Bölüm 4: İş Parçacıkları

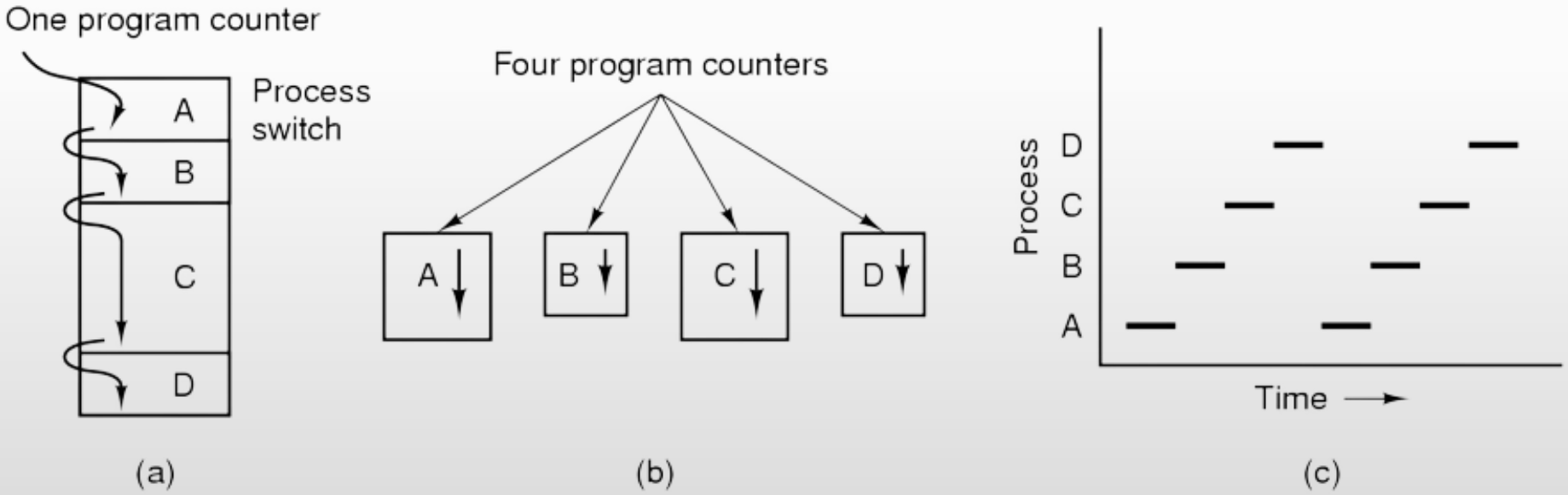
İşletim Sistemleri

Sözde Paralellik

- Tüm modern bilgisayarlar aynı anda birçok iş yapar.
- Tek işlemcili bir sistemde, herhangi bir anda, işlemci sadece bir işlem yürütebilir.
- Ancak çoklu programlama sisteminde işlemci, her biri onlarca veya yüzlerce ms boyunca çalışan işlemler arasında hızlıca geçiş yapar.
- Sözde paralellik kullanıcılar için çok faydalıdır. Ancak; yönetimi bir o kadar zordur.

Çoklu Programlama Süreç Modeli

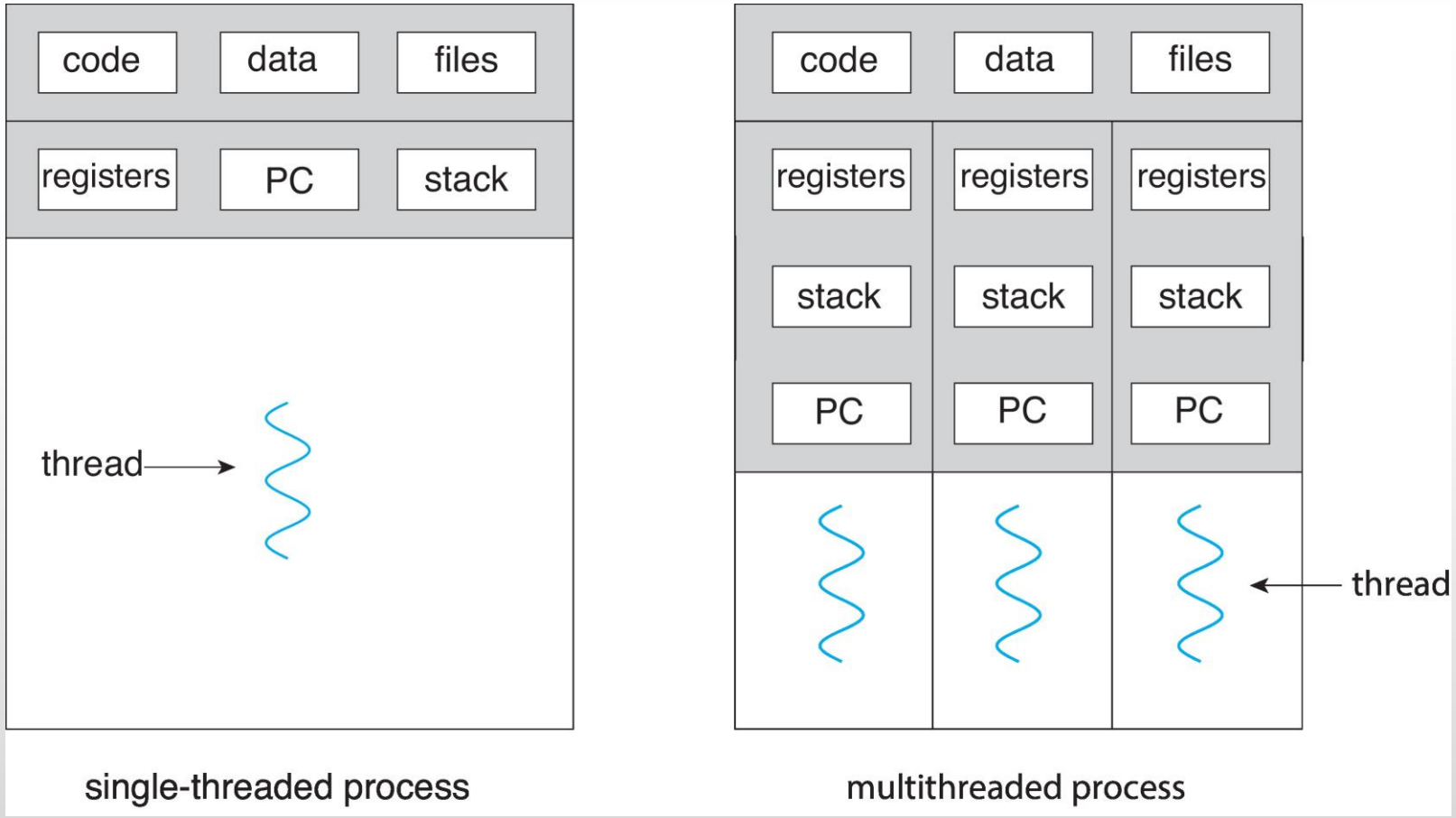
(a) Dört programın çoklu programlanması. (b) Birbirinden bağımsız dört ardışık sürecin kavramsal modeli. (c) Aynı anda bir program etkindir.



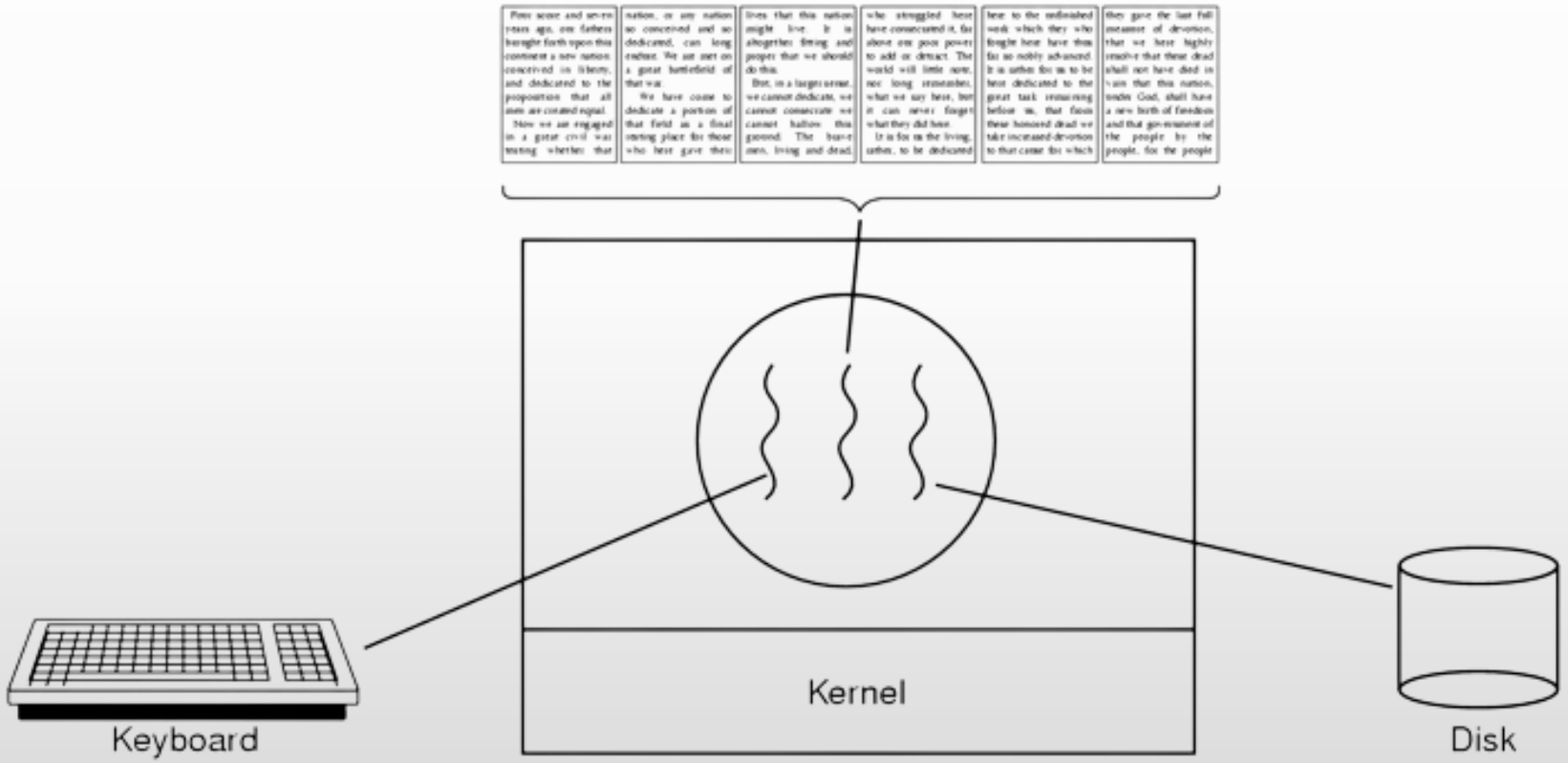
İş Parçacığı

- Modern uygulamalar çok iş parçacıklıdır
- İş parçacıkları uygulama içerisinde çalışır
- Uygulama içerisinde farklı görevler, ayrı iş parçacıkları tarafından uygulanabilir
  - Ekranı güncelleme
  - Veri getirme
  - Bir ağ isteğini yanıtlama
- Süreç oluşturma masraflı, iş parçacığı oluşturma ise daha hafif bir işlemdir
- Kodu basitleştirebilir, verimliliği artırabilir
- Çekirdekler genellikle çok iş parçacıklıdır

Tekli ve Çoklu İş Parçacığı

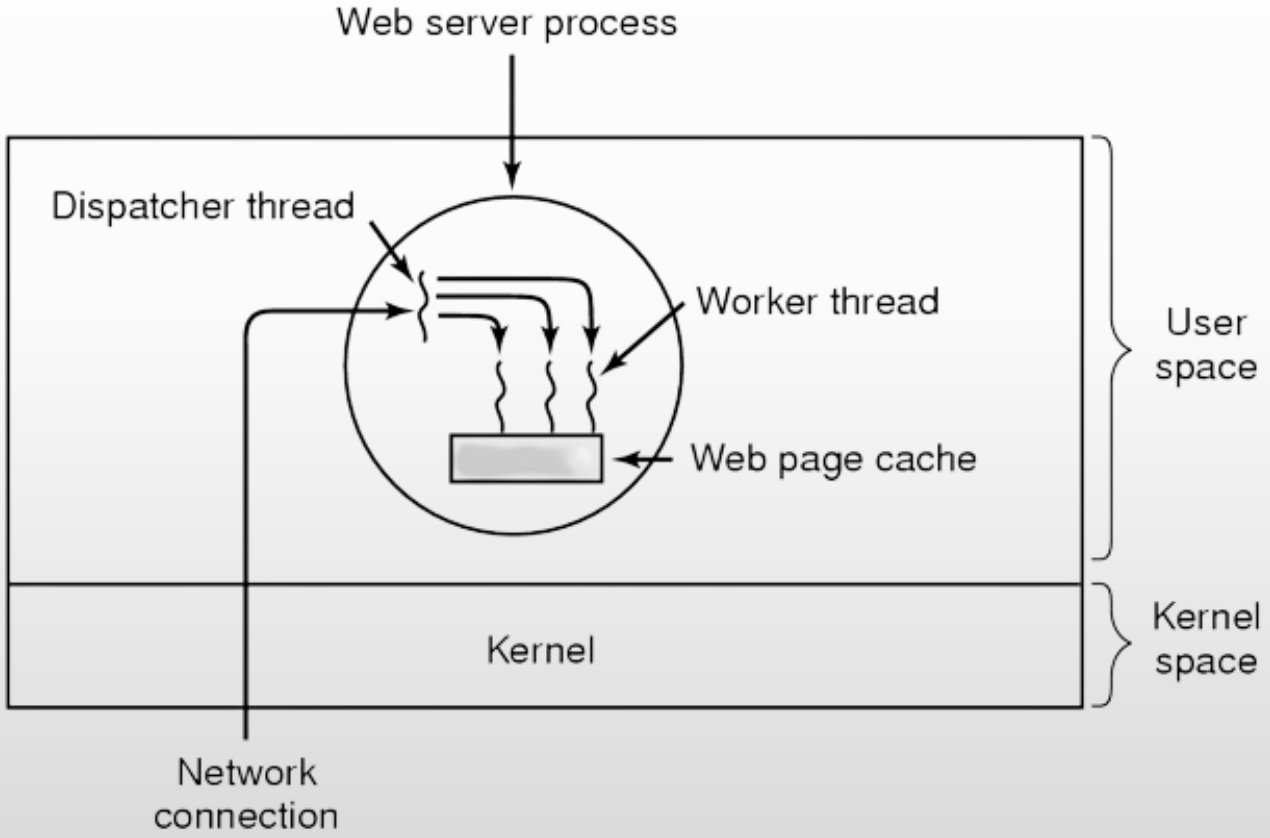


Üç İş Parçacığına Sahip Uygulama



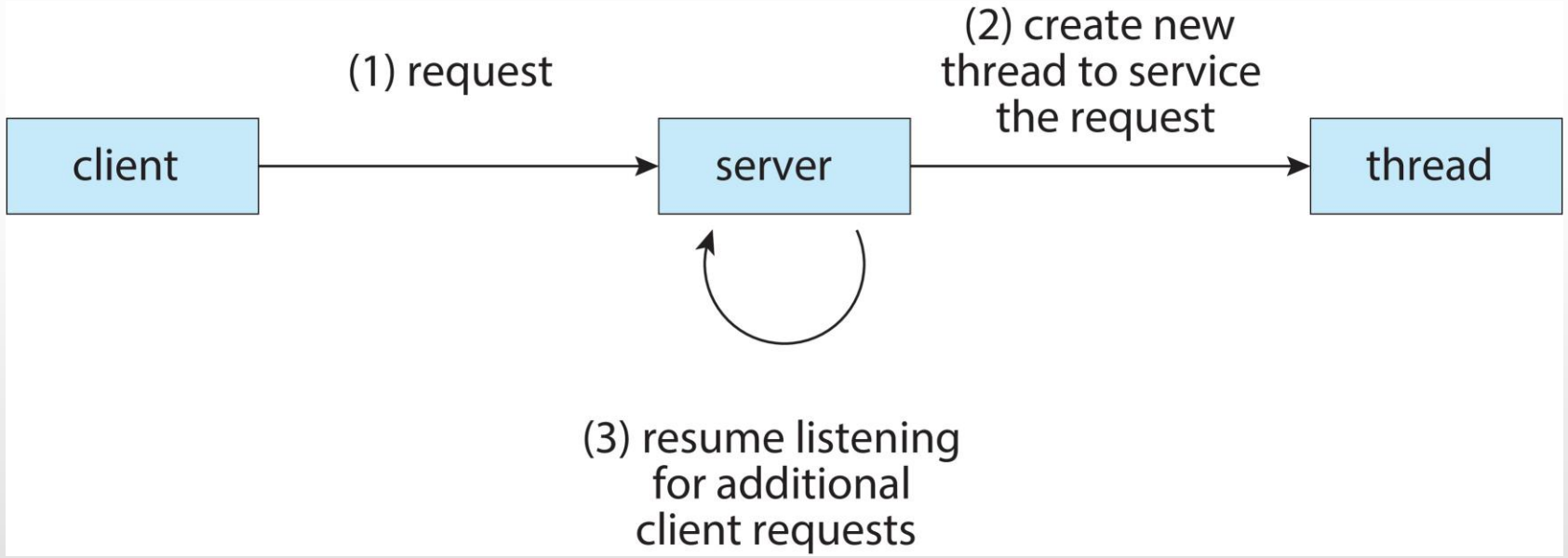
Web Sunucusu

- .



Web Sunucusu

- .



İş Parçacığı Kullanımı

(a) İşlemci zamanlayıcı (dispatcher) iş parçacığı

(b) İşçi (worker) iş parçacığı

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)

Web Sunucusu

- Eğer sayfa önbellekte değilse,
  - iş parçacığı bloke olur
- Sayfanın hazır olması beklenirken,
  - işlemci hiçbir şey yapamaz
- İş parçacığı kullanıldığında ise,
  - sunucu sayfanın hazırlanmasını bir iş parçacığına aktarır ve
  - çalışmaya devam eder.

Web Sunucusu Geliştirmek için Üç Farklı Yol

- İş Parçacığı
  - Paralellik, sistem çağrıları bloke olur
- Tek iş parçacıklı süreç
  - Paralellik yok, sistem çağrılarını bloke olur
- Sonlu durum makinesi
  - Paralellik, bloke olmayan sistem çağrıları, kesilmeler

Faydaları

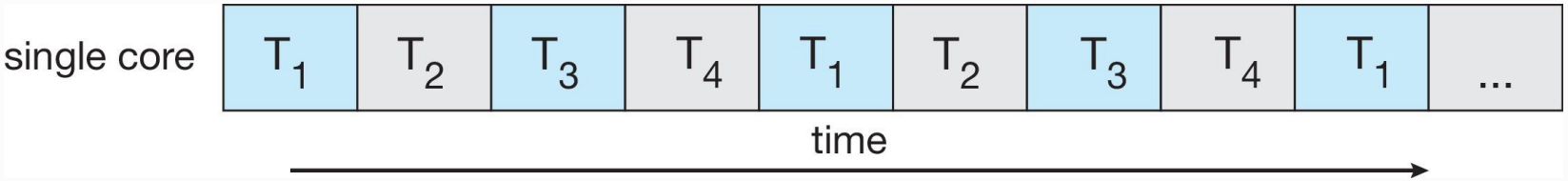
- **Duyarlılık** – özellikle kullanıcı arabirimleri için önemli olan, işlemin bir kısmı bloke olursa yürütmenin devam etmesine izin verir
- **Kaynak Paylaşımı** – iş parçacıkları, süreç kaynaklarını paylaşır, paylaşılan bellek veya mesaj iletmeye göre daha kolaydır
- **Ekonomi** – süreç oluşturmadan daha ucuz, iş parçacığı değiştirme, bağlam anahtarlamaya göre daha az ek yük getirir
- **Ölçeklenebilirlik** – süreç, çok çekirdekli mimarilerden yararlanabilir

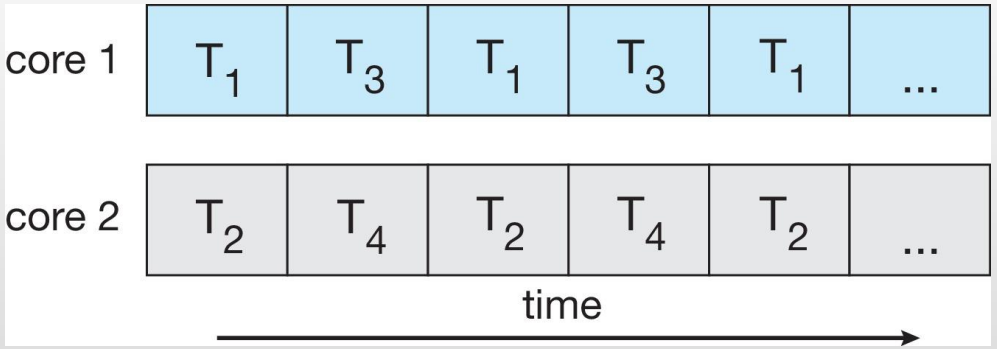
Çok Çekirdekli Programlama

- **Paralellik (parallelism)**, sistemin aynı anda birden fazla görevi gerçekleştirebileceğini ifade eder.
- **Eşzamanlılık (concurrency)**, birden fazla görevin ilerleme kaydetmesini destekler.
- **Getirdiği zorluklar**
  - Etkinlikleri bölmek (dividing activities)
  - Denge (balance)
  - Veri bölme (data splitting)
  - Veri bağımlılığı (data dependency)
  - Test etme ve hata ayıklama (test and debug)

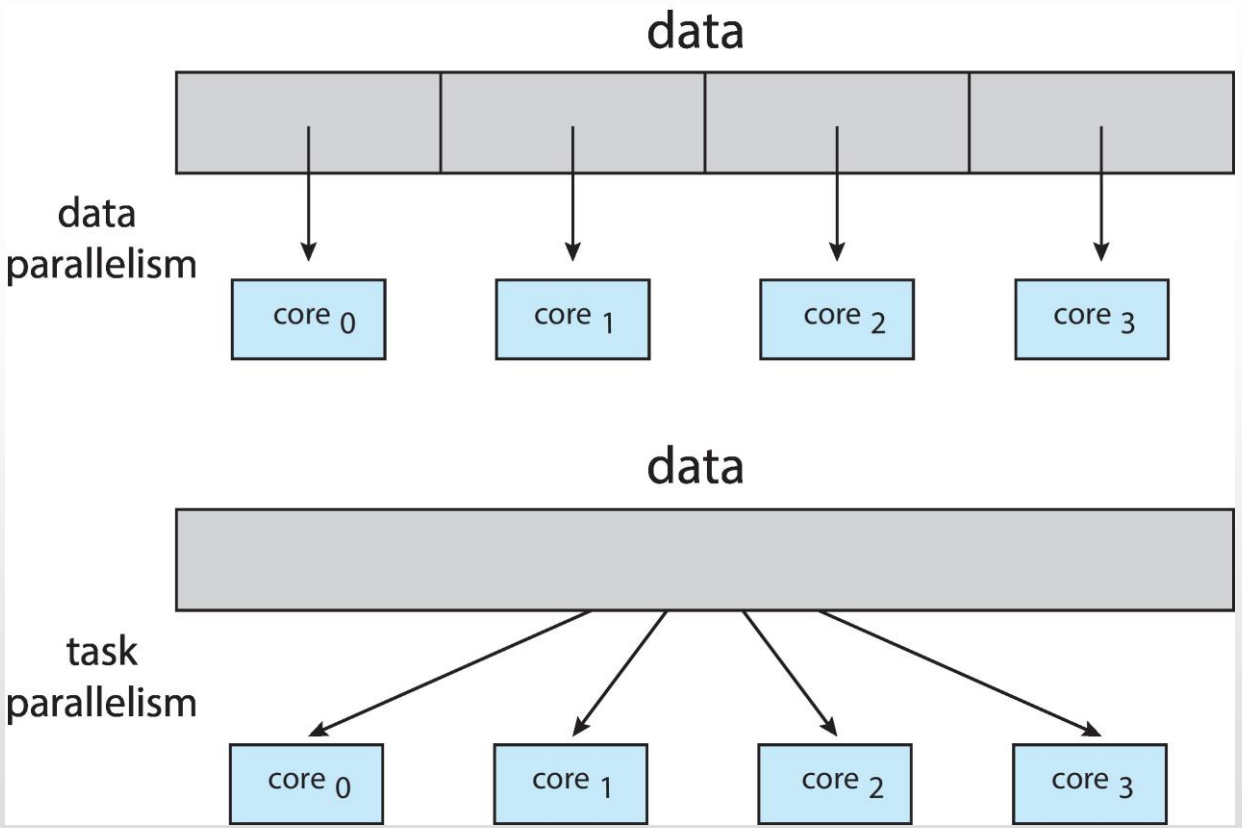
Parallellik ve Eşzamanlılık

- .



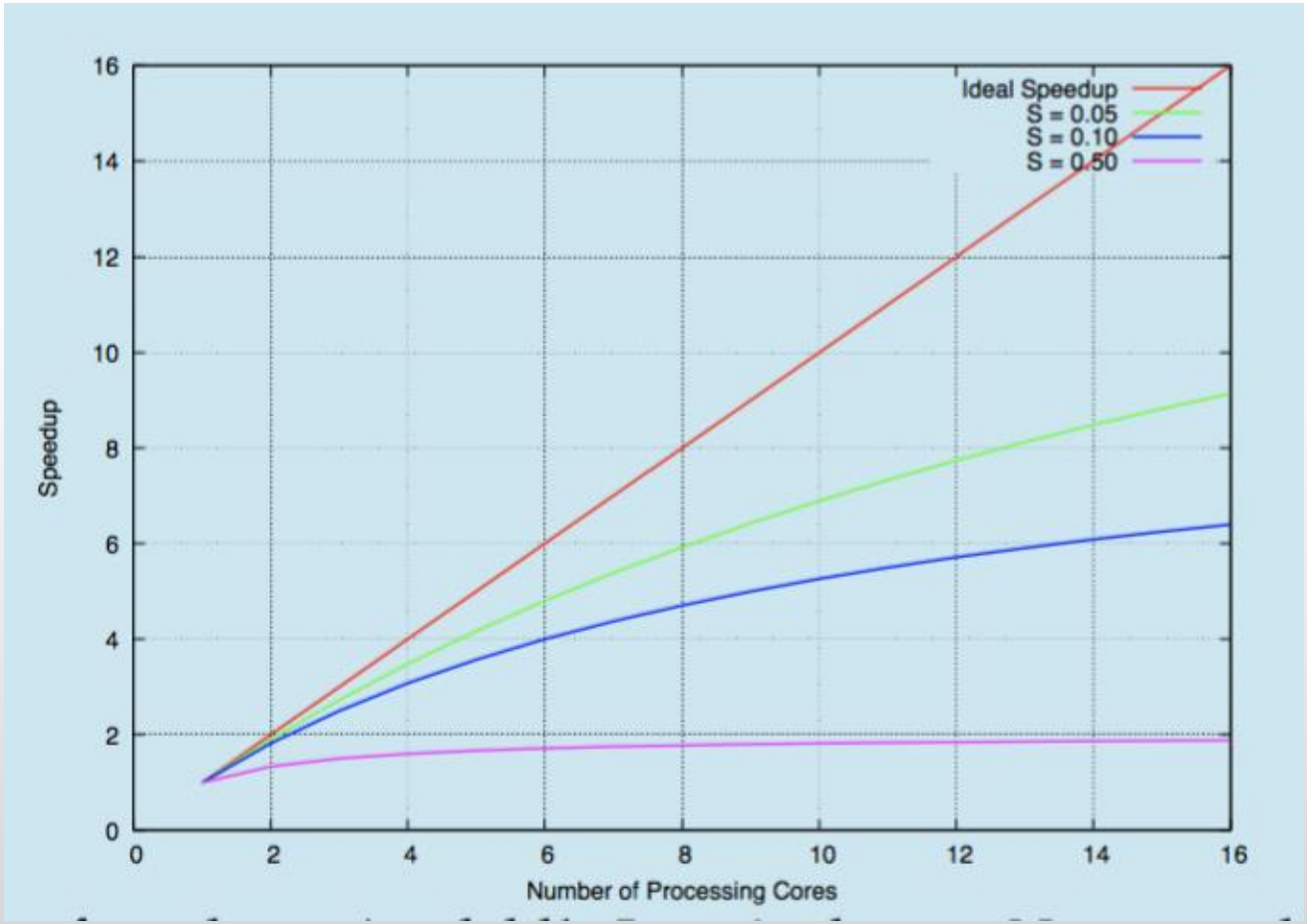


Veri ve Etkinlik Bölme



Amdahl Kuralı

- .



İş Parçacığı Modeli

- Süreç içerisindeki tüm iş parçacıkları ile paylaşılan veriler
  - Adres alanı (address space), Global değişkenler (variables), Açık dosyalar (open files), Çocuk süreçler (child processes), Bekleyen alarmlar (waiting alarms), Sinyal ve Sinyali ele alacak süreçler (signal handlers)

- Her bir iş parçacığına özel veriler
  - Program sayacı (counter), Yazmaçlar (register), Yığın (stack), Durum (state)

İş Parçacığı

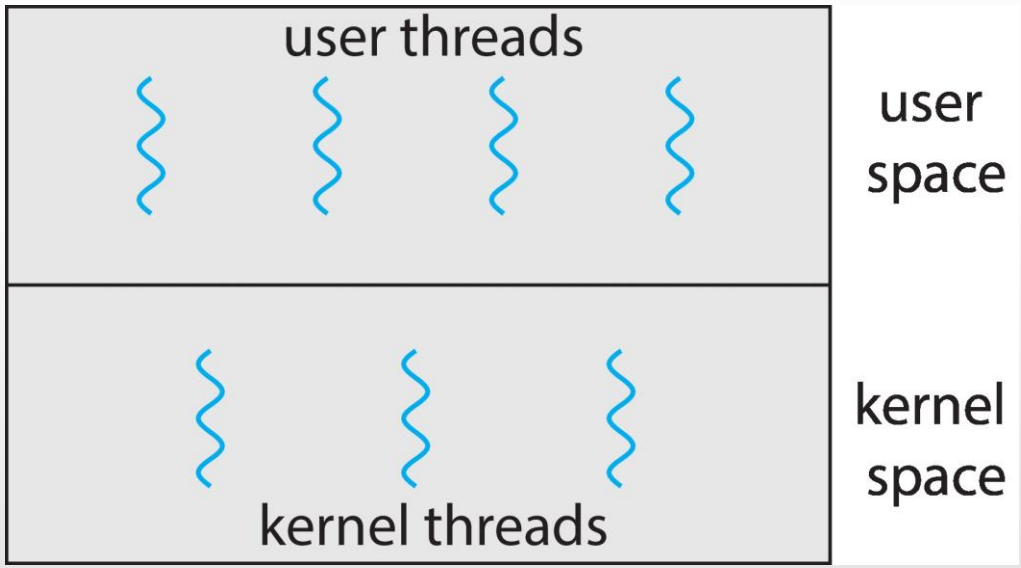
- Kendi program sayacı, yazmaç kümesi ve yığını vardır
- Kod (text), global veri ve açık dosyaları paylaşır
  - Aynı süreci sonlandırmak için paralel çalıştığı iş parçacıkları ile
- Kendi süreç kontrol bloğuna (PCB) sahip olabilir
  - İşletim sistemine bağlıdır
  - Bağlam, iş parçacığı kimliğini, program sayacını, kayıt kümesini, yığın işaretçisini içerir
  - Aynı süreçteki diğer iş parçacıklarıyla bellek adres uzayı paylaşılır
    - bellek yönetimi bilgileri paylaşılır

İş Parçacıkları Nasıl Çalışır

- Bir süreçte bir iş parçacığı ile başlar
- İş parçacığı içeriği (kimlik, yazmaçlar, nitelikler)
- Yeni iş parçacıkları oluşturmak ve kullanmak için kütüphane çağrıları kullanılır
  - Thread_create parametre olarak aldığı prosedürü başlatır
  - Thread_exit iş parçacığını sonlandırır
  - Thread_join başka bir iş parçacığının bitmesi beklenir
  - Thread_yield diğer iş parçacıklarına çalışma şansı verir

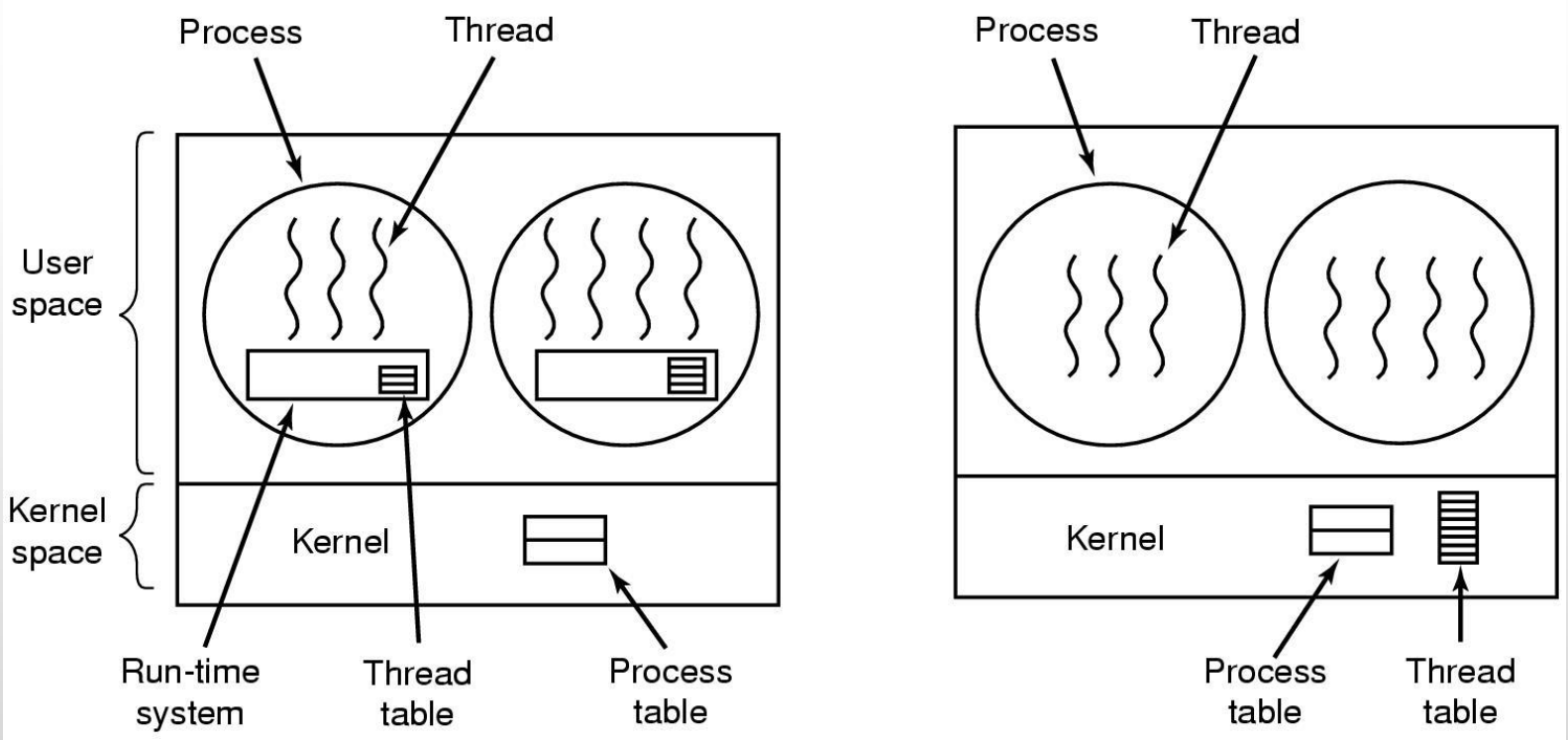
Kullanıcı ve Çekirdek Modu İş Parçacıkları

- .



Kullanıcı ve Çekirdek Modu İş Parçacıkları

(a) Kullanıcı düzeyinde iş parçacığı yönetimi. (b) Çekirdek tarafından yönetilen bir iş parçacıkları.



Çoklu İş Parçacığı Modelleri

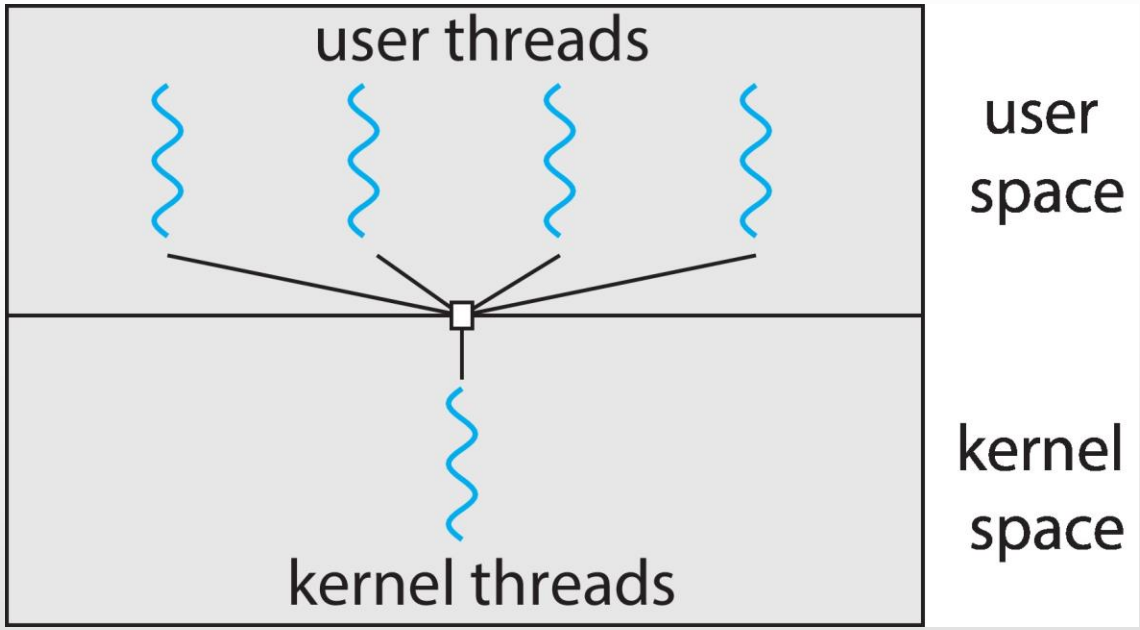
- Çoktan bire (many-to-one)
- Bire bir (one-to-one)
- Çoktan çoğa (many-to-many)

Çoktan Bire Yaklaşım

- Tek çekirdek iş parçacığına eşlenen birçok kullanıcı düzeyinde iş parçacığı
- Bir iş parçacığının bloke olması, tümünün bloke olmasına neden olur
- İş parçacıkları, çok çekirdekli sistemde paralel olarak çalışmayabilir
  - çünkü aynı anda yalnızca bir tanesi çekirdekte olabilir.
- Şu anda çok az sistem bu modeli kullanıyor
  - Örnekler:Solaris Green Threads, GNU Portable Threads

Çoktan Bire Yaklaşım

- .

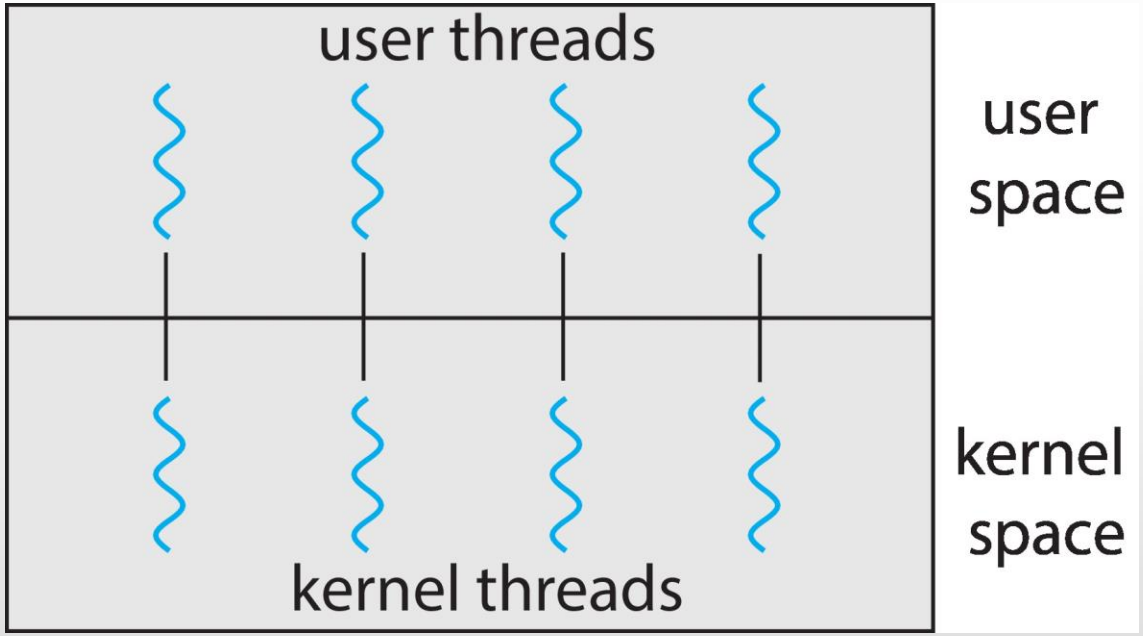


Bire Bir Yaklaşım

- Kullanıcı düzeyindeki her iş parçacığı, çekirdek iş parçacığına eşlenir
- Kullanıcı düzeyinde bir iş parçacığı oluşturmak, bir çekirdek iş parçacığı oluşturur
- Çoktan bire göre daha fazla eşzamanlılık sağlar
- Süreç başına iş parçacığı sayısı bazen ek yük nedeniyle kısıtlanır
- Örnekler; Windows, Linux

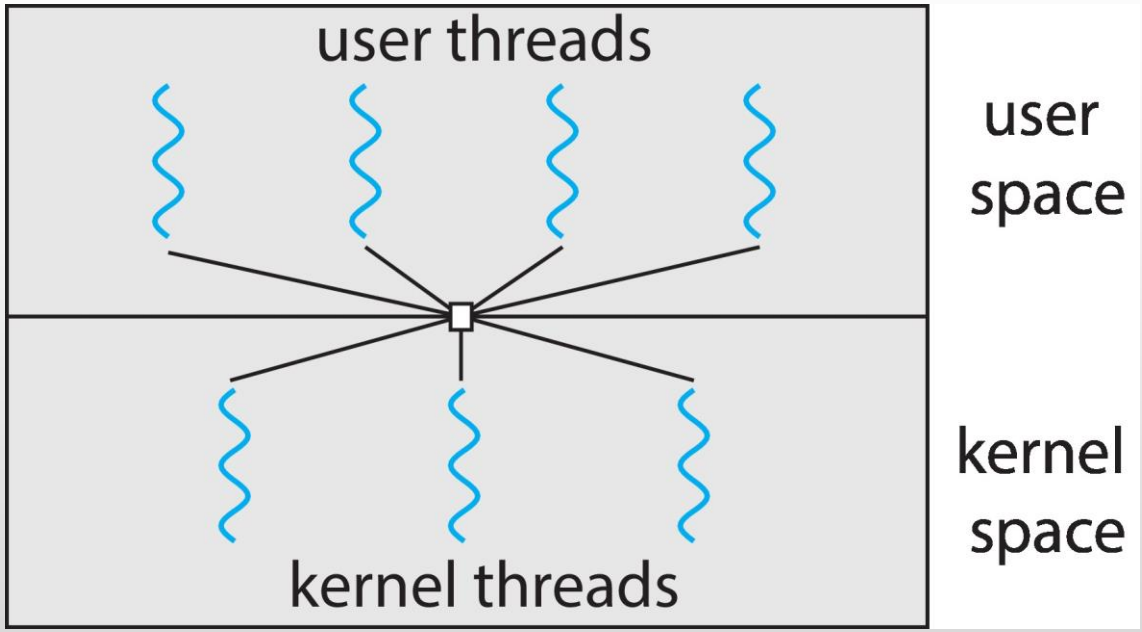
Bire Bir Yaklaşım

- .



Çoktan Çoğa Yaklaşım

- Birçok kullanıcı düzeyinde iş parçacığının birçok çekirdek iş parçacığına eşlenmesine izin verir
- İşletim sisteminin yeterli sayıda çekirdek iş parçacığı oluşturmasına izin verir
- Yaygın değil



Kullanıcı Modu İş Parçacıkları (+)

- İş parçacığı tablosu, iş parçacığı hakkında bilgi içerir (program sayacı, yığın işaretçisi...), böylece çalışma zamanı sistemi bunları yönetebilir
- İş parçacığı bloke olursa, çalışma zamanı sistemi iş parçacığı bilgilerini tabloda saklar ve çalıştırılacak yeni iş parçacığını bulur
- Durum kaydetme ve çizelgeleme, çekirdek modundan daha hızlı çağrılır (tuzak kapı yok, önbellek temizleme yok) (no trap, no cache flush)

Kullanıcı Modu İş Parçacıkları (-)

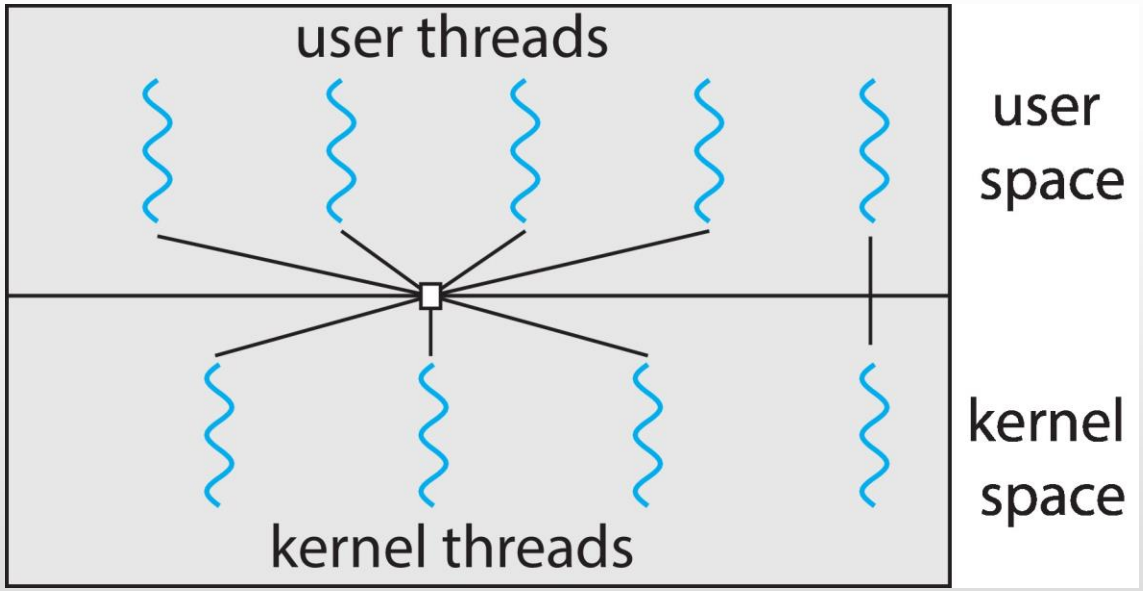
- İş parçacığının sistem çağrısını yürütmesine izin verilemez, çünkü diğer tüm iş parçacıklarını bloke eder.
- Zarif bir çözüm yok
  - Çağrıları engellememek için sistem kütüphanesi kırılabilir (hack)
  - Unix'in bazı sürümlerinde aynı işi yapan benzer sistem çağrıları kullanılabilir
- İş parçacıkları gönüllü olarak işlemciyi bırakmaz
  - Kontrolü sisteme vermek için periyodik olarak kesilmeye uğrar
  - Bu çözümün maliyeti de bir sorundur

Çekirdek Modu İş Parçacıkları

- Çekirdek, kullanıcı modu ile aynı iş parçacığı tablosunu tutar
- İş parçacığı bloke olursa, çekirdek başka bir tanesini seçer
  - Aynı süreçten olması gerekmez!
- Çekirdekte iş parçacıklarını yönetmek çok maliyetli ve değerli olan çekirdek alanında yer kaplar

Hibrit Yaklaşım

- Kullanıcı düzeyindeki iş parçacıklarını çekirdek düzeyindeki iş parçacıklarına eşler

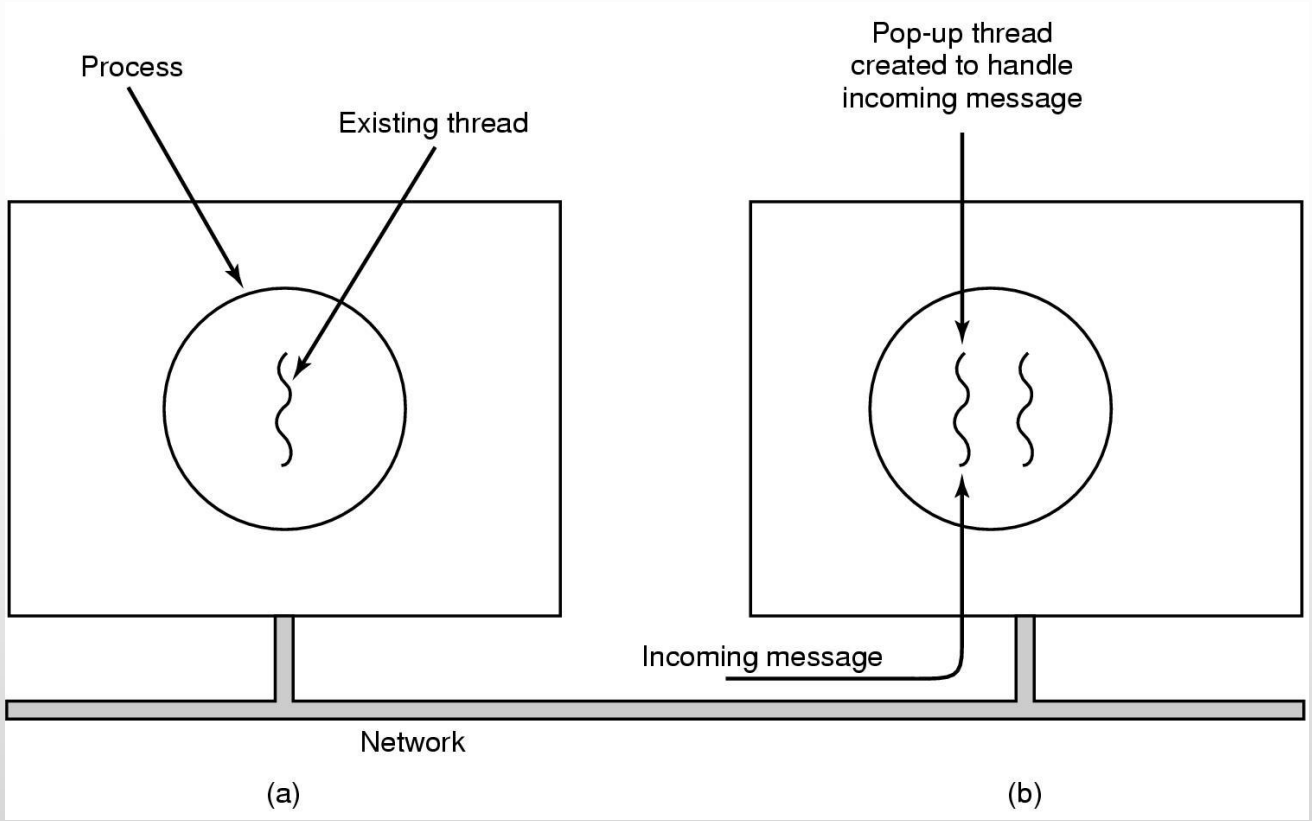


Hibrit Yaklaşım

- Çekirdek, sadece çekirdek iş parçacıklarından haberdardır
- Kullanıcı düzeyinde iş parçacıkları, çekirdekten bağımsız oluşturulur, çizelgelenir, sonlandırılır
- Programcı, kaç tane kullanıcı seviyesi ve kaç tane çekirdek seviyesi iş parçacığı kullanacağını belirler

Açılır (Pop-up) İş Parçacıkları

- Bir mesaj geldiğinde yeni bir iş parçacığı yaratılır



Açılır İş Parçacıkları

- Sistem alma çağrısına bloke olmuş ve mesaj geldikçe işleyen bir iş parçacığı kullanılabilir
- Her mesaj geldiğinde iş parçacığının geçmişinin geri yüklenmesi gerekir
- Açılır iş parçacıkları yenidir ve geri yüklenecek bir verisi yoktur
- Bu nedenle daha hızlıdırlar

POSIX kütüphanesi (pthreads)

- Kullanıcı düzeyinde veya çekirdek düzeyinde sağlanabilir
- İş parçacığı oluşturma ve senkronizasyon için bir POSIX standardı (IEEE 1003.1c) API'si
- Şartname (specification), uygulama (implementation) değil
- API, iş parçacığı kitaplığının nasıl davranması gerektiğini belirtir
- Uygulama, kitaplığın geliştirilmesine bağlıdır
- UNIX işletim sistemlerinde yaygın (Linux ve Mac OS X)

POSIX kütüphanesi (pthreads)

- IEEE Unix standart kütüphane çağrıları

İşlev çağrısı	Açıklama
Pthread_create	Yeni bir iş parçacığı oluştur
Pthread_exit	Çağıran iş parçacığını sonlandır
Pthread_join	Belirli bir iş parçacığının sonlanmasını bekle
Pthread_yield	Başka bir iş parçacığının çalışması için CPU'yu serbest bırak
Pthread_attr_init	Bir iş parçacığının öznitelik yapısını oluştur ve başlat
Pthread_attr_destroy	Bir iş parçacığının öznitelik yapısını kaldır

Üstü Örtülü İş Parçacığı Oluşturma

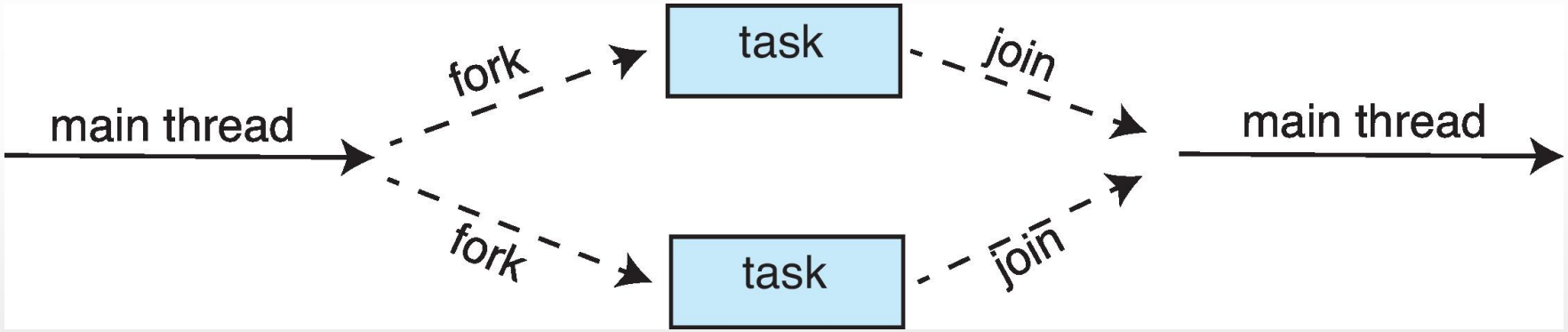
- İş parçacığı sayısı arttığında, programın doğruluğunu kontrol etmek zorlaşır
- İş parçacıkları, derleyiciler ve çalışma zamanı kütüphaneleri tarafından oluşturulur ve yönetilir
- Yöntemler
  - İş parçacığı havuzları (thread pool)
  - Fork - join
  - OpenMP
  - Sevk (grand cental dispatch)
  - Intel İş Parçacığı Oluşturma Yapı Taşları (intel threads building blocks)

İş Parçacığı Havuzu

- Bir havuzda belirli sayıda iş parçacığı tutulur
- Mevcut bir iş parçacığıyla bir isteğe hizmet vermek, yeni bir iş parçacığı oluşturmaktan hızlıdır.
- Uygulama(lar)daki iş parçacığı sayısı havuzun boyutu ile sınırlandırılır

Fork - Join

- .



OpenMP

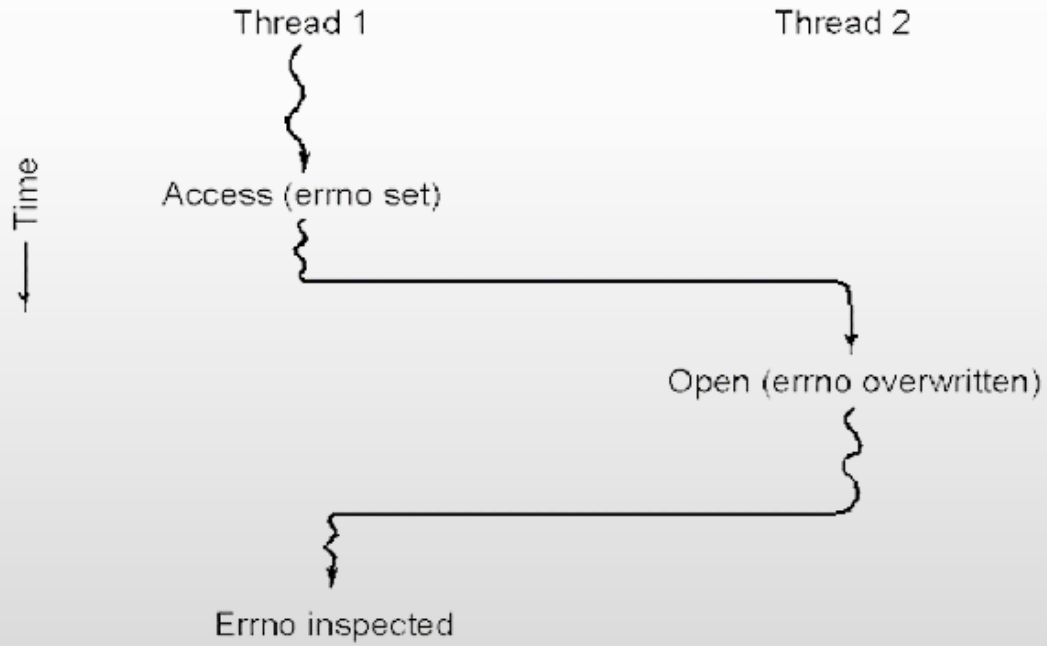
- C, C++, FORTRAN için bir API
- Derleyici yönergeleri kümesi
- Paylaşılan bellek ortamlarında paralel programlama için destek sağlar
- Paralel bölgeleri (paralel olarak çalışabilen kod blokları) tanımlar
- `#pragma omp paralel`
- Çekirdek sayısı kadar iş parçacığı oluşturur

Grand Central Dispatch

- macOS ve iOS işletim sistemleri için Apple teknolojisi
- C, C++ ve Objective-C dillerine, API'ye ve çalışma zamanı kütüphanesine yönelik eklemeler
- Paralel çalıştırılacak kod blokları tanımlanmasına izin verir
- İş parçacığı oluşturma işlemlerini yönetir
- "^{ }" içerisine yazılır
  - `^{ printf("Ben bir bloğum"); }`
- Kod blokları sevk kuyruğuna yerleştirilir
  - Kuyruktan çıkan havuzdaki kullanılabilir iş parçacığına atanır

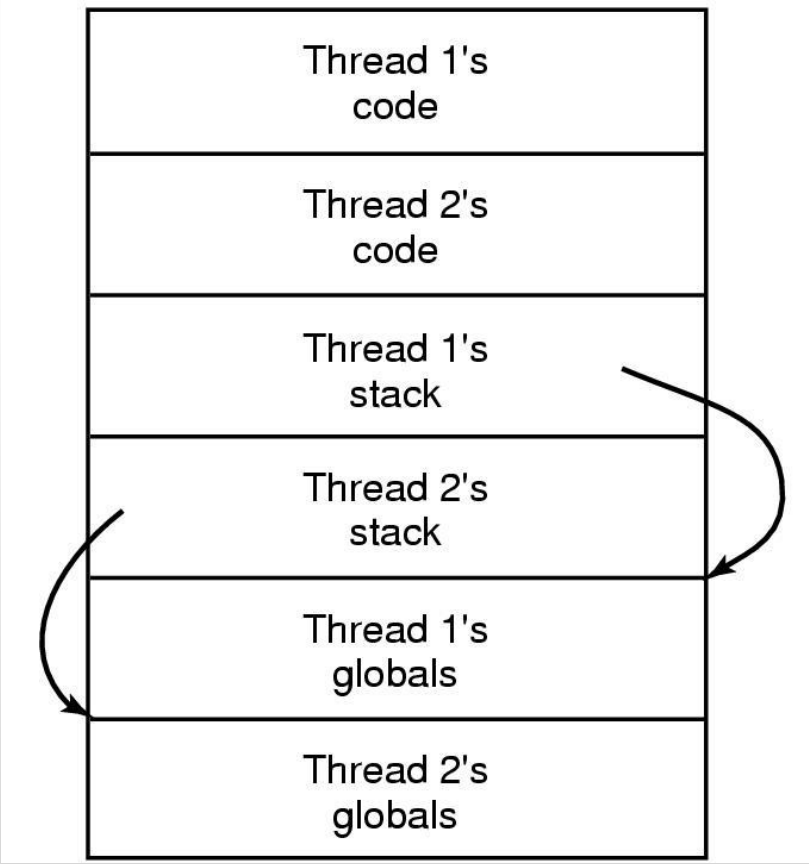
İş Parçacıkları Arasında Çakışma

- Global bir değişkenin kullanımıyla ilgili iş parçacıkları arasında yaşanabilecek çakışma



Çoklu İş Parçacıklı Programlama

İş parçacıkları kendilerine ait global değişkenlere sahip olabilir



Problemler

- Yeniden girilmeyen (not re-entrant) kütüphane yordamı.
  - Bir iş parçacığı mesajı bir ara belleğe koyar, yeni bir iş parçacığı mesajın üzerine yazar
- Bellekten yer alma programları (geçici olarak) tutarsız bir durumda olabilir
  - Yeni iş parçacığı yanlış işaretçi almış olabilir
- İş parçacığına özgü sinyalleri uygulamak zor mu?
  - İş parçacıkları kullanıcı alanındaysa, çekirdek doğru iş parçacığını adresleyemez.

İş Parçacıklarının Kullanılma Nedeni

- Sistem çağrılarını bloke ederek paralelliği etkinleştirir (web sunucusu)
- İş parçacıkları oluşturmak ve yok etmek, süreçlerden daha hızlıdır
- Çoklu çekirdekli sistemler için doğal
- Kolay programlama modeli

İş Parçacıklarının Avantajları

- Kullanıcı duyarlılığı
  - Bir iş parçacığı bloke olduğunda, diğeri kullanıcı G/Ç'sini işleyebilir. Ancak: iş parçacığı uygulamasına bağlı
- Kaynak paylaşımı: ekonomi
  - Bellek paylaşılır (yani adres alanı paylaşılır), Açık dosyalar, soketler
- Hız
  - İş parçacığı oluşturma süreç oluşturmaya göre yaklaşık 30 kat daha hızlı, bağlam geçişi 5 kat daha hızlı
- Donanım paralelliğinden yararlanma
  - Ağır süreçler, çoklu işlemcili mimarilerden de faydalanabilir

İş Parçacıklarının Dezavantajları

- Senkronizasyon
  - Paylaşımlı bellek ve değişkenlere erişim kontrol edilmelidir.
  - Program koduna karmaşıklık, hatalar ekleyebilir. Yarış koşullarından, kilitlenmelerden ve diğer sorunlardan kaçınmak gerekir
- Bağımsızlık eksikliği
  - Ağır Ağırlık İşlemde (HWP) iş parçacıkları bağımsız değildir
  - Adres uzayı paylaşıldığından bellek koruması yoktur
  - Her iş parçacığının yığınları bellekte ayrı yerde olması amaçlanır, ancak bir iş parçacığının hatası nedeniyle başka bir iş parçacığının yığınının üzerine yazma yapılabilir.

Pthreads Mutex - Producer

```
pthread_mutex_t the_mutex;
pthread_cond_t condc, condp;
int buffer = 0; /* buffer used between producer and consumer */

void *producer(void *ptr) { /* produce data */
  for (i= 1; i <= MAX; i++) {
    pthread_mutex_lock(&the_mutex); /* get exclusive access to buffer */
    while (buffer != 0) pthread_cond_wait(&condp, &the_mutex);
    buffer = i; /* put item in buffer*/
    pthread_cond_signal(&condc); /* wake up consumer */
    pthread_mutex_unlock(&the_mutex); /* release access to buffer */
  }
  pthread_exit(0);
}
```

Pthreads Mutex - Consumer

```
void *consumer(void *ptr) /* consume data */
{
  for (i = 1; i <= MAX; i++) {
    pthread_mutex_lock(&the_mutex); /* get exclusive access to buffer */
    while (buffer == 0) pthread_cond_wait(&condc, &the_mutex);
    buffer = 0; /* take item out of buffer */
    pthread_cond_signal(&condp); /* wake up producer */
    pthread_mutex_unlock(&the_mutex); /* release access to buffer */
  }
  pthread_exit(0);
}
```

SON