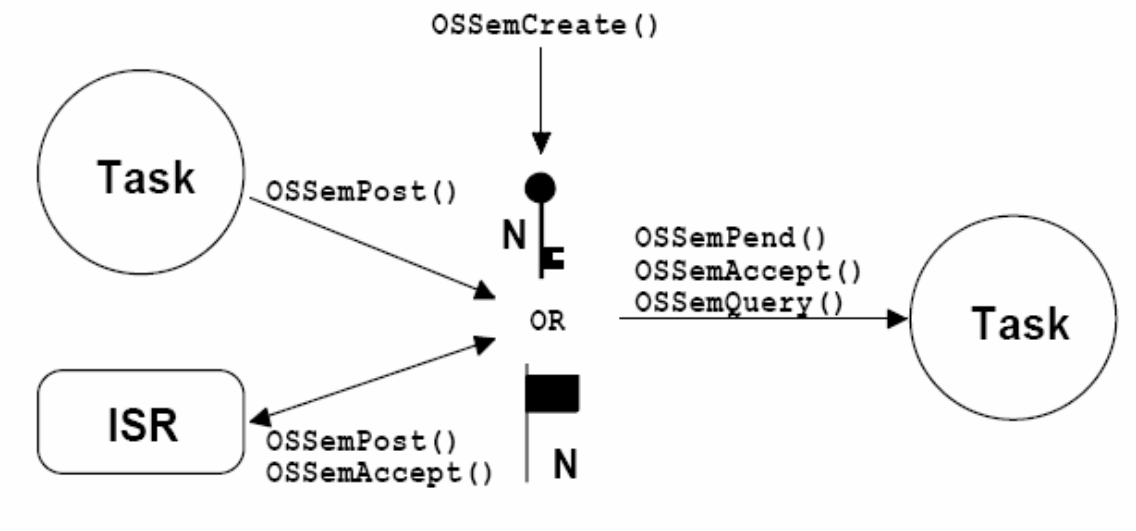
```
typedef struct {
    void   *OSEventPtr;                    /* Ptr to message or queue structure */
    INT8U   OSEventTbl[OS EVENT TBL SIZE]; /* Wait list for event to occur      */
    INT16U  OSEventCnt;                    /* Count (when event is a semaphore) */
    INT8U   OSEventType;                   /* Event type                        */
    INT8U   OSEventGrp;                    /* Group for wait list               */
} OS_EVENT;
```

Şekil 4.7 Durum kontrol bloğu veri yapısı (CMP Books - MicroC-OS-II The Real-Time Kernel 2nd Ed-2002)

OSEventPtr sadece ECB mesaj kutusu yada kuyruk olarak kullanılırsa kullanır. Bu durumda bir mesaja yada kuyruğa gösterici olarak kullanılır. OSEventTbl[] ve OSEventGrp bu olay için bekleyen görevlerin tutulduğu listedir. OSEventCnt ECB semafor olarak kullanıldığında bir sayaç olarak kullanılır. OSEventType ise olayın semafor mu, mesaj kutusu mu, yoksa kuyruk mu olduğunu belirler. Bu olayın olmasını bekleyen bütün görevler bekleme listesine alınır ve olayın gerçekleşmesi ile birlikte bekleyen en yüksek öncelikli görev listeden çıkarılır (Labrosse, 2002). Uygulamadaki AdcMbox ECB si örnek olarak alınabilir.

4.3.2 Semaforlar

MicroC/OS-II nin semaforları semafor sayacını tutmak için kullanılan 16 bit işaretsiz bir tamsayı değişkeni ve semafor sayacının 0 dan farklı bir değer almasını bekleyen görevler listesinden oluşur.



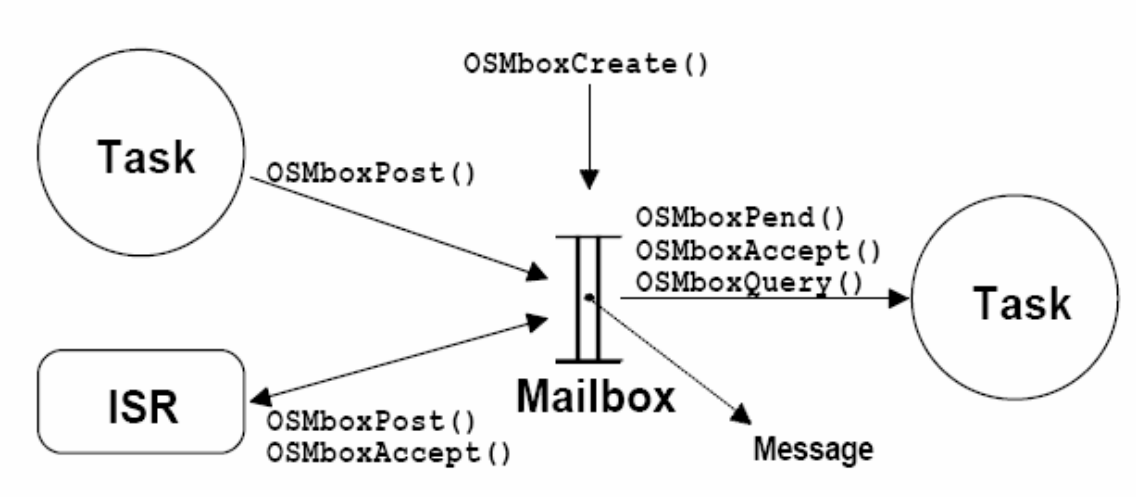
Şekil 4.8 Görevler, KHY ler ve semaforlar arasındaki ilişki (CMP Books - MicroC-OS-II The Real-Time Kernel 2nd Ed-2002)

Bir semaforu kullanmadan önce OSSemCreate() fonksiyonu ile semaforu oluşturmak gerekir. Semaforu oluşturmak semafora ilk değerini vermekten ibarettir. Bir semaforun ilk değeri 0 ile 65535 arasında olabilir. Eğer semafor bir yada daha fazla olayın olmasını işaret edecekse ilk değeri 0 olarak verilmelidir. Eğer ortak bir kaynağa erişim için kullanılacaksa ilk değer olarak 1 verilmelidir.Şekil 4.8 semaforların görevler ve KHY lerle ilişkisini açıklamaktadır. Semafor ortak bir kaynağa erişim için kullanılacaksa anahtar sembolü, olayın gerçekleşmesini gösterir ise bayrak sembolü kullanılmıştır. OSSemPend() fonksiyonu ile semaforun ortak bir kaynağın erişimine izin vermesini yada bir olayın gerçekleşmesi beklenir. OSSemPost() fonksiyonu ile semafora işaret verilir, böylece olay gerçekleşmiş yada ortak kaynağa erişime izin çıkmıştır. OSSempAccept() fonksiyonu ile beklemeden semafora ulaşılır. OSSemQuery() fonksiyonu ile semaforun durumu öğrenilebilir.

4.3.3 Uygulamada Gerçekleştirilen Mesaj Kutusu Nedir?

Uygulamada ADC sonucunun hazır olduğunun diğer görevlere bildirilmesini sağlamak için biir mesaj kutusu kullanılmıştır. Bu mesaj kutusu aslında C dilindeki bir göstericidir. BU yapıların detayları aşağıda anlatılmıştır.

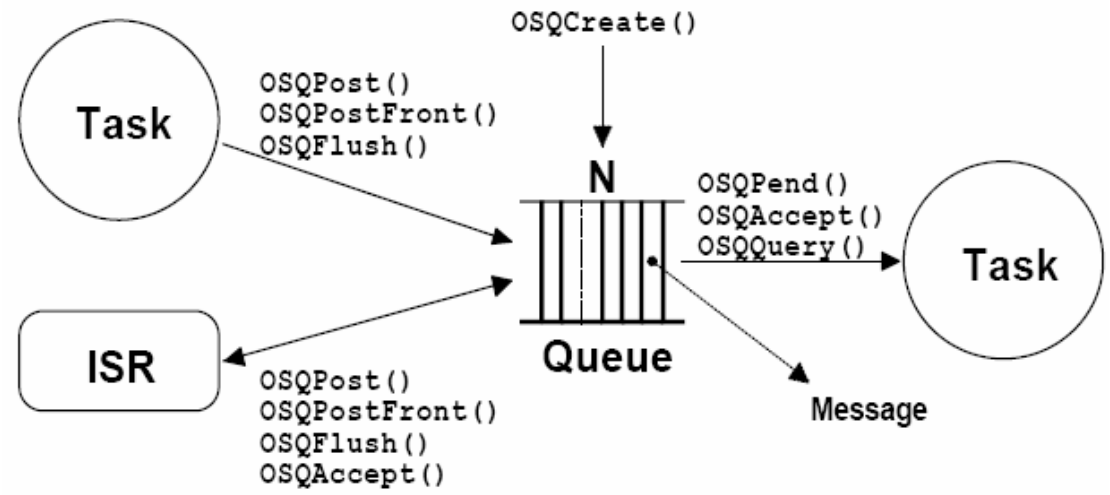
Bir mesaj kutusu basit anlamda bir görevin diğerine bir gösterici değişkeni göndermesinden ibarettir. Gösterici bir mesaj içeren bir veri yapısını gösterir. Bir mesaj kutusu kullanılmadan önce OSMboxCreate() fonksiyonu ile oluşturulmalıdır. İlk değer bir NULL göstericidir.



Şekil 4.9 Görevler, KHY ler ve mesaj kutuları arasındaki ilişki (CMP Books - MicroC-OS-II The Real-Time Kernel 2nd Ed-2002)

OSMboxPend() fonksiyonu ile mesaj kutusunda bir mesajın olup olmadığına bakılır ve eğer erişilebilir bir mesaj yoksa yazılım burada bekler. Tabi ki yazılımın buraya gelmesi bir başka görevden yada KHY'den mesaj beklediğini belirtir. Mesaj kutusuna mesaj koymak için OSMboxPost() fonksiyonu kullanılır. Bir mesaja beklemeden erişmek için OSMboxAccept() fonksiyonu kullanılır. Mesaj kutusunun durumunu öğrenmek için OSMboxQuery() fonksiyonu kullanılır.

4.3.4 Mesaj Kuyrukları



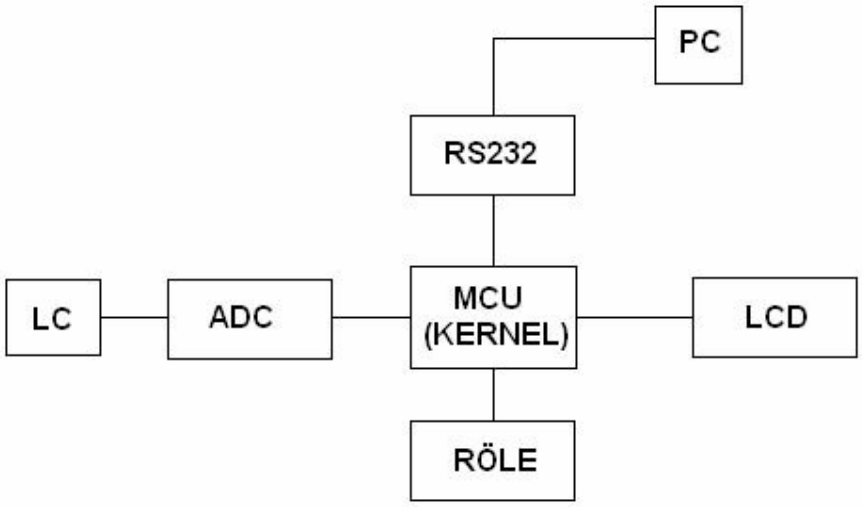
Şekil 4.10 Görevler, KHY ler ve mesaj kuyrukları arasındaki ilişki (CMP Books - MicroC-OS-II The Real-Time Kernel 2nd Ed-2002)

Mesaj kuyrukları bir görevin yada KHY'nin diğer görevlere bir gösterici değişkeni göndermesinden ibarettir. Her bir gösterici bir mesajı belirten belirgin veri yapılarını gösterir. Mesaj kuyruğu bir dizi mesaj kutusu ve tek bir bekleme listesi varmış gibi düşünülebilir.

5. GERÇEK ZAMANLI İŞLETİM SİSTEMİ KULLANARAK BİR GÖMÜLÜ SİSTEM TASARIMI VE KONTROL UYGULAMASI

Uygulama için önceki bölümde detaylı bir şekilde anlatılan MicroC/OS-II gerçek zamanlı işletim sistemi kullanılmıştır. Uygulamada gerçek zamanlı işletim sistemlerinin temel özelliklerinin anlaşılması için birkaç göreve sahip bir sistem seçilmiştir. Sistem dört görevli olarak tasarlanmıştır.

Basitçe sistemin fonksiyonel olarak çalışması şu şekilde anlatılabilir. Sistemde ADC, Röle Kontrol, Display ve PC görevleri vardır. 80ms de bir ADC entegresinden sonuç okunur, bu sonuca göre Röleler kontrol edilir, ADC sonucu belli iki değer arasında ise röle çektirilir, bu iki değer arasında değil ise diğer röleler çekilir. Ölçülen her ADC sonucu RS232 protokülü ile bilgisayara gönderilir ve 100 ms de bir LCD göstergeye ADC sonucu yazdırılır. Sistemin blok diyagramı Şekil 5.1' de gösterilmiştir.



Şekil 5.1 Sistemin blok diyagramı

5.1 Uygulamada Kullanılan Donanımın Açıklanması

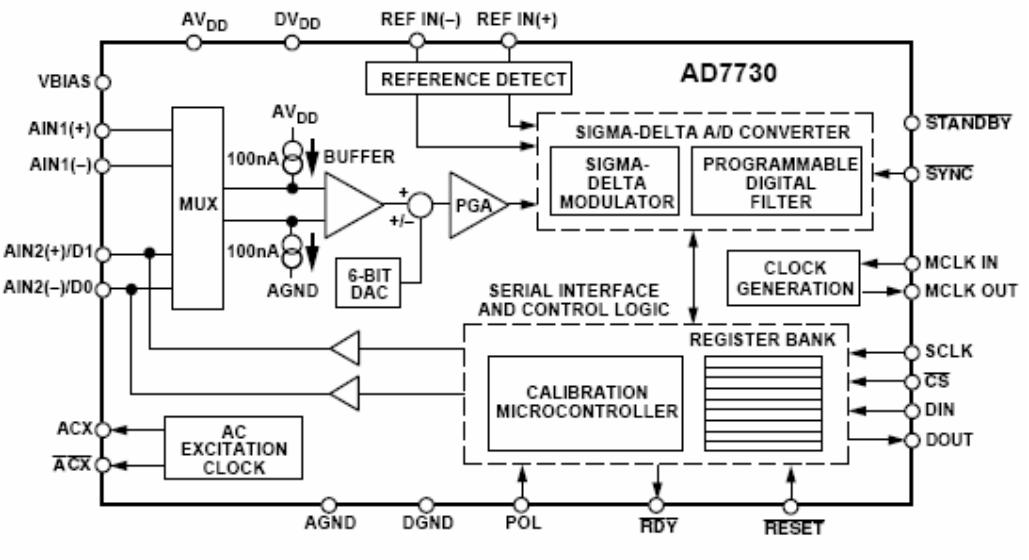
5.1.1 Kullanılan Mikro Denetleyici

Sistemde diğer kartlardan gelen bilgileri alan ve bu kartları kontrol eden FREESCALE M68HC908LK24 modelinde 64 pinli, 24Kbyte ROM ve 768 byte RAM alanlarına sahip bir mikro denetleyici kullanılmıştır. Bu mikro denetleyici diğer kartlardaki mikro denetleyicilerle

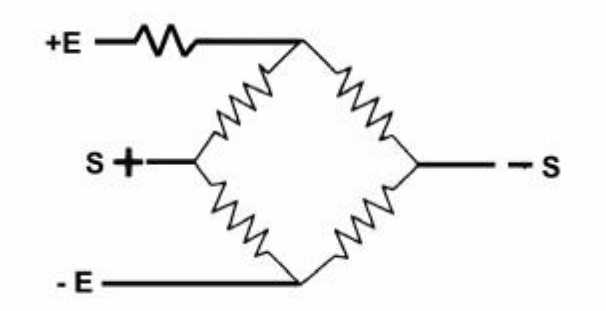
I2C protokolü ile haberleşmektedir. Diğer kartların enerjileri de ana kart üzerinden verilmektedir.

5.1.2 Analog Sayısal Dönüştürücü Kartı

ADC kartı üzerinde 24 bitlik ∑−Δ bir ADC kullanılmıştır. 2mv/V luk bir yük hücresi ADC'ye bağlanmıştır. Yük hücresi 5V ile beslenir, dolayısıyla 2mv/V luk yük hücresi 0-10mv aralığında bir çıkış verir. Bu şu anlama gelir; yük hücresi minimum yükte 0V maksimum yükte 10mV verir. Yük hücresi aslında direnci basınçla değişen bir direnç köprüsüdür. Yük hücresindeki dirençlerin değerleri 350 ohm dur. Yük hücresinden elde edilen gerilimi ölçmek için Analog Devices firmasının AD7730 entegresi kullanılmıştır.



Şekil 5.2 AD7730 Analog sayısal dönüştürücüsü iç yapısı (www.Analog.com)



Şekil 5.3 Yük hücresi

5.1.3 RS232 Kartı

RS232 kartı üzerinde ana işlemciden I2C protokolü ile gelen bilgileri RS232'ye çeviren bir işlemci ve TTL seviyesini RS232 lojik seviyesine çeviren MAX232 entegresi vardır. Haberleşme ayarları aşağıdaki gibidir:

Saniyedeki bit sayısı : 9600

Veri bitleri :8

Eşlik: yok

Dur bitleri: 1

Akış denetimi: yok

5.1.3.1 RS232 Haberleşme Protokolü

RS232 standardı ile iletim seri bir şekilde ve asenkron olarak yapılmaktadır. RS232 hatları TTL sinyal seviyelerini (+5V, OV) taşımazlar. Tipik olarak gerilim seviyeleri +12 V ve -12V'dur. Fakat RS232 hatları, +25V DC'ye kadar yüksek olan sinyal seviyeleri ile -25 V DC'ye kadar düşük olan sinyalleri taşıyabilir. Bilindiği üzere bilgisayardaki veri iletimi ikilik sistemde olmaktadır. Lojik 1'e +5V karşılık gelirken, lojik 0'a OV seviyesi denk gelir. Bu tür bir çevrime TTL (Transistor, Transistor Lojik Level) çevrimi denir. Bu, bilgisayar içindeki haberleşme standardı kabul edilir. Bilgisayar içindeki veri transferlerinde TTL seviyeli sinyallerin kullanılması birkaç sebepten dolayı avantajlıdır.

Güç Yönünden

Isı dağılımının az olmasından

Bu tür çalışan aletler için sürücüye ve alıcıya ihtiyaç duyulmadan doğrudan bağlantı yapılabilir.

TTL aletler yüksek hızda çalışabilir. Bu durum bilgisayar içindeki veri transferleri için çok uygundur.

TTL haberleşmesinde mesafe arttıkça sonra çok ciddi problemler ortaya çıkmaktadır. Ayrıca TTL, dışarıdan gelen sinyallerden çok çabuk etkilenir. Dolayısıyla sinyaldeki birkaç Voltluk kayıp sinyalin belirsiz bölgeye düşmesine sebep olur.

Normalde bir konnektörün pinleri 4 kısımda incelenebilir.

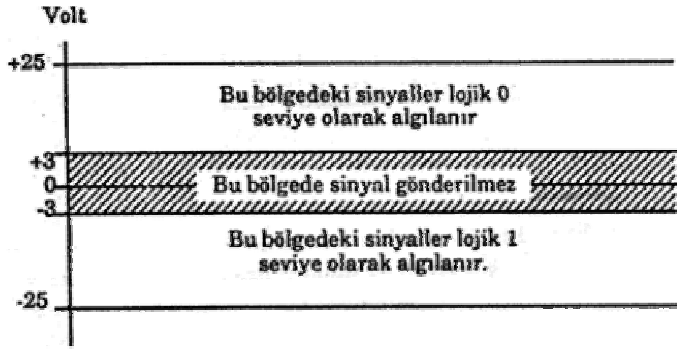
Data

Kontrol

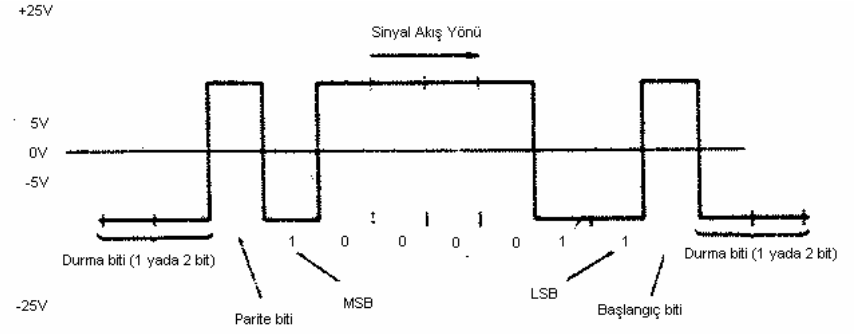
Zamanlama

Asenkron bir iletişim kullanıldığı için zamanlama kısmını kullanmayız.

PC'lerde tek gerilim (genellikle 5V) lojik 1 olarak ve OV ve toprak da lojik 0 olarak belirtildiğini söylemiştik. Bu tür iletim bilgisayar içinde sorun çıkarmaktadır. RS232 hatları gürültülü hatlar üzerinde çok uzun mesafelere sinyalleri göndermek zorunda kalabilmektedir. +5V, uzak mesafelere göndermede bir zayıflamaya maruz kalacaktır. Başarılı bir iletimin sağlanabilmesi için RS232 sinyalleri pozitif bir sinyal için +5V ile +15 V arasında ve negatif bir sinyal içinde -5V ile-15V arasında ve negatif bir sinyal içinse -5V ile -15 V arasında bir değer almalıdır. Bu aralığı bu şekilde tutarak, gürültüden dolayı oluşan gerilim dalgalanmalarından etkilenmesini de minimuma indirmiş oluruz. Bu şekilde bir sinyal göndericiden gönderildi diyelim. RS232 alıcısı için ise bu sinyal aralığı +3V'dan yukarısı için pozitif sinyal, -3V ve bundan aşağı için ise negatif sinyal olduğu anlaşılır. Burada -3V ile +3V arasındaki bölgede kararsız bir bölgedir. Bu bölgede bulunan bir sinyal, gürültü olarak kabul edilir. Sonuç olarak RS232 hatları pozitif olarak +25 V'a kadar, negatif olarak da -25V'a kadar olan sinyalleri taşıyabilir.



Şekil 5.4 RS232 hatlarındaki verilerin gerilime göre lojik yapısı



Şekil 5.5 RS232 haberleşmesinde bir byte veri yollama yapısı

5.1.4 Gösterge Kartı

Gösterge kartı üzerinde 6 dijitli bir segment LCD ve bu LCD'yi sürmek için HT1621 LCD sürme entegresi mevcuttur.

5.1.5 Röle Kartı

Röle kartı bir giriş/çıkış kartıdır. Röle kartı üzerinde optocoupler lı dört adet giriş ve 5V luk röleler ile dört adet çıkış bulunmaktadır..

5.2 Programın Çalışması

Sistem dört görev üzerine kurulmuştur. Bunlar sırasıyla ADC, Röle Kontrol, Display ve PC görevleridir. Her bir görev kendine ait işi istenen zamanda yada verilen işaret ile yapar. Program çalışmaya başladığında ilk önce donanımsal ilklendirmeleri yapar (giriş çıkış pin koşullamaları , timer gibi çevre birimlerinin ayarlanması vs.). Daha sonra işletim sisteminin ilklendirmesi yapılır . Görevler oluşturulur ve OSStart() fonksiyonu ile işletim sistemi başlatılır. İşletim sistemi çalışmaya başladıktan sonra görevlerde sırası geldikleri zaman çalışırlar. Görevlerin içerikleri ve çalışması aşağıda verilmiştir.

5.2.1 Analog Sayısal Çevrim Görevi – Öncelik Seviyesi 8

ADC görevi 80 ms de bir ADC kartından sonuç okur. Sonucu filtreden geçirir ve bu değeri ADC sonucu olarak saklar. OSMboxPost() fonksiyonu ile ADC sonucunun hazır olduğunu bu sonucu bekleyen görevlere bildirir.

ADC görevi sonucu oluşturduğunu belirtmek için mesaj kutusuna bir mesaj koyar. OSMboxPost() fonksiyonu içinde işletim sistemi , bu mesajı mesaj kutusuna koyduktan sonra

mesaj kutusuna bir mesaj konmasını bekleyen görev var mı diye bakar. İşletim sistemi Röle Kontrol görevini bu mesajı bekler konumda bulur. İşletim sistemi Röle Kontrol görevini ECB' nin (Event Control Blok) bekleme listesinden OSEventTaskRdy() fonksiyonu ile çıkarır. Böylece Röle Kontrol görevi çalışmaya hazır hale gelmiştir. OSSched() fonksiyonu daha sonra bekleme listesinden çıkarılan fonksiyonun mesajı kutuya koyan fonksiyondan daha yüksek öncelikli olup olmadığına bakar. ADC görevi Röle Kontrol görevinden daha yüksek öncelikli olduğu için görev değişimi yapılmaz. Eğer Röle Kontrol görevi ADC görevinden daha yüksek öncelikli olsaydı. OSSched() fonksiyonu çalıştığında bir görev değişimi olacak ,OSSched() fonksiyonu çağırıldığı yere dönmeyecek (Görev değişimi) ve Röle Kontrol görevi çalışmaya başlayacaktı. Fakat Röle Kontrol görevinin önceliği düşük olduğu için OSSched() fonksiyonu OSMboxPost() fonksiyonuna geri döner.OSMboxPost() fonksiyonunun içeriği ve kodları Ekler kısmında verilmiştir.

```
/*--------------------------------------------------------------------*/

void ADCTask(void *pdata)
{

INT8U   err;

    ADCInit();              // ilk ayarlamaları yap
    OSStartHardware();
    OSTaskCreate(ROLETask, (void*)0, (void*)&ROLETaskStk[TASK_STK_SIZE],10
    OSTaskCreate(PCTask, (void*)0, (void*)&PCTaskStk[TASK_STK_SIZE], 11);
    OSTaskCreate(LCDTask, (void*)0, (void*)&LCDTaskStk[TASK_STK_SIZE],12);


    for(;;)
    {

    GetADCResult (ADCBuff);                         // ADC sonucunu oku
    ADCResult=MedianFilter(ADCBuff);                //filtre
    OSMboxPost (AdcMbox, (void *)&ADCResult);       //sonucun alındığını bildi
    OSTimeDly(80);                                  //80 ms bekle


    }
}
/*--------------------------------------------------------------------*/
```

Şekil 5.6 ADC görevi

ADC görevi ADC sonucu hazırlamış ve bu sonucu bekleyen görevi de waiting(bekleme) konumundan çıkarıp ready(hazır) konuma almıştır. Program ADC görevinin işletilmesiyle devam edilir.ADC görevi en sonunda OSTimeDly() fonksiyonu ile 80ms lik bir beklemeye girer. Böylece 80 ms sonra waiting (bekleme) konumundan kalkıp runing (çalışır) konuma geçer.

ADC görevi OSTimeDly() fonksiyonunu işletip kendisini waiting(bekleme) konumuna

aldıktan sonra işletim sistemi Hazır Listede bekleyen görev olup olmadığına bakar. Az önceki mesaj verme işlemi içinde Röle Kontrol görevi çalışmaya hazır hale gelmiş ve ready (hazır) konuma alınmıştı. Uygulamamızda Röle Kontrol görevinin önceliği ADC görevinden sonra sistemdeki en yüksek öncelikli görev olduğu için hazır listede başka görevler de çalışmaya uygun konumda olsalar dahi Röle Kontrol görevi çalışmaya başlar.

5.2.2 Röle Kontrol Görevi – Öncelik Seviyesi 10

Röle Kontrol görevi ADC sonuçlarını elindeki iki sınır değer ile karşılaştırır ve sonuç bu iki değer arasında ise röle konumlarını değiştirir. ADC sonucunun hazır olmasını OSMboxPend() fonksiyonu ile bekler eğer mesaj kutusunda erişilebilir bir mesaj yoksa işletim sistemi bu görevi waiting(bekleme) konumuna alır. Görev bir mesaj gelene yada bir zaman aşımı olana kadar waiting (bekleme) konumunda kalır.

```
void ROLETask(void *pdata)
{
char *rxmsg;
INT8U err;
asm nop;

    for(;;)
    {
    rxmsg= (char *)OSMboxPend (AdcMbox, 0, &err); //adc sonucunu bekle
    asm nop;
    RoleControl (&ADCResult);                      //role kontrollerini yap
    OSMboxPost (AdcMbox, (void *)&ADCResult);    //sonucun alındığını bildir
    }
}
/*---------------------------------------------------------------------*/
```

Şekil 5.7 Röle kontrol görevi

Mesaj kutusunda bir mesaj var ise ve Röle Kontrol görevi çalışmaya hazır görevler arasında en yüksek öncelikli görev ise işletim sistemi bu görevi çalıştırmaya başlar. Görev ADC sonucuna göre röle kontrollerini yaptıktan sonra mesaj kutusuna ADC sonucunun erişilebilir olduğunu belirtmek için bir mesaj koyar. Bu esnada bu mesajı bekleyen PC görevi vardır. PC görevi aslında röle görevi ile beraber ADC sonucunu bekliyordu fakat sonuç mesaj kutusuna konduğunda Röle Kontrol görevinin önceliği PC görevinden yüksek olduğu için Röle Kontrol görevi waiting (bekleme) konumundan ready (hazır) konuma gelmiş fakat daha sonra işletilmişti. Şimdi PC görevi Röle Kontrol görevinin çağırdığı OSMboxPost() fonksiyonu içindeki OSEventTaskReady() fonksiyonu ile bekleme listesinden çıkarılır ve ready (hazır) konuma alınır. Ancak yine PC yi hazır hale getiren ve çalışan Röle Kontrol görevi yüksek öncelikli olduğu için çalışmaya devam eder. Röle görevi bu işlemlerden sonra yine

OSMboxPend() fonksiyonu ile yine ADC sonucunun hazır olmasını bekler. Mesaj kutusunda sonucun hazır olduğunu belirten bir mesaj olmadığı için ise waiting (bekleme) konuma alınır. ADC ve Röle Kontrol görevleri waiting (bekleme) konumundadırlar. Artık PC görevi diğer görevlere göre daha yüksek öncelikli olduğu için çalışmaya hazırdır.

5.2.3 PC görevi – Öncelik Seviyesi 11

PC görevi aldığı ADC sonucunu RS232 üzerinden bilgisayara aktarır. Alınan veriler bilgisayarda hyper terminal aracılığıyla görülebilir. PC görevi sadece ADC sonucunun hazır olmasını bekler, sonucun hazır olduğunu algıladıktan sonra mesaj kutusunu temizler, RS232 den verileri gönderir ve tekrar ADC sonucun hazır olmasını bekler.

```
/*---------------------------------------------------------------------*/

void PCTask(void *pdata)
{
char *rxmsg;
INT8U err;
asm nop;
    for(;;)
    {
    rxmsg= (char *)OSMboxPend (AdcMbox, 0, &err);    //ADC sonucunun hazır
    SendADCResult(&ADCResult);                       //PC ye gönder

    }
}
/*---------------------------------------------------------------------*/
```

Şekil 5.8 PC görevi

5.2.4 Gösterge görevi – Öncelik Seviyesi 12

LCD görevi 100 ms de bir ADC sonucunu LCD göstergeye yazar. ADC sonucun güncel olup olmadığına bakmaz.

```
/*---------------------------------------------------------------------*/
void LCDTask(void *pdata)
{
    for(;;)    {

        PrintADCResult(&ADCResult);          //LCD ye yazdır
        OSTimeDly(100);                      //100 ms bekle
    }
}
/*---------------------------------------------------------------------*/
```

Şekil 5.9 LCD görevi

Görev LCD'ye veriyi yazıp kendini 100ms lik bir beklemeye OSTimeDly() fonksiyonu ile

sokar. LCD görevi ile ilgili önemli olan şey bu görevin sistemdeki diğer görevlerden daha düşük öncelikli olmasıdır. Dolayısıyla bu görevin süresinde sapmalar olabilir ama bu sapmalar hayati önem taşımamaktadır çünkü göstergeyi gören bir kullanıcı yada bir operatör olacaktır. Bu yüzden sistemde gösterge görevinin yaptığı işler çok önemli değildir, bunun için de gösterge görevine en düşük öncelik verilmiştir.

5.2.5 Görevlerin Öncelik Seviyelerinin Belirlenmesi

ADC görevi sistemdeki bütün görevlerin ve sistemin temelini oluşturur. ADC sonucu elde edilmelidir ki diğer görevler kendi işlerini yapabilsinler. Dolayısıyla ADC görevine en yüksek öncelik verilmiştir.

İkinci en yüksek öncelik Röle Kontrol görevine verilmiştir. Çünkü ADC sonucu elde edildikten sonra yapılması gereken en önemli şey bu sonuca bakarak bazı röleleri kontrol etmektir.

PC görevi Röle Kontrol kadar önemli değildir çünkü PC görevinde yapılan şey bir kayıt tutulabilmesi için elde edilen her ADC sonucunu bilgisayara göndermektir.Bu sonuçlar bir rapor yada veri tabanı için kullanılabilir. Önemli olan bunların tam zamanında gönderilmesi değil kayıpsız bir şekilde gönderilmesidir.

LCD görevi sadece operatör için bilgilendirme amaçlı olduğundan en düşük önceliklidir. En düşük öncelikli görev olduğu için de bu görev çalışırken diğer görevlerden biri hazır hale gelirse LCD görevinin çalışması durur ve diğer görevler işletilmeye başlar.

5.3 RTOS kullanmaya ihtiyaç var mıydı?

Bu tez çalışmasında gömülü sistemler için gerçek zamanlı işletim sistemleri incelenmeye çalışıldı. Burada bir kavramı yanlış anlamamak gerekir. Gerçek zamanlı demek işlerin çok hızlı bir şekilde yapılacağı olarak algılanmamalıdır, bu terim işin zamanında yapılacağını belirtir. İşin çok hızlı yapılabilmesi için sistemin çok iyi tanınması ve buna göre donanım seçilmesi gibi faktörler de devreye girer. Burada önemli olan nokta gerçekten RTOS kullanmaya ihtiyacımız var mıdır? Bu soruya cevap verebilmek için tasarlayacağımız sistemi çok iyi tanımamız ve RTOS lar hakkında yeterli bilgiye sahip olmamız gerekir.

Günümüzde RTOS ların kullanımı giderek artmasına rağmen hala tasarımların çoğu klasik yazılım mantığı ile yapılmaktadır. Klasik yazılım mantığı basit ve kolay uygulamalar için kullanışlı olarak görülebilir fakat sistem karmaşık bir hal almaya, zamanlama önem

kazanmaya başladıktan sonra bir RTOS kullanımı yada en azından basit bir işletim sistemi mantığıyla yazılım yapmaya gerek vardır.

Mühendisler olarak bizleri gerçek zamanlı işletim sistemi kullanmaya iten en önemli faktör zamandır. Burada zamandan kastettiğim şey sadece tasarlanan sistemin istenen zamanlama beklentilerine uyması değildir.

Bir tasarım, daha en başında proje tanımının çıkarılıp üretime sokulması ve sahada düzgün bir şekilde çalışmasına kadar olan süreç olarak değerlendirilmelidir. Burada birinci aşama tasarımın yapılıp prototipin ortaya çıkarılmasıdır. Prototip bir an önce ortaya çıkarılmalıdır ki, gerçek ortam testleri de yapılabilsin. Günümüzdeki acımasız ve rekabetçi piyasa koşullarında da ürünlerin tasarım planı yapıldıktan sonra bir an önce ortaya çıkarılması gerekmektedir. Bu yüzden tasarımların da hızlı bir şekilde sorunsuz olarak ortaya çıkarılması gerekir. Gerçek zamanlı işletim sistemi ilk olarak bu aşamada devreye girer.

Bir işletim sistemi kullanmak yazılımcının bir çok ayrıntıyla uğraşmamasını sağladığı için tasarım sürecini hızlandırır.

Gerçek zamanlı işletim sistemi kullanmak ise sistemin ihtiyacı olan zamanlamalara da uyulmasını ve zaman beklentilerinin yakalanmasını sağlar. Yapılan değişiklikler ile yazılıma yapılan ekler mevcut olan zamanlamaları bozmaz. Herhalde tasarımcılar için en zor işlerden biri de budur. Çünkü yaptığınız değişikliklerin eski zamanlamaları etkilememesi için çok dikkatli ve iyice düşünerek değişiklik yapmanız gerekir. Halbuki bir RTOS kullanıyorsanız yaptığınız ek önemli ise önceliğini yüksek vererek sisteme eklerseniz istediğiniz zamanlamayı elde edersiniz.

6. SONUÇ VE ÖNERİLER

Tezin başında ortaya konan gömülü sistemlerin giderek karmaşıklaşması, kritik zamanlı çalışması ve tasarım sürelerinin azalması gerekliliği sonucunda gömülü sistemlerde de gerçek zamanlı işletim sistemleri kullanma zorunluluğu ortaya çıkmıştır.

Bu tezde öncelikle böyle bir işletim sistemine ihtiyaç duyabilecek sistemler anlatılarak, daha sonra da bu sistemler için kullanılabilecek gerçek zamanlı işletim sistemleri detaylı bir şekilde incelenerek, gömülü sistemler için gerçek zamanlı işletim sistemlerinin kullanımının gerekliliği üzerinde durulmuştur. Örnek olarak Micrium firmasının MicroC/OS-II işletim sistemi alınarak, incelenmiş ve bir uygulama gerçekleştirilmiştir. Uygulama için bir RTOS'un temel özelliklerini gösterebilecek 4 basit görev seçilmiştir. Böyle bir uygulama hatta daha karmaşık sistemler de klasik yaklaşım ile yapılabilir ancak bu yeni yaklaşım bazı avantajlar sağlamaktadır. Tabii ki avantajları yanında dezavantajları da bulunmaktadır.

Gerçek zamanlı bir işletim sisteminin avantajları olarak aşağıdaki özellikler sıralanabilir;

- Zaman beklentileri olan sistemlerin, kolayca tasarlanmasını, beklentilerin karşılanmasını ve kritik zamanlamaların yakalanmasını sağlar.
- Eklenen yeni fonksiyonlar sistemdeki diğer zamanlamaları etkilemez ve büyük değişiklikler yapmadan yazılıma eklenebilir.
- Uygulama kodunu birbirinden bağımsız görevlere bölerek tasarımı basitleştirir.
- Tasarım sürecini hızlandırır

Dezavantajları olarak aşağıdakiler sıralanabilir;

- Ekstra maliyet getirir
- Ekstra RAM/ROM ihtiyacı duyar
- %2 ile %4 arasında CPU kullanımını arttırır.

Bir RTOS kullanmak için öncelikle bu RTOS'un satın alınıp alınmayacağına yada yazılacağına karar verilmelidir. Bir çok ticari firma, kullanıcılar için tasarlanmış, hatadan ayıklanmış ve test edilmiş RTOS lar sağlar. Böyle bir RTOS satın almak tasarımcıyı RTOS'u yazma süresinden kurtarır. Tamamen test edilmiş, denenmiş, test sonuçları belli olan bir RTOS kullanılmasını sağlar. Öte yandan böyle bir durumda tasarımcı kodun tamamına hakim

olamaz. Büyük firmalar kodun tamamına hakim olabilmek ve istedikleri gibi geliştirebilmek için kendi RTOS larını yazarlar. Fakat bunun için sağlam bir iş gücü gerekmektedir. Benim tez çalışması sonunda elde ettiğim sonuç hazır RTOS'u kullanmanın onu yazmaktan daha etkili olacağıdır. Çünkü bazen geliştirme aşaması aylar alabilmektedir. Onun yerine eğer zamanlamanın çok önemli olduğu kritik bir sistem yapılıyor ise güvenilir katı zamanlamalı bir RTOS almak daha mantıklıdır. Eğer zamanlama çok kritik değilse ve daha basit bir sistem yapılıyor ise fiyatı uygun yada ücretsiz bir RTOS temin edip kullanılabilir.

RTOS seçiminde dikkat edilmesi gereken konular:

- Desteklediği mikroişlemciler ve programlama dilleri
- Tasarımda kullanılabilecek semafor, kuyruk yapıları gibi hizmetleri sağlayıp sağlamadığı
- Bazı RTOS ların özellikleri kullanıcı tarafından kapatılıp açılarak ölçeklenebilir, böyle bir özelliğe sahip mi?
- Seçilecek RTOS'un benchmark sonuçları var mı? Bu sonuçlar tasarlanan sistemin ihtiyacı olan zamanlamaları karşılıyor mu?
- Protokol yığınları, haberleşme servisleri, grafik kütüphaneleri gibi yazılım bileşenleri var mı?
- Hata ayıklama araçları var mı ?
- Teknik destek
- Kaynak kodu açık mı değil mi? Bazı RTOS lar kaynak kod olarak satılırken bazıları obje kodlu olarak satılır. Kaynak kodu açık olan RTOS ların tüm yazılımını görülebilir ve kodun tamamına hakim olunabilir.
- Lisansı var mı? Bir RTOS temin edildiğinde genelde başlangıç için bir para ve daha sonra her bir sistem içinde küçük paralar verilir.

Şu anda piyasada 100-150 adet RTOS var ve bunlar 8-16-32 bit işlemciler için kullanılmaktadır. Bazıları tamamen bir işletim sistemi halindedir ve sadece Gerçek zamanlı çekirdek değil giriş çıkış yöneticisi, dosyalama sitemi, gösterge sistemi, kütüphaneleri, hata ayıklayıcılar derleyicileri içerir. Bir RTOS'un maliyeti 50$ ile 30.000$ arasında olabilir. RTOS'un konulduğu her çip için ise ayrıca bir ücret gerekir bu ücrette 5$ ile 250$ arasında

değişmektedir.

RTOS ların zamanlamanın çok önemli olduğu askeri savunma sistemleri, araç kontrol sistemleri gibi sistemlerin yanında artık, küçük işlemciler ile daha basit sistemlerde de kullanılması giderek artmaktadır ve giderek gerekli hale gelmektedir. Zamanla küçük gömülü sistemlerin çoğunda RTOS kullanmak zorunlu hale gelecektir.

KAYNAKLAR

Barr, M. (1999), Programming Embedded Systems in C and C++, O'Reaily, New York.

Futacı, S. (2002), "Object Oriented Programming in Microcontroller Based Systems with Extremely Limited Resources", İstanbul Universty Engineering Faculty Journal of Electrical and Electronics, 593.599.

Gerhardt, M. ve Locke, D. (2005), "Real-Time Architecture – Past, Present, and Future" Embedded Systems Conference, 10 March 2005, San Fransisco.

Joseph, M., (2001), Real-Time Systems Specification, Verification and Analysis, Tata Research Development and Design Centre, London.

Labrosse, J. (2002), MicroC/OS-II The Real Time Kernel Second Edition, CMP Books, San Fransisco.

Labrosse, J. (2005), a,"Desisging with Real Time Kernel",b,"Inside Real Time Kernel", Embedded Systems Conference, 10 March 2005, San Fransisco.

Li, Y., Potkonjak, M., Wolf, W.(1997), "Real-Time Operating Systems for Embedded Computing", IEEE Computer Society Journal, 1063:6404/97.

Moore, R. (2001), How to Use Real-Time Multitasking Kernels in Embedded Systems, Micro Digital Associates, California.

Tomiyama,H.,Chikada, S., Honda,S.,Takada,H. (2005), "An RTOS-Based Approach to Design and Validation of Embedded Systems" 0:7803-9060-1/05, IEEE.

Paulin, P., Liam, C., Cornero, M., Nacabal, F., Gossens, G. (1997), "Embedded Software in Real-Time Signal Processings Systems : Aplication and Architecture Trends", Proceedings of the IEEE, Vol. 85, No.3, March 1997, 0018:9219/97$10.00.

Qing, L. ve Yao, C. (2003), Real-Time Concepts for Embedded Systems, CMP Books, San Fransisco.

Ward, R., (2003), "Practical Real-Time Techniques", Embedded Systems Conference, 26 April 2003, San Fransisco.

Zeidman, B. (2005), "Real-Time Operating Systems for Systems On a Chip", Embedded Systems Conference, 10 March 2005, San Fransisco

Zurel, K. (200), C Programming for Embedded Systems, R&D Books , Kansas

INTERNET KAYNAKLARI

www.analog.com

www.freertos.org

www.freescale.com

www.micrium.com

www.techonline.com

ÖZGEÇMİŞ

Doğum tarihi	15.01.1980	
Doğum yeri	İstanbul	
Lise	1993-1997	Şehremini Lisesi
Lisans	1997-2001	Yıldız Teknik Üniversitesi Elektrik Mühendisliği Bölümü

Çalıştığı kurum

2002-Devam ediyor Tüm Elektronik Mühendislik Ltd Şti.