1906003052015

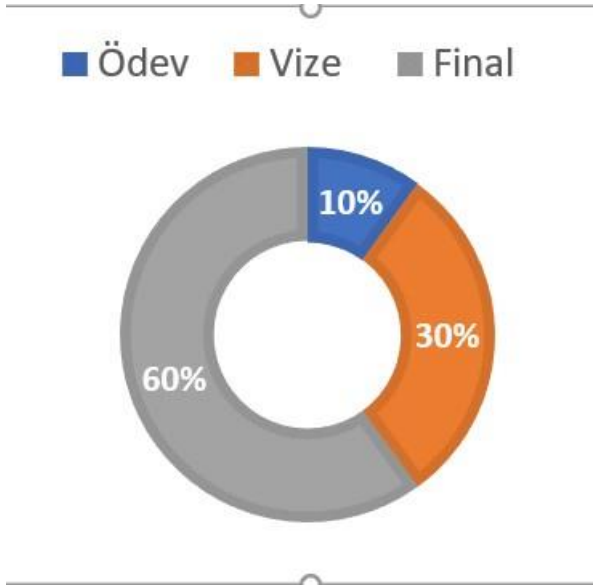
İşletim Sistemleri

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.



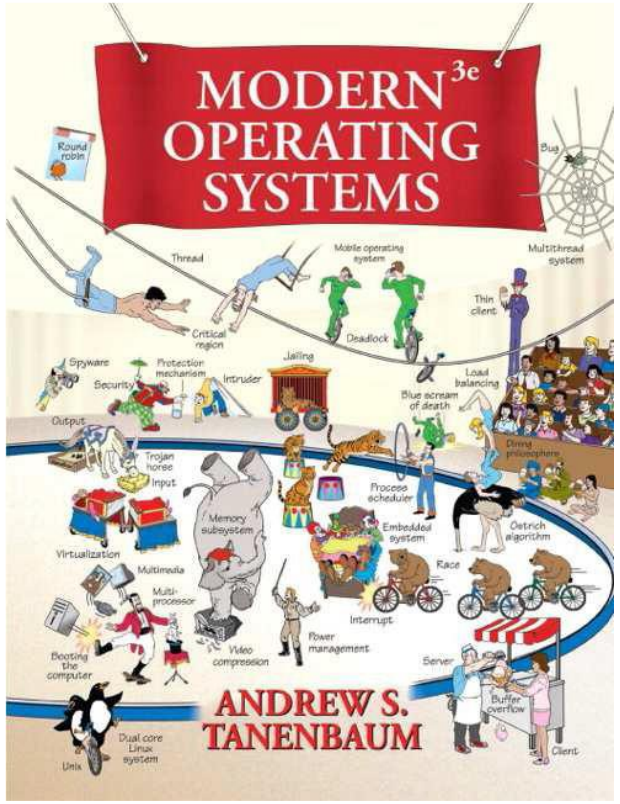
HAFTA	KONULAR
Hafta 1	: İşletim sistemlerine giriş, İşletim sistemi stratejileri
Hafta 2	: Sistem çağrıları
Hafta 3	: Görev, görev yönetimi
Hafta 4	: İplikler
Hafta 5	: İş sıralama algoritmaları
Hafta 6	: Görevler arası iletişim ve senkronizasyon
Hafta 7	: Semaforlar, Monitörler ve uygulamaları
Hafta 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 - 4

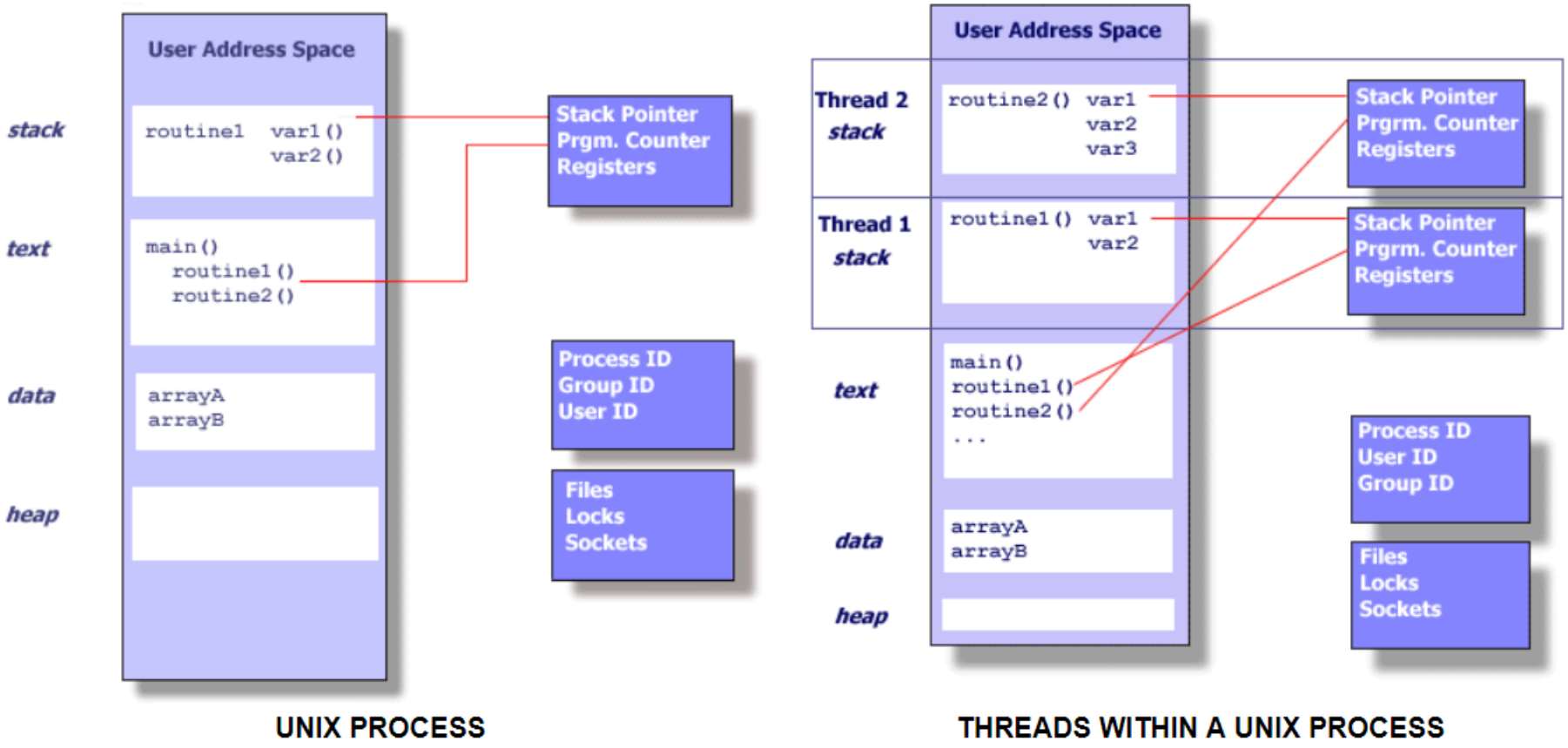
İPLİK (Thread) YÖNETİMİ

IPLIKLER (THREADS)

- Geleneksel (veya ağır) işlem tek bir kontrolüne sahiptir. süreç modeli, iki bağımsız kavrama dayanır: kaynak gruplandırma ve yürütme.
- Bir iplik (aynı zamanda hafif işlem LWP olarak da adlandırılır) CPU kullanımının temel bir birimdir. Aşağıdakilerden oluşur
  - bir iş parçacığı kimliği,
  - bir program sayacı,
  - bir kayıt kümesi,
  - ve bir yığın.
- Bir işlemdeki tüm ipliklerin tam olarak aynı adres alanına sahip olması; aynı global değişkenleri paylaştıkları anlamına gelir.

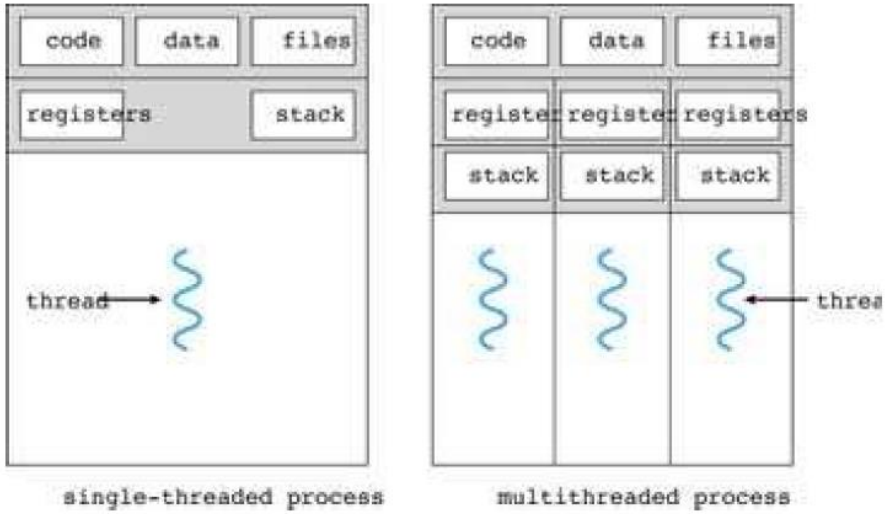
Per process items	**Per thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

IPLIKLER (THREADS)



IPLIKLER (THREADS)

- Kod bölümünde aynı işleme ait veri bölümü ve açık dosyalar ve sinyaller gibi diğer OS kaynakları paylaşır.
- Bir işlemin aynı adres alanında birden çok thread kontolü varsa, sanki yarı-paralel çalışan ayrı işlemlermiş gibidüşünülebilir.
- «Multiithreading», birden çok sürecin çaştığı gibi çalışır. İşlemci, iplikler arasında hızlı bir şekilde ileri geri hareket ederek ipliklerin paralel işlendiği gibi bir durum oluşturur.
- Gerçek te Bir işlemde 3 iplik varsa, CPU nun hızına bağlı olarak CPU zamanı 3 iplik arasında paylaşılır.



IPLIKLER (THREADS)

- Bir thred, bir sürece bağlı olarak çalıştırılsa da, iplik ve süreç farklı kavramlardır ve ayrı olarak ele alınabilir.
  - İş parçacığı bir adres alanını, açık dosyaları ve diğer kaynakları paylaşır
  - İşlemler, fiziksel bellek, diskler, yazıcılar ve diğer kaynakları paylaşır.
- • Her iplik, işlemin adres alanındaki her bellek adresine erişebildiğinden, iplikler arasında herhangi bir koruma yoktur, Çünkü,
  - İmkansızdır.
  - Gereksizdir. İplikler rakip değil ortaktırlar.

Geleneksel bir işlemde olduğu gibi (yani yalnızca bir iş parçacığına sahip bir işlem), bir iş parçacığı birkaç durumdan herhangi birinde olabilir. İplik durumları arasındaki geçişler, işlem durumları arasındaki geçişlerle aynıdır.

YARARLARI

- Çok iş parçacıklı programlamanın yararları dört bölüme ayrılabilir.

Bunlar:

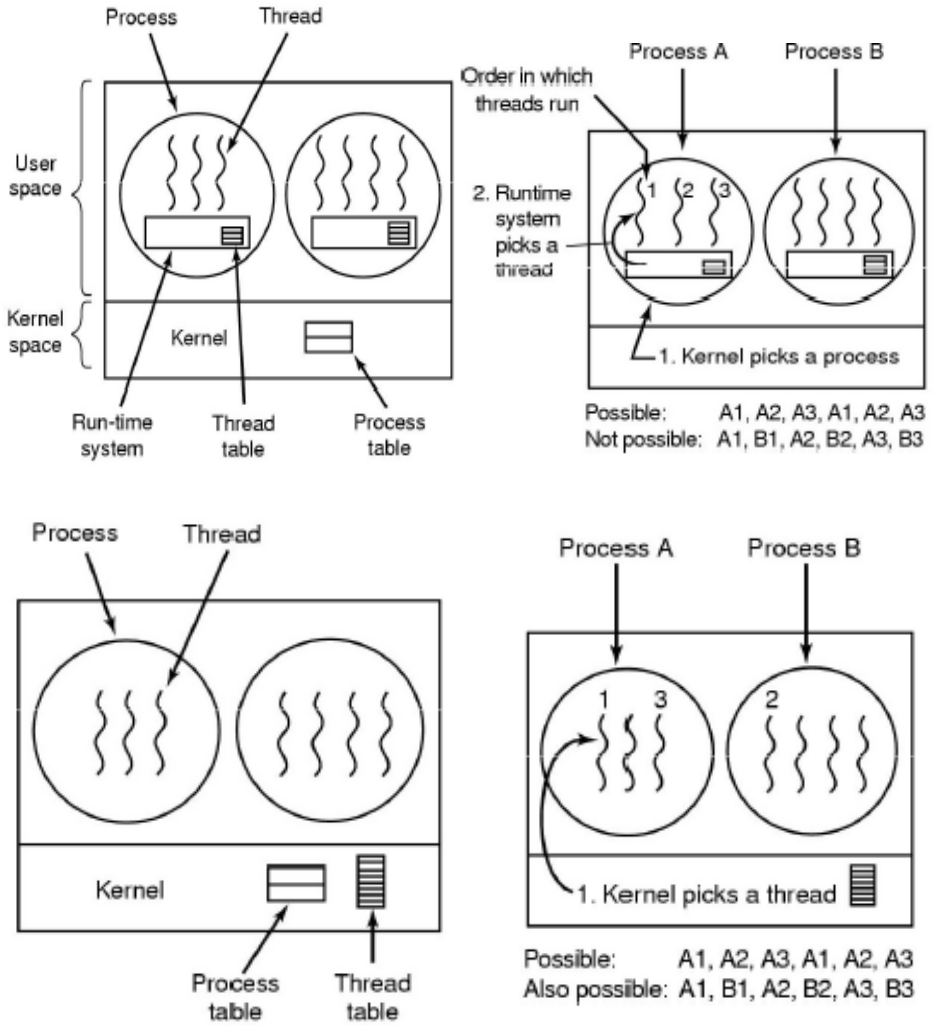
1. Esneklik
2. Kaynak paylaşımı
3. Genel gider ekonomisi
4. Çok işlemcili mimarilerin kullanılabilmesi

MULTITHREADING MODEL

İplik yönetimi genelde iki değişik biçimde gerçeklesebilir:

Kullanıcı Evreninde: İsletim sistemi ipliklerin varlığından habersizdir. Herbir görev kendine ayrılan zaman dilimi içinde, kendine ait iplikler arasında anahtarlama yönetimini yapar. İsletim sistemi durumdan haberdar edilmez.
Örnek: POSIX P-Threads, Mach C-Threads

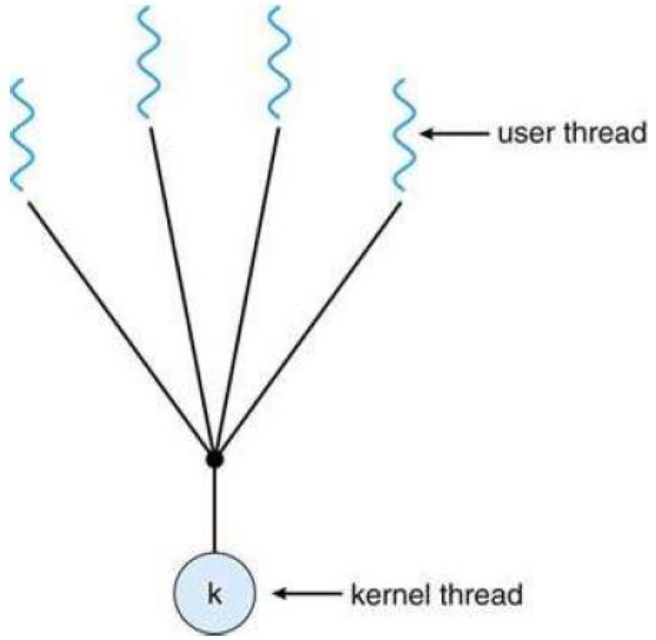
Çekirdek Evreninde: İsletim sistemi, görevlerin yanı sıra görevler altında yer alan ipliklerin yönetimini de üstlenir. İsletim sistemi, herbir ipliğin anahtarlama yönetimini yapar.
Örnek: Windows NT



MULTITHREADING MODEL

Kullanıcı evrenindeki iplikler ile çekirdek evrenindeki iplikler arasında ilişkiyi belirleyen 3 model vardır.

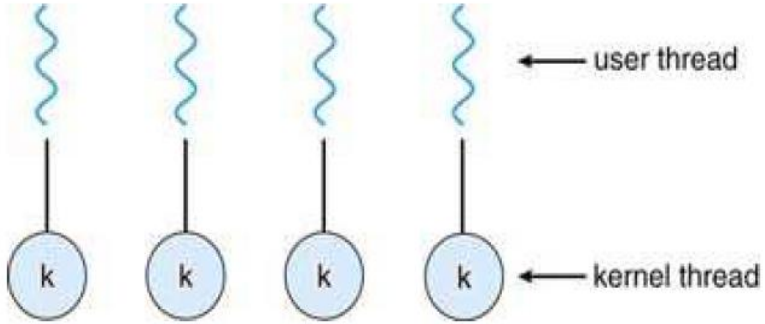
1. **Çoktan bire model:**
   + İş parçacığı yönetimi kullanıcı alanındaki iş parçacığı kitaplığı tarafından yapılır, bu nedenle verimli olur;
   - Ancak bir iş parçacığı bir engelleme sistemi çağrısı yaparsa tüm işlem engellenecektir.
   - Aynı anda yalnızca bir iş parçacığı, çekirdeğe erişebildiğinden, çoklu iş parçacıkları, çok işlemcili sistemlerde paralel olarak çalıştıramaz.
2. Birden Bire model:
3. Çoktan Çoğa model:



MULTITHREADING MODEL

2. Birden Bire model:

- Bir iş parçacığı bir engelleme sistemi çağrısı yaparken başka bir iş parçacığının çalışmasına izin vererek çoktan-bire modele göre daha fazla eşzamanlılık sağlar;
- Ayrıca, çoklu işlemciler üzerinde birden fazla iş parçasının paralel çalışmasına izin verir.
- Bu modelin tek dezavantajı, bir kullanıcı iş parçacığı oluşturmak için ilgili çekirdek iş parçacığı oluşturulmasını gerektirmesi.
- Çekirdek iş parçacığı oluşturma yükü bir uygulamanın performansını yükleyebildiğinden, bu modelin çoğu uygulaması, sistem tarafından desteklenen iş parçacığı sayısını sınırlar.
- Linux, Windows İşletim Sistemi ailesi ile birlikte bire bir modeli uygular.

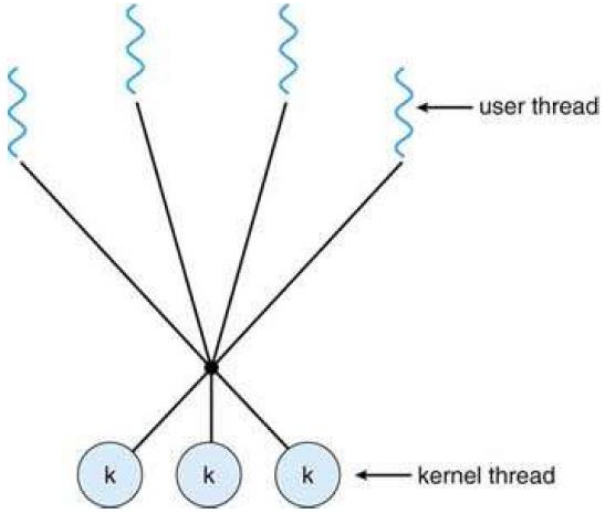


Çoktan Çoğa model:

MULTITHREADING MODEL

2. Çoktan Çoğa model:

- Birçok kullanıcı düzeyindeki iş parçacığını daha küçük ya da eşit sayıda çekirdek iş parçacığı ile çoğaltılır.
- Çekirdek iş parçacığı sayısı belirli bir uygulamaya veya belir bir makineye özeldir.
- Çoktan bire model, geliştiricinin istediği sayıda kullanıcı i parçacığı oluşturmasına izin verirken, gerçek eşzamanlılı kazanılmaz, çünkü çekirdek bir seferde yalnızca bir i parçacığı zamanlayabilir.
- Birden bire model daha fazla eşzamanlılık sağlar, anca geliştirici bir uygulamada çok fazla iş parçacığı yaratmaya dikkat etmelidir
- Çok-çok model bu eksikliklerin hiçbirine sahip değildir:
  - Geliştiriciler, gerektiği kadar çok kullanıcı iş parçacığ oluşturabilir ve karşılık gelen çekirdek iş parçacığı çok işlemcili olarak paralel çalışabilir.
  - Ayrıca, bir iş parçacığı bir engelleme sistemi çağrısı gerçekleştirdiğinde, çekirdek yürütülecek başka bir iş parçacığı zamanlayabilir.



THREAD KÜTÜPHANELERİ

Bir iş parçacığı kitaplığı, programlayıcıya iş parçacığı oluşturma ve yönetme için bir API sağlar. Bir iş parçacığı kitaplığını uygulamak için iki temel yol vardır.

1. Birinci yaklaşım, tamamen çekirdek desteği olmayan bir kütüphane sağlamaktır. Kütüphane için tüm kod ve veri yapıları kullanıcı alanında bulunur. Bu, kitaplıktaki bir işlevi çağırmak kullanıcı alanındaki bir yerel işlev çağrısı anlamına gelir ve bir sistem çağrısı demektir.

2. İkinci yaklaşım doğrudan OS tarafından desteklenen bir çekirdek düzeyinde kütüphane uygulamaktır. Bu durumda, kütüphane için kod ve veri yapıları çekirdek alanında bulunur. Kitaplık için API içindeki bir işlevi çağırmak, genellikle çekirdeğe bir sistem çağrısı ile sonuçlanır.

THREAD KÜTÜPHANELERİ

Bugün üç ana iş parçacığı kitaplığı kullanılmaktadır:

1. **POSIX Pthreads.** POSIX standardının iş parçacığı uzantısı olan Pthreads, kullanıcı veya çekirdek düzeyinde bir kitaplık olarak sağlanabilir.
2. **Win32**. Win32 iş parçacığı kitaplığı, Windows sistemlerinde kullanılabilen bir çekirdek düzeyinde kitaplıktır.
3. **Java**. Java iş parçacığı API, doğrudan Java programlarında iş parçacığı oluşturma ve yönetme imkanı verir. Bununla birlikte, çoğu durumda JVM, bir ana bilgisayar işletim sisteminin üstünde çalıştığı için, Java iş parçacığı API'si, genellikle, ana makine sisteminde bulunan bir iş parçacığı kitaplığı kullanılarak uygulanır.

PTHREAD KÜTÜPHANESİ (POSIX)

- Pthreads, iş parçacığı oluşturma ve senkronizasyon için bir API tanımlayan POSIX standardını (IEEE 1003.1c) belirtir.
- Bu, bir uygulama değil, iş parçacığı davranışı için bir belirtmedir. İşletim sistemi tasarımcıları şartnameyi istedikleri şekilde uygulayabilir.
- Pek çok sistem, Solaris, Linux, Mac OS X ve Tru64 UNIX de dahil olmak üzere Pthreads özelliklerini uygular. Shareware uygulamaları, çeşitli Windows işletim sistemleri için de kamu malıdır.

Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread's attribute structure
Pthread_attr_destroy	Remove a thread's attribute structure

PTHREAD KÜTÜPHANESİ (POSIX)

Compiler / Platform	Compiler Command	Description
INTEL **Linux**	`icc -pthread`	C
	`icpc -pthread`	C++
PGI **Linux**	`pgcc -lpthread`	C
	`pgCC -lpthread`	C++
GNU **Linux, Blue Gene**	`gcc -pthread`	GNU C
	`g++ -pthread`	GNU C++
IBM **Blue Gene**	`bgxlc_r / bgcc_r`	C (ANSI / non-ANSI)
	`bgxlC_r, bgxlc++_r`	C++

DERS - 4

1. Giriş

Prosesler arası haberleşme ihtiyacı çeşitli nedenlerle ortaya çıkabilir. Bazen bir proses diğer prosesin ürettiği sonuca ihtiyaç duyar, bazen ortaklaşa çalıştıkları bir problemin çözümünde biri birlerini beklemeleri gerekebilir. Prosesler arası iletişim farklı şekillerde gerçekleşebilir:

1. Haberleşecek prosesler aynı ya da bilgisayar ağı ile bağlı farklı makinalarda olabilir
2. Prosesler arasında bağlantılı (veri aktarımlı) ya da bağlantısız (mesaj aktarımlı) haberleşme kurulabilir
3. Randevu yöntemi: Haberleşecek prosesler haberleşmeyi nasıl başlatacaklar
    a. Dosya sisteminde yaratılan bir nesne kullanılabilir
    b. İnternet adresi kullanılabilir

1. Giriş

Prosesler Arası İletişim	Karakteristik özelliği	Makine uzaklığı	Randevu
exit()	Tamsayı döndürür	Aynı makina	Çocuk prosesten ebeveyn prosese doğru
signal()	Sinyal numarası	Aynı makina	Sinyal numarası
mmap	Sanal belleğe Bellek bölgesi	Aynı makina	Sanal adres
Pipe	Kuyruk, ilk- giren ilk-çıkar	Aynı makina	Dosya tanımlayıcısı
Mesaj kuyruğu	Mesaj	Aynı makina	IPC ID
Paylaşılan bellek	mmap gibi	Aynı makina	IPC ID
Semafor	Senkronizasyon	Aynı makina	IPC ID
Soket	Mesaj	Aynı ya da farklı makina	Dosya tanımlayıcısı

2. Prosesler Arası Haberleşme Mekanizmaları

Unix işletim sistemi prosesler arası etkileşim için üç temel yapı sunar:

1. **Mesaj aktarımı**
2. **Paylaşılan bellek**
3. **Semafor**

Unix işletim sisteminde, çekirdek tüm kaynaklar için tekil bir anahtar kullanır. Bu üç proses arası iletişim yapısı için kullanılan kaynaklar için tekil bir anahtar üretilmesi gerekir. Bunun için ftok() fonksiyonu kullanılır.

ftok () - Dosya Adı'dan IPC Anahtarı Yaratır.
(https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_73/apis/p0zftok.htm)
Tanımlayıcı tabanlı işlemler arası iletişim yöntemleri, süreçler arası iletişim tanımlayıcıları elde etmek için **msgget (), semget (), shmget ()** fonksiyonlarına bir anahtar sağlamanızı gerektirir. ftok () fonksiyonu, bu anahtarı üretmek için kullanılan bir mekanizmadır.

Geri Dönüş Değerleri:

value	**ftok()** başarılı.
(key_t)-1	**ftok()** başarılı değil. Errno değişkeni, hatayı belirtmek için tanımlanır.

ftok1.c

Mesaj Kuyrukları

Mesaj kuyruk sistemi ile çalışmak için aşağıdaki fonksiyonları kullanabilirsiniz:

- `msgget()`
  Mesaj kuyruğuna erişmek için kullanılır
- `msgsnd()`
  Mesaj göndermek için kullanılır
- `msgrcv()`
  Mesaj almak için kullanılır
- `msgctl()`
  Mesaj kuyruğunu yönetmek için kullanılır

```
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

`key`: mesaj kuyruğunu tanımlayan tekil anahtar

msgflg	Anlamı
0	Mesaj kuyruğunu kimliğini döner
IPC_CREAT | 0640	Mesaj kuyruğu yoksa yaratır ve kimliğini döner
IPC_CREAT | IPC_EXCL | 0640	Mesaj kuyruğu yoksa yaratır ve kimliğini döner, varsa hata döner

Mesaj Kuyrukları

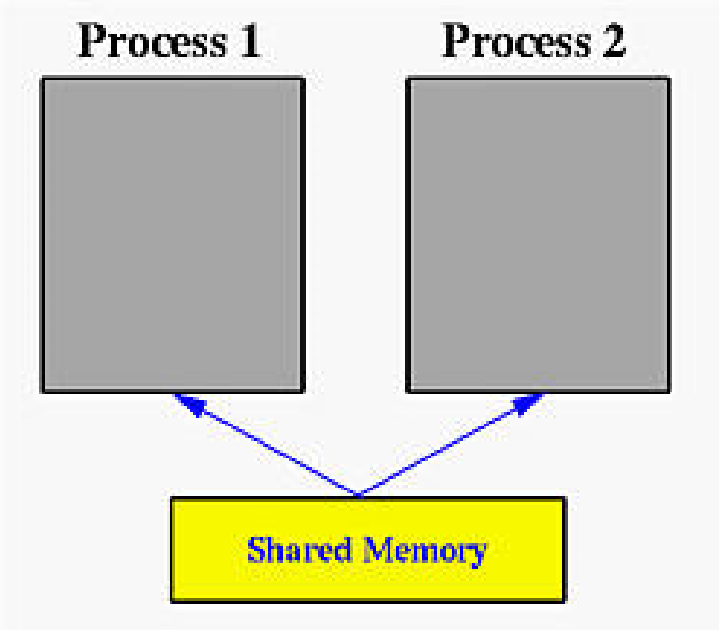
#include <sys/msg.h>
int **msgsnd**(int msqid, const void *msgp, size_t msgsz, int msgflg);
int **msgrcv**(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);

Burada **msgp** mesajı içeren **msgbuf** tipinden bir bellek gözünün adresidir:
struct **msgbuf** {
long mtype; char mtext[1];
}
Mesaj herhangi bir yapıda olabilir. **mtext[1]** burada sadece verinin başlangıcını işaret etmek için kullanılmaktadır.
msgsz, **msgp** işaretçisinin gösterdiği mesajın boyutunu tanımlar.
msgflg bayrağının değeri **0** ise **msgsnd** çağrısı için mesaj kuyruğu doluyken, **msgrcv** çağrısı için mesaj kuyruğu boşken çağrıyı yapan prosesin bloke olmasına neden olur. Bu bayrağın değeri IPC_NOWAIT ise kesintisiz çalışır. **msgsnd** çağrısı için mesaj kuyruğu doluyken, **msgrcv** çağrısı için mesaj kuyruğu boşken çağırılırsa hata kodu ile döner. Hata kodunun, **errno**, değeri **msgsnd** çağrısı için **EAGAIN**, **msgrcv** çağrısı için **ENOMSG**'dir.

msg_queue.c

Paylaşılan Bellek

Paylaşılan bellek, prosesin bellek uzayının bir kısmını başka bir proses ile paylaşmasıdır (Şekil-1). Paylaşılan bellek alanı, Proses A'nın ve Proses B'nin bellek adres uzaylarında farklı bölgelerine karşı düşer. Paylaşıla bellek sistemi ile ilgili sistem çağrıları aşağıda listelenmiştir. Paylaşılan bellek alanı ayırmak, bağlamak ve geri vermek işlemlerini bu sistem çağrılarını kullanarak gerçekleştiriyoruz.



Paylaşılan Bellek

Şimdi sırasıyla bu fonksiyonların parametrelerine ve kullanımına bakalım.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
```

key: paylaşılan belleği tanımlayan tekil anahtar

size: paylaşılan bellek bölümünün boyutunu tanımlar. Var olan bir paylaşılan bellek alanı içi bu değer göz ardı edilir.

shmflg	Anlamı
0	Paylaşılan belleğin kimliğini döner
IPC_CREAT | 0640	Paylaşılan bellek yoksa yaratır ve kimliğini döner
IPC_CREAT | IPC_EXCL | 0640	Paylaşılan bellek yoksa yaratır ve kimliğini döner, varsa hata döner

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);
int shmdt(void *shmaddr);
```

shmflg	Anlamı
0	Paylaşılan bellek alanı hem okumak hem de yazmak için kullanılabilir
SHM_RDONLY	Paylaşılan bellek alanı salt okunur olarak kullanılabilir

Görev Denetimi

Planlama Algoritmaları

1. FCFS (First Come First Served – İlk Gelen Önce) Algoritması:

Bu algoritmaya göre; AIB yi ilk talep eden görev, ilk olarak işlemciyi kullanır. FIFO kuyruğu ile çalıştırılabilir. Görev hazır görevler kuyruğuna sunulduktan sonra, onun görev denetim bloğu (PCB) kuyruğun sonuna ilave edilir. AIB boş olduğu zaman kuyruğun başındaki görev çalışması için AIB ye sunulur ve kuyruktan silinir. Bu algoritmada görevlerin bekleme süresi yüksek olur.

Örnek: P1, P2, P3 görevlerinin sırasıyla kuyrukta yerleştiklerini kabul edelim:

Görev	Çalışma Zamanı (sn)
P1	24
P2	3
P3	3

Görev Denetimi

Toplu Sistemlerde Zamanlama

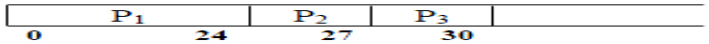
Görev Denetimi

1. FCFS (First Come First Served – İlk Gelen Önce) Algoritması:

Örnek: P1, P2, P3 görevlerinin sırasıyla kuyrukta yerleştiklerini kabul edelim:

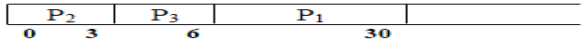
Görev	**Çalışma Zamanı (sn)**
P1	24
P2	3
P3	3

1.Görevler P1, P2, P3 ardışıklığı ile sunulmuş olduğunu varsayalım. Buna göre planlama:



Ortalama bekleme süresi : (24+27+0) / 3 = 17 msn 'dir.

2.Eğer görevlerin gelme P2, P3, P1 şeklinde sıralanırsa, planlama:



Ortalama bekleme süresi : (3+6+0) / 3 = 3 msn'dir.

Görev Denetimi

2.SJF (Shortest Job First – En Kısa İşletim Süresi Olan Önce) Algoritması:

Bu algoritmada CPU boş olduğunda, kalan görevler içinde çalışma süresi en küçük olan görev, çalışması için işlemciye sunulur. Eğer iki görevin kalan süreleri aynı ise o zaman FCFS algoritması uygulanır. Bu algoritmada: Her görev, o görevin bir sonraki MİB işlem zamanı ile değerlendirilir. Bu, en kısa zamanlı işin bulunması için kullanılır.

SJF Türleri:

1.Kesilmesiz SJF : Eğer AİB bir göreve tahsis edilmişse, AİB işlem zamanı bitmeyince görev kesilemez.

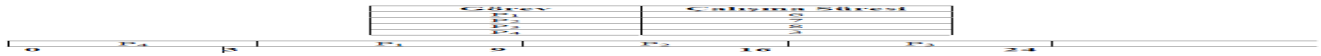
2.Kesilmeli SJF : Eğer AİB işlem zamanı, şu anda çalışan görevin kalan işlem zamanından küçük olan yeni bir görev sisteme sunulmuşsa, eski görev kesilecek. Bu yönteme, SRTF(Shortest Remaining Time First – En kısa işlem zamanı kalan, birinci) yöntem denir.

SJF verilmiş görevler kümesi için en küçük ortalama bekleme zamanı oluşması için optimizasyon yapar.

Görev Denetimi

2.SJF (Shortest Job First – En Kısa İşletim Süresi Olan Önce) Algoritması:

Örnek: : P1, P2, P3, P4 görevleri aşağıdaki ardışıklık ile sunulmuş olduğunu varsayalım. Buna göre kesilmesiz SJF yöntemine göre ortalama bekleme süresini bulalım:



- **SJF :** tob = tP1 + tP2 + tP3 + tP4 = (3+9+16+0) / 4 = 7 msn
- **FCFS :** tob = tP1 + tP2 + tP3 + tP4 = (0+6+13+21) / 4 =10.75 msn

Görev Denetimi

3.Çok Kuyruklu Planlama Algoritması:

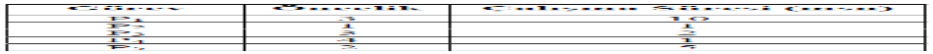
Bu algoritmaya göre görevler belli sınıflara ayrılır ve her sınıf görev kendi kuyruğunu oluşturur. Başka deyişle hazır görevler çok seviyeli kuyruğa dönüştürülür. Görevin türüne önceliğine, bellek durumuna veya başka özelliklerine göre görevler belli kuyruğa yerleştirilirler. Her kuyruk için planlama algoritması farklı olabilir. Bununla birlikte görevlerin bir kuyruktan diğerine aktarılmasını sağlayan algoritmada oluşturulur.

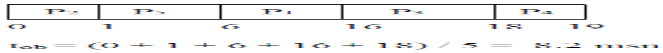
Bu algoritmaya göre yüksek öncelikli kuyruktaki görevler önce işlenir. Eğer bu kaynak boş ise ondan aşağı seviyedeki görevler çalıştırılabilir.

Görev Denetimi

4.Öncelikli Planlama Algoritması:

Bu algoritmaya göre her bir göreve öncelik değeri atanır ve görevler öncelik sırasına göre işlemciyi kullanırlar. Aynı öncelikli görevler FCFS algoritması ile çalıştırılırlar.





Görev Denetimi

5.Döngülü Planlama (Round Robin - RR) Algoritması:

Her görev küçük bir AİB zaman dilimini alır. Bu zaman bittiğinde, görev kesilir ve hazır görevler kuyruğunun sonuna eklenir.

- **Örnek:** P1, P2, P3, P4 görevleri aşağıdaki ardışıklık ile sunulmuş olduğunu varsayalım. Zaman dilimi 20 msn ise buna göre:

Görev	İşlem Zamanı (msn)
P1	53
P2	17
P3	68
P4	24