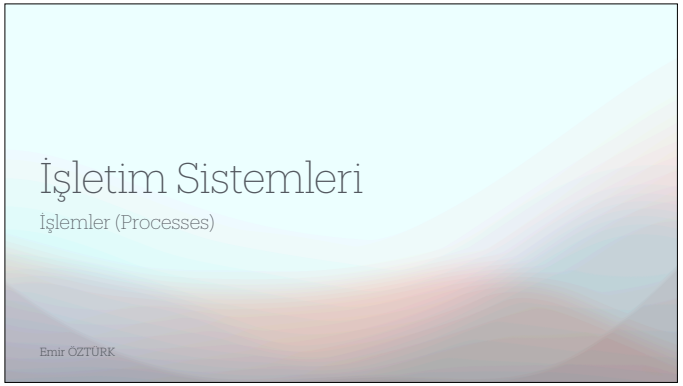
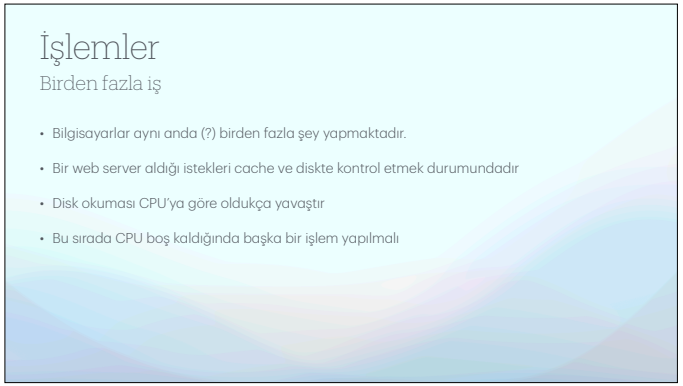
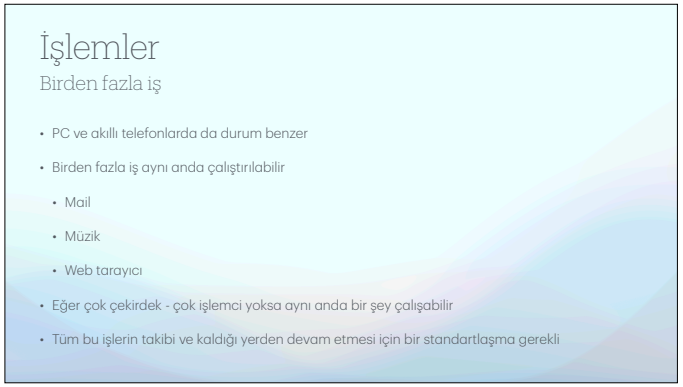
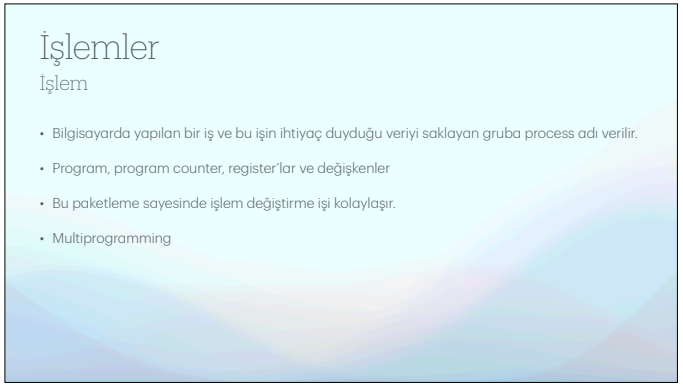




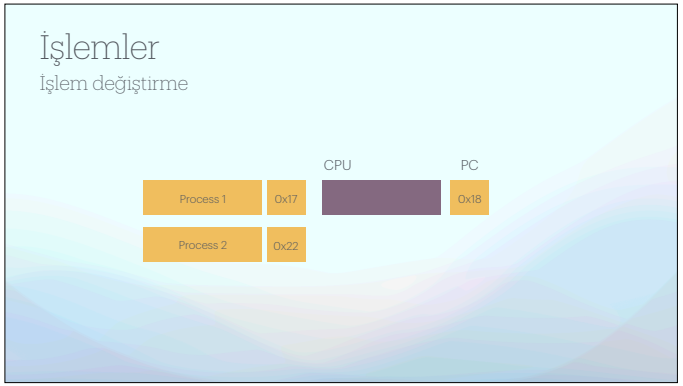
Bilgisayarlarda aynı anda birden fazla işlem yapılması gerekmektedir. Örneğin bir web sunucusu için isteklerin alınması ve cevap döndürülmesi, birden fazla isteğin karşılanması gerekir. Fakat bir zaman aralığında (eğer çok çekirdekli - çok işlemcili bir sistem bulunmuyorsa) en fazla bir işlem çalıştırılabilmektedir. Ayrıca disk okuma gibi işlemler cpu'da işlem yapılmasına göre oldukça yavaştır. Bir işlem disk okuması veya io işlemi gibi bir işlem istiyor ve bu sırada cpu'yu kullanmıyorsa cpu'nun başka bir işleme tahsis edilmesi uygun olacaktır.

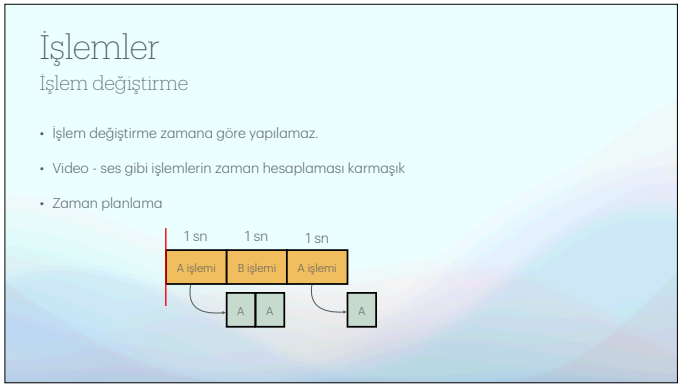


Birden fazla işlemin kullanıcıya aynı anda çalıştığı izleniminin verildiği tek ortam server'lar değildir. PC ve akıllı telefonlarda da aynı şekilde bir anda birden fazla işlem açık olabilir. Bir kullanıcı aynı anda müzik dinlerken maillerine de bakmak isteyebilir. Eğer çok çekirdek - çok işlemci mevcutsa bu durumda her işlemcide bir işlem çalıştırılabilmesi mümkündür fakat tek işlemci bulunduğu durumda işlemlerin bir şekilde işlemciyi ortak olarak kullanması gerekmektedir. İşlem değiştirme işleminin verinin bozulmadan gerçekleştirilebilmesi için ise bir standarda ihtiyaç duyulmaktadır.

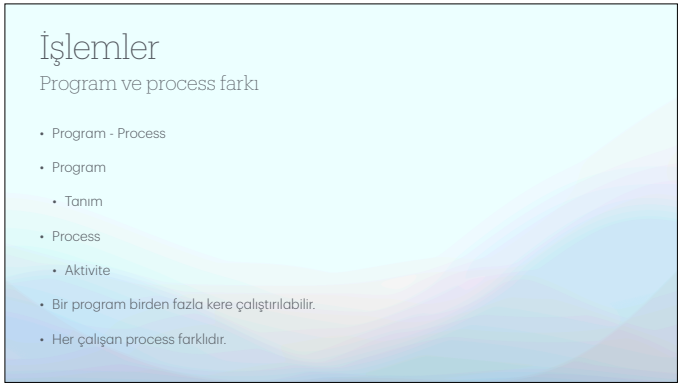


Standartlaşma için bilgisayarda çalıştırılan her iş, verisi ile birlikte bir grup halinde saklanır. Bu gruba ise işlem (process) adı verilir. Bir işlemde işlemin kodu (program), işlemin nerede kaldığını belirten program counter, işlemin o anki verilerini tutan register ve değişken değerleri bulunur. Bir işlem değişikliği istendiğinde paket içerisindeki tüm değerler saklanır ve bir başka işleme geçilir. Böylece veri kaybı yaşanmadan işlemler arası geçiş sağlanabilir (ilerleyen slaytlarda saklanan verilerin tablosu verilmiştir).





İşlem değiştirme işlemi belirli kurallara göre yapılmalıdır. Örneğin zamana göre bölme yapılmak istenirse her işlemin ihtiyaç duyduğu zaman eşit olmayacağından ve bazı işlemlerin önceliğe sahip olması gereğinden dolayı iyi bir tercih gerçekleştirilmiş olmaz. Özellikle video-ses oynatımı, ve gerçek zamanlı çalışması gereken işlemler zamanla sıranın kendisine gelmesini beklemek zorunda kalır.



Program ile process arasında fark bulunmaktadır. Program çalıştırılacak bir işin yönergesiyken process ise çalışan halidir. Bir program birden fazla kez çalıştırılabilir ve birden fazla process aynı anda aynı işi yapabilir. Her çalışan process birbirinden bağımsızdır.

Bir işlem dört şekilde oluşturulabilir.
Sistem başladığında init() bir process olarak çalıştırılır.
Bir diğer durumda çalışan bir process sistem çağrısında bulunup yeni bir process oluşturabilir. Buna fork adı verilir.
Kullanıcı gui veya shell üzerinden bir komut çalıştırarak process başlatabilir.
Son olarak bir toplu iş işlemi çalıştırılır processler sıra ile çalışır.
Bazı işlemler kullanıcı ile etkileşim halindeyken bazı işlemler ise arkaplanda çalışır. Örneğin belirli aralıklarla yeni gelen mailleri kontrol eden bir işlem arkaplan işlemi olarak kabul edilirken, kullanıcının çalıştırdığı bir uygulama programında ise durum farklıdır.

İşlemler

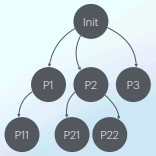
İşlem sonlandırma

- İşlemin tamamlanması - Normal exit
- İşlemin hata vererek tamamlanması - Error exit
- İşlemin tamamlanmadan hata vermesi - Fatal error
- İşlemin başka bir işlem tarafından kapatılması - Kill by another process
  - Kill
  - TerminateProcess

İşlemler

İşlem hiyerarşisi

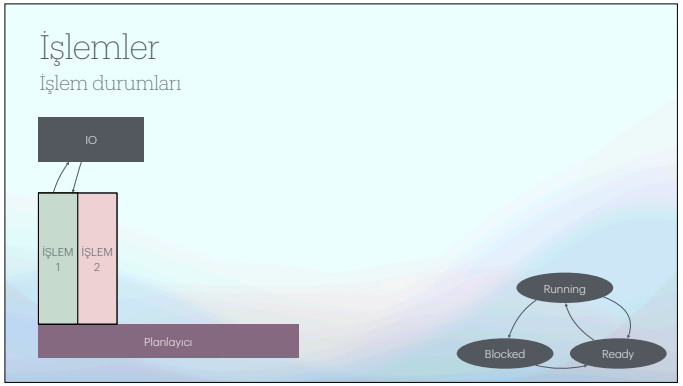
- Her işlem bir çocuk işlem oluşturabilir
- Her çocuk işlem de başka işlemin ebeveyni olabilir
- İşletim sisteminin başlangıcı buna örnek verilebilir
  - init

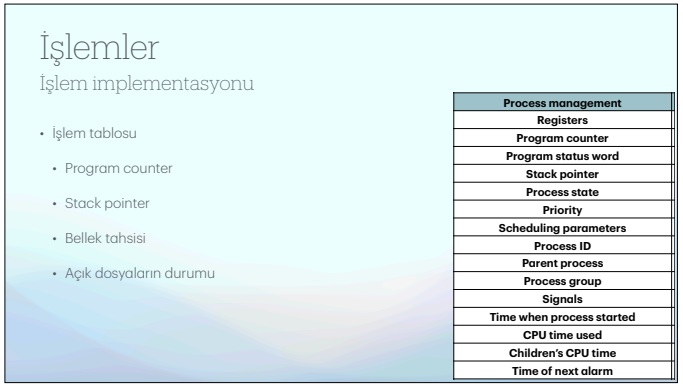


İşlemler

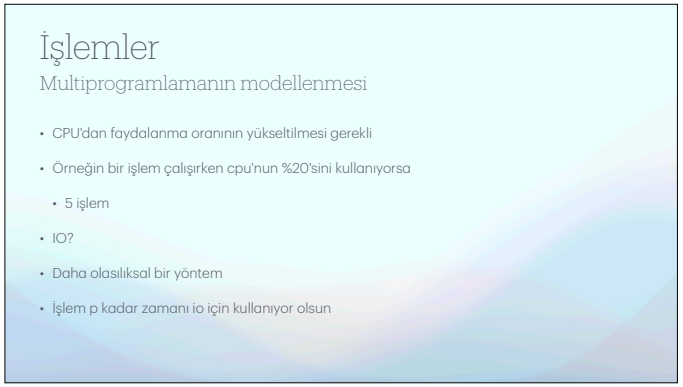
İşlem durumları

- İşlemler başka işlemleri bekleyebilir
- İşlemler bir girdi bekleyebilir
- İşlemlerin bulunduğu durumun bilinmesi gerekir
- Üç işlem durumu bulunmaktadır
  - Çalışma durumu (Running)
  - Hazır durum (Ready)
  - Engellenmiş durum (Blocked)





Bir işlem için saklanan değerler işlem tablosunda tutulur. Bir işlem tablosu içeriği tablodaki gibi verilmiştir.

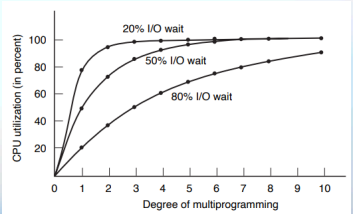


Birden fazla uygulama çalıştırılmak istendiğinde aynı anda kaç işlem çalıştırılması gerektiğinin bir hesabının yapılması cpu'nun verimli kullanılması için gereklidir. Örneğin her işlem cpu'nun %20'sini kullanıyorsa temelde 5 işlem çalıştırarak %100 cpu kullanımına erişilir diyebiliriz. Fakat her işlem zamanının %100'ünde cpu kullanmayabilir. io veya disk okuma yazma işlemlerinin beklemesinin de hesaba katılması gerekmektedir. Bu sebeple daha olasılıksal bir yöntem seçilebilir. Her çalıştırılan işlemin cpu zamanının p kadar kısmını io için beklediğini varsayalım

İşlemler

Multiprogramlamanın modellenmesi

- n adet işlem
  - p^n kadar süre CPU boşta
- CPU kullanımı = 1-p^n
- p'nin farklı oranları için



Böyle bir durumda n adet işlem çalıştırıldığında p^n kadar bir süre cpu boşta kalacaktır. Olasılık değer toplamı 1 ettiği için CPU kullanımını 1-p^n olarak hesaplayabiliriz. P ve N değerlerini farklı seçtiğimizde oluşan grafikler şekilde verilmiştir.

İşlemler

Thread

- İşlemlerin adres alanı mevcut
- Her işlemin adres alanı izole
- Thread'ler bir işlemin içerisinde
- Ortak bir adres alanı

Her işlem izole bir şekilde kendi adres alanına sahiptir (bellek yönetimi konusunda detaylandırılacaktır). Ortak adres alanı kullanmak isteyen paralel işlemler için thread'ler kullanılabilir.

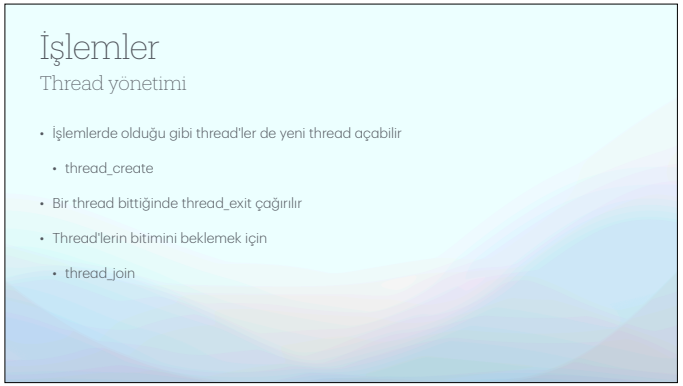
İşlemler

Thread ihtiyacının sebepleri

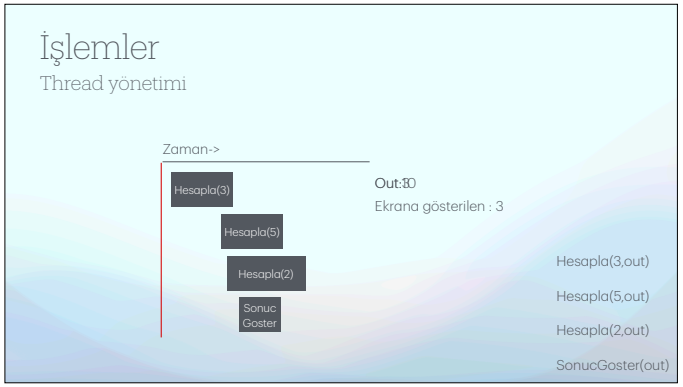
- Ortak bir veriyi işlemek isteyen paralel işlem ihtiyacı
- Thread'ler daha hafif-siklet (lightweight)
  - 10-100 kat hız
- IO ve CPU aynı anda kullanılacaksa thread'ler avantajlı
  - Paralel işlem
- Çok CPU'lu sistemlerde gerçek paralelizm

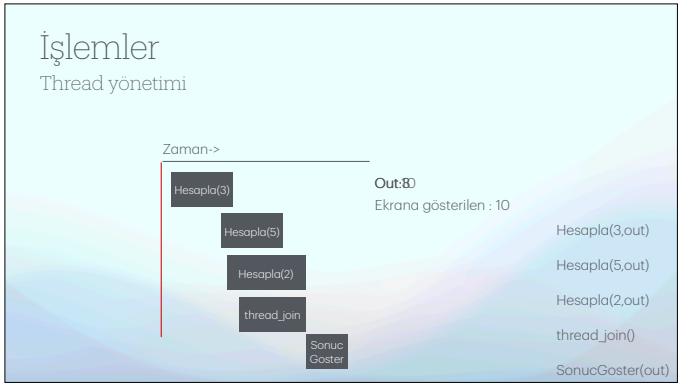
Thread oluşturulması process oluşturulmasına göre daha az vakit harcar. Thread tablosu daha küçüktür ve process kadar veri saklanmasına gerek yoktur. Aynı zamanda io ve cpu'yu aynı anda kullanacak bir işlem varsa bunu birden fazla thread ile paralel bir şekilde gerçekleştirebilir. Ayrıca yine çok cpu'ya sahip sistemlerde gerçek paralel çalışma sağlanabilir.

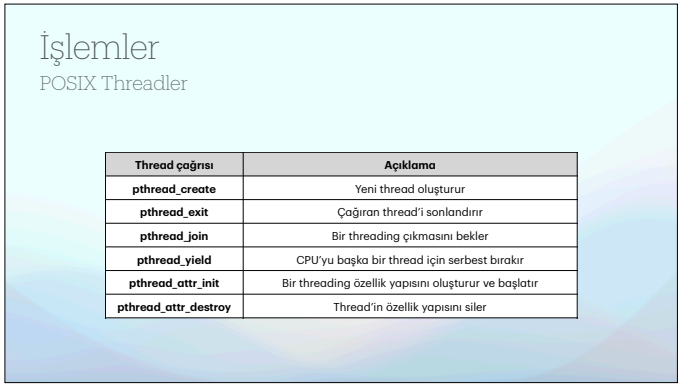
Çalışma zamanına bakılırsa tek işlemcinin olduğu sistemlerde aynı işlemin yapıldığı varsayılırsa bir işlemin oluşturduğu 3 thread ve 1'er thread'e sahip 3 işlem aynı zamanı harcayacaktır. Oluşturma maliyetleri sayılmadığında hangisinin seçileceği yapılacak işe bağlıdır. Eğer ortak bir alana sahip olan paralellik isteniyorsa thread'lerin kullanımı daha uygundur. Ayrıca thread'ler ortak bir dosyaya ya da ortak global değerlere erişebilirler.



İşlemlerin fork edebilmesi gibi thread'ler de yeni thread oluşturabilir. Bir thread bittiğinde thread_exit ile sonlanır. Eğer oluşturulacak son değerin tüm threadlerin bitmesini beklemesi gerekiyorsa thread_join kullanılmalıdır.







POSIX Thread çağrıları

Thread'lerin tamamı kullanıcı alanında oluşturulabilir. Bu durumda thread yönetimi işleme aittir. Kernel yalnızca işlemleri yönetir. Paralelizm desteği olmayan sistemlerde thread ile paralellik sağlanabilir. Tabi ki bu durumda işlemlerde olduğu gibi her işlemin içerisinde bir thread tablosu bulunmalıdır ve işlemler de thread'lerin zaman yönetimini gerçekleştirmelidir.

İşlemler

User space thread'leri

- User space thread'lerinin dezavantajları da bulunmaktadır.
- Bloklama gerçekleştiren sistem çağrılarının nasıl yönetileceği belirsizdir.
- Thread bir çağrı yaptığında bu çağrı process tarafından kernel'a ileteceğinden tüm thread'ler duracaktır.
  - Tüm sistem çağrıları bloklanmayacak (non-blocking) şekilde değiştirilebilir ama işletim sistemini değiştirmek gerekir (!)
- Çağrının bloklama yapıp yapmayacağı kontrol edilerek thread'in isteği pas geçilebilir
  - Sistem çağrılarının durumunu öğrenecek şekilde değişiklik yapılması gerekir (!)

Eğer thread'lerden biri bloklama, io erişimi veya disk okuması isterse diğer thread'ler devam edebileceği halde process bu işi istemiş olduğu için kernel tarafından process ve buna bağlı tüm thread'ler bloklanır.

İşlemler

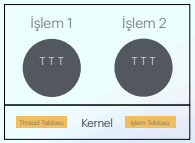
User space thread'leri

- Kernel thread'lerden habersiz olduğu için thread'lerden biri page fault ile karşılaştığında (Bkz: bellek yönetimi) diğer thread'ler çalışabilir olduğu halde tüm işlemi iptal edebilir.
- Planlama doğru yapılmazsa bir thread gönüllü bir şekilde CPU'yu bırakmadığı sürece başka thread'lere sıra gelmeyebilir.

Yine thread'lerden biri hata ile karşılaştığında diğer thread'ler düzgün çalıştığı halde kernel tarafından işlem sonlandırılabilir. Thread'lerin nerede duracağı veya zaman planlaması doğru yapılmazsa CPU'yu bırakmayacağı için diğer thread'lere geçilemeyebilir.

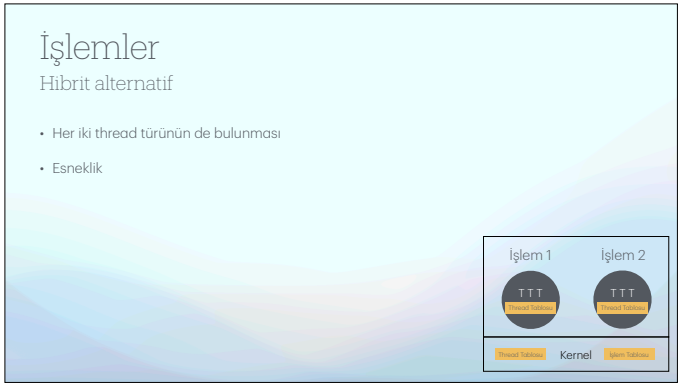
İşlemler

Kernel space thread'leri



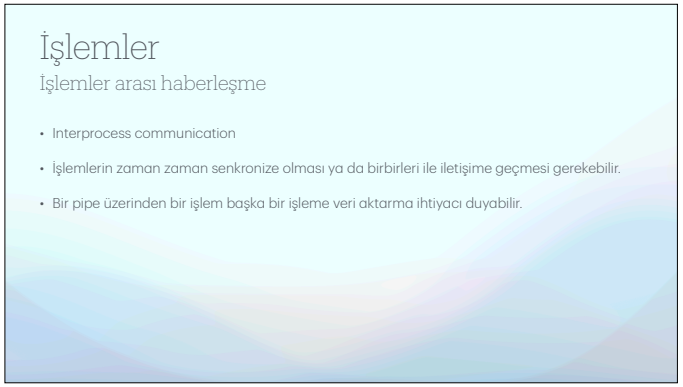
- Thread tablolarının da kernel space'e çekilmesi
- User space'e göre performans düşük
- User space thread'lerinin problemlerine çözüm getirmekte
- Birden fazla thread'e sahip bir işlemin fork durumu hala çözümsüz
  - Yeni process'in thread sayısı
- İşletim sistemlerinin sinyallerini üstlenecek thread'in seçimi

Böyle durumlar için thread yönetimi de kernel'a bırakılabilir. Fakat kernel tüm yönetimi yapacağı ve tüm thread'lerin tablolarını saklayacağı için performans daha düşük olacaktır.

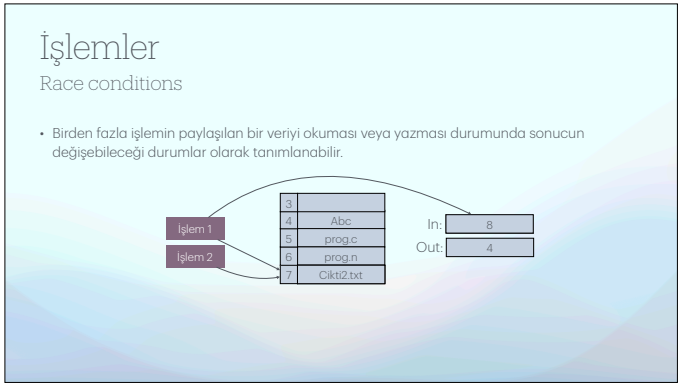


İki farklı problemin ortak çözümü için ise hibrit bir alternatif kullanılabilir. Böyle bir durumda istenirse user, istenirse kernel taraflı bir thread oluşturma işlemi gerçekleştirilebilir. Bu da ihtiyaca göre oluşturulabilecek thread'ler sayesinde esneklik sağlar.

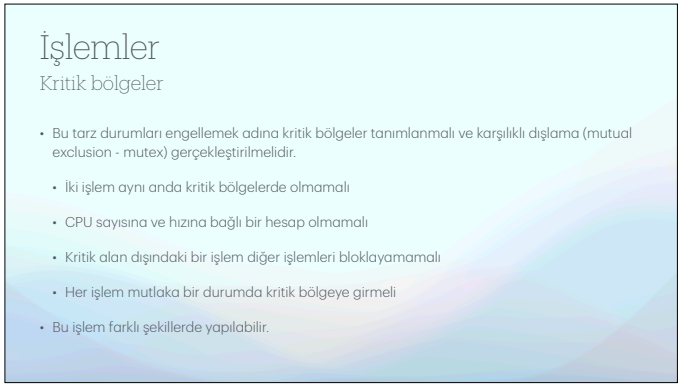
Multithread uygulamalarda bir alana ortak erişim veri kaybına sebep olabilir. Bu sebeple bazı önlemler alınması gerekmektedir.



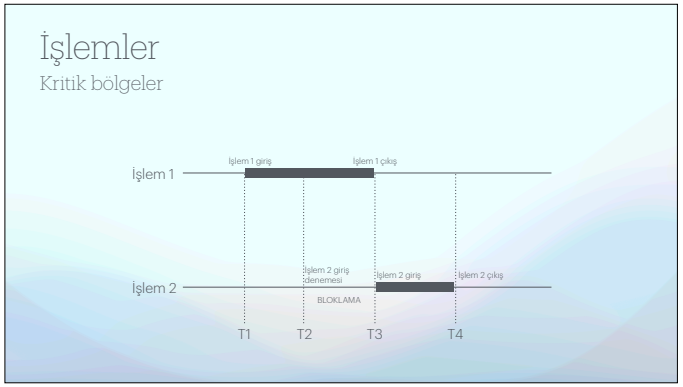
İşlemler aynı zamanda birbirleri ile haberleşme ihtiyacı duyabilmektedirler. Bazı işlemlerin zaman zaman senkronize olması veya birbirlerini beklemesi de gerekebilir. Aynı şekilde bir işlem başka bir işleme veri aktarma ihtiyacı da duyabilir.



Birden fazla işlem aynı anda bir ortak alana yazmaya kalktığında oluşan durumdur. Örneğin işlem1 ortak alana yazma işlemi yaptıktan sonra “sıradaki” değişkenini değiştirir. Yazma işleminden sonra, “sıradaki” değişkeninin değişmesinden önce işlem2 devreye girerek yazma işlemi yapabilir. Bu durumda işlem2 henüz “sıradaki” değişkeni değişmemişken yazma işlemi yapacağından işlem1’in yazdığı alana yazar ve işlem1’in yazdığı veri kaybolur.



Böyle durumların engellenmesi için kritik bölge adı verilen alanlar tanımlanmalıdır. Bu alanların belirli kuralları veri bütünlüğünü sağlamak adına tanımlanmıştır. Kritik alanlara aynı anda erişimin engellenmesi için farklı mekanizmalar kullanılabilmektedir.



Kritik bölgelerde bir işlem bölgede olduğunda diğer işlemin bloklanması gerekmektedir. T1 anında işlem1 kritik bölgeyi alır ve T3 anına kadar kullanmaya devam eder. Bu sırada T2 anında işlem2 kritik bölgeye giriş denemesi yapsa bile bloklanacaktır. İşlem2 en erken T3 anında yani işlem1 çıkış yaptığında devreye girebilmektedir.

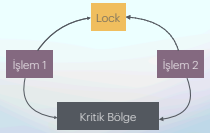
İşlemler
Kesme iptali

- Kesmeleri iptal etmek (?)
  - Bu hakka sahip işlem tekrar kesmeleri açmayabilir
- Kernel belirli işlemler için bunu gerçekleştirebilir
- Çok işlemcili ve çok çekirdekli sistemlerle beraber bu yöntemden uzaklaşılmıştır.

Kritik bölgeye ortak erişimin engellenmesi için kesmeler engellenebilir böyle bir işlem kritik bölgeye girdiğinde kesme yapılamadan sonuna kadar çıkması beklenebilir fakat bu seçimin bazı problemleri bulunmaktadır.
Kesme yapılamayacak işlem başlatıldığında tekrar kesmeleri geri açmayabilir. Bu durumda bir işlem kritik bölgeden çıkmadığında sistemin sadece o işlemde beklemesine sebep olacaktır. Kernel üzerinde yalnızca bazı işlemlere bu imkan verilebilir fakat çok çekirdek ve çok işlemcili sistemlerle beraber bu yöntemin denenmesinden vazgeçilmiştir.

İşlemler
Kilit değişkenleri

- Kilit değişkenleri (Lock variables)
- Yazıcı kuyruğundaki durum yaşanabilir.
- Ortak lock değişkeni aynı anda 1 yapılıp iki thread aynı anda kritik alanda bulunabilir.



Bir işlem kritik bölgeye giriş yapmadan önce bir kilit mekanizması kullanılabilir. İşlem1 kilit değişkenini değiştirdikten sonra kritik bölgeyi alır ve kritik bölgeyi bıraktığında kilit değişkenini de tekrar geri bırakabilir. Böyle bir durumda eğer iki işlem de aynı anda kilit değişkenine erişirse, bir işlem kilit değişkenini değiştirmeden ikisi de müsait olduğunu görüp kritik bölgeyi alabilir (race condition).

İşlemler
Sıra numaralı kilit değişkenleri

- Bir değişkenin sıra numarası ile değiştirilmesi
- Bir thread diğerinin kritik olmayan alanını beklememeli

```
while(true){                    while(true){
   while(turn!=0){}                 while(turn!=1){}
   critical_region();               critical_region();
   turn = 1;                        turn = 0;
   noncritical_region();            noncritical_region();
}                               }
```

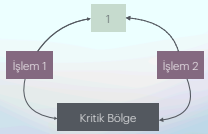
Kilit değişkenleri bir sıra numarası ile değiştirilebilir

İşlemler

Sıra numaralı kilit değişkenleri

```
while(true){
   while(turn!=0){}
   critical_region();
   turn = 1;
   noncritical_region();
}
```

```
while(true){
   while(turn!=1){}
   critical_region();
   turn = 0;
   noncritical_region();
}
```



İşlemler

Peterson'ın çözümü

```
int turn;
int interested[N];
void enter_region(int process){
   int other;
   other = 1 - process;
   interested[process]= true;
   turn = process;
   while(turn == process && interested[other] == true)
}
void leave_region(int process){
   interested[process]=false;
}
```

Birden fazla işlem olduğu durumda kritik bölge ile ilgilenen işlemler bir bayrak dizisinde gerekli alanı atayarak sıra bekleyebilir. Her işlem dizi içerisinde farklı bir alanı değiştirdiği için race condition oluşmaz

---

İşlemler

Peterson'ın çözümü

İşlem 1

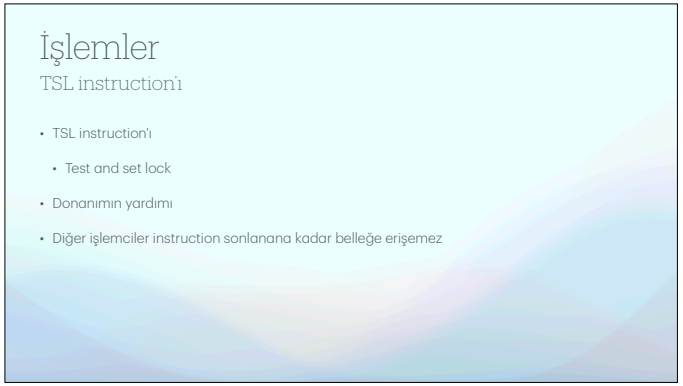
Other = 0 Interested Turn |while|işlem LR

İşlem 2

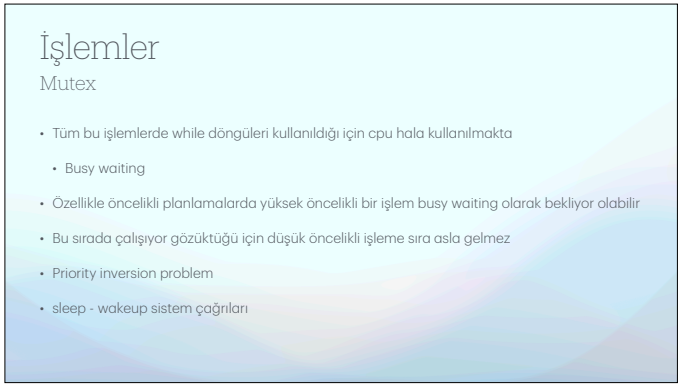
Other = 1 Interested Turn |while. | işlem

Interested
[0] False
[1] True
Turn: 0

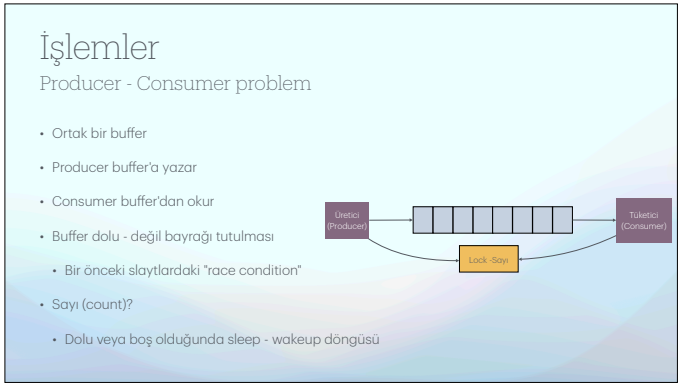
```
int turn;
int interested[N];
void enter_region(int process){
   int other;
   other = 1 - process;
   interested[process]= true;
   turn = process;
   while(turn == process
      && interested[other] == true)
}
void leave_region(int process){
   interested[process]=false;
```



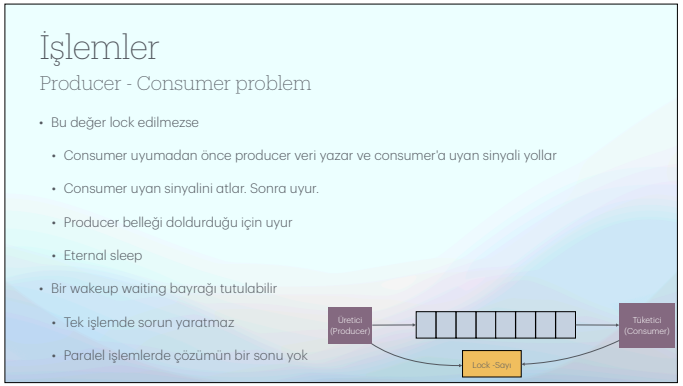
Race condition durumunun engellenmesi donanım taraflı da gerçekleştirilebilir. TSL buyruğu ile bellekteki bir alan diğer işlemcilerin erişimine kapatılabilir.



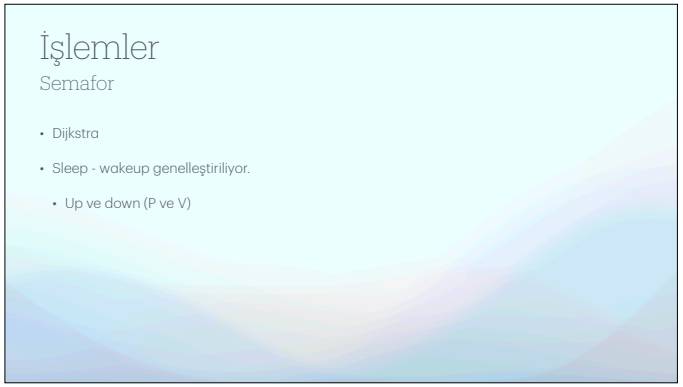
Önceki verilen örneklerde bekleme işlemi bir while döngüsü ile yapılmaktadır. Bu da aslında işletim sistemi için işlemi çalışır durumda göstermektedir. Aslında iş yapmadığı halde iş yapar bir şekilde gözüken işlem “Busy waiting” durumundadır. Özellikle bu işlem yüksek öncelikliyse işletim sistemi işlemi çalışır bir durumda bırakacaktır. Fakat bu işlem bu sıra işlemciyi kullanmıyor durumdadır. Bu durumda önceliklerin belirli durumlarda değiştirilmesi gerekmektedir. Bunun dışında bekleme işlemi bir döngü yerine sleep ve wakeup çağrıları kullanılarak yapılabilir. Böylece beklemeye geçecek olan işlem işletim sistemine çalışıyor gözükmez ve gerekirse yüksek öncelikli olsa bile blocked durumuna geçebilir.



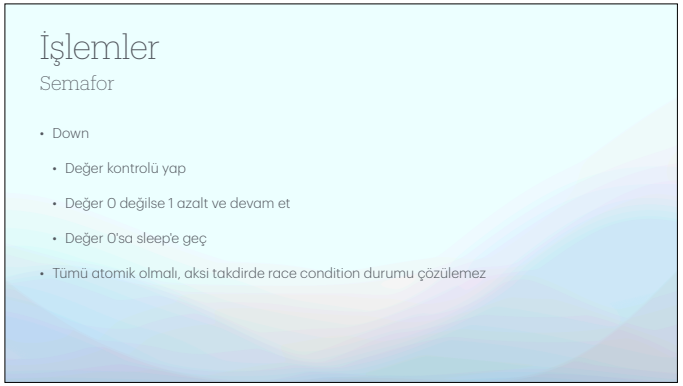
Ortak bir buffer'a yazma ve okuma durumlarında da belirli problemler bulunmaktadır. Bu problemlerden biri üretici ve tüketicinin ortak alanı yönetmesidir.



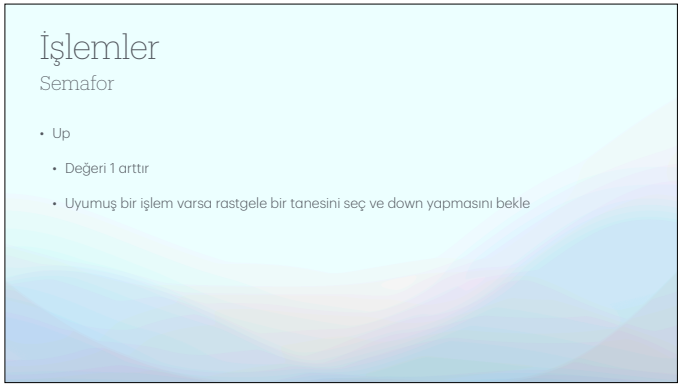
Uyuma uyanma döngüsü ile bu işlem yönetilebilir fakat yine de bir kilit değişkenine ihtiyaç vardır. Aksi takdirde slayt maddelerinde belirtildiği gibi üretici veya tüketici ilk race condition durumuna düştüğü andan itibaren buffer'ın dolması ya da boşalması sebebi ile iki işlem de uykuya geçer. Bu durumda iki işlemi de uyandıracak bir işlem bulunmadığı için iki işlem de uyku durumundan kurtulamaz. Bunun çözülmesi için uyanma bekleyen işlem var mı gibi bir bayrak tutulabilir fakat n paralel işlem için n bayrak tutulması gerekir.



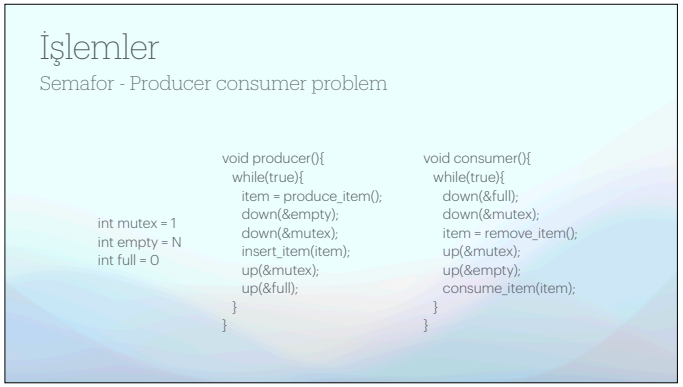
Dijkstra'nın önerisi ile sleep wakeup genelleştirilerek "up" ve "down" olarak iki işleme çevrilmiştir. Bu iki işlemde yapılması gereken adımlar detaylıca tanımlanmış ve bu tanımlara uyulduğunda sistemin "eternal sleep" problemi yaşatmayacağı savunulmuştur.



Down durumunda üç farklı işlem yapılmalıdır. Ayrıca bu işlemlerin biri esnasında arada başka bir up veya down işlemi gerçekleştirilememeli, sistem bu 3 işlem bitene kadar beklemelidir.



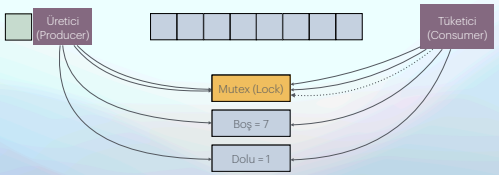
Up durumunda ise iki farklı işlem bulunmaktadır.



Üretici-Tüketici problemi için semafor çözümü

İşlemler

Semafor - Producer consumer problem



İşlemler

Readers and Writers Problem

- Veritabanı erişimi
- Paralel okuma konusunda bir problem yok
- Herkesin okuduğu veriyi bir işlem güncellemek istiyorsa?
  - Diğer tüm okuma ve yazma işlemlerinin beklemesi gerekir
- Nasıl?

Bir diğer çözülmesi gereken problem ise readers and writers problem'dir. Veritabanı erişiminde de karşılaşılabilecek bu problemde ortak alandan okuma esnasında bir problem yaşanmamaktadır fakat okuma yapılan değişken güncellenmek istediğinde ve bu sırada bu değişkene okuma isteği atıldıysa tüm okuma işlemlerinin bitmesinin beklenmesi gerekir.

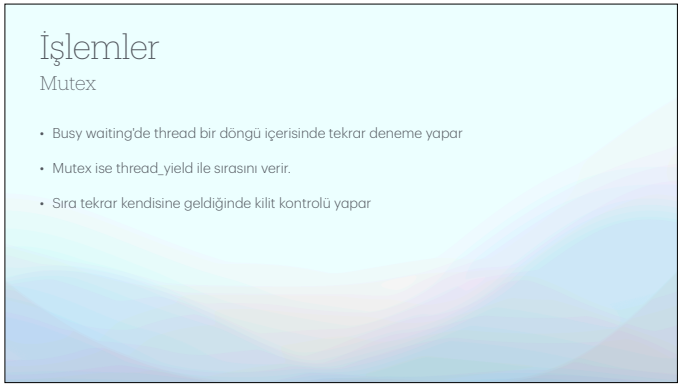
İşlemler

Readers and Writers Problem

- Reader okuduğu sürece writer bekler
- Hep reader gelirse writer'a sıra gelmeyecek
- Writer'a öncelik vermek?

Fakat böyle bir durumda okuma işlemi bitmeyeceği için yazmaya asla sıra gelmeyecektir. Bunun yerine yazmaya öncelik verilebilir. Tabi böyle bir durumda yazma işlemi kendisinden önce çağırılan okuma işlemlerini beklemeli fakat kendisinden sonra çağırılan okuma işlemlerinin önüne geçmelidir.

Mutex kilitli ve değil durumunda bulunan basit kilit mekanizmasıdır. Basit bir mekanizma olduğu için kullanıcı alanında geliştirilebilir. Ayrıca sleep mekanizmasının kullanılması ile Busy waiting durumundan farklı şekilde de çalıştırılabilir.



Sleep için thread_yield ile sıranın verilmesi gerçekleştirilebilir. Böylece mevcut thread sırasını başka bir thread'e verir ve tekrar sıra kendisine geldiğinde Lock kontrolünü yapmış olur. Bu sayede sistem busy waiting durumunda beklemez.

İşlemler

Monitörler

- Producer'da down sırası değiştiğinde
- Mutex'i 0 yapar, bloklanır
- Consumer da mutex'i 0 gördüğü için bloklanır
- Deadlock

```
void producer(){
  while(true){
    item = produce_item();
    down(&empty);
    down(&mutex);
    insert_item(item);
    up(&mutex);
    up(&full);
  }
}
void consumer(){
  while(true){
    item = produce_item();
    down(&full);
    down(&mutex);
    item = remove_item();
    up(&mutex);
    up(&empty);
    consume_item(item);
  }
}
```

İşlemler

Monitörler

- Daha üst seviye
- Derleyici kontrolü üstlenir
- Sadece bir işlem aynı anda monitöre erişebilir
- Derleyici ve yazılmış uygulama kontrolüne bırakıldığı için kullanıcı hatası olmaz
- Java - .Net

İşlemler

Mesaj aktarımı

- İşlemler arası haberleşme
- Gönder ve al değişkenleri (send - receive)
- Sender - receiver
- Herhangi biri yapılacak bir iş yoksa bloklanır

İşlemler

Mesaj aktarımı

- Haberleşme problemleri
  - Özellikle ağ üzerinde
  - Syn - Ack
- Cevap alınamadığı durumlarda tekrarın önlenmesi
- Cevabı verenin kimliğinin doğrulanması
- Performans

İşlemler

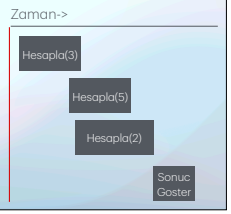
Mesaj aktarımı - Producer Consumer Problem

- Mailbox yaklaşımı
  - Bir mesaj alanı (message buffer)
- Buffer ortadan kaldırılabilir
- MPI'in kullandığı bir sistem

İşlemler

Bariyerler

- Belirli zaman aralıklarında tüm işlemlerin bekleme gerekliliği
- Ortak alan hesaplamalarında kullanılır



İşlemler

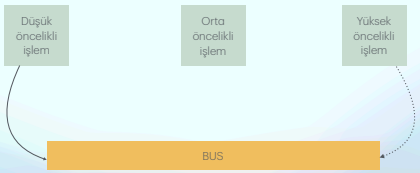
Priority inversion

- NASA - Pathfinder
- Veri gönderiminin durması ve reset gerektirme
- Paylaşılan bellek
- Düşük öncelikli bir işlem bus'ı hava tahmini için kullanır
- Yüksek öncelikli bir işlem bunu okumak için bus'a bakar
- Orta öncelikli haberleşme işlemi ise mutex'i kullanmaz
- Orta öncelikli thread düşük öncelikli thread mutex'i kullanırken devreye girer
- Yüksek öncelikli thread mutex'i almaya çalışır ve başaramadığı için okuma yapamaz

---

İşlemler

Priority inversion



İşlemler

Priority inversion

- Tüm interrupt'ları kapatmak?
- User space'te tehlikeli olduğu söylenmişti
- Priority ceiling
  - Mutex'e ayrı bir öncelik tanımak
  - Mutex'i kullanan işlemler daha öncelikli kabul edilir ve kullanmayanlar işlemler devreye alınmaz
- Priority inheritance
  - Yüksek seviyeli bir thread mutex istediğinde düşük seviyeli teslim eder
  - Mutex kullanmayan thread'ler devreye giremez

İşlemler

Planlama

- Birden fazla işlem hazır durumundaysa
- Planlayıcı (Scheduler)
- Planlama algoritmaları (Scheduling algorithms)
- Toplu işlemden farklı
- CPU'nun tam anlamı ile efektif kullanılabilmesi
- Ayrıca yeni nesilde pil kullanımının ayarlanma işlemi ihtiyacı

İşlemler

Planlama

- İşlem değiştirme maliyetli
- User - kernel modları arasında değişim maliyetli
- Belleğin durumu zaman zaman dinamik hesaplanmak zorunda kalabilir
- Bu durumda her değişimde yeniden bellek değeri hesaplamaları maliyetli
- Bir işlemin tüm içeriği ve gerekirse bellek içeriği tekrar okunup yazılmalı
  - Context-switching

İşlemler

İşlem davranışı

- CPU-bound
- I/O bound
- Kesintisiz (Non-preemptive)
- Kesintili (Preemptive)

İşlemler

Planlama algoritmaları

- Her sistemde
  - Eşit kullanım
  - Zorunlu bazı kuralların mutlaka uygulanması
  - Sistem kaynak kullanımında denge
- Toplu (Batch)
  - İş/saat oranını maksimize etme
  - Bir işin başlama ve bitiş zamanını minimize etme
  - CPU'yu her daim meşgul tutma
-
- İnteraktif (Interactive)
  - Cevap süresini dengeli hale getirme
  - Kullanıcı beklentilerini karşılama
- Gerçek zamanlı (Real-time)
  - Veri kaybını önleme ve süre kısıtları kontrolü
  - Tahmin edilebilirlik

İşlemler

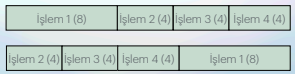
Planlama algoritmaları - First Come, First Served

- İlk gelen ilk devreye alınır
- Veri yapılarında benzer uygulamalar
- Gelen işlem I/O bound olduğunda cpu bekler, planlama yok

İşlemler

Planlama algoritmaları - Shortest Job First

- Kesintisiz
- En kısa iş öne alınır
- En temelde tüm işlemler hazır durumda olduğunda ideal
- İşlem bekleme süreleri devreye girdiğinde dezavantaj



İşlemler

Planlama algoritmaları - Shortest Remaining Time Next

- Kesintili
- Bir önceki yöntemin aynısı
- Her kesintide işlemlerin kalan sürelerine bakılır
- Bunun elde edilmesi için kalan çalışma zamanının bilinmesi gerekir

---

İşlemler

Planlama algoritmaları - Round Robin

- Zaman aralıkları - quantum
- Her işleme bir quantum verilmesi
- Quantum sonuna gelindiğinde işlem hala çalışıyorsa sıradaki işleme geçiş
- Quantum sonuna gelmeden işlem bitiyorsa yine sıradaki işleme geçiş
- Quantum'unun sonuna gelen işlem liste sonuna gider

İşlemler

Planlama algoritmaları - Round Robin

- Örnek

İşlemler

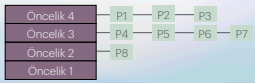
Planlama algoritmaları - Round Robin

- Quantum çok kısa olursa değişim maliyeti yüksek olabilir
- Quantum çok uzun olduğunda yavaş cevap süreleri
- Eğer quantum süresi cpu kullanım süresinden genellikle uzun olursa
  - IO için beklerken quantum süresi dolduran işlemlerin sayısı artar
  - Zaman kaybı
- 20-50 ms quantum süreleri ideal olarak elde edilmiştir (Sisteme göre değişiklik göstermekle beraber)

İşlemler

Planlama algoritmaları - Öncelikli planlama

- Bazı kritik işlemlerin öncelikleri olması gerekir
- Round-robin algoritmasında her işlem eşit önceliktedir
- Öncelik değişimi
  - UNIX - nice komutu
- Farklı öncelik kuyrukları tanımlanıp her kuyruk için round robin uygulanması



İşlemler

Planlama algoritmaları - Garantili planlama

- N kullanıcı için her kullanıcıya işlemcinin 1/N süresinin ayrılması istenir
- Her kullanıcı için de M işleme sahip olduğu durumda sahip olduğu zamanın 1/M kadarlık kısmı işlemlere ayrılmak istenir
- Tüm işlemlerin ve her işlemin çalışma zamanı bilindiğinden bunu hesaplamak mümkündür
- Linux - CFS

İşlemler

Planlama algoritmaları - Loto planlaması

- Her kaynak için bilet oluşturulması
- Belirli sürelerle çekiliş yapılması
- Öncelikli işlemlere daha fazla bilet verilmesi
- Toplam bilete göre kaynağı alma şansının oranı
- Ortak çalışacak işlemlerin bilet paylaşımı
- Belirli bir hesaba göre kesin olarak oran belirlenebiliyorsa hesaplama başarılı bir yöntem
  - Farklı fps'lere sahip videoların işlenmesinde farklı sayıda bilet verilmesi
    - 10 ,20 ,25 fps için 10, 20 ve 25 bilet verilmesi

İşlemler

Planlama algoritmaları - Gerçek zamanlı sistemlerin planlanması

- Hard realtime
- Soft realtime
- İstenen çağrıya belirli sabit zamanda cevap verilmeli
- Sağlık uygulamaları, video-ses senkronizasyonu vb.
- Periyodik çağrılar
- Aperiyodik çağrılar
- Kaynak planlama sistem açılmadan statik ve dinamik olarak yapılabilir
  - Statik durumda girdi çıktı ve istek sayısı bellidir

İşlemler

Planlama algoritmaları - Thread planlanması

- Kullanıcı modu (User mode)
  - Planlama uygulamaya kalır
  - Daha az kaynak tüketimi
  - İşlemlere bağımlı olunduğu için kapsülleme mevcut, durum kaybı yaşanmaz
- Çekirdek modu (Kernel mode)
  - İşlemlerde olduğu gibi yönetim mevcut
  - Seçimi kernel yapar
  - Belirli durumlarda belleğin de veri değişimi ihtiyacı olabilir