

İşletim Sistemlerine Giriş

Süreçler ve İş Parçacıkları(Thread)

Süreç

-Tüm modern bilgisayarlarda bir çok iş aynı anda yapılabilir.

- *kullanıcı programları çalışır
- *disk okunabilir.
- *yazıcıya ya da ekrana çıktı verilebilir.
- *cd ye veri yazılabilir.
- *
...

Çoklu programlamalı sistemlerde, işlemci bir programdan diğerine çok küçük sürelerde geçerek tüm programların aynı anda çalışıyormuş izlenimini verilir.

Süreç

Bilgisayarda bulunan tüm çalışabilir programlar süreçler şeklinde organize edilir.

Süreç yürütme anında(execution) bulunan bir programdır.

Her sürecin kendine özel değişkenleri vardır.

- *Program sayacı

- *Yazmaçları

- *Program durum kelimesi(PSW)

- *Yığın Göstercisi

- *

Süreç Kavramı

Bir bilgisayar mühendisi düşünelim. Mühendis kek yapıyor olsun.

Mühendisin elinde bir kek tarifi ve malzeme(yumurta, un, şeker,...) ile dolu bir mutfağı bulunsun.

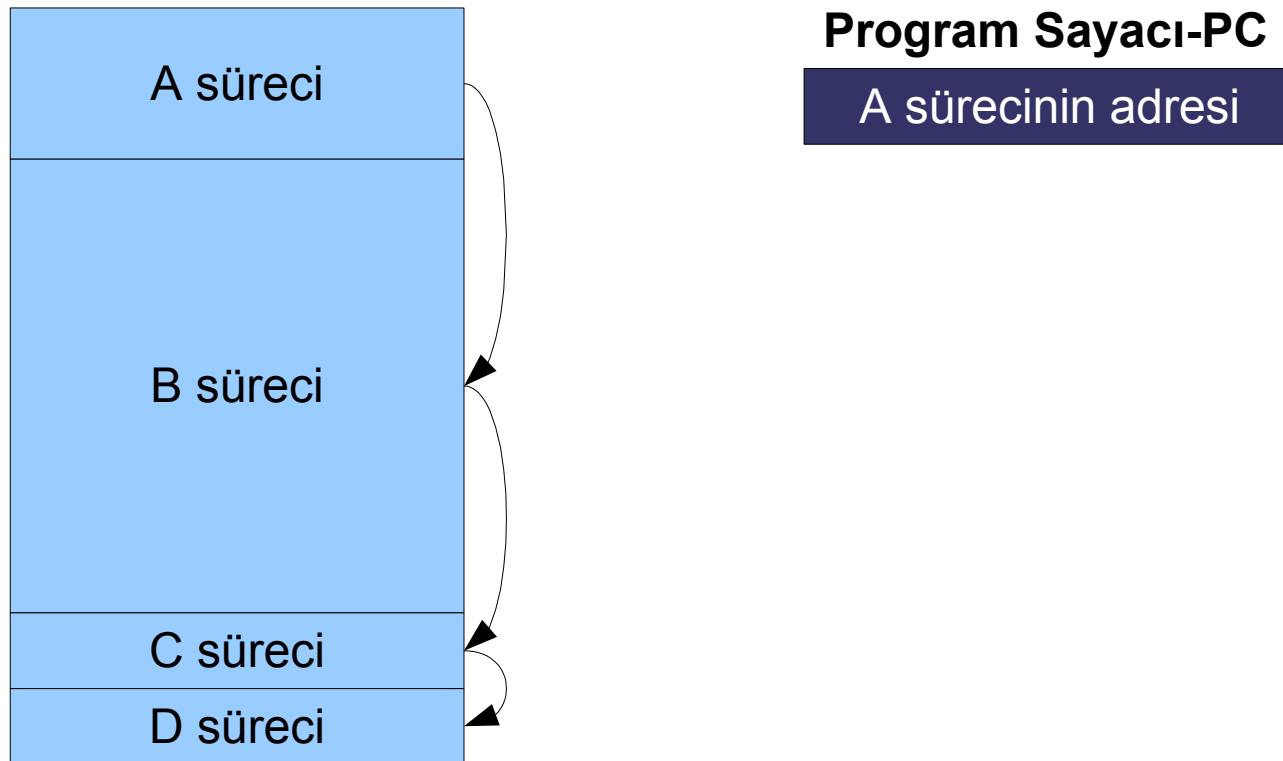
Bu anlatımdaki kek tarifi programdır.

Bilgisayar Mühendisi İşlemcidir.

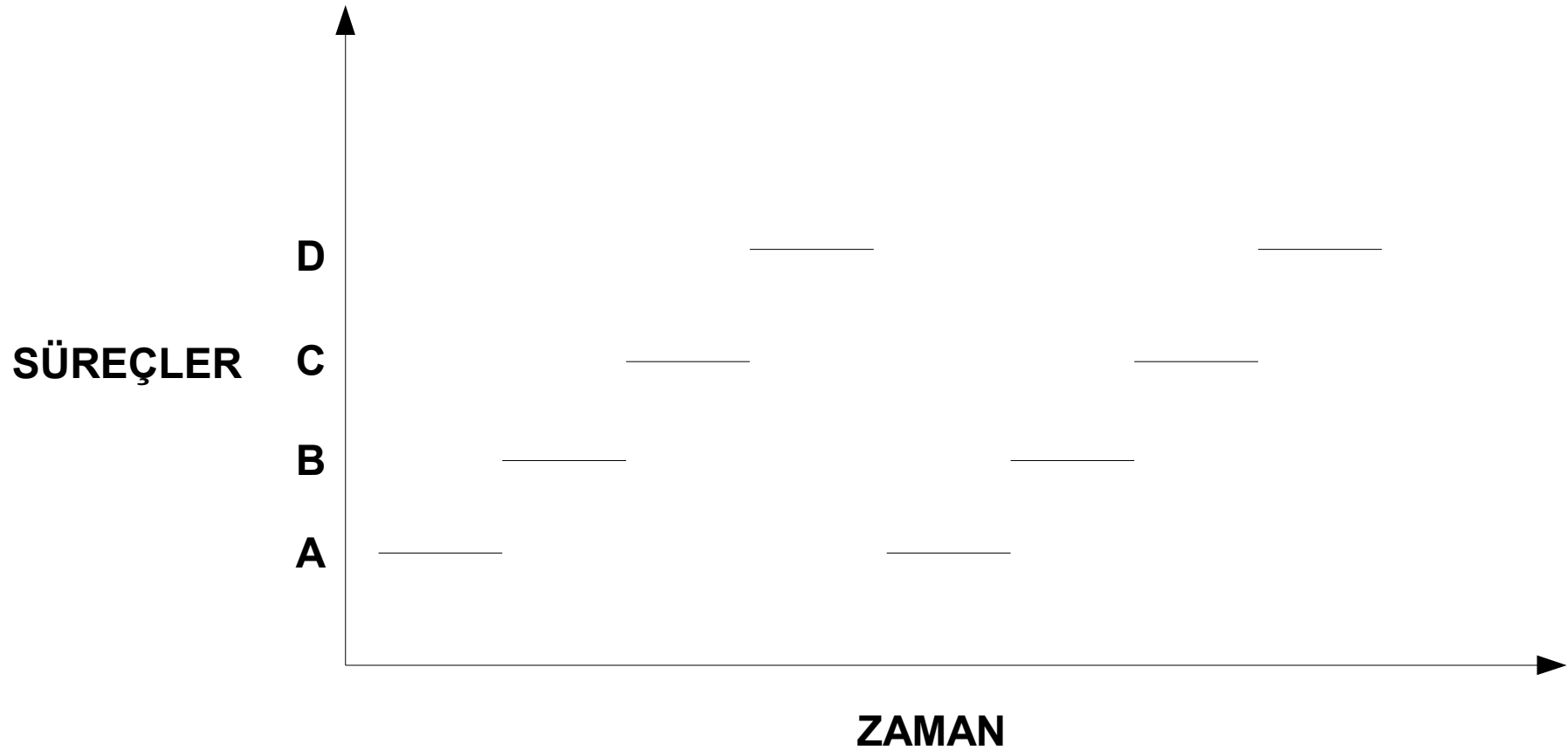
Kek malzemesi ise programın girdi verisidir.

Süreç; mühendisin tarifi okuması, malzemeleri alıp kullanması ve keki yapmasıdır.

Süreçler arası geçiş



Süreçler arası geçiş



Süreç

İşletim sistemi tüm süreçlerin bilgilerini, sistemde bulunan bir tabloda(dizi ya da bağlı liste) tutar.

Bu tabloya süreç tablosu(process table) denilir.

Bu tabloda her süreç için bir girdi bulunur. Bu girdi içerisinde süreç ile ilgili olan tüm bilgiler tutulur.

*program sayacı, yığın işaretçisi, bellek paylaşım bilgisi, açık dosyalarının durumları, zamanlama(scheduling) bilgileri, ...

Süreç Tablosu



Süreç Tablosu

A süreci
B süreci
C süreci
.
.
.
n. süreç



Yazmaçlar(registers)
Program sayacı
Program durum kelimesi
Yığın işaretçisi
Süreç durumu
Öncelik(priority)
Zamanlama parametreleri
Süreç numarası
Ana süreç
Süreç grubu
Sinyaller
Başlama zamanı
Kullanılan işlemci zamanı

...

Süreç Durumları

Her sürecin kendi iç durumu ve program sayacı(program counter) vardır.

Her süreç farklı birer varlık olmasına rağmen birbirleri ile etkileşimde bulunabilir ve birbirleri ile haberleşebilirler.

Bir sürecin çıktısı başka bir sürecin girdisi olabilir. Bu tip durumlarda örneğin, girdiyi alan süreç çıktıdan daha hızlı çalışırsa girdinin olmadığı durumlarda bekler(bloklanır).

İkinci durum ise işlemcinin çalışan süreci durdurarak başka bir süreci çalıştırmasıdır.

Süreç Durumları

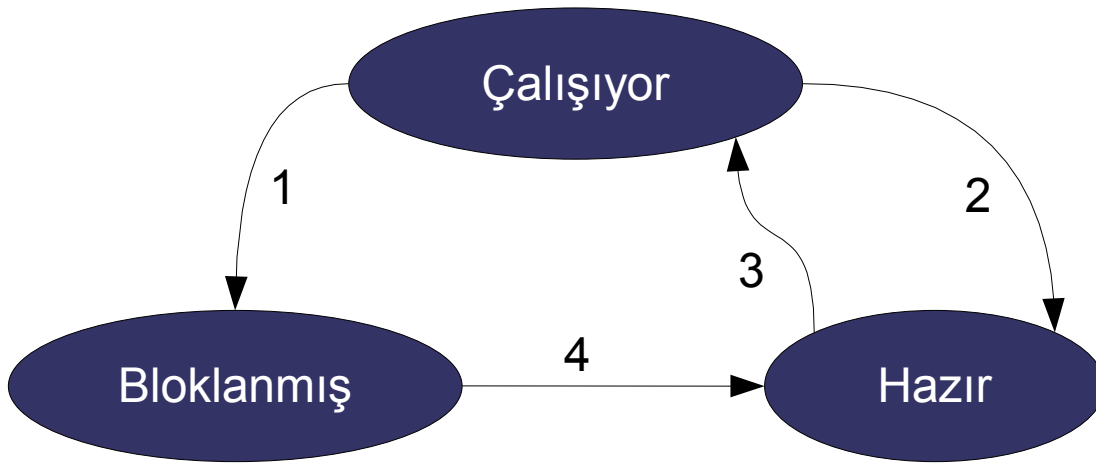
Süreçler;

1. Çalışıyor(Running): Şu anda işlemciyi kullanıyor

2. Hazır(Ready) : Çalışabilir durumdadır, başka bir sürecin çalışması için geçici olarak durmuştur.

3.Bloklanmış(Blocked): Harici bir olay gerçekleşmeden çalışmaz.

Süreç Durumları



1. Süreç girdi beklediği için bloklanır.
2. Zamanlayıcı(Scheduler) başka bir süreç seçer.
3. Zamanlayıcı bu süreci çalışması için seçer.
4. Girdi kullanılabilir olur.

İşlemci ve I/O kesmeleri

ZAMANLAYICI(Scheduler), o anda çalışan süreçlerden hangisinin durdurulup, hangisinin çalıştırılacağına karar veren ve kesmeleri(interrupt) yöneten yazılımdır.

Bilgisayarda işlemci ve I/O aygıtlarının çoklu programlama ortamındaki birlikte çalışmaları şu şekilde gerçekleşir;

Bilgisayarın belleğinde her I/O aygıt sınıfı ile ilişkili olan bir tablo bulunmaktadır. Bu tabloya kesme vektörü(interrupt vector) adı verilmektedir. Bu vektör kesme işlemciye geldiğinde bu kesmeye karşılık gelen kesme servis prosedürünün çalıştırılması için gerekli adres bilgisini tutar. Her kesmeye karşılık bir prosedür vardır ve adresi bu tabloda tutulur.

İşlemci ve I/O kesmeleri

Örneğin, 3 numaralı kullanıcı süreci çalışırken bir disk kesmesi(disk interrupt) gelsin. Bu durumda çalışan program ile ilgili tüm bilgi(program sayacı, yığın işaretçisi,PSW,yazmaçlar) yığına saklanır. Kesme numarasına göre kesme vektöründeki ilgili prosedüre gidilir. Bu prosedürün çalışması sonunda disk kesmesine yol açan süreç numarası bulunur ve bu süreç bloklanmış durumundan hazır duruma geçirilir.

Zamanlayıcı(scheduler) çalıştırılır. Zamanlayıcı hazır durumdaki süreçlerden birisini kendi algoritmasına göre seçerek çalıştırır.

İş Parçacığı(Thread) Modeli

Genelektel işletim sistemlerinde, her sürecin belirli bir adres uzayı vardır. Bu adres uzayında yapması gerekli olan işlemleri kendisi işlemcide çalışırken yapar. İşlemci aynı anda sadece tek bir komut(instruction) çalıştırabilir. İşlemcide çalışacak olan süreç seçildikten sonra işlemcide belirli bir süreye kadar çalışır.

Bazen aynı adres uzayında başka bir ifadeyle aynı süreç içerisinde aynı programın farklı bölümleri (iş parçacığı-thread) yarı paralel olarak çalıştırılabilir. Bu şekilde aynı süreç içerisinde birden fazla süreç parçacıkları paralel olarak işlem yaparlar. Bu modele iş parçacığı modeli denilir.

İş Parçacığı(Thread) Modeli

Bir süreç iki farklı bölümden oluşmaktadır:

- * Kaynakları(açık dosyalar, çocuk süreçler,program texti, program verileri,süreçe özel olan diğer veriler...)
- * Çalışan kod kısmı, yani şu anda çalışan kod ve bir sonraki çalışacak kod bölümü.(program sayacı, yazmaçlar, yığın(stack), çağrılmış fakat geri dönmemiş prosedürler).

İş Parçacığı(Thread) Modeli

İş Parçacığı(thread) kavramının bize getirdiği, aynı süreç ortamında çoklu çalıştırma(execution-yürütme) işleminin yerine getirilmesidir.

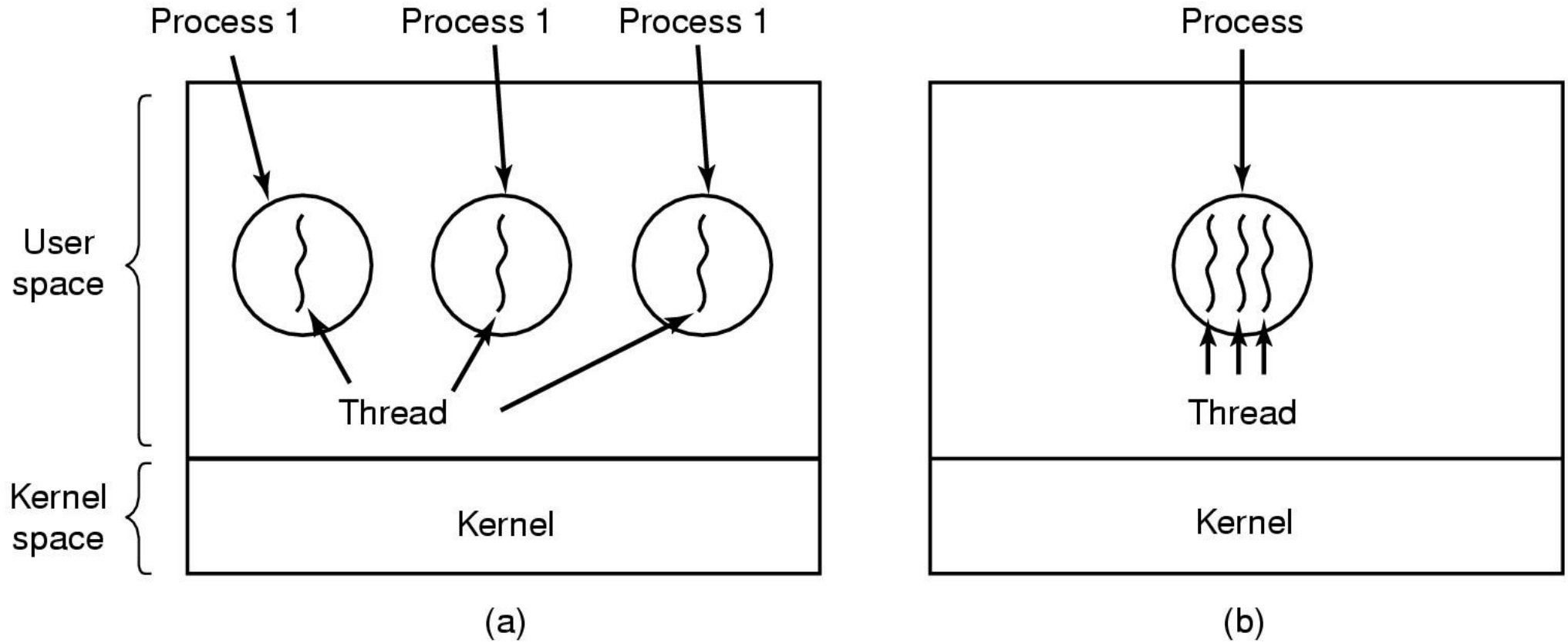
Tek bir süreç içerisinde çoklu iş parçacıklarının(multi thread) çalışması, çoklu programlamada birden fazla sürecin aynı anda (işlemcide sırayla) çalışmasına eşdeğerdir.

İş Parçacığı(Thread) Modeli

Bir süreç içerisindeki tüm threadler, aynı adres uzayını, açık dosyaları, genel değişkenler gibi kaynakları paylaşırlar.

Süreçler ise fiziksel belleği, diskleri, yazmaçları, ve diğer genel kaynakları paylaşırlar.

İş Parçacığı(Thread) Modeli



- a) Tek thread li üç süreç çalışmaktadır.
b) Üç threadli tek süreç çalışmaktadır.

İş Parçacığı(Thread) Modeli



Bir süreç içerisindeki tüm threadler, aynı adres uzayını, açık dosyaları, genel değişkenler gibi kaynakları paylaşırlar.

Süreçler ise fiziksel belleği, diskleri, yazmaçları, ve diğer genel kaynakları paylaşırlar.

Her thread kendi sürecinin içerisindeki tüm adreslere erişebilir, bir thread başka bir thread'in yığının(stack) yazabilir, okuyabilir, ya da tamamen boşaltabilir.

Threadler aynı işi gerçekleştirmek üzerinde ortaklaşa çalışırlar.

İş Parçacığı(Thread) Modeli

Her süreç için	Her thread için
 Adres Uzayı(Address Space) Genel değişkenler Açık dosyalar Çocuk süreçler Askıya alma ikazları (pending alarms) sinyal ve sinyal işleyiciler hesap bilgisi	 Program sayacı (program counter) Yazmaçlar(registers) Yığın(Stack) Durum bilgisi(state) 

Birinci satırda bir süreç içerisindeki tüm threadlerce paylaşılan değişkenler, ikinci satırda ise thread'e özel olan değişkenler listelenmiştir.

İş Parçacığı(Thread) Modeli

Süreçler gibi threadlerde hazır(ready), bloklanmış(blocked), çalışır(running) ve sonlanmış(terminated) durumlarında bulunabilir.

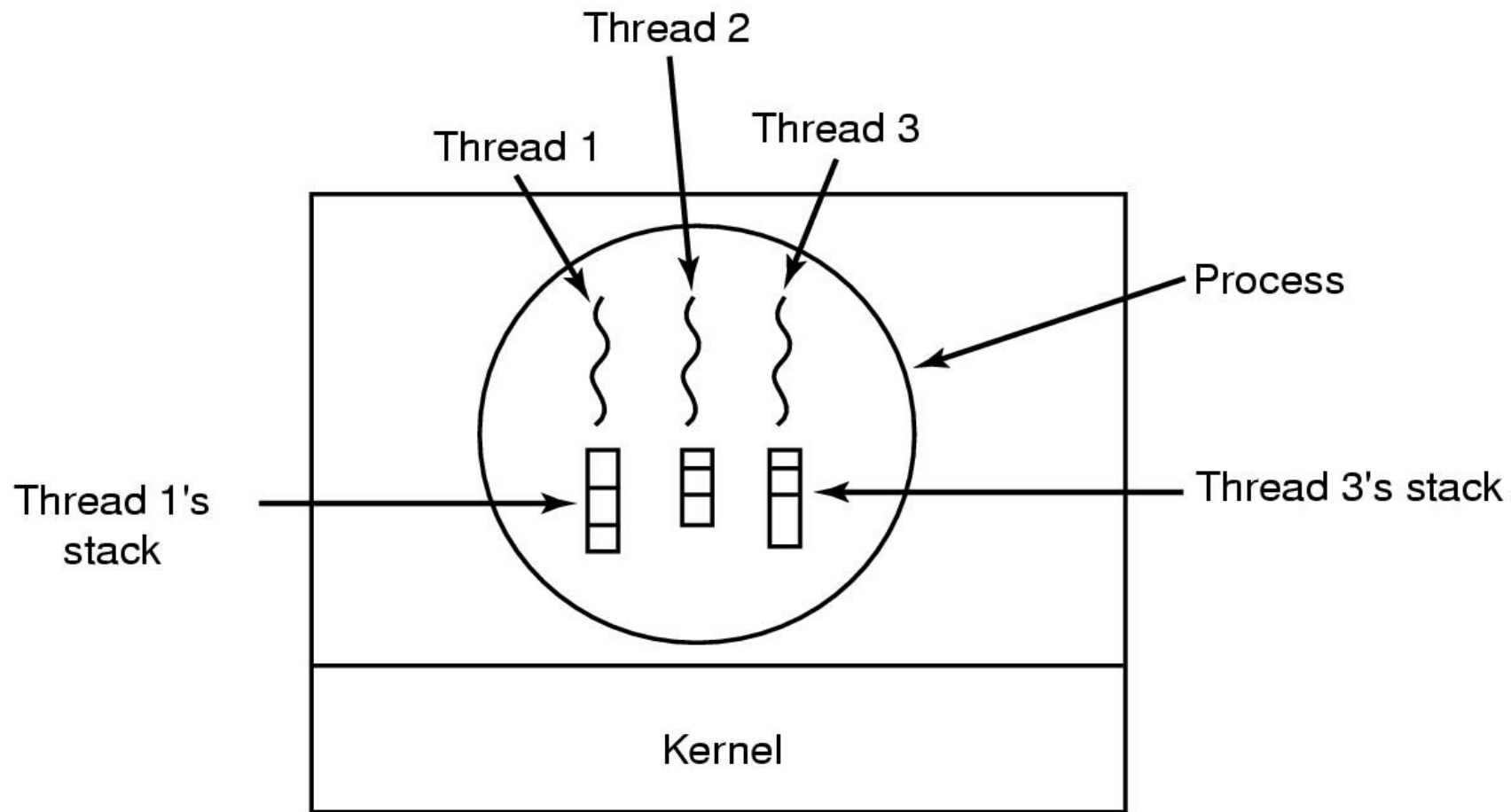
Çalışır durumda olan thread şu anda aktiftir ve işlemciyi kullanmaktadır.

Bloklanmış bir thread, bir olayın olmasını beklemektedir. Örneğin, klavyeden bir verinin girilmesini bekleyen thread bloklanmıştır.

Hazır olan thread ise çalışabilir durumdadır ve kendisine sıra geldğinde çalışacaktır.

İş Parçacığı(Thread) Modeli

Her threadin kendi yığını(stack) bulunmaktadır.



İş Parçacığı(Thread) Modeli

Yazılan programın çoklu iş parçacıklı(multi threaded) çalışabilmesi için yazılan programın thread mekanizmalarını dektekliyor olması ya da dektekleyen kütüphaneleri kullanabiliyor olması gereklidir.

GNU/Linux temelli işletim sistemlerinde, POSIX standart thread API (pthreads) kullanılmaktadır.

Threadler ile ilgili olan tüm fonksiyonlar <pthread.h> kütüphanesinde tanımlıdır. Bu kütüphane standart C kütüphaneleri arasında olmadığı için programın bu kütüphane eklenerek derlenmesi gereklidir.

```
#include <pthread.h>
#include <stdio.h>
//threadin calistiracagi fonksiyon bu sekilde tanimlanmalidir.
//void* F(void* data)
void* print_xs(void* unused){
    while(1){
        fputc('c',stderr);
    }
    return NULL;
}

int main(){
    pthread_t thread_id; //threadi tutan degisken, thread numarasi

    //yeni thread olusturur,
    //ilk parametre thread_noyu tutacak degisken
    //ikinci parametre threadin ozellikleri (thread_attribute nesnesi)
    //ucuncu degisken threadin calistiracagi fonksiyon
    //dorduncu degisken threadin fonksiyonuna gonderilecek degisken
    pthread_create(&thread_id,NULL,&print_xs,NULL);

    while(1){
        fputc('o',stderr);
    }
    return 0;
}
```



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

struct karakter_cikti_parametreleri{
    char karakter; //bu karakter
    int kackez;    //kac kez ekrana yazilacak
};

void* karakter_yaz(void* parametreler){
    struct karakter_cikti_parametreleri* p =
        (struct karakter_cikti_parametreleri*) parametreler;
    int i;
    for (i=0;i<p->kackez;i++){
        fputc(p->karakter,stderr);
    }
    return NULL;
}

int main(){
    pthread_t thread_1;
    pthread_t thread_2;
    struct karakter_cikti_parametreleri p1={'c',3000};
    struct karakter_cikti_parametreleri p2={'o',1000};

    pthread_create(&thread_1,NULL,&karakter_yaz,&p1);

    pthread_create(&thread_2,NULL,&karakter_yaz,&p2);

    return 0;
}
```

İş Parçacığı(Thread) Modeli

`pthread_join(thread_id,void*)` : bir threadin başka threadin çalışmasını bitirmesini bekler.

`pthread_equal(thread_id1,thread_id2)`: iki threadin eşitliğini kontrol eder.

... gibi başka kütüphane fonksiyonları bulunmaktadır.

İş Parçacığı(Thread) Modeli

Bir süreçte üç adet thread bulunsun. Eğer bilgisayarda tek işlemci var ise, bu threadlerin her biri CPU nun $1/3$ hızda çalışabilir.

Eğer sistemde birden fazla işlemci var ise, her thread farklı işlemcide çalıştırılır, bu şekilde uygulama işlemci ve thread sayısına bağlı olarak hızlanır.

Thread kullanımının kötü yanları ise; threadleri kullanan uygulamanın yazımının özenli yapılması gereklidir. Her thread ortak veri kullandığı için birinin bir veri üzerindeki etkisi diğer threadleri etkiler. Örneğin, bir thread bir dosya ile işlemini bitirdiğinde dosyayı kapatsın. Eğer süreçteki başka thread bu dosyayı kullanıyorsa, uygulama çalışmaz ya da sonuçlar yanlış olabilir.

İş Parçacığı(Thread) Modeli

Diğer bir örnek, bir thread kullandığı belleğin tükendiğini görse ve kendine bellek tahsis etmek istese ve bu işleme başlasa; bu noktada başka bir thread'e geçiş olduğunda aynı işlemi ikinci thread'de gerçekleştirirse iki thread de iki kez bellek tahsisi yapmış olacaktır.

Bu tip problemlerin çözümü için, kodun çok dikkatli yazılması gereklidir.

İş Parçacığı(Thread) Modeli

Thread modeli iki şekilde gerçekleştirilebilir.

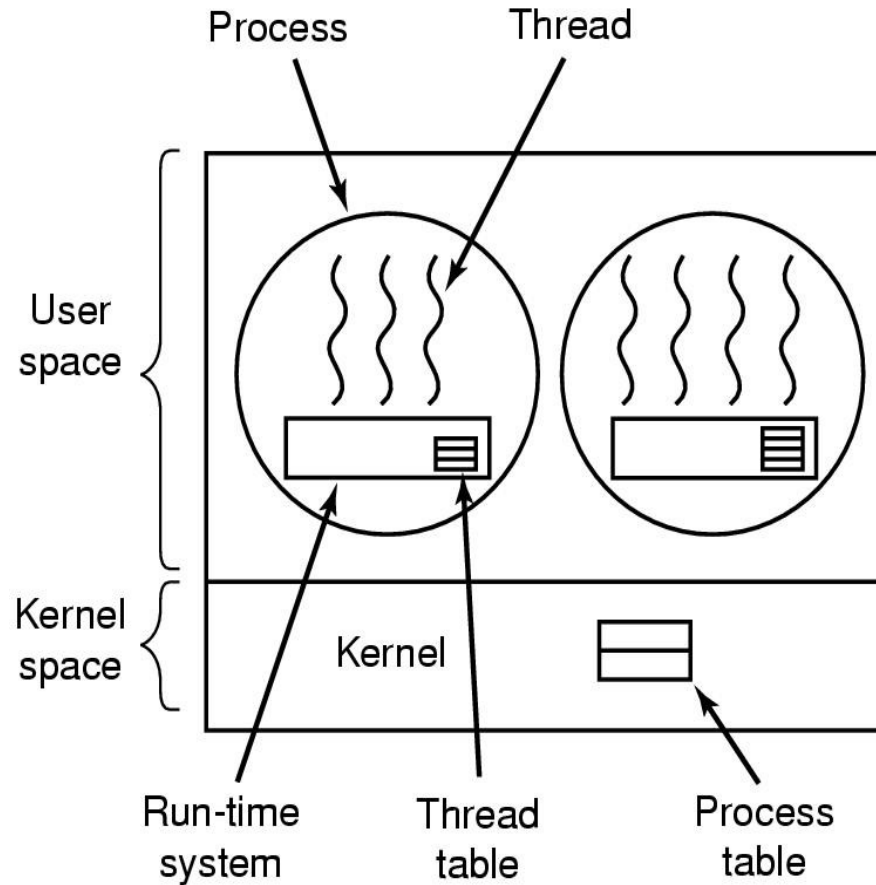
- *Kullanıcı uzayında (user space)
- *Çekirdek uzayında (kernel space)

Thread leri Kullanıcı Uzayında Gerçekleştirme

Bu yöntemde thread mekanizması kullanıcı uzayında çalışmaktadır. Çekirdek, çalışan bir süreç içerisindeki threadlerden habersizdir ve bu sürece tek thread'li gibi davranır.

Bu yöntem işletim sisteminin kendisi Thread leri desteklemese dahi gerçekleştirilebilir.

Thread leri Kullanıcı Uzayında Gerçekleştirme



Thread leri Kullanıcı Uzayında Gerçekleştirme

Bu yöntemde threadleri yöneten prosedürler vardır.(thread_create,thread_exit,... gibi) (pthread kütüphanesi)

Her süreç kendi içerisindeki thread bilgisini tutan bir tabloya sahiptir. Bu tabloya **thread tablosu** denilir.

Her sürecin kendi thread zamanlama(thread scheduler) algoritması olabilir.

Bir thread bloklandığında, aynı süreç içerisindeki tüm threadler ve süreç bloklanır. Thread kendi isteği ile çalışmasını durdurabilir, başka türlü durduramaz.

Thread leri Çekirdek Uzayında Gerçekleştirme

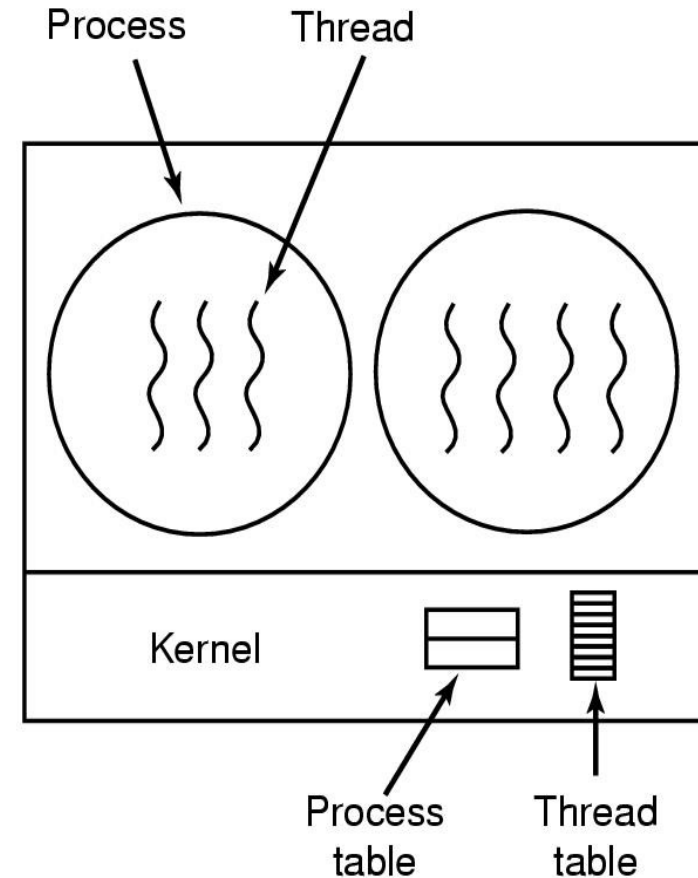
Çekirdek süreçleri nasıl yöneteceğini bilir ve threadlerden haberdardır.

Her süreç için thread tablosu kullanılmaz. Genel bir thread tablosu bulunur. Çekirdekte tüm threadlerin bilgisi saklanır.

Bir thread yeni bir thread oluşturmak ya da yok etmek isterse, sistem çağrısı yapar.

Çok fazla thread oluşturma ve silme, sistem çağrıları ile gerçekleştirildiği için ağır bir yük getirir ve performansın düşmesine yol açar.

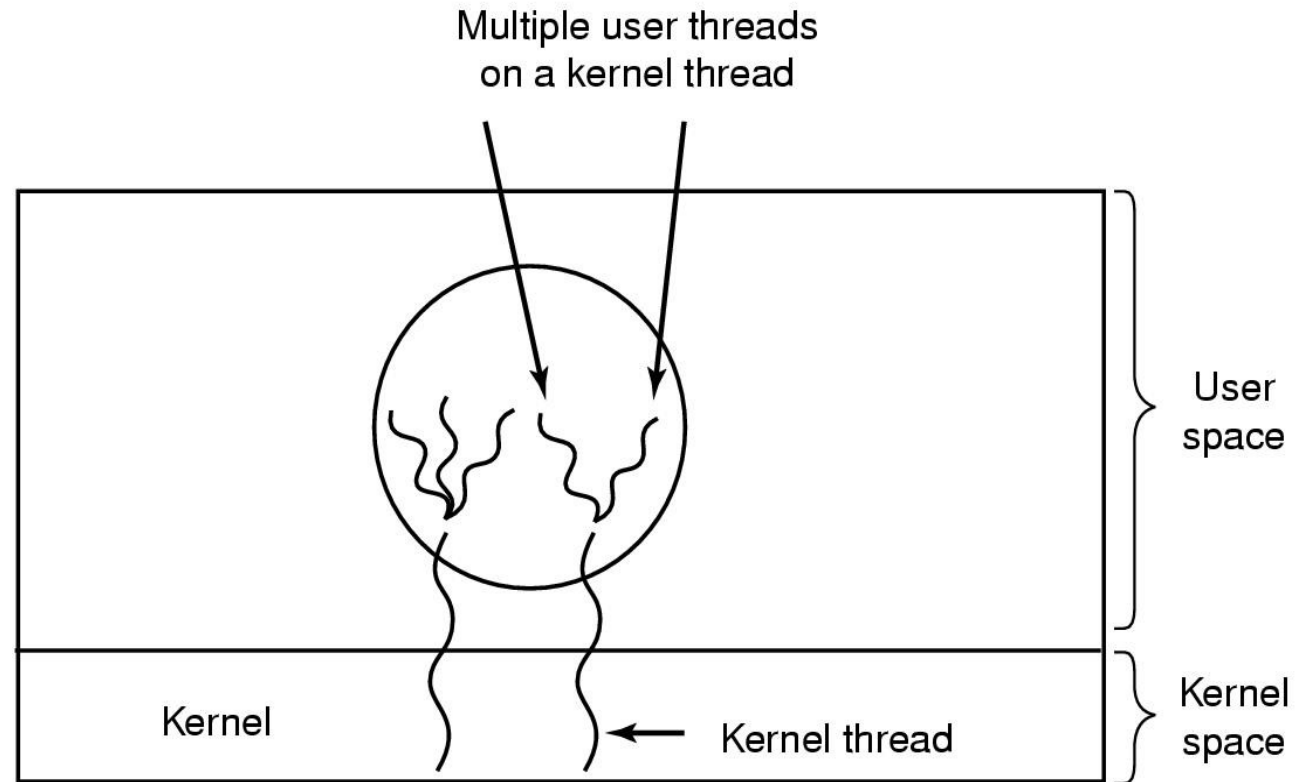
Bir thread bloklandığında çekirdek aynı sürecin başka bir threadine değil, başka bir sürece ait olan thread'e geçiş yapabilir.



Melez yöntem

Çekirdek, kendi seviyesindeki threadlerde zamanlama(scheduling) yapar, kullanıcı seviyeli olanlardan habersizdir.

Çekirdek seviyeli thread de kullanıcı seviyeli olan threadlerin zamanlamasından sorumludur.



İşletim Sistemlerine Giriş

Süreçler ve İş Parçacıkları(Thread)