YZM 3102
İşletim Sistemleri

Yrd. Doç. Dr. Deniz KILINÇ
Celal Bayar Üniversitesi
Hasan Ferdi Turgutlu Teknoloji Fakültesi
Yazılım Mühendisliği

BÖLÜM - 3

Bu bölümde,

- Proses Kavramı
- Proses Üzerindeki Bilgiler
- Proses Durumları
- Proses Control Block
- Planlama Kuyrukları
- Planlayıcılar
- Proses Yaratma / Sonlandırma / Listeleme

konularına değinilecektir.

Proses Kavramı

- Bir prosesi:

“Çalışmakta olan program

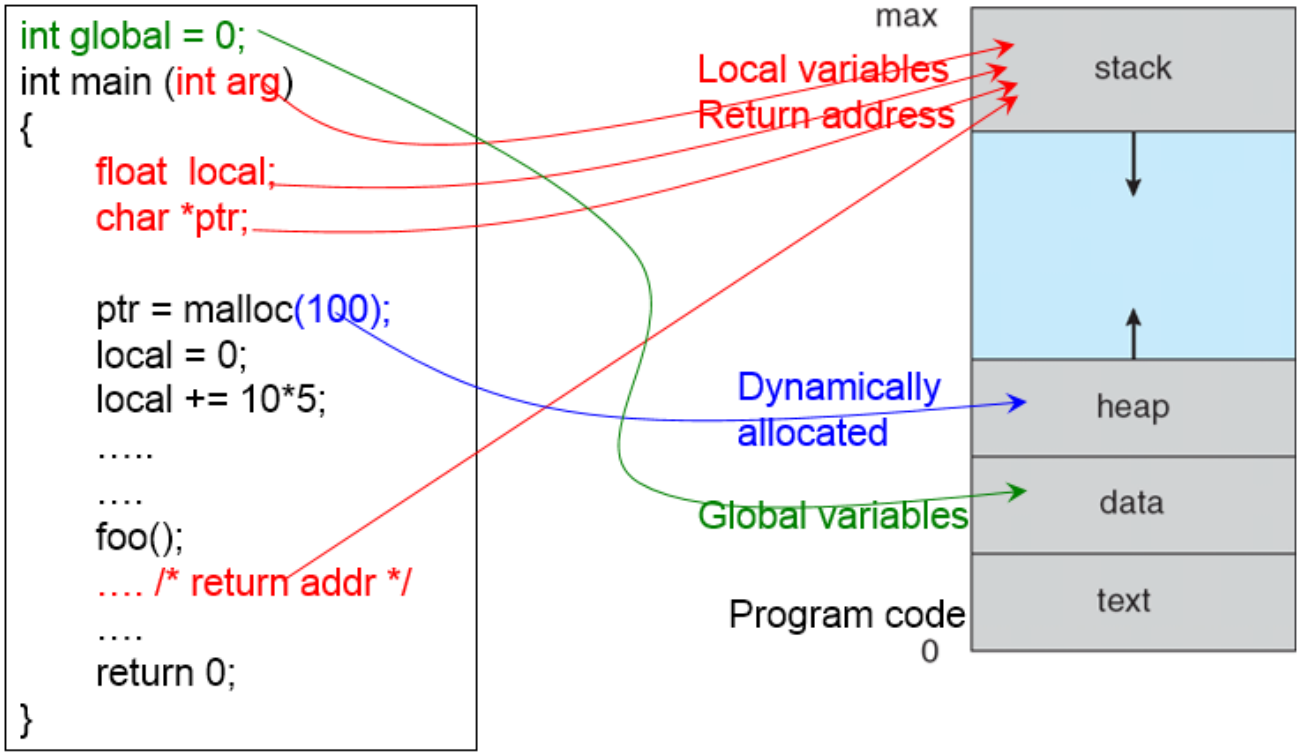
(program in execution)”

- olarak (bu tanım, kısa ve eksik de olsa) tanımlayabiliriz.
- Proses terimi yerine **iş**, **görev**, **işlem** ve **süreç** (job, task) gibi *farklı terimler de* **sıkça** kullanılmaktadır.
- Bir proses, bir program kodundan **çok daha fazlasıdır** ve çalıştığı sürece birçok kaynağa ihtiyaç duyar ve birçok bilgiyi barındırır.

Proses Üzerindeki Bilgiler

- **Metin bölümü (text section)** veya **kod bölümü (code section)** olarak adlandırılan, program kodlarının bulunduğu bölüm,
- Bir sonraki işletilecek olan komutu gösteren **sayaç** ve işlemci kayıtçılarının içeriği olan program sayacı (**program counter**) bölümü,
- *Fonksiyon parametreleri*, *geri dönüş değerleri* ve *lokal değişkenler* gibi **geçici bilgileri (temporary data)** tutan **stack (yığın)** bölümü,
- Global değişkenleri tutan **veri bölümü (data section)**,
- Prosesler çalıştığı sürece kullandığı ve dinamik olarak tahsis edilen **bir bellek alanı (heap) bölümü.**

Proses Üzerindeki Bilgiler (devam…)



Proses Üzerindeki Bilgiler (devam…)

- Program, **pasif** bir varlıkken (içerisinde bir takım *komutlar barındıran disk üzerindeki bir dosya*dır – çalıştırılabilir dosya – executable file),
- Proses **aktiftir**.
- Program, **diskten→belleğe** yüklendikten sonra prosese dönüşür.
- Bunun için *program ikonuna çift tıklanır* veya çalıştırılabilir dosyanın ismi *CLI'dan komut olarak* girilir.

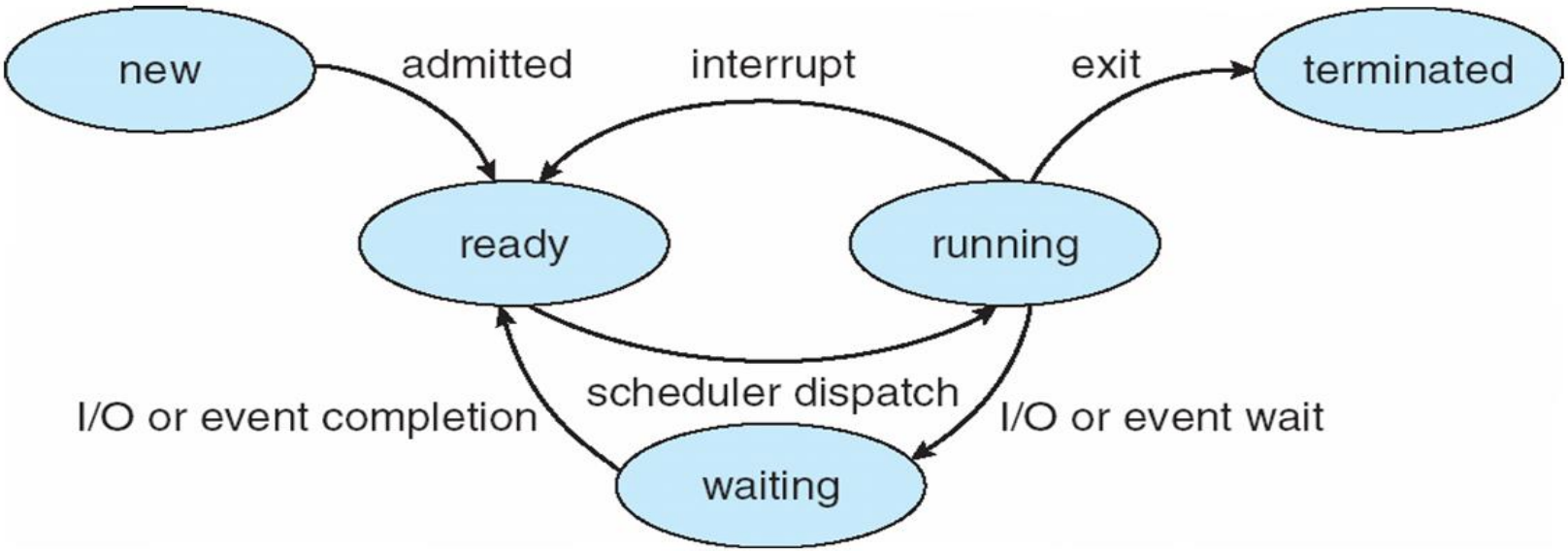
Proses Durumları

- Bir prosesin o anki aktivitesine göre ortaya çıkan durumları aşağıdaki gibidir:
  - **Yeni - New:** Proses yaratıldı.
  - **Çalışıyor - Running:** Proses komutları CPU'da çalıştırıyor.
  - **Bekliyor - Waiting:** Proses bir olayın gerçekleşmesi için bekliyor. Örneğin bir G/Ç işlemi.
  - **Hazır - Ready:** Proses bir işlemciye atanmak için hazır ve bekliyor.
  - **Bitti - Terminated:** Proses çalışmasını bitirdi.

Proses Durumları (devam…)

- Proses durumları işletim sistemlerinde *farklılık gösterse de*, bahsettiğimiz durumların her OS'da mutlaka bir karşılığı vardır.
- **Unutulmaması gereken** önemli noktalardan bir tanesi, bir işlemcide *bir anda* sadece bir proses **çalışıyor (running)** durumdadır.
- Diğer taraftan, birden fazla proses, **hazır (ready)** ve **bekliyor (waiting)** durumunda olabilir.

Proses Durumları (devam…)



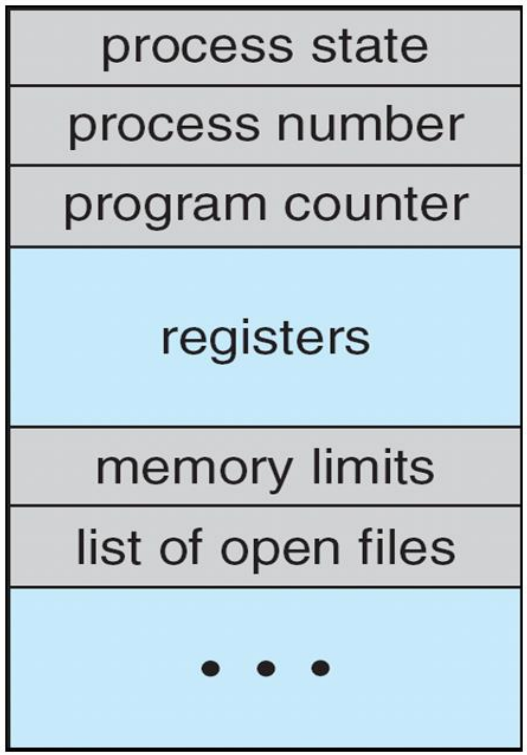
PCB - Process Control Block

- İşletim sistemindeki her proses içerisinde prosese spesifik bilgiler barındıran **PCB** (Process Control Block) ile gösterilir.
- Proses Durumu: Yeni, çalışıyor, bekliyor, hazır veya bitti şeklinde durumlar olabilir.
- Program Sayacı: Proseste çalıştırılacak bir sonraki komutu gösterir.
- CPU Kayıtçıları (CPU Registers): İşlemci mimarisine göre kayıtçıların sayısı ve türü değişmektedir. Proseste kullanılan kayıtçıların (AX, BX vb.) sahip olduğu bilgiler ve program sayacı bilgisi bir *kesme (interrupt) geldiğinde* mutlaka saklanmalıdır. Çünkü kesme sonrası proses kaldığı yerden devam edebilmelidir.

PCB - Process Control Block (devam…)

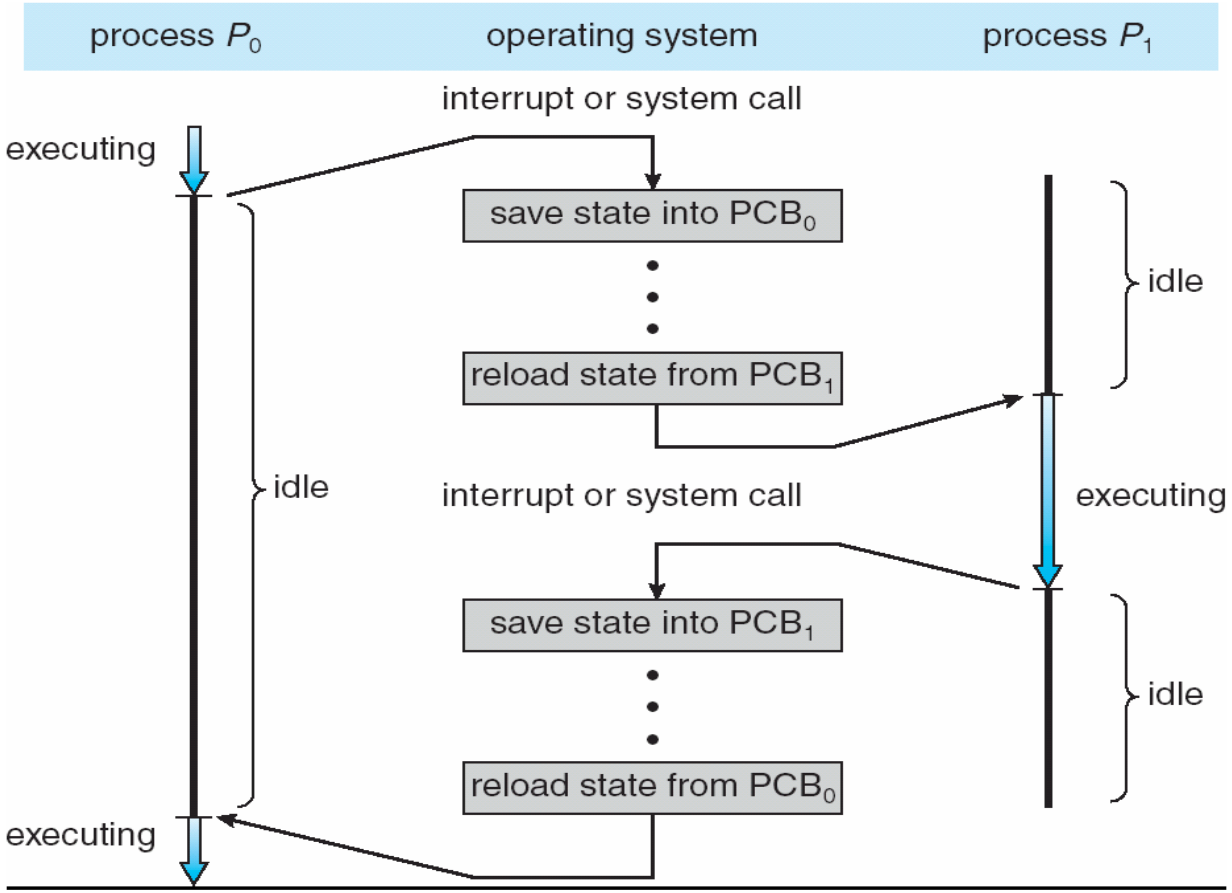
- CPU Planlama Bilgisi (CPU-Scheduling Information): Proses öncelik (priority) bilgisi, planlama kuyruğuna olan işaretçiler ve diğer planlama parametreleri ile ilgili bilgiler bulunmaktadır.
- Bellek Yönetimi Bilgisi: İşletim sistemi tarafından kullanılan bellek sistemleri; taban ve limit (tavan) kaydedicileri, sayfa tablosu veya bölüm tablosu gibi bilgileri içermektedir.
- Hesap Bilgileri (Accounting Information): Proses tarafından kullanılan CPU miktarı ve diğer parametrelerin kulanım zamanlarını içermektedir.
- G/Ç durum bilgisi: Proses tarafından kullanılan G/Ç aygıtlarının listesi, açılan dosyaların listesi, ağ bağlantıları vb. bilgileri içermektedir.

PCB - Process Control Block (devam…)



PCB - Process Control Block (devam…)

- PCB'ler prosesleri yönetmek için kullanılır. Örneğin **CPU bir prosesten diğer prosese geçiş yaparken** PCB kullanılır.



PCB - Process Control Block (devam…)

- P_0 prosesi çalışırken bir interrupt (kesme) veya sistem çağrısı geldiğinde P_0'ın durum (state) bilgisi (program sayacı bilgisi, CPU kayıtçıları vb.) PCB_0'da saklanmalıdır.
- Daha sonra P_1 prosesine ait durum bilgisi PCB_1'den yüklenir ve CPU yeni prosesi execute etmeye başlar.
- CPU'nun bir prosesten diğerine geçiş işlemi **context switch** olarak adlandırılır.
- Sistem, Context switch **süresince kullanılamaz** olduğu için bu işlemin süresi önemli bir ek yüktür.
- *Switch süresi* bilgisayara, bellek hızına register sayısına göre **değişkenlik gösterebilir**.
- Tipik olarak **birkaç milisaniye sürer**.

PCB - Process Control Block (devam…)

- Linux işletim sistemindeki PCB, bir C structure olan ve kernel kaynak kod dizinindeki <linux/sched.h> dosyasında bulunan **task_struct** ile gösterilir.
- Aşağıda task_struct'a ait bazı üye değişkenleri gösterilmektedir:

```
long state; /* state of the process */
struct sched_entity se; /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```

Proses Planlama(Process Scheduling)

- **Çoklu programlamanın** amacı sürekli çalıştırılacak bir proses bulunmasını sağlayarak CPU'yu sürekli meşgul (çalışır) tutmaktır.
- **Zaman paylaşımının** amacı CPU'yu prosesler arasında çok sık şekilde switch (yer değiştirerek) ederek, kullanıcının her programla etkileşimini devam ettirmektir.
- Bu amaçlara ulaşabilmek için **proses planlayıcı (process scheduler)** hazır birçok proses arasından uygun bir tanesini **CPU execution** için seçer.

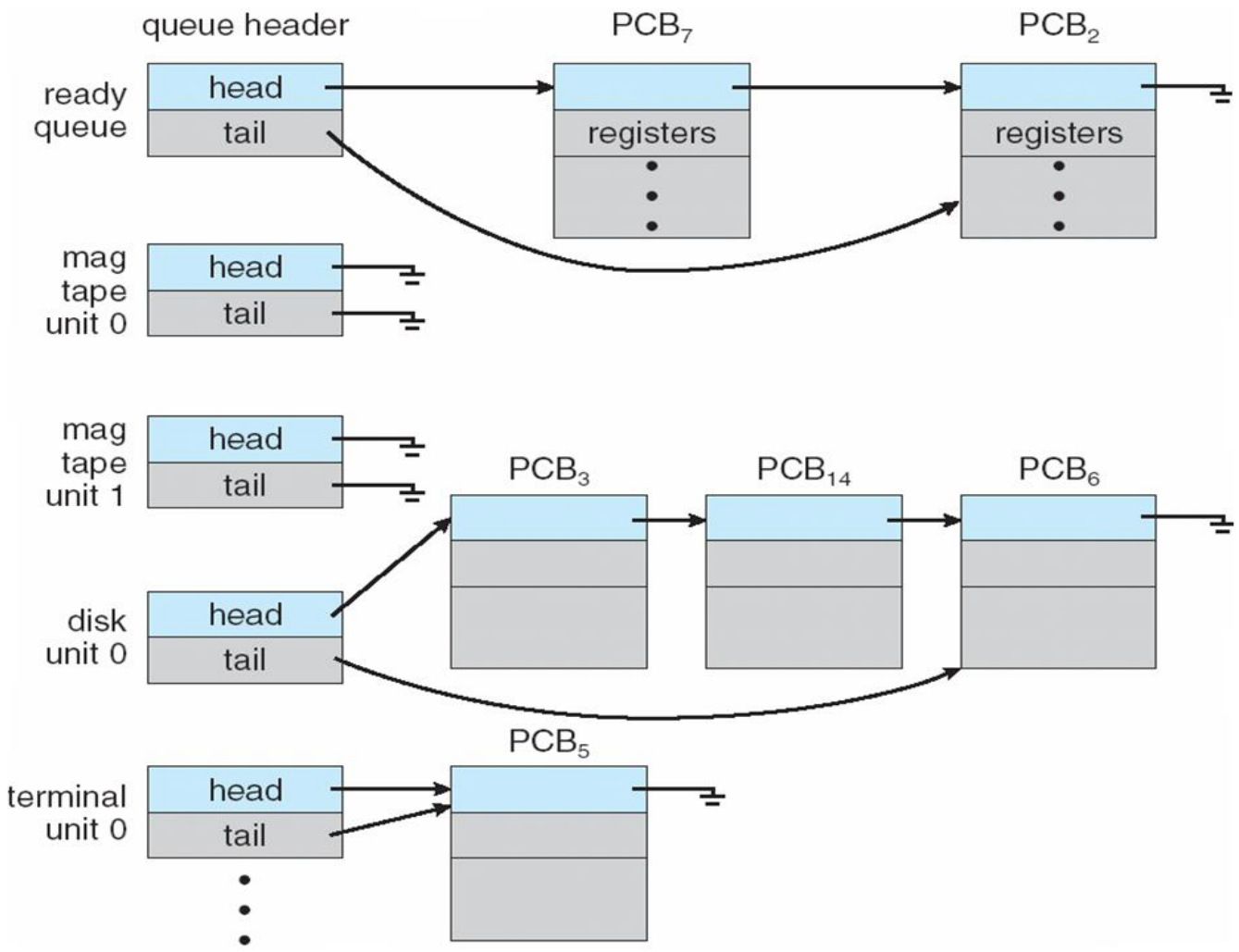
Planlama Kuyrukları (Scheduling Queues)

- Prosesler sisteme girdiklerinde, tüm proseslerin yer aldığı **iş kuyruğuna (job queue)** girerler.
- Ana bellekteki, ready veya wait durumunda olan prosesler, **ready queue** isimli *genelde linked-list türünde tasarlanmış* bir listede tutulurlar.
- Her ready-queue, başlık değişkeni *ilk ve son PCB'leri gösteren işaretçilere* sahiptir.
- Ready-queue'daki her PCB'de de *bir sonraki PCB'yi gösteren* **işaretçi** mevcuttur.

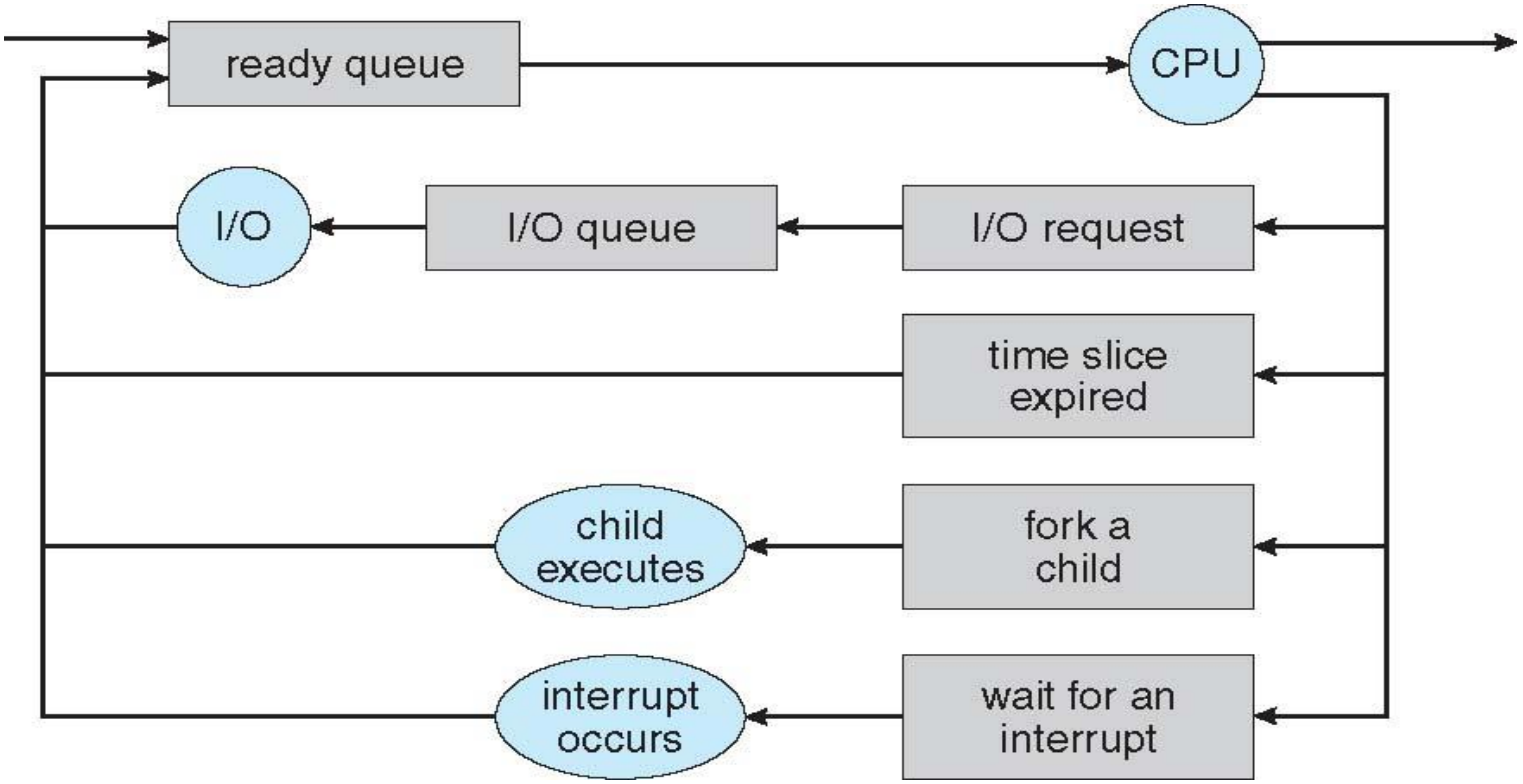
Planlama Kuyrukları (Sched. Queues) (devam…)

- Bir proses CPU'da bir süre çalışmışken er ya da geç ***CPU'dan bir süreliğine ayrılır***:
  - **Kesmeler** nedeniyle ayrılabilir.
  - **Başka eventin sonucu**nun bitmesini beklemeye girebilir.
  - Örneğin bir device için I/O talebinde bulunur ve beklemeye başlar. Bir prosesin (P) *disk gibi paylaşımlı bir kaynağı kullanma* **talebinde** bulunduğunu varsayalım. Sistemde birçok proses olduğu için disk diğer proseslerin I/O talepleriyle meşgul olabilir ve **sonuç olarak**, **P prosesi disk için beklemeye girebilir**. Spesifik bir I/O device için bekleyen proseslerin olduğu listeye **device queue** adı verilir. Her device'a ait bir **device queue** bulunmaktadır.

Planlama Kuyrukları (Sched. Queues) (devam…)



Queueing Diagram (Kuyruk Diyagramı)



- Her kutu, **bir kuyruğu** *ifade etmektedir*. İki tip kuyruk: **ready queue** diğeri de **device queue**

Planlayıcılar (Schedulers)

- Bir proses, **yaratılmasından sonlandırılmasına kadar** olan yaşam sürecinde farklı kuyruklarda yer alır.
- Kuyruklardaki bu prosesler bir şekilde *OS tarafından seçilmelidir*. İlgili kuyruktaki seçme işlemini ilgili scheduler gerçekleştirir.
- Long-term Scheduler-LtS (job scheduler): Disk gibi cihazlarda kuyruklanan prosesleri bu havuzdan seçen ve belleğe yükleyen (aslında ***ready kuyruğuna*** taşıyan) scheduler türüdür.
- Short-term Scheduler-StS (CPU scheduler): Çalışmaya hazır olan proseslerden bir tanesini seçerek çalıştırılmasını sağlar.

Planlayıcılar (Schedulers) (devam…)

- Her iki planlayıcı arasındaki *temel fark*, **çalıştırma sıklığıdır**.
- StS çok daha sık ve hızlı şekilde CPU için proses seçmelidir.
- Genelde bir StS, en az 100 milisaniyede bir kez çalışır. Çalışma süresi ***çok hızlı olmalıdır***.
  - **Örneğin:** StS 10 milisaniyede karar verirse, 100 milisaniyede bir kez çalıştığını düşündüğümüzde 10/110 = ~%9 bir çalışma zamanı *CPU adına boşa geçirilmiş bir zaman olarak düşünülebilir*.
- LtS çok daha yavaş çalışabilir (saniye, dakika).

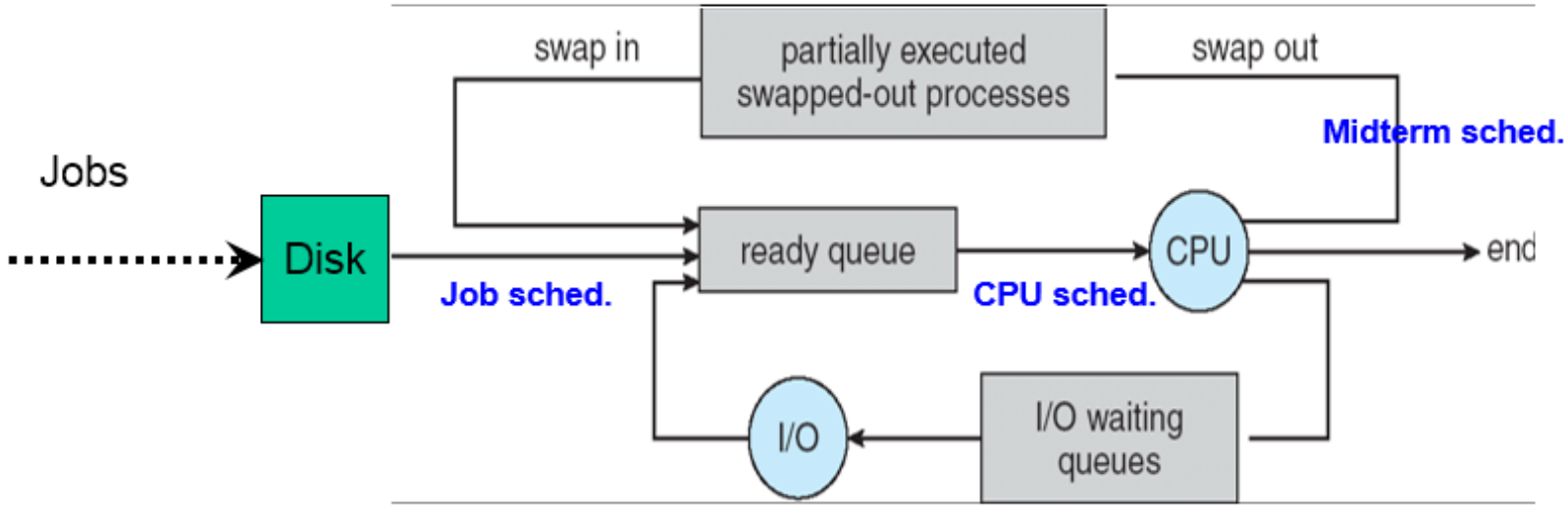
Planlayıcılar (Schedulers) (devam…)

- Prosesleri **I/O-ağırlıklı** ve **CPU-ağırlıklı** olmak üzere ***ikiye ayırmak*** mümkündür.
- I/O-ağırlıklı olanlar, genelde I/O işlemi yapıp, CPU üzerinde hesaplama işlemi gerçekleştirmez.
- CPU-ağırlıklı olanlar ise tam aksine, neredeyse hiç I/O işlemi yapmazken, genelde hesaplama işlemi gerçekleştirir.
- **LtS'in en önemli görevi**, her iki proses türünü dikkate olarak iyi bir *proses karması* (kombinasyonu) seçmesidir (***process mix***),
  - hem I/O hem de ready kuyruğu boş kalmamalıdır.

Planlayıcılar (Schedulers) (devam…)

- Medium-term Scheduler-Mts: Bellek yönetimi sürecinde gerçekleşir.
  - CPU'daki prosesleri alır ve gerektiğinde tekrar CPU'ya koyar (swap).
  - **Detayları** bellek yönetiminde tartışılacaktır.
  - Hızı LtS ve StS arasındadır.

Planlayıcılar (Schedulers) (devam…)



Prosesler Üzerindeki Operasyonlar

- OS prosesler üzerinde:
  - Proses Yaratma
  - Proses Sonladırma
  - Proses İzleme ve Yönetme

işlerini yönetebilecek mekanizmaya sahip olmalıdır.

Prosesler Üzerindeki Operasyonlar (devam…)

- Genelde tüm işletim sistemlerinde prosesler **process identifier** (**pid**) ile tanımlanır ve yönetilirler.
- Prosesler **parent-child** ilişkisi içerisinde birbirlerine bağlıdırlar.

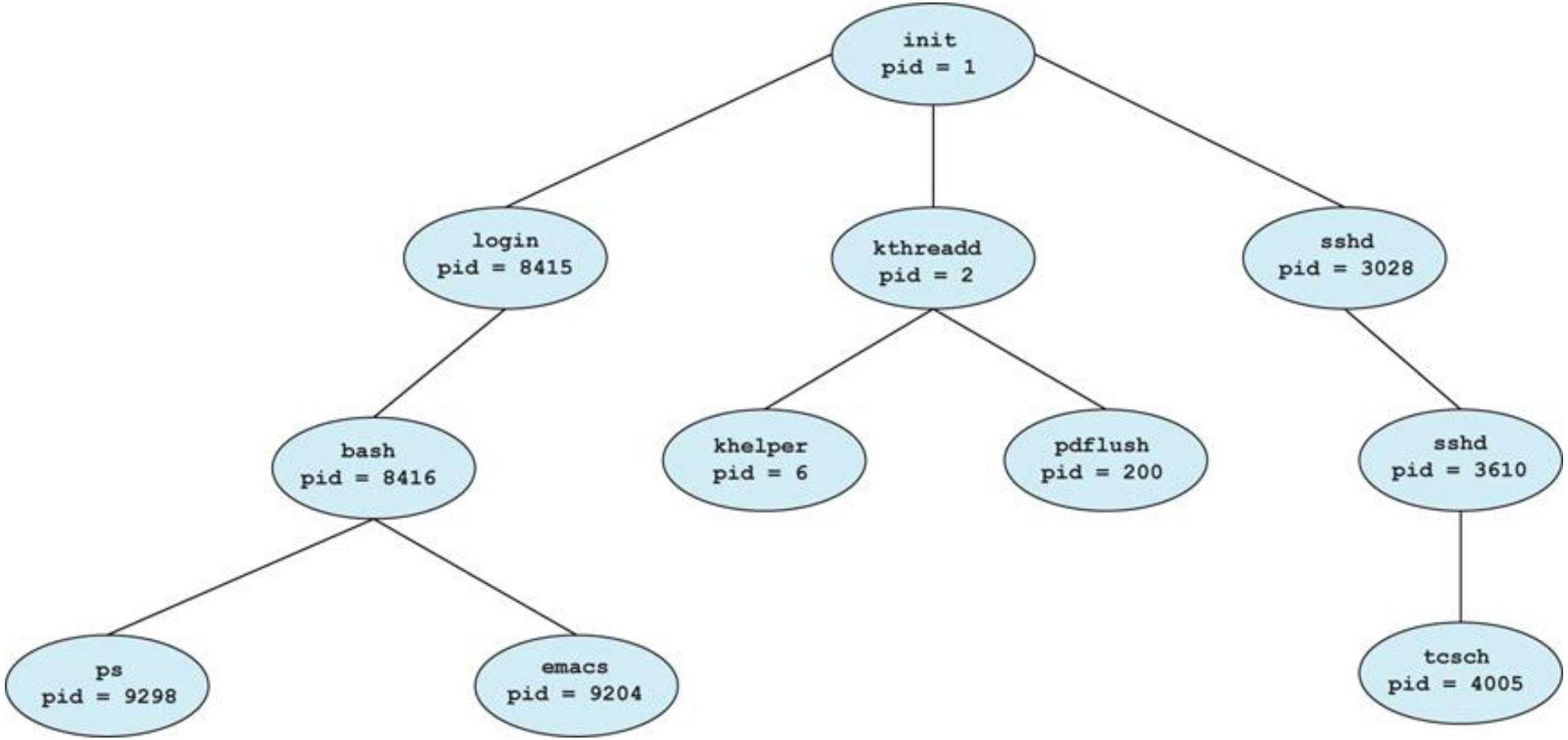
```
long state; /* state of the process */
struct sched_entity se; /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```

Prosesler Üzerindeki Operasyonlar (devam…)

- Tüm UNIX ve Linux OS'larda prosesleri listelemek için ps komutunu kullanabiliriz.
- Örneğin aşağıdaki şekilde bir kullanım sistemdeki aktif olan prosesleri tüm bilgileriyle listelememizi sağlar.

```
ps -el
```

Linux Proses Ağacı



Proses Yaratma (devam…)

- **Parent** proses çalışma süresince **children** prosesler yaratabilir.
- Bir child proses yaratıldığında kendi işlerini yapabilmek için **kaynaklara** (CPU time, bellek, dosyalar, I/O cihazları) ***ihtiyaç duyacaktır***. Bu durumdaki **kaynak paylaşımı** aşağıdaki gibi üç şekilde olabilir:
  - Parent ve children tüm *kaynakları paylaşır*.
  - Children parent'ın kaynaklarının *bir kısmını alabilir*.
  - Parent ve children *kaynak paylaşmaz*. Direk olarak OS tarafından yeni kaynaklar child'a atanır.

Proses Yaratma (devam…)

- Parent ve children yaratıldıktan sonraki **çalışma durumlarını** iki şekilde olabilir:
  - Parent ve children **concurrent** çalışır.
  - Parent, children **terminate edene kadar bekler.**
- **Adres alanı (address space)** durumlarına da baktığımızda aşağıdaki iki olasılık karşımıza çıkmaktadır:
  - Child proses parent prosesin *kopyasıdır* (Aynı program ve veriye sahiptir).
  - Child prosese *yeni bir program* yüklenir.

Proses Yaratma (devam…)

getpid() ve getppid() Sistem Çağrıları

UNIX'de aktif prosesin, proses id'sini (pid) almak için **getpid()** ve aktif prosesin Parent pid'sini almak için de **getppid()** sistem çağrıları kullanılır.

```
…
printf("The process ID is %d\n", (int) getpid());
printf("The parent process ID is %d\n", (int) getppid());
…
```

fork() Sistem Çağrısı

UNIX'de yeni bir proses fork() sistem çağrısı ile yaratılır. Neredeyse yaratıldığı parent prosesin tam bir kopyası oluşur (pid hariç). Parent'ın code, veri, stack, açık dosya tanımlayıcıları ve sinyal tablosunu kopyalar. Farklı bellek alanına ve stack'e sahip olsalar da her iki proses de aynı kod bloğu üzerinde concurrent olarak **çalışmaya devam ederler**.

Proses Yaratma (devam…)

```
pid_t pid;
pid = fork();
if (pid == -1)
{
    printf("Fork gerçekleştirilemedi...\n");
    exit(1);
}
if (pid == 0)
{
        printf("I am child…\n");
}
else
{
        printf("I am parent…\n");
}
```

Proses Yaratma (devam…)

exec() Sistem Çağrısı

Aktif prosesin bellek alanına yeni programı yükleyerek, aslında bir şekilde yeni programın çalıştırılmasını sağlar. fork() ve exec() genelde birlikte çalıştırılırlar. Örneğin child proses bloğunda aşağıdaki kod çalıştırılarak yeni prosesin dosyaları listelemesi sağlanabilir.

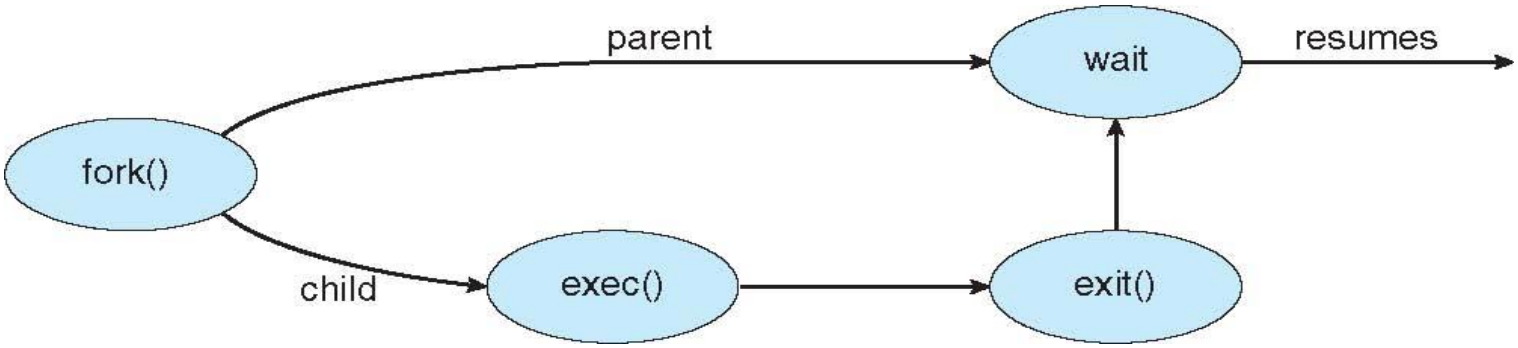
```
execlp("/bin/ls","ls",NULL);
```

Proses Sonlandırma

- Çalışan her proses işi bittiğinde kendi kendine sonlanır. Ancak Parent proses bir child proses yarattığında, child prosesten önce sonlanması istenmiyorsa (**orphan - öksüz** bir proses olmasın), **wait() sistem çağrısı** kullanılarak önce child prosesin sonlanması beklenir.

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

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

Proses Sonlandırma (devam…)



İYİ ÇALIŞMALAR…

Yararlanılan Kaynaklar

- **Ders Kitabı:**
  - Operating System Concepts, Ninth Edition, Abraham Silberschatz, Peter Bear Galvin, Greg Gagne
- **Yardımcı Okumalar:**
  - İşletim Sistemleri, Ali Saatçi
  - Şirin Karadeniz, Ders Notları
  - İbrahim Türkoğlu, Ders Notları