İŞLETİM SİSTEMLERİ

İleri İş Sıralama Konuları

Prof.Dr. Ahmet Zengin

Öğrenme Hedefleri

Bu konuyu çalıştıktan sonra:

- İş Parçacığı (thread) Zamanlama
- Çoklu-işlemci Zamanlama
- İşletim Sistemleri Örnekleri
- Algoritma Değerlendirme

Hedefler

- Çeşitli CPU iş sıralama algoritmalarını tanımlamak.
- CPU iş sıralama algoritmalarını değerlendirmek.
- Çok işlemcili ve çok çekirdekli iş sıralama ile ilgili sorunları açıklamak.
- Çeşitli gerçek zamanlı iş sıralama algoritmalarını tanımlamak.
- Windows, Linux ve Solaris işletim sistemlerinde kullanılan iş sıralama algoritmalarını açıklamak.
- CPU iş sıralama algoritmalarını değerlendirmek için modelleme ve simülasyon yönteminden faydalanmak.
- Belirli bir sistem için CPU İş sıralama algoritma seçim kriterlerini değerlendirmek.

İş Parçacığı (Thread) Zamanlama

- Kullanıcı-seviyeli (user-level) ve çekirdek-seviyeli (kernel-level) iş parçacıkları arasında ayrım.
- İş parçacıkları desteği varsa prosesler değil iş parçacıkları sıralanır.
- Çoktan-teke ve çoktan-çoka modelleri, iş parçacığı kütüphanesi lightweight process - LWP üzerinde çalışmak için kullanıcı seviyeli iş parçacıklarını sıralar.
  - İş sıralama yarışı proses içinde olduğu için **proses çekişme kapsamı (process-contention scope-PCS)** olarak bilinir.
  - Programcı tarafından öncelik kümesi yardımıyla yapılır.
- Mevcut CPU üzerinde sıralanan çekirdek iş parçacığı **sistem çekişme kapsamı (system-contention scope-SCS)** dır– sistemdeki tüm iş parçacıkları arasında yarış.

Pthread İş Zamanlama

- Pthread API'si, iş parçacığı oluşturulurken, zamanlamanın process-contention scope (PCS) veya system-contention scope (SCS) olarak ayarlanmasına izin verir.
  - PTHREAD_SCOPE_PROCESS, iş parçacıklarını PCS ile zamanlar.
  - PTHREAD_SCOPE_SYSTEM SCS ise iş parçacıklarını SCS ile zamanlar.

  İşletim Sistemi tarafından sınırlandırılabilir.

  Linux ve Mac OS X sadece PTHREAD_SCOPE_SYSTEM'i kullanır.

Pthread Zamanlama API

```c
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

/* Each thread will begin control in this function */
void *runner(void *param){
   /* do some work ... */
   printf("Thread %d works..\n", pthread_self());
   pthread_exit(0);
}

int main(int argc, char *argv[]) {
   int i, scope;
   pthread_t tid[NUM_THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
```

```c
/* first inquire on the current scope */
   if (pthread_attr_getscope(&attr, &scope) != 0)
      fprintf(stderr, "Unable to get scheduling scope\n");
   else {
      if (scope == PTHREAD_SCOPE_PROCESS)
         printf("PTHREAD_SCOPE_PROCESS \n");
      else if (scope == PTHREAD_SCOPE_SYSTEM)
         printf("PTHREAD_SCOPE_SYSTEM \n");
      else
         fprintf(stderr, "Illegal scope value.\n");
   }
   /* set the scheduling algorithm to PCS or SCS */
   pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
   /* create the threads */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_create(&tid[i],&attr,runner,NULL);
   /* now join on each thread */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_join(tid[i], NULL);
}
```

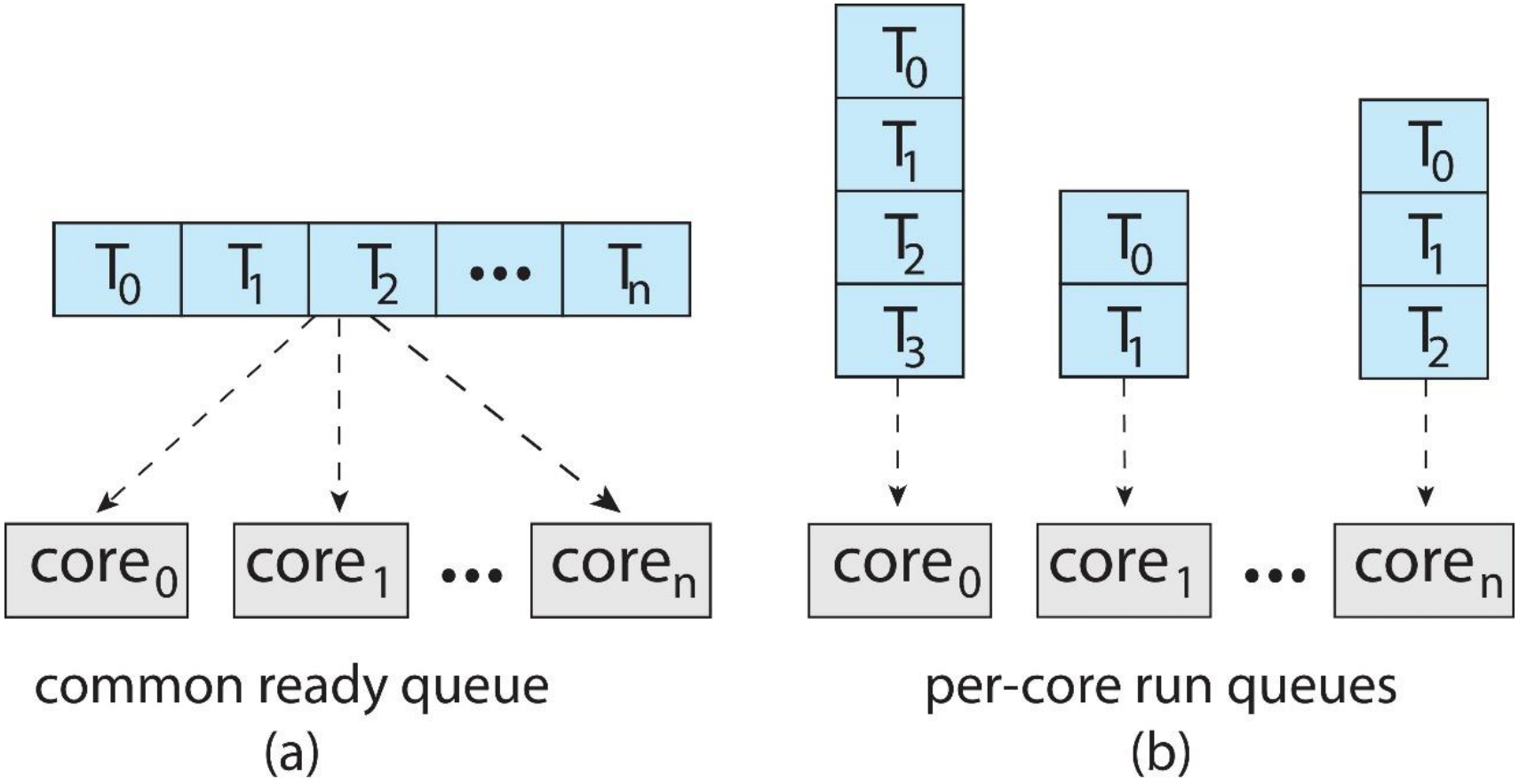
Çok İşlemcili İş Zamanlama

- Birden fazla CPU varsa iş sıralama daha karmaşık olur.

- Çok işlemci aşağıdaki mimarilerden biridir:
  - Çok çekirdekli CPUlar
  - Çoklu İş Parçacıklı çekirdekler
  - NUMA sistemleri
  - Heterojen çok işlemcili sistemler

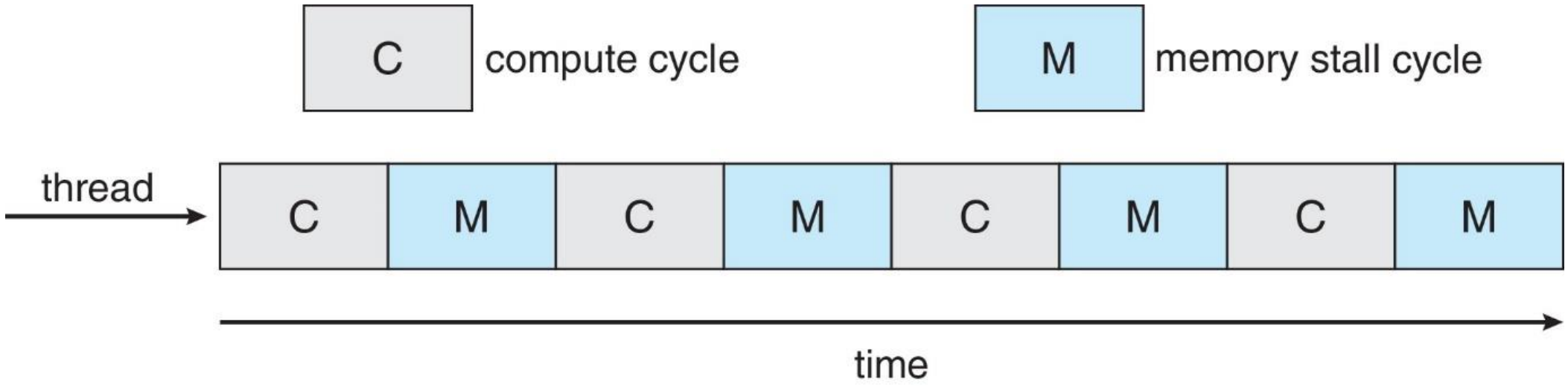
Çok İşlemcili Zamanlama

- Simetrik çoklu işlemci (SMP), her işlemcinin kendi kendini sıraladığı bir sistemdir.
- Tüm iş parçacıkları ortak bir hazır kuyruğunda olabilir (a)
- Her işlemcinin kendi özel iş parçacığı kuyruğu olabilir (b)



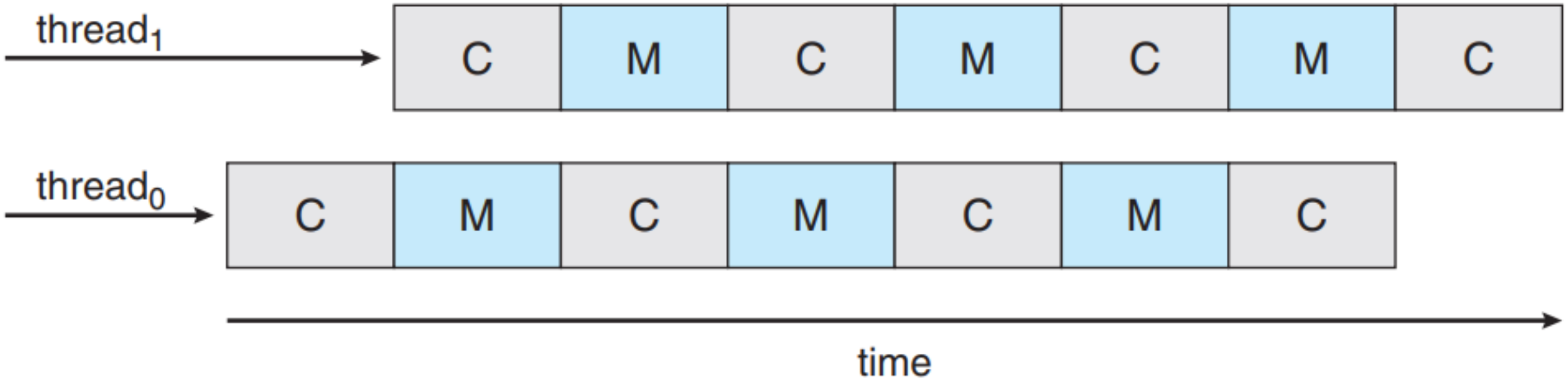
Çok Çekirdekli İşlemciler

- Son eğilim birden fazla işlemci çekirdeğini tek bir fiziksel çip üzerine yerleştirmektir.
- Daha hızlı ve daha az güç harcar.
- Çekirdek başına birden fazla iş parçacığı ortaya çıkıyor
    - Bellekten veri alınırken bir başka iş parçacığı üzerinde devam etmek için ara verilmesinin avantajını kullanır .



Çoklu İş Parçacıklı Çok Çekirdekli Sistem

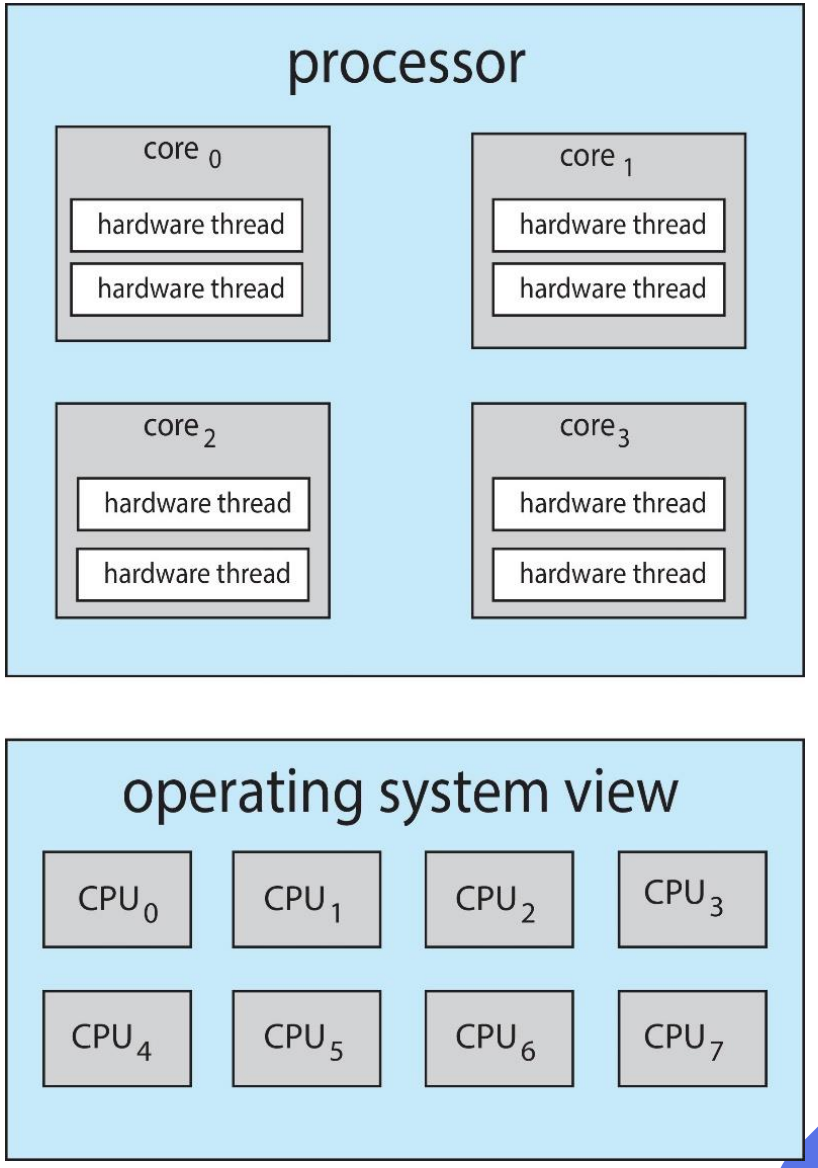
- Her çekirdek > 1 donanım iş parçacığına sahiptir.
- Bir iş parçacığının bellek okuma işlemi varsa, başka bir iş parçacığına geçin!



Çoklu İş Parçacıklı Çok Çekirdekli Sistem

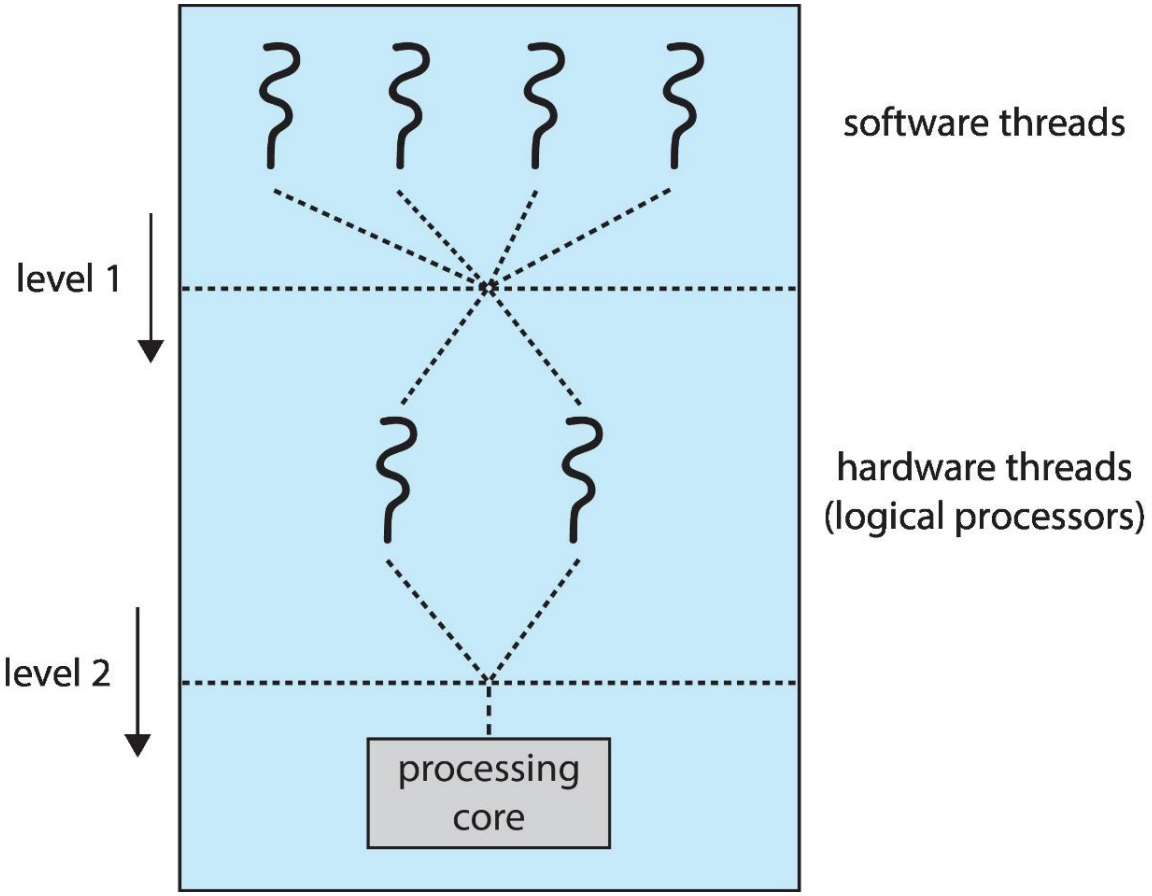
- Çip çoklu iş parçacığı (CMT), her çekirdeğe birden çok donanım iş parçacığı atar. (Intel **hyperthreading** olarak adlandırır)

- Çekirdek başına 2 donanım iş parçacığına sahip dört çekirdekli bir sistemde, işletim sistemi 8 mantıksal işlemci görür.



Çoklu İş Parçacıklı Çok Çekirdekli Sistem

- İki sıralama düzeyi:
  1. Işletim sistemi mantıksal CPU'da hangi yazılım iş parçacığının çalıştırılacağına karar verir.
  2. Her çekirdek, fiziksel çekirdekte hangi donanım iş parçacığının çalıştırılacağına karar verir.



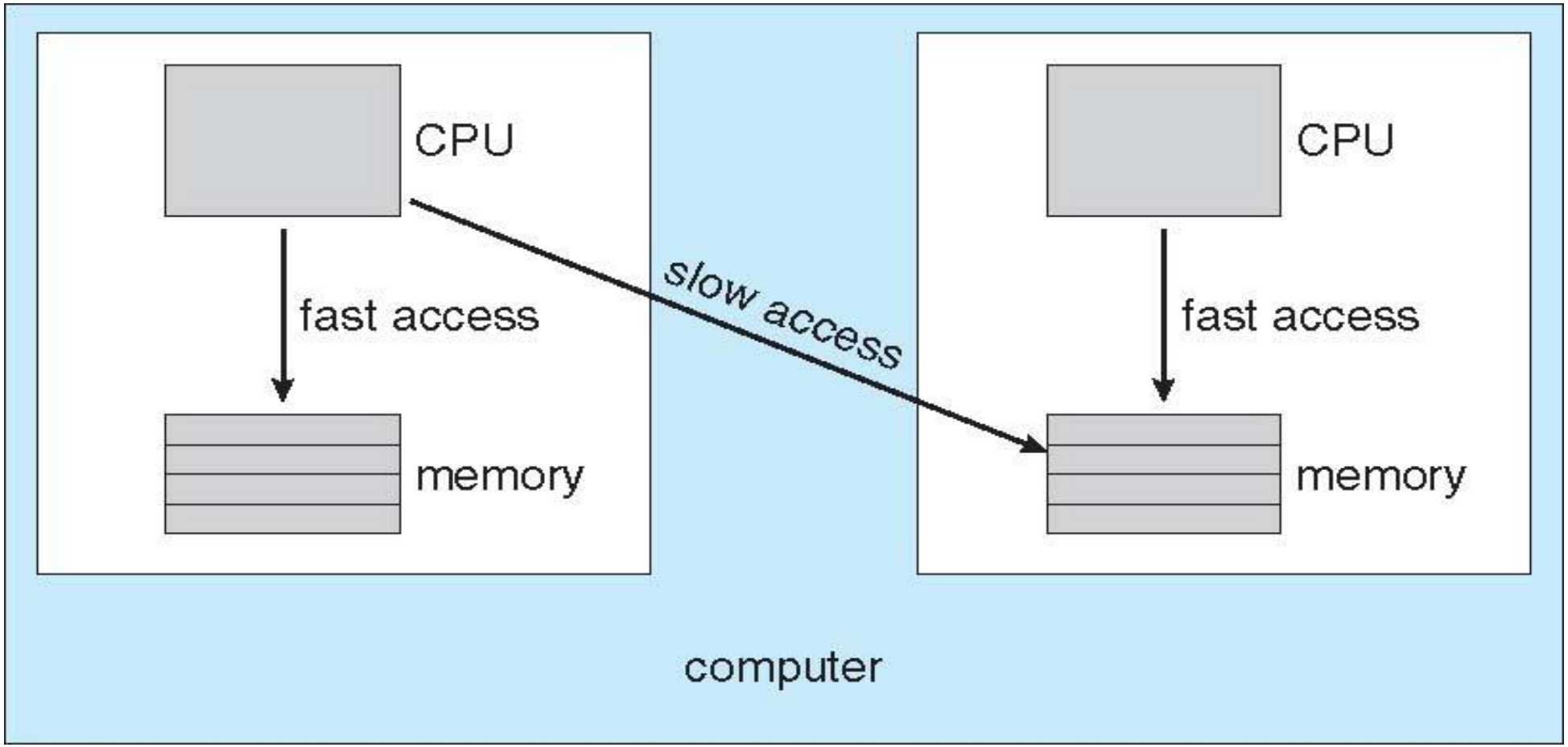
Multiple-Processor Scheduling – Load Balancing

- SMP ise, verimlilik için tüm CPU'ları yüklü tutmanız gerekir Yük dengeleme, iş yükünü eşit şekilde dağıtılmış tutma girişimleri İtme geçişi - periyodik görev her işlemciye yük bindirme sağlar ve bulunursa görevi aşırı yüklenmiş CPU'dan diğer CPU'lara iter Çekme geçişi - boşta kalan işlemciler bekleme görevini meşgul işlemciden çeker.

Multiple-Processor Scheduling – Processor Affinity

- Bir iş parçacığı bir işlemcide çalıştığında, o işlemcinin önbellek içeriği o iş parçacığı tarafından yapılan bellek erişimlerini depolar. Buna bir iş parçacığının bir işlemciye bağlılığı denir (yani, "işlemci bağlılığı").
- Yük dengeleme (Load balancing) , bir iş parçacığının yükleri dengelemek için bir işlemciden başka bir işlemciye taşınabileceğinden, işlemci bağlılığını etkileyebilir; ancak bu durumda iş parçacığı, taşındığı işlemcinin önbelleğindeki içeriği kaybeder.
- Yumuşak bağlılık (Soft affinity) - işletim sistemi bir iş parçacığını aynı işlemcide çalıştırmaya çalışır, ancak garanti vermez.
- Sert bağlılık (Hard affinity) - bir işlemin çalışabileceği bir dizi işlemciyi belirtmesine izin verir.

NUMA ve CPU İş Zamanlama



Bellek yerleşim algoritmalarının da CPU ilişkisini (affinity) göz önünde bulundurduğuna dikkat ediniz

Sanallaştırma ve İş Zamanlama

- Sanallaştırma yazılımı birden fazla konuğu CPU üzerine programlar

- Her bir konuk kendi iş sıralama işlemini yapar
  - CPU'ya sahip olup olmadığını bilmeyerek
  - Düşük bir cevap zamanına neden olabilir
  - Konukların güncel zamanları etkilenebilir

- Konukların iyi iş sıralama algoritması çabalarını telafi edebilir

İşletim Sistemi İş Zamanlama Örnekleri

ORACLE®

Solaris

https://www.oracle.com/solaris/solaris11/

Linux

https://kernel.org/

https://www.microsoft.com/tr-tr/windows

Solaris Zamanlaması

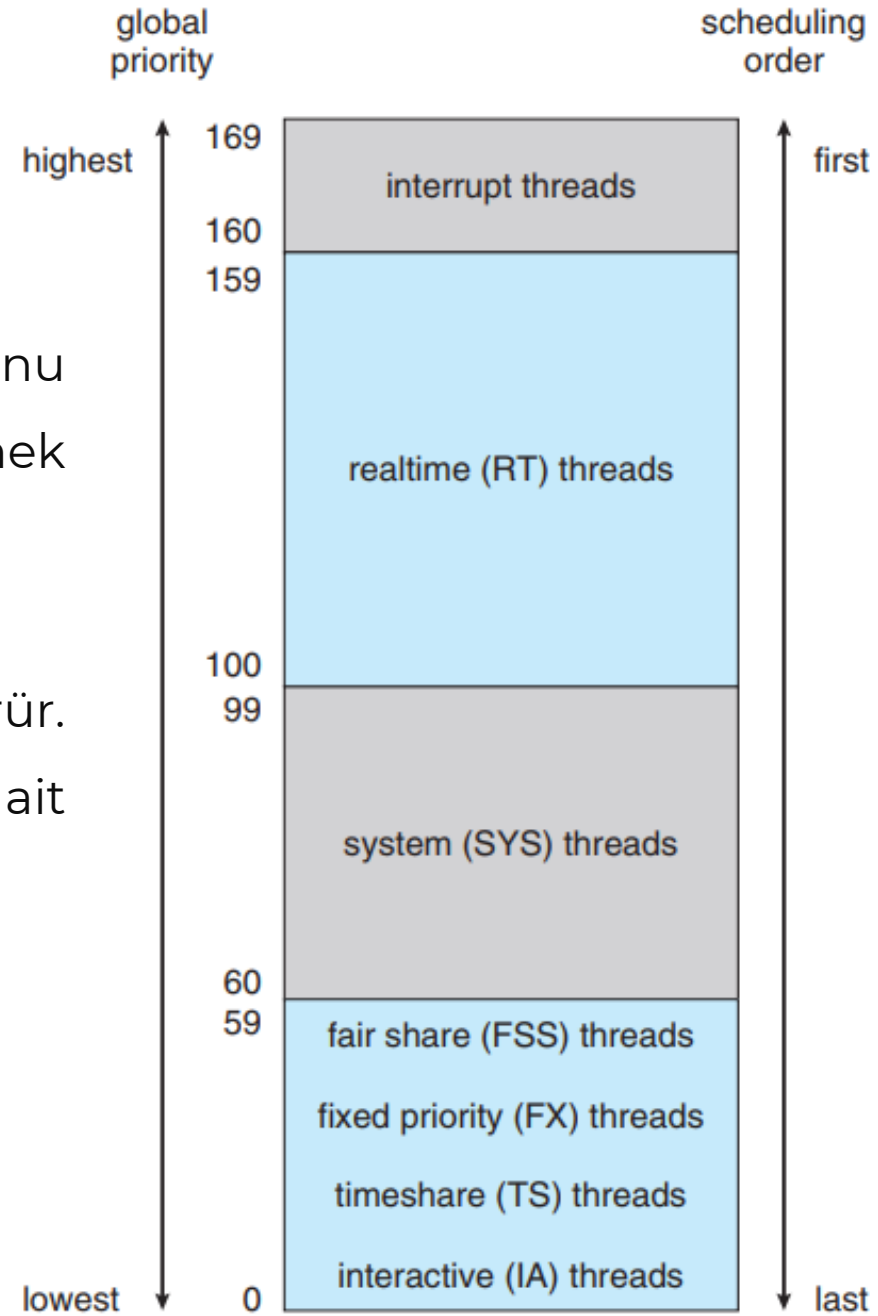
ORACLE®
Solaris

- Solaris, öncelik tabanlı zamanlama kullanılmaktadır.
- Her bir thread 6 adet zamanlama sınıfına aittir.
    1. Zaman paylaşımlı – time sharing (TS)
    2. İnteraktif – interactive (IA)
    3. Gerçek zamanlı – real time (RT)
    4. Sistem – system (SYS)
    5. Adil paylaşım – fair share (FSS)
    6. Sabit öncelik – fixed priority (FP)
- Belirli bir iş parçacığı herhangi bir zamanda sadece bir sınıfta olabilir.
- Her bir sınıf kendi iş sıralama yaklaşımına sahiptir.
- Zaman paylaşımı çok seviyeli geri beslemeli kuyruktur.
    - Sistem yöneticisi tarafından yüklenebilir tablo konfigüre edilebilir.

Solaris İş Sıralama

Altı zamanlama sınıfının birbirleriyle nasıl ilişkili olduğunu ve nasıl global önceliklere eşlendiğini gösteren bir örnek şekilde gösterilmektedir.

Burada, kernel kesmeleri için on adet iş parçacığı sürdürür. Bu iş parçacıkları herhangi bir zamanlama sınıfına ait değildir ve en yüksek öncelikte (160–169) çalışır.



Solaris Görevlendirme Tablosu

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Öncelik: Zaman paylaşımı ve etkileşimli sınıflar için sınıf bağımlı öncelik. Daha yüksek bir sayı, daha yüksek bir önceliği gösterir.

Zaman kuantum: Öncelikler ile zaman kuantumları arasındaki ters ilişkiyi gösterir: En düşük öncelik (öncelik 0), en yüksek zaman kuantumuna (200 milisaniye) sahiptir ve en yüksek öncelik (öncelik 59), en düşük zaman kuantumuna (20 milisaniye) sahiptir.

Zaman kuantum süresi doldu: Zaman kuantumu süresi doldu durumu, bir iş parçacığının CPU'da çalışırken kendisine ayrılan zaman dilimini tükettiği anlamına gelir. İş parçacığı CPU'da çalışırken belirli bir zaman dilimine (zaman kuantumu) sahiptir. Eğer bu zaman dilimi dolarsa, işletim sistemi bu iş parçacığını durdurur ve başka bir iş parçacığına CPU'ya erişim hakkı verir.

Uykudan dönüş: Giriş/çıkış işlemlerini beklemek gibi bir süreden sonra uykudan dönen bir iş parçasının önceliği. Tablo, bekleyen bir iş parçası için giriş/çıkış işlemi mevcut olduğunda, önceliğinin 50 ile 59 arasına yükseltildiğini gösterir. Bu, etkileşimli işlemler için iyi yanıt süresi sağlama planlama politikasını destekler.

To see the full dispatch table on a Solaris system or VM, run **dispadmin -c TS -g**

Windows İş Sıralama

- Windows, öncelik tabanlı, kesme yetenekli bir zamanlama algoritması kullanarak iş parçacıklarını zamanlar.
- Windows zamanlayıcısı, her zaman en yüksek önceliğe sahip iş parçacığının çalışacağını garanti eder.
- Zamanlama ile ilgilenen Windows çekirdeğinin bölümüne "dağıtıcı" (dispatcher) denir.
- Dağıtıcı tarafından çalıştırılmak üzere seçilen bir iş parçacığı, daha yüksek bir öncelikli bir iş parçacığı tarafından kesilene, sona erene, zaman kotası sona erene veya G/Ç gibi bir engelleyici sistem çağrısına (I/O için) çağrılana kadar çalışır.
- Daha yüksek bir öncelikli gerçek zamanlı bir iş parçacığı, daha düşük bir öncelikli iş parçacığı çalışırken hazır duruma geldiğinde, daha düşük öncelikli iş parçacığı kesilir. Bu kesme, gerçek zamanlı bir iş parçacığının, ihtiyaç duyduğunda CPU'ya öncelikli erişim sağlar.

Windows İş Sıralama

- Dağıtıcı, iş parçacığı yürütme sırasını belirlemek için 32 seviyeli bir öncelik düzeni kullanır.
- Öncelikler iki sınıfa ayrılır.
  - Değişken sınıf, 1 ila 15 arasında önceliğe sahip iş parçacıklarını içerir
  - gerçek zamanlı sınıf, 16 ila 31 arasında önceliğe sahip iş parçacıklarını içerir.
  - (Ayrıca bellek yönetimi için kullanılan 0 önceliğinde bir iş parçacığı da bulunur.)
- Dağıtıcı, her zamanlama önceliği için bir kuyruk kullanır ve hazır durumda bir iş parçacığı bulana kadar en yüksekten en düşüğe doğru kuyrukların kümesini geçer.
- Hazır durumda iş parçacığı bulunamazsa, dağıtıcı özel bir iş parçacığı olan boşta iş parçacığını çalıştırır.

Windows Öncelik Sınıfları

- Windows çekirdeğinin sayısal öncelikleri ile Windows API arasında bir ilişki vardır. Windows API, bir işlemin aşağıdaki altı öncelik sınıfından birine ait olabileceğini tanımlar:
  - IDLE_PRIORITY_CLASS
  - BELOW_NORMAL_PRIORITY_CLASS
  - NORMAL_PRIORITY_CLASS
  - ABOVE_NORMAL_PRIORITY_CLASS
  - HIGH_PRIORITY_CLASS
  - REALTIME_PRIORITY_CLASS
- Öncelik sınıfı ve bağıl öncelik nümerik önceliği oluşturur.
- Taban önceliği NORMAL dir

Windows Öncelik Sınıfları

- Belli bir öncelik sınıfındaki iş parçacığı (thread) aynı zamanda bu sınıf içerisinde tanımlı bir bağıl önceliğe sahip Bağıl öncelik değerleri
  - TIME_CRITICAL
  - HIGHEST
  - ABOVE_NORMAL
  - NORMAL
  - BELOW_NORMAL
  - LOWEST
  - IDLE

Windows Thread Öncelik Tablosu

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Linux İş Sıralama

- Sabit dereceli *O*(1) iş sıralama zamanı
- Kesintili öncelik tabanlı
- İki adet öncelik aralığı: zaman paylaşımlı ve gerçek zamanlı
- **Gerçek zamanlı** aralık 0 dan 99 a dır
- Daha yüksek önceliği gösteren nümerik düşük değerlere sahip global öncelik
- Daha yüksek öncelik daha büyük q ya neden olur
- Zaman aralığında kalan zaman süresi kadar görev çalışabilir (**active**)
- Eğer zaman kalmamışsa (**expired**), diğer görevler kendi zaman aralıklarını kullanana kadar çalışamaz
- Tüm çalışabilir görevler CPU başına bir adet çalışır kuyruğunda tutulur.
  - İki adet öncelikli dizi (active, expired)
  - Görevler önceliğe göre sıralanır
  - Aktif görev kalmadığında diziler yer değişir

Linux

Algoritma Değerlendirme

- Bir işletim sistemi için CPU iş sıralama algoritması nasıl seçilir ?

- Kriterleri belirle ve daha sonra algoritmaları değerlendir.

Deterministik modelleme

- Bir tür analitik değerlendirme
- Belirli bir iş yükünü alır ve o iş yükü için her bir algoritmanın performansını hesaplar.
- Örneğin, aşağıda gösterilen iş yüküne sahip olduğumuzu varsayalım. Tüm beş işlem, verilen sırayla, 0 zamanında varır ve CPU patlamasının uzunluğu milisaniye cinsinden verilmiştir.

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

FCFS, SJF ve RR (quantum = 10 milisaniye) zamanlama algoritmalarını düşünelim. Hangi algoritmanın minimum ortalama bekleme süresini sağlayacağına bakalım.

Deterministik modelleme

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

FCFS (First-Come, First-Served): İlk gelen ilk hizmet alır algoritması, işlemleri sıraya koyar ve işlem sırasına göre çalıştırır.

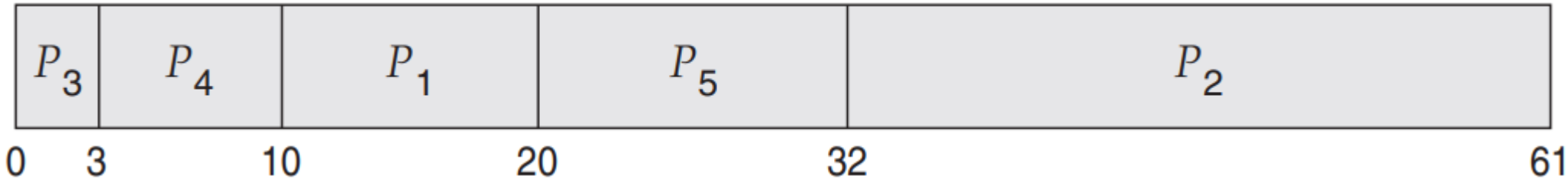
Bekleme süresi, P1 işlemi için 0 milisaniye, P2 işlemi için 10 milisaniye, P3 işlemi için 39 milisaniye, P4 işlemi için 42 milisaniye ve P5 işlemi için 49 milisaniyedir.

Dolayısıyla, ortalama bekleme süresi şu şekildedir: (0 + 10 + 39 + 42 + 49) / 5 = **28 milisaniye**.

Deterministik modelleme

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

SJF (Shortest Job First): En kısa işlem önce algoritması, en kısa CPU patlama süresine sahip işlemi önceliklendirir. Bu durumda, işlemler öncelik sırasına göre sıralandığında ve en kısa süreli işlem en önce çalıştırıldığında minimum bekleme süresi elde edilir.



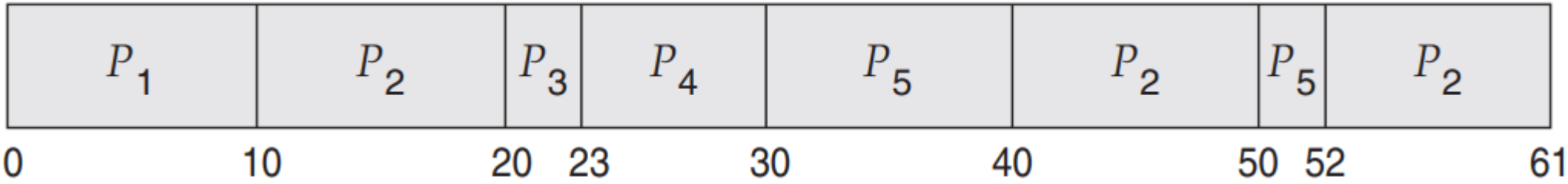
P1 işlemi için 10 milisaniye, P2 işlemi için 32 milisaniye, P3 işlemi için 0 milisaniye, P4 işlemi için 3 milisaniye ve P5 işlemi için 20 milisaniye bekleme süresi vardır.

Dolayısıyla, ortalama bekleme süresi şu şekildedir: (10 + 32 + 0 + 3 + 20) / 5 = **13** milisaniye.

Deterministik modelleme

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

RR (Round Robin): Sabit bir zaman dilimine (quantum) sahip bir zamanlayıcıdır. Her işlem için belirlenen zaman dilimi kadar çalışır, ardından diğer işlemlere geçilir. Eğer bir işlem tamamlanmamışsa, sıradaki işlemlere geçilir ve işlem devam eder.



Bekleme süresi, P1 işlemi için 0 milisaniye, P2 işlemi için 32 milisaniye, P3 işlemi için 20 milisaniye, P4 işlemi için 23 milisaniye ve P5 işlemi için 40 milisaniyedir.

Dolayısıyla, ortalama bekleme süresi şu şekildedir: (0 + 32 + 20 + 23 + 40) / 5 = **23** milisaniye

Deterministik modelleme

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

SJF (Shortest Job First): 13 milisaniye

RR (Round Robin): 23 milisaniye

FCFS (First-Come, First-Served): 28 milisaniye

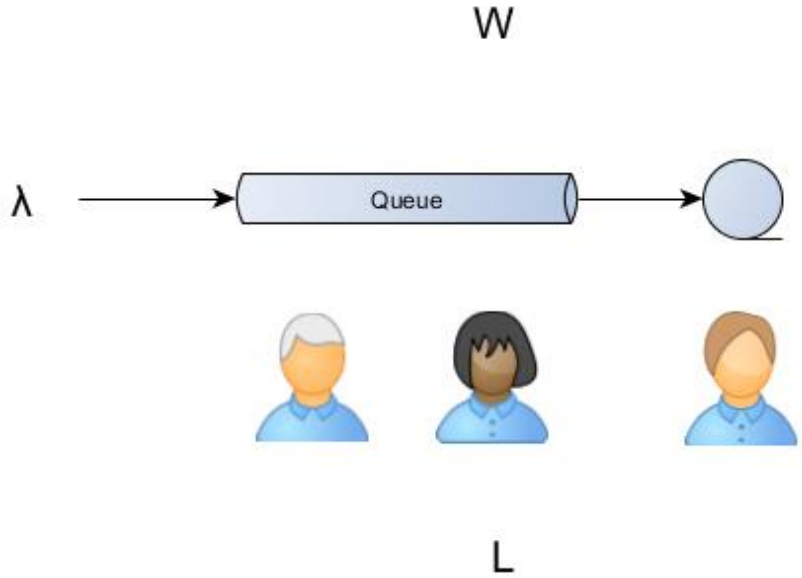
Bu durumda, SJF algoritması ile elde edilen ortalama bekleme süresinin FCFS zamanlama ile elde edilen ortalama bekleme süresinin yarısından daha az olduğunu görebiliyoruz; RR algoritması ise bize ara bir değer vermektedir.

Deterministik (belirleyici) modelleme basit ve hızlıdır. Bize kesin sayılar verir ve algoritmaları karşılaştırmamıza olanak tanır. Ancak, giriş için kesin sayılara ihtiyaç duyar ve cevapları yalnızca bu durumlara uygulanır. Belirleyici modellemenin başlıca kullanımları, zamanlama algoritmalarını tanımlamak ve örnekler sağlamaktır. Aynı programı sürekli olarak çalıştırdığımız ve programın işlem gereksinimlerini tam olarak ölçebildiğimiz durumlarda, belirleyici modellemeyi bir zamanlama algoritması seçmek için kullanabiliriz.

Kuyruk Modelleri

- Proseslerin varışını, CPU ve I/O patlamalarını olasılıksal olarak tanımlar.
  - Genelde üstel ve ortalama ile tanımlanır.
  - Ortalama çıkış (throughput), kullanım oranı, bekleme zamanını hesaplar.
- Bilgisayar sistemi, bir dizi sunucunun ağı olarak tanımlanır. Her sunucunun bekleme sürecinin bir kuyruğu vardır. CPU, hazır kuyruğu olan bir sunucudur, I/O sistemi ise cihaz kuyruklarıyla bir sunucudur.
- Geliş oranlarını ve hizmet oranlarını bildiğimizde, kullanımı, ortalama kuyruk uzunluğunu, ortalama bekleme süresini vb. hesaplayabiliriz. Bu çalışma alanı kuyruk ağı analizi olarak adlandırılır.

Little Formülü



- *L* = ortalama kuyruk boyutu
- *W* = kuyruktaki ortalama bekleme zamanı
- *λ* = ortalama kuyruğa varış oranı
- Little'ın kuralı – kararlı durumda, kuyruğu terk eden prosesler kuyruğa varanlarla eşit olmalıdır, bu nedenle
- ***L = λ x W***
  - Herhangi bir iş sıralama algoritması ve geliş dağılımı için geçerlidir.
- Mesela, eğer saniyede ortalama 7 proses varıyorsa ve normalde kuyrukta 14 proses varsa, prosesler için ortalama bekleme zamanı = 2 saniye olur.
- *λ=7, L=14, W=2*

Simülasyonlar

- Zamanlama algoritmalarının daha doğru bir değerlendirmesini elde etmek için simulasyonlar kullanabiliriz.
- Simulasyonları çalıştırmak, bilgisayar sisteminin bir modelini programlamayı içerir.
- Yazılım veri yapıları, sistemin ana bileşenlerini temsil eder.
- Simülatörde bir saatı temsil eden bir değişken bulunur. Bu değişkenin değeri arttıkça, simülatör, cihazların, işlemlerin ve zamanlayıcının etkinliklerini yansıtmak için sistem durumunu değiştirir.
- Simülasyon yürütüldükçe, algoritma performansını gösteren istatistikler toplanır ve yazdırılır.

CPU İş Sıralayıcıların Simülasyon ile Değerlendirilmesi