1906003052015

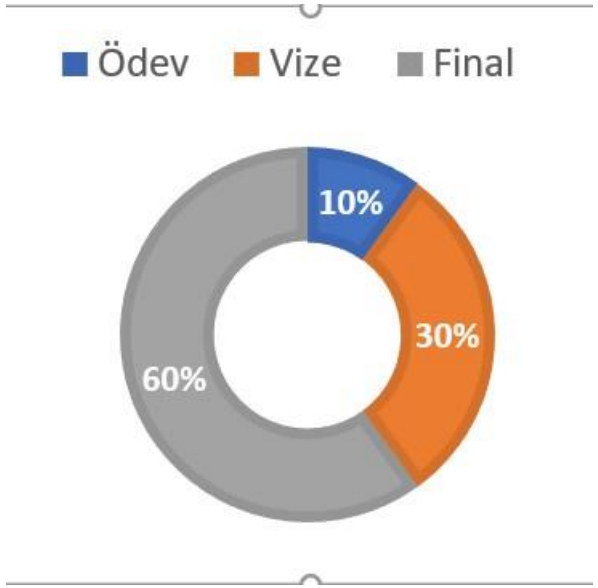
İşletim Sistemleri

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

Giriş

Ders Günü ve Saati:
Çarşamba: 13:00-16:00

- Uygulama Unix (Linux) İşletim sistemi
- Devam zorunluluğu %70
- Uygulamalar C programlama dili üzerinde gerçekleştirilecektir. Öğrencilerden programlama bilgisi beklenmektedir.



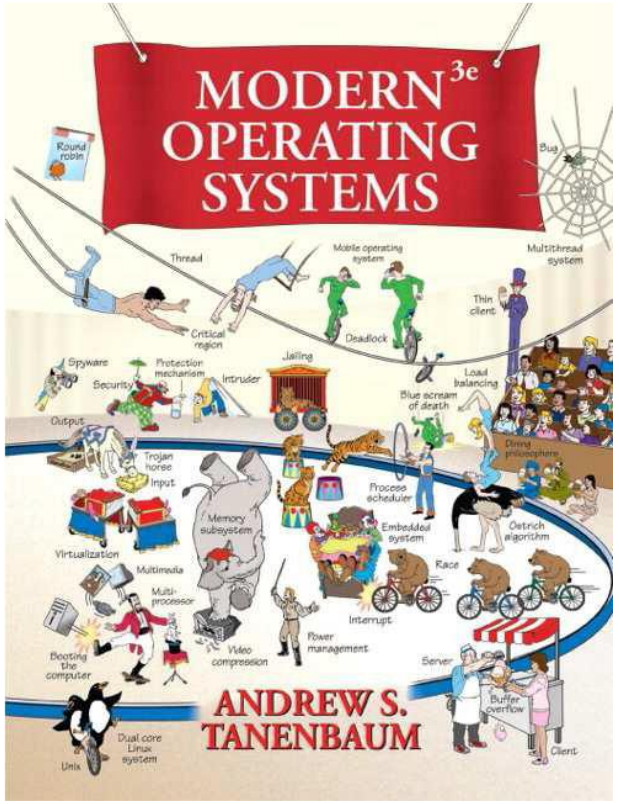
HAFTA	KONULAR
Hafta 1	: İşletim sistemlerine giriş, İşletim sistemi stratejileri
Hafta 2	: Sistem çağrıları
Hafta 3	: Görev, görev yönetimi
Hafta 4	: İplikler
Hafta 5	: İş sıralama algoritmaları
Hafta 6	: Görevler arası iletişim ve senkronizasyon
Hafta 7	: Semaforlar, Monitörler ve uygulamaları
Hafta 8	: Vize
Hafta 9	: Kritik Bölge Problemleri
Hafta 10	: Kilitlenme Problemleri
Hafta 11	: Bellek Yönetimi
Hafta 12	: Sayfalama, Segmentasyon
Hafta 13	: Sanal Bellek
Hafta 14	: Dosya sistemi, erişim ve koruma mekanizmaları, Disk planlaması ve Yönetimi
Hafta 15	: Final

Giriş

Kaynaklar:

– Modern Operating Systems, 3rd Edition by Andrew S. Tanenbaum, Prentice Hall, 2008.

– Bilgisayar İşletim Sistemleri (BIS), Ali Saatçi, 2. Baskı, Bıçaklar Kitabevi.



DERS - 3

Sistem Çağrıları(System Calls)

Sistem Çağrılarının Türleri

- İşlem Kontrolü
- Dosya Yönetimi
- Cihaz Yönetimi
- Bilgi Sağlama
- İletişim
- Koruma

POSIX tarafından tanımlanan, birkaç sistem çağrısını inceleyelim.

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Süreç Yönetimi için Sistem Çağrıları

fork() sistem çağrısından başlayabiliriz. fork() sistem çağrısı UNIX sistemlerde yeni bir süreç oluşturmak için tek yoldur.

Bu fonksiyon asıl sürecin bire bir kopyasını oluşturur (dosya tanınlayıcıları, yazmaçlar,... herşey).

Kopyalama işleminden sonra iki süreç (ana ve çocuk) birbirlerinden tamamen ayrılırlar.

Kullandıkları veriler kendilerine özgü olur. (Fakat programın text kısmı aynı olduğu için iki süreç tarafından paylaşılır)

Örnek (Linux)

print-pid.c

```
#include <stdio.h>
#include <unistd.h>

int main(){
    printf("süreç numarası : %d\n", (int)getpid() );
    printf("Ana sürecin süreç numarası:%d\n", (int)getppid() );
}
```

-getpid(): geriye sürecin süreç numarasını çevirir.
*-getppid():*çalışan sürecin ana sürecinin süreç numarasını geri çevirir. (get_parent_program_id)

Linux'da çalışan süreçleri ***ps*** komutu ile öğrenebilirsiniz.
$ps -e -o pid,ppid,command

fork() ve exec() in kullanımı

fork() : mevcut sürecin birebir kopyasını oluşturur, iki süreçde fork() fonksiyonundan sonraki satırdan itibaren kendi başlarına çalışmaya devam eder.

exec(): fonksiyonları kümesi, mevcut sürecin çalıştırmak istenilen başka bir programın sürecine dönüşmesini sağlar.

fork() ve exec() in kullanımı

fork.c

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
    pid_t cocuk_pid;

    printf("Ana sürecin pid = %d\n", (int)getpid() );

    cocuk_pid=fork(); if (cocuk_pid!=0){
        printf("burası ana sürectir, süreç id
        pid=%d\n",(int)getpid());
        printf("çocuk sürecin idsi pid =
        %d\n",(int)cocuk_pid);
    }else{
        printf("burası çocuk süreçtir, pid=%d\n",
        (int)getpid());
    }
    return 0;
}
```

fork-exec.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
//fonksiyon yeni bir programı yumurtlar(spawn)
//yani calistirir, mevcut süreci bu programa cevirir
int spawn(char *program,char** arg_list){
    pid_t cocuk_pid;

    cocuk_pid =fork();
    if (cocuk_pid!=0){
          return cocuk_pid;
    }else{
          execvp(program,arg_list);
          //eger hata olmus ise alt kisim calisir
          fprintf(stderr,"execvp de hata olustu\n")
          abort();
    }
}
int main(){
    char * arg_list[]={"ls", "-l","/", NULL};
    spawn("ls",arg_list);
    printf("basarili olarak ana program bitti\n");
    return 0;
}
```

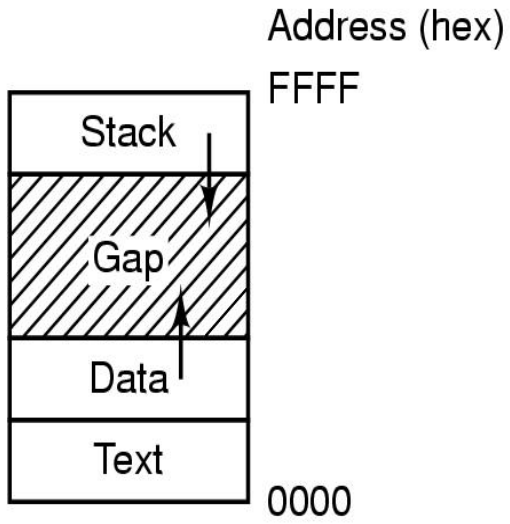
Süreçlerin Bellek Görünümü

UNIX temelli sistemlerde bellek üç kesime(segment) parçalanmıştır.

Veri Kesimi(Data Segment) : değişkenler

Stack Kesimi(Stack Segment): malloc() ile ayrılan yerler

Yazı Kesimi(Text Segment): Program kodunun olduğu bellek bölümü



Windows Win32 API

Programlama modelleri Windows ve Unix türevi işlerim sistemlerinin farklıdır.

Bir UNIX programı belirli bir görevi yerine getiren ve bu görevi yerine getirirken sistem çağrılarını kullanan bir programdır.

Bir Windows programı ise olay temellidir.Ana program belirli olayların gerçekleşmesini bekler. Bu olay gerçekleştiğinde de, olayı işleyecek(handle) olan prosedürü çalıştırır. Windowsunda sistem çağrıları bulunmaktadır. Unix sistemlerde genellikle sistem çağrısı ile çağrılacak olan kütüphane fonksiyonunun ismi aynıdır. Bu isimler POSIX tarafından tanımlanmıştır.

Windowsda durum bu şekilde değildir. Microsoft Win32 API (Application Programin Interface) adını verdiği bir prosedür kümesi tanımlamıştır. Programcılar işletim sisteminin servislerini kullanmak için bu prosedürleri kullanırlar. Bu arayüz tüm Windows işletim sistemleri tarafından kısmi olarak desteklenmektedir.

Yeni windows sistemlerinde bu prosedürler ve kullanımları farklılaştırılmaktadır.

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

İçerik

- Süreç Konsepti (Process Concept)
  - Tanımı
  - Süreç Durumları
  - Süreç Kontrol Bloğu (PCB)
- Süreç Operasyonları (Process Operations – POSIX)

- Süreç Zamanlama (Process Scheduling)
- Süreçler Arası İletişim (Inter Process Communication)

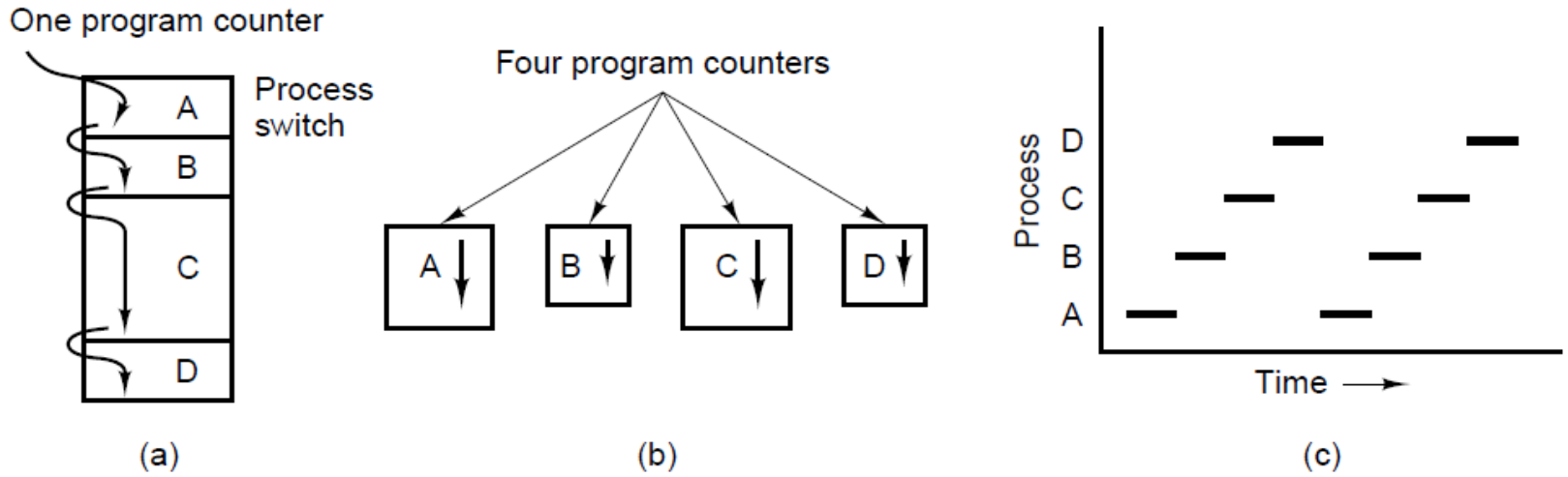
SÜREÇ YÖNETİMİ

Süreç: Çalışan bir programın soyutlamasıdır. Bir süreç yürütülmekte olan bir program olarak düşünülebilir. Program durgun komut dizisini tanımlarken, görev bu komut dizisinin devingen işletimine karşılık gelir.
Kaynaklar, ya oluşturulduğunda ya da yürütülürken süreçlere dağıtılır.
OS, süreç ve iplik yönetimi ile bağlantılı aşağıdaki faaliyetlerden sorumludur::

- Kullanıcı ve sistem süreçlerinin oluşturulması ve silinmesi
- süreçlerin zamanlaması;
- senkronizasyon mekanizmalarının sağlanması,
- süreçler arası iletişim ve kilitlenmeyle mücadele

SÜREÇ YÖNETİMİ

- Bir süreç, çalışan program olarak düşünülebilir. Fakat bir «text section» olarak adlandırılan programdan daha fazlasıdır.
- Süreç, mevcut faaliyetleri, ?program sayaç? Değerlerinin ve « register » kayıtlarını içerir.

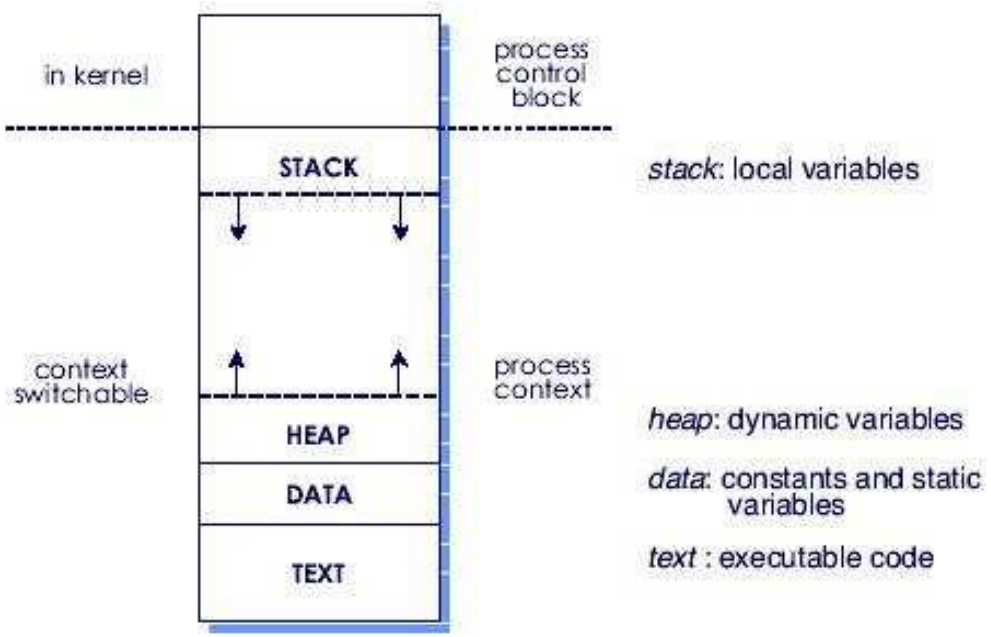


Süreçlerin Bellekteki Yapısı

Belleğe yüklenen bir görev;

Data: programın global değişkenlerini içerir.
Stack: Yerel değişkenler, fonksiyon parametreleri gibi geçici verileri tutar.
Heap: Süreç çalışma zamanında, dinamik olarak tahsis edilen bellek alanıdır.

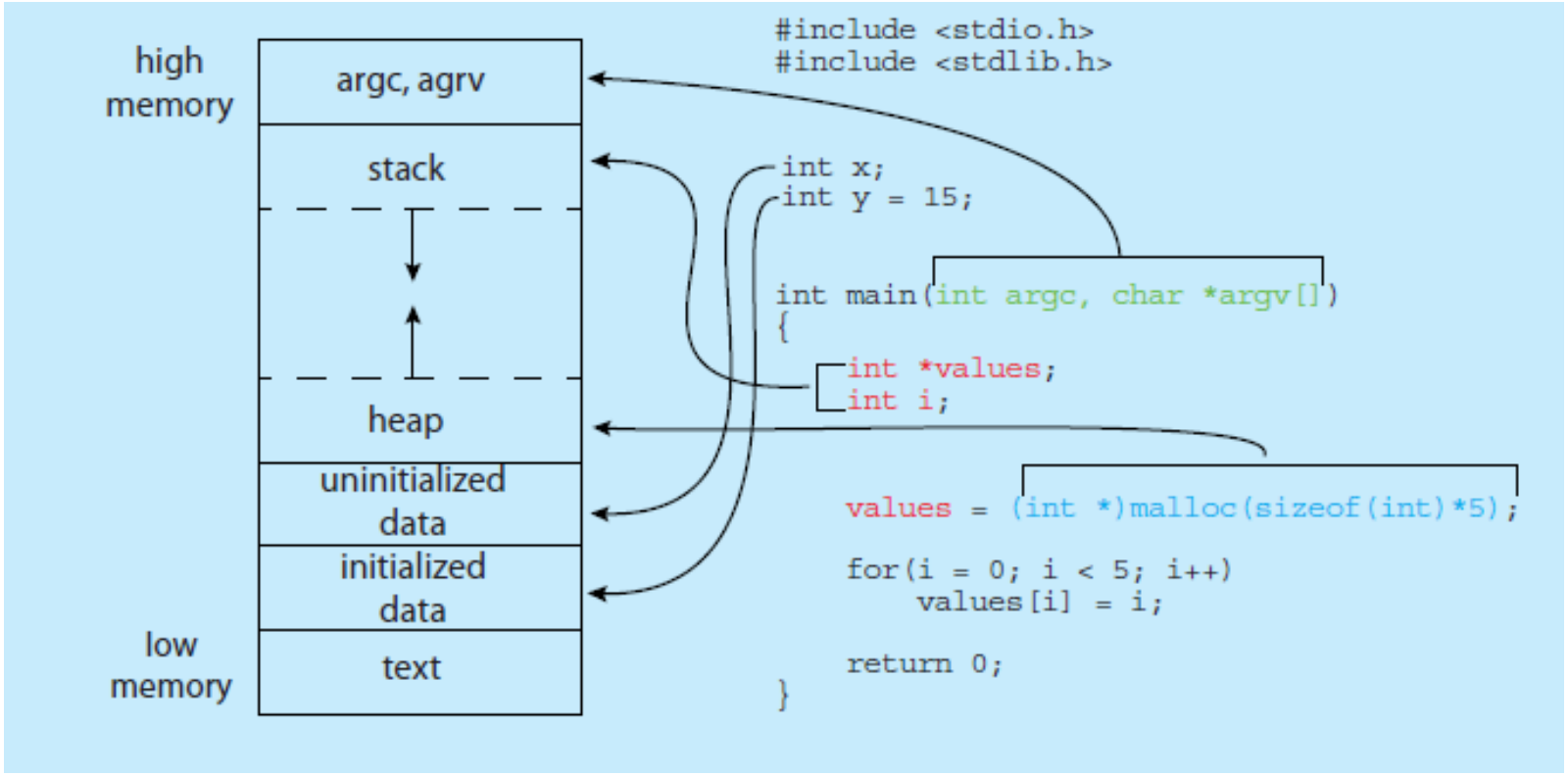
Bilgilerini içerir.



Bellekteki görev

- Bir program, çalışabilir bir dosya olarak belleğe yüklendiğinde, süreç halini alır.
- Aynı programa karşılık birden fazla görev olabilir. Bu görevler farklı görevlerdir. Bunların text kısımları aynı, data, heap ve stack kısımları farklıdır.

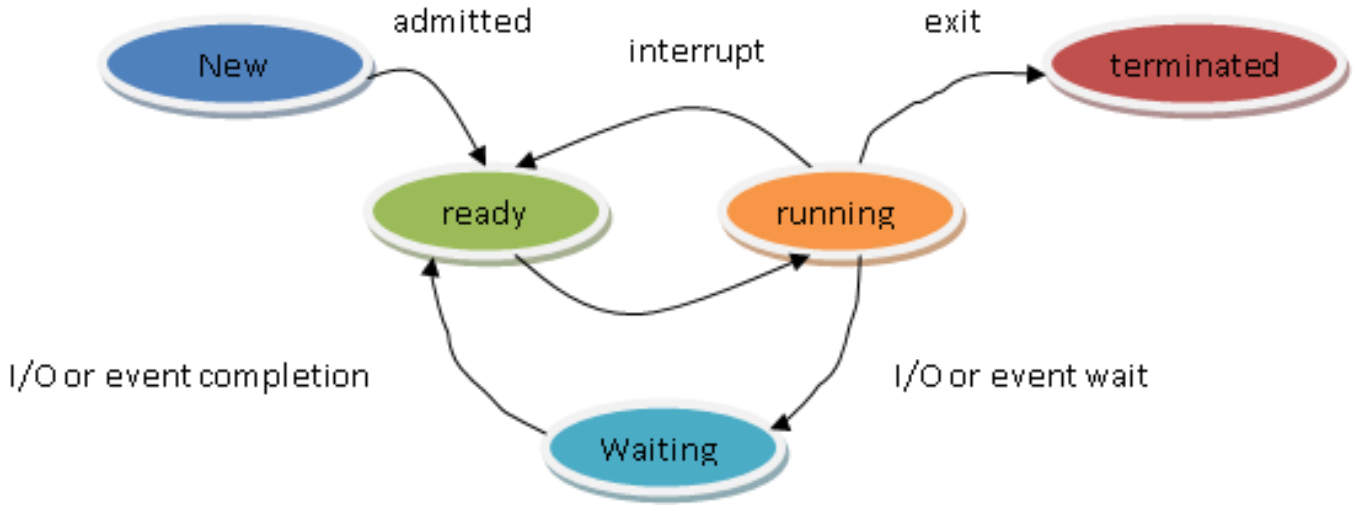
Süreçlerin Bellekteki Yapısı



Süreç Durumları

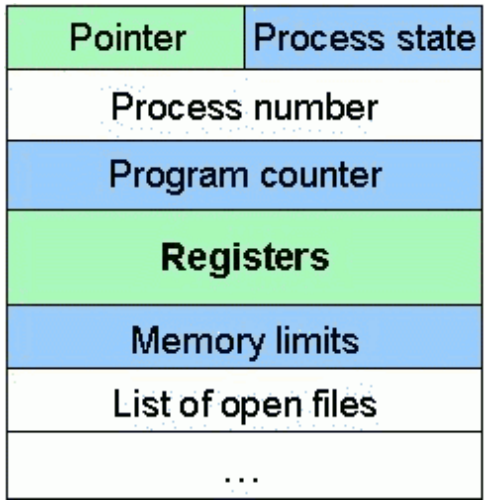
Bir süreç yürürlükte olduğu müddetçe aşağıdaki durumlardan birisinde bulunur.

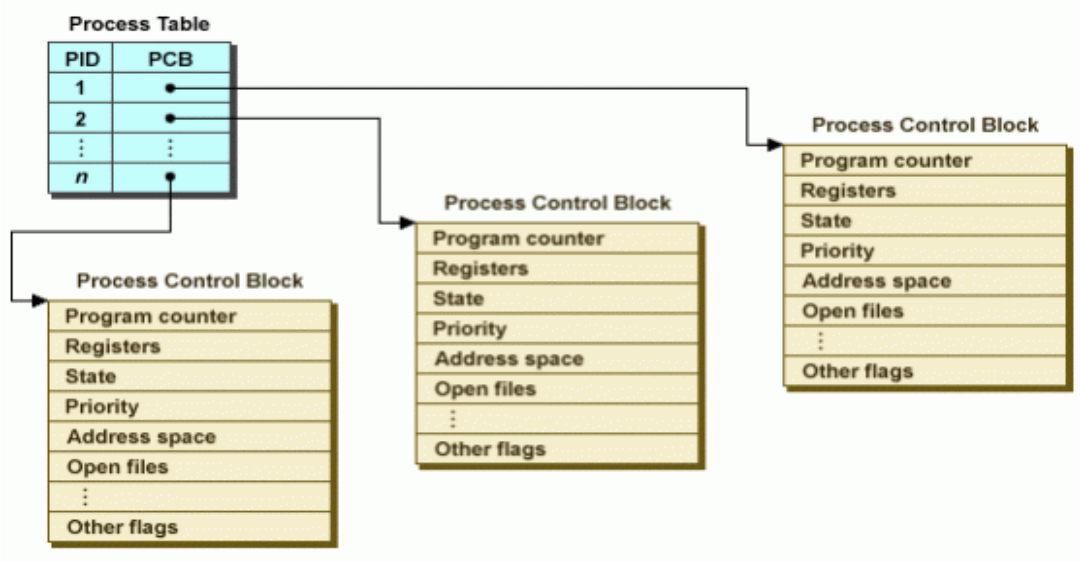
New : Süreç oluşturulur.
Ready : Süreç, hazır kuyruğundadır. Beklemeye neden olacak olay (event) tamamlanmıştır.
Running : Süreç komutları (instructions) yürütülür.
Waiting : Süreç, herhangi bir olay beklemektedir. (Ör. I/O event)
Terminated : İşlem yürütmeyi bitirmiştir.



Süreç Kontrol Bloğu (PCB)

İşletim Sistemi, süreç tablosu adı verilen ve işlem başına bir giriş olan bir tablo (yapı dizisi) tutar.
Bu girdilere **süreç kontrol blokları (PCB)** denir - buna görev kontrol bloğu da denir. Bunlar, süreçle ilgili bütün bilgileri saklar;
PCB; süreç çalışır durumdan, hazır veya bekliyor durumuna geçtiğinde saklanması gereken ve süreç çalışır duruma geçtiğinde sanki hiç durmamış gibi çalışması iin gereken program sayacı, stack göstericileri, bellek tahsisi, açık dosyaların durum ve bilgileri, hesap ve zamanlama bilgileri, içerir.





Süreç Kontrol Bloğu (PCB)

Bu bilgiler; Süreç Durum bilgileri ve Süreç kontrol bilgileri olmak üzere iki sınıfta toplanır. Bunlar;

Süreç Durumu : «New», «Ready», «wating», «running», ve «terminated» durumları

Program sayacı : Gelecek komutun (instruction) adresini tutar.

CPU Kayıtçıları : CPU mimarisine bağlı

CPU Zamanlama Bilgileri : İşlem önceliği ve kuyruk göstericileri gibi zamanlama (scheduling) parametreleri

Bellek Yönetim Bilgileri : İşletim Sistemine bağlı, «base» ve «limit» kayıcıları, sayfa ve segman tabloları gibi bellek yönetim bilgileri

Hesap Bilgileri : Kullanılan CPU ve gerçek zaman miktarını, zaman sınırlarını, hesap numaralarını, iş veya işlem numaraları

I/O Durum Bilgileri: Tahsis edilen I/O cihazları açık dosyalar

Süreç Kontrol Bloğu (PCB)

Tipik bir süreç tablosu girişinin bazı alanları.

Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Süreç Anahtarlama (Process Switching)

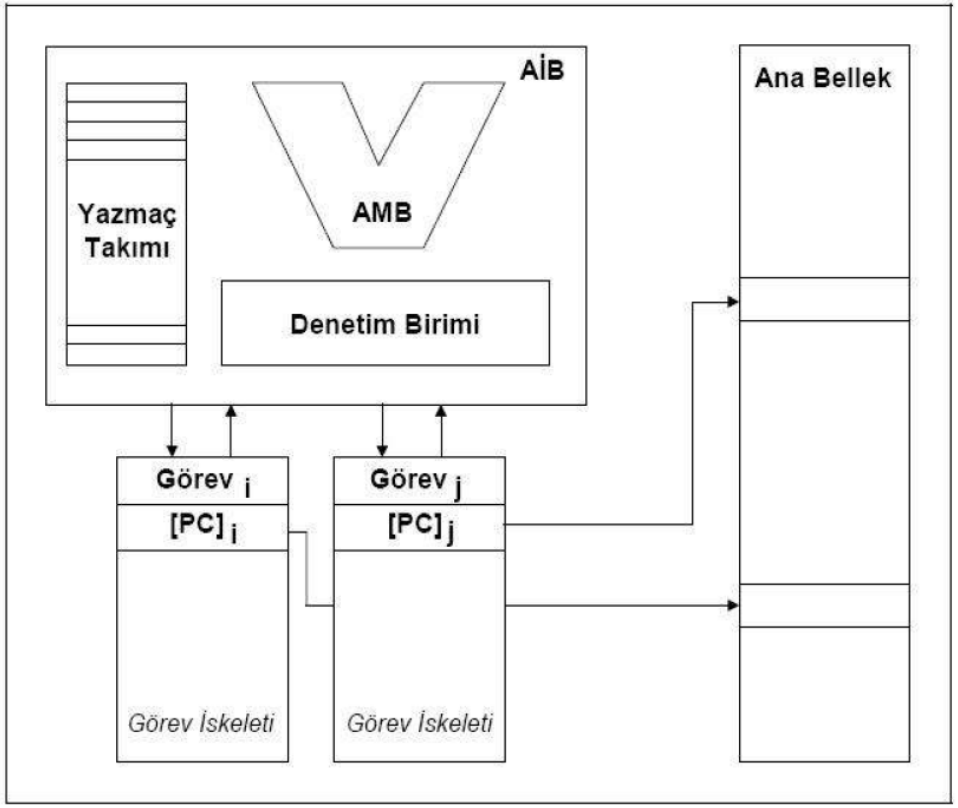
Kesintiler işletim sisteminin bir CPU çekirdeğini mevcut görevinden değiştirmesine ve bir çekirdek rutini çalıştırmasına neden olur. Bu tür işlemler genel amaçlı sistemlerde sıklıkla gerçekleşir. Bir kesinti meydana geldiğinde, sistemin CPU çekirdeğinde çalışan işlemin mevcut bağlamını kaydetmesi gerekir, böylece işlem bittiğinde bu bağlamı geri yükleyebilir, esasen işlemi askıya alır ve ardından devam ettirir. Bağlam, sürecin PCB'sinde temsil edilir. CPU kayıtlarının değerini, işlem durumunu ve bellek yönetimi bilgilerini içerir. Genel olarak, çekirdek veya kullanıcı modunda olsun, CPU çekirdeğinin mevcut durumunun bir durum kaydetmesini ve ardından işlemleri sürdürmek için bir durum geri yüklemesini gerçekleştiririz.

Süreç Anahtarlama (Process Switching)

CPU çekirdeğini başka bir işleme geçirmek, mevcut işlemin durum kaydetmesinin ve farklı bir işlemin durum geri yüklenmesinin yapılmasını gerektirir. Bu görev, bağlam anahtarı olarak bilinir ve Şekil 3.6'da gösterilmektedir. Bir bağlam geçişi gerçekleştiğinde, çekirdek eski işlemin içeriğini kendi PCB'sine kaydeder ve çalışmak üzere programlanan yeni işlemin kaydedilmiş bağlamını yükler. Bağlam değiştirme süresi tamamen ek yüktür, çünkü sistem geçiş yaparken hiçbir işe yaramaz. Anahtarlama hızı, makineden makineye değişir.

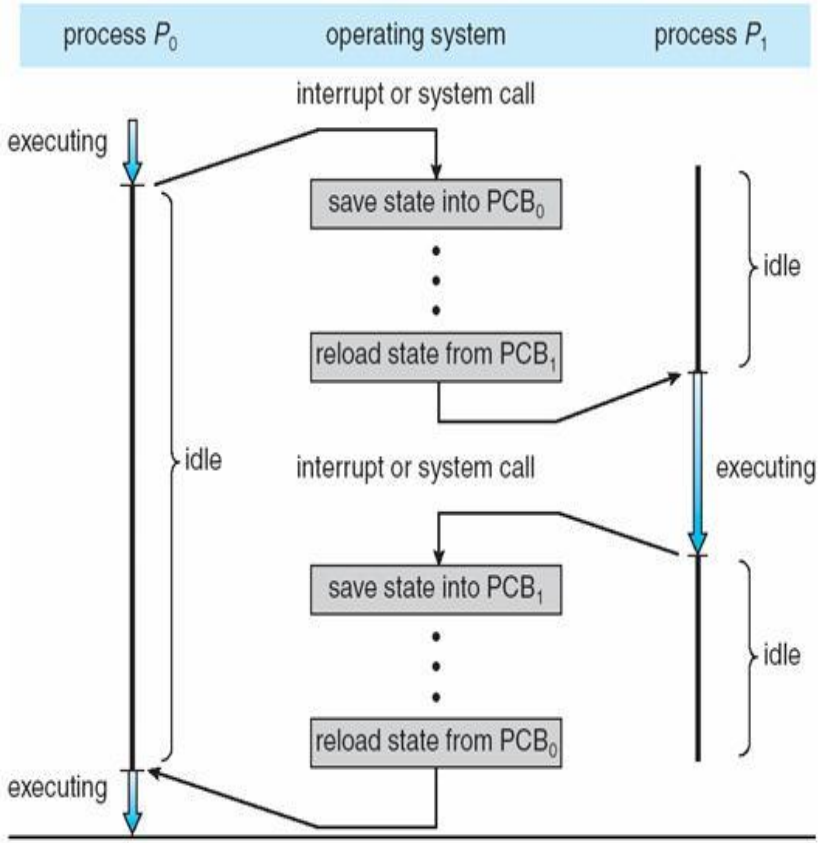
Süreç Anahtarlama (Process Switching)

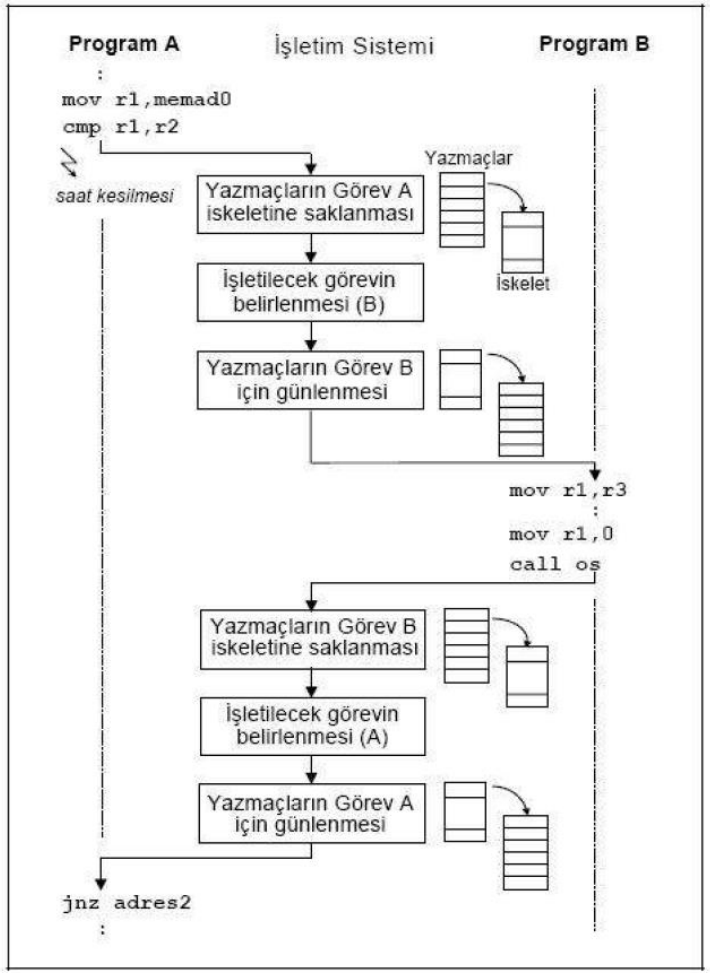
Süreçler arasındaki geçişler, Süreç yöneticisi tarafından gerçekleştirilir.



Süreç Anahtarlama (Process Switching)

Süreçler arasındaki geçişler, Süreç yöneticisi tarafından gerçekleştirilir.





Süreç Zamanlama

«Multi programming» amacı, CPU kullanımını en üst düzeye çıkarmak için bazı işlemlerin her zaman çalışmasını sağlamaktır.

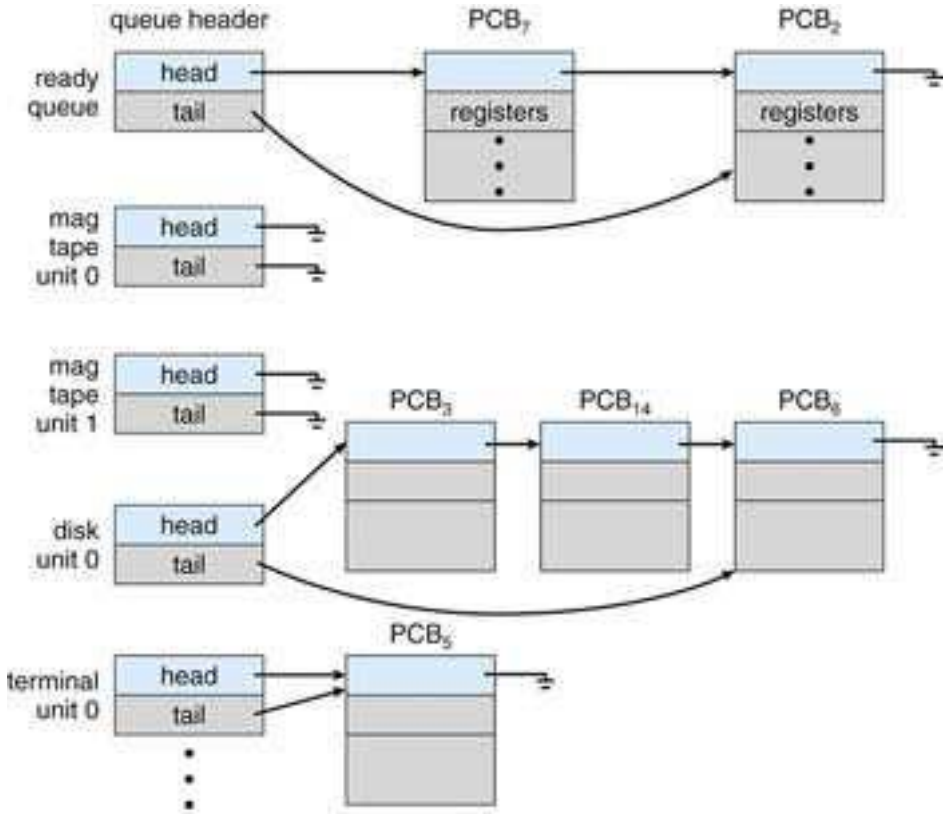
- Zaman paylaşımının amacı, CPU'yu işlemler arasında çok sık değiştirerek kullanıcıların her program çalışırken etkileşime girmesini sağlamaktır.
- Bu hedefleri gerçekleştirmek için işlemci zamanlayıcı, CPU üzerinde program yürütülmesi için kullanılabilir bir işlemi seçebilir (muhtemelen birkaç kullanılabilir işlem arasından seçer).
- Tek işlemcili bir sistem için asla birden fazla çalışan işlem olmaz.
- Eğer daha fazla işlem varsa, geri kalan CPU boşalıncaya ve beklenmelidir.

Süreç Kuyruklama

- Süreçler sisteme girerken, bunlar sistemdeki tüm süreçlerden oluşan bir iş kuyruğuna konur.
- Ana belleğe konan ve hazır olan süreçler hazır kuyruk adı verilen bir listede bağlı liste olarak saklanır.
- Hazır kuyruk başlığı, listedeki ilk ve son PCB'lere işaretçiler içerir. Her bir PCB, hazır sıradaki bir sonraki PCB'yi işaret eden bir işaretçi alanı içerir.

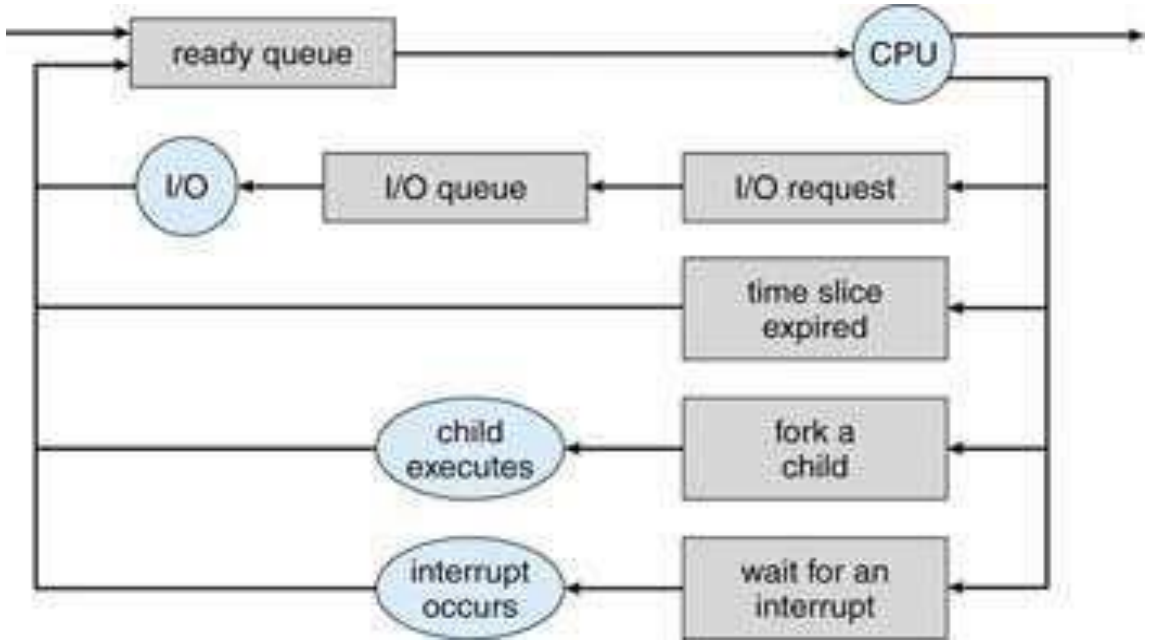
Süreç Kuyruklama

Belirli bir G / Ç aygıtını bekleyen işlemlerin listesine bir aygıt sırası denir. Her cihazın kendi cihaz sırası vardır



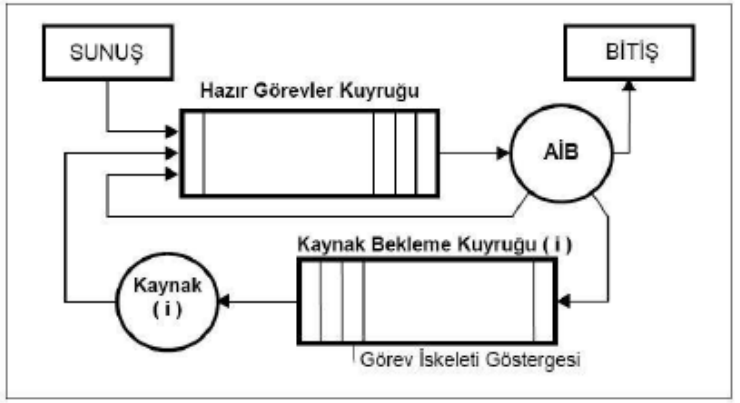
Süreç Kuyruklama

Süteç çizelgeleme kuyruk diyagramı

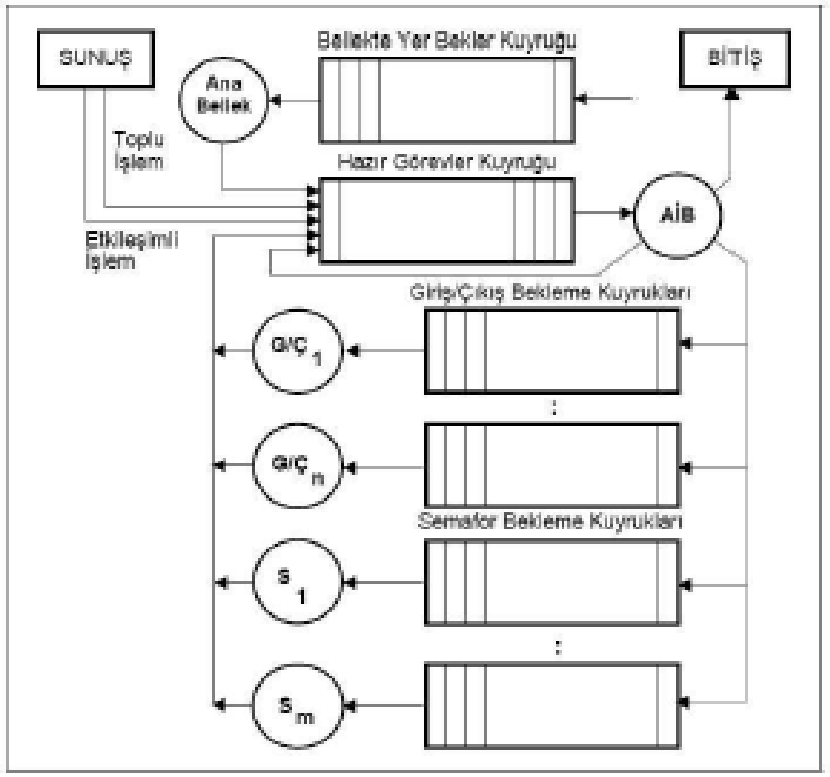


Her dikdörtgen kutu bir kuyruğu temsil eder. Hazır kuyruğu ve bir dizi aygıt kuyruğu olmak üzere iki tür kuyruk bulunur.
Daireler kuyruklara hizmet eden kaynakları temsil eder ve oklar sistemdeki süreçlerin akışını gösterir.

Süreç Kuyruklama



- Görevler tarafından paylaşılan kaynakların herbiri için ayrı bir kuyruk öngörülür



Görevlerle İlgili İşlemler

create(görev-kimligi, öznitelik-alani): Yeni bir gorev yaratılıp, bu goreve iliskin bir iskeletin olusturulmasını sağlayan sistem cağrısıdır. create komutunu calıstıran gorev (ata gorev), yaratılacak gorevin (oğul gorevin) kimliğini alır.

delete(görevkimligi): Kimliği verilen görevin iskeletini bos iskeletler listesine ekleyerek gorevin sistem icindeki varlığına son vermek icin kullanılır.

suspend (görev-kimligi,kuyruk-kimligi): Kimliği verilen gorevi, kimliği verilen bekler turu bir kuyruğa bağlamak amacıyla kullanılır.

`resume` (görev-kimligi,kuyruk-kimligi): Kimliği verilen gorevi, bağlı bulunduğu bekler turu kuyruktan hazır gorevler kuyruğuna aktarmak amacıyla kullanılır.

suspend, G/C isteminde bulunma, gorevler arası zamanuyumlama gibi durumlarda gorevin kendisi tarafından calıstırılır. resume cağrısı, ilgili gorevin dısındaki gorevler tarafından calıstırılabilir.

`delay(süre)`: Çalıstır gorevin, arguman olarak verilen sure kadar bekler durumunda kalması sağlanır.

`change-priority` (görev-kimligi, öncelik) change-attributes (görev-kimligi, öznitelik-alanı): Kimliği verilen gorevin oncelik ve oznitelik bilgilerinin gozlenmesi ve gunlenmesi amacıyla kullanılırlar.

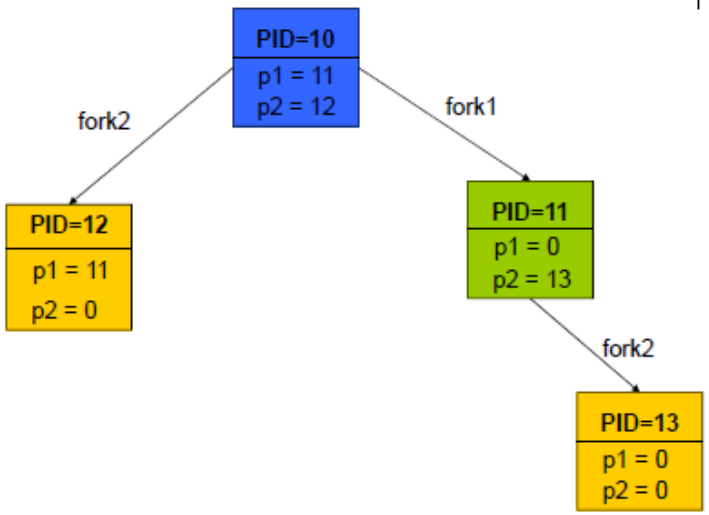
POSIX Görev yaratma

fork() sistem cağrısı, bu cağrıyı isleten gorevle (kimlik bilgileri dısında) aynı gorev iskeletine sahip bir diğer gorevi yaratıp hazır gorev durumuna getirmek için kullanılmaktadır.

execlp sistem cağrısı, gorev iskeletinde, işletilen programı temsil eden *code segment* ve islenen verileri temsil eden *user data segment* kesimlerini gunler.

```
int execlp(char* path, char* program, char* arg1, char* arg2, ...., char* argn, char* NULL)
```

POSIX Görev yaratma




```
int main()
{
   int p1, p2;

   p1 = fork();          /*fork1*/
   p2 = fork();          /*fork2*/
   printf ("%d\n", p1 + p2);
}
```

POSIX Görev yaratma

`fork()` ve `exec()` Kullanımı