İşletim Sistemleri
CPU Scheduling

Dr. Öğr. Üyesi Ertan Bütün

Bu dersin içeriği hazırlanırken Operating System Concepts (Silberschatz, Galvin and Gagne) kitabı ve
Prof. Dr. M. Ali Akcayol'un (Gazi Üniversitesi Bilgisayar Mühendisliği Bölümü) ders sunumlarından faydalanılmıştır.

https://codex.cs.yale.edu/avi/os-book/OS9/slide-dir/index.html

http://w3.gazi.edu.tr/~akcayol/BMOS.htm

Konular

- Temel kavramlar
- Scheduling kriterleri
- Scheduling algoritmaları
- Thread Scheduling
- Multiple-Processor Scheduling
- Gerçek zamanlı (Real-Time) CPU scheduling

Temel kavramlar

- **İşletim sistemlerinin en önemli yönlerinden biri, birden çok programı çalıştırabilmeleridir.**
- **Multiprogramming** ile process'ler organize edilerek CPU'nun her zaman yürütecek bir process'e sahip olması sağlanır, **Multiprogramming** işletim sistemlerinde birden fazla program aynı anda bellekte tutulur, işletim sistemi bu programlardan birini yürütmek için seçer.
  - böylece hem CPU'nun verimliliği artırılır hem de kullanıcılar *(genellikle birden fazla program çalıştırmak isterler)* memnun edilir.
- Yürütülmekte olan process'in tamamlanması için I/O gibi bazı işlemler gerekebilir. **Multiprogramming** olmayana bir sistemde process'in beklediği işlemler tamamlanana kadar CPU boşta bekler.
- **Multiprogramming** sitemde ise yürütülmek üzere başka bir process seçilir, bu process de tamamlanmak için beklemek zorunda kalırsa yürütülmek üzere başka bir process seçilir.

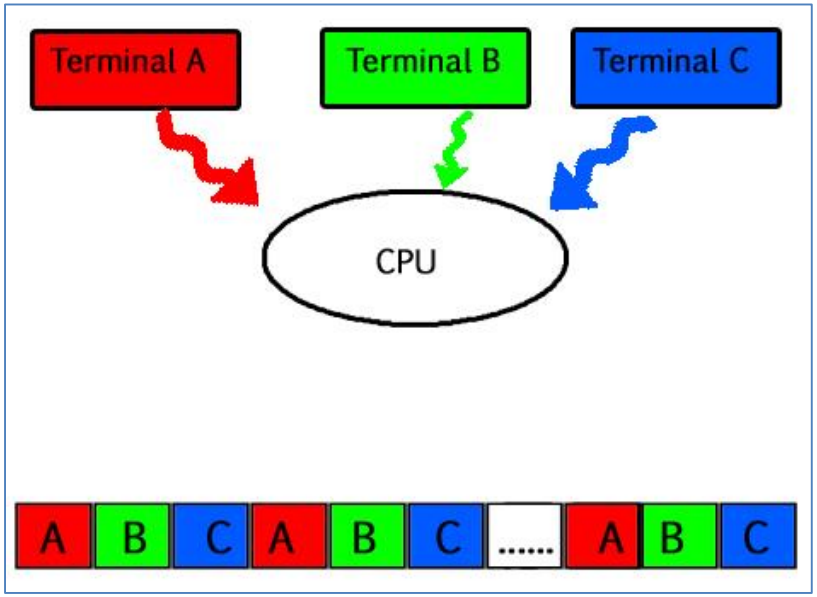
Temel kavramlar

- Process'in beklediği işlemler tamamlandığında process OS tarafından tekrar yürütülür.
- **En az bir process yürütülmek üzere hazır olarak beklediği müddetçe, CPU asla boşta kalmaz.**
- Multicore bir sistemde, yürütülmeye hazır process olduğu müddetçe CPU'yu meşgul tutma kavramı, sistemdeki tüm işlemci çekirdeklerine genişletilir.
- **Multitasking,** multiprogramming'in genişletilmiş halidir. **Multitasking** sistemlerde CPU, aralarında geçiş yaparak birden fazla process'i yürütür, ancak bu geçişler sık sık meydana gelir.
  - Böylece hem CPU verimliliği hem de sistemin kullanıcıya cevap verebilirliliği artmış olur.
- Birden fazla process aynı anda çalışmaya hazırsa, işletim sistemi hangi process'in çalışacağını seçmelidir. Bu karar **CPU Scheduling** ile verilir.

Temel kavramlar

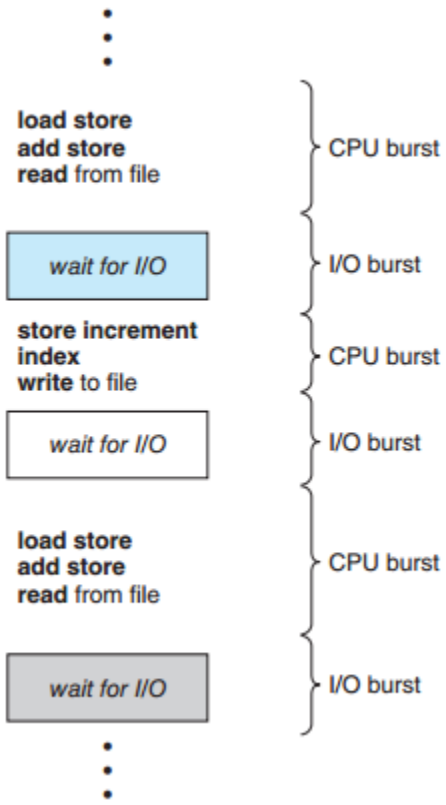
Multiprogramming

Multitasking



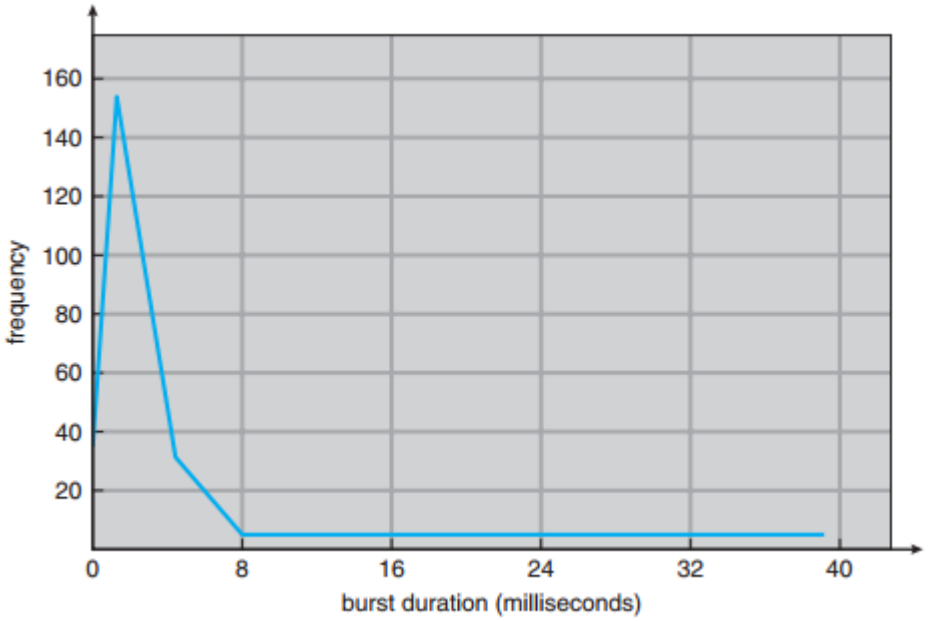
CPU Burst Cycle ve I/O Burst Cycle

- Hemen hemen tüm process'lerin yürütülmesi iki durum arasında dönüp duran bir döngüde şeklindedir: **CPU execution** ve **I/O wait.**
- **CPU burst** : Process'in, bir I/O için beklemeye başlayana veya başka bir process tarafından interrupt edilinceye kadar CPU'yu kullandığı süredir
- **I/O burst:** Process'in CPU zamanına ihtiyaç duymadan önce girdi-çıktı için beklediği zamandır.
- Process yürütme, bir **CPU burst** (CPU patlaması) ile başlar, bunu bir **I/O burst** (I/O patlaması) izler, ardından başka bir **CPU burst**, ardından başka bir **I/O burst** takip eder. En son **CPU burst** ile processin bir sistem talebiyle yürütülmesi sonlandırılır.



CPU Burst Cycle ve I/O Burst Cycle

- CPU burst süresi; process'ten process'e, bilgisayardan bilgisayara çok farklılık gösterse de şekildeki grafiğe benzer bir frekans eğrisine sahip olma eğilimindedirler.
- Eğri genellikle, **çok sayıda kısa** CPU burst ve **az sayıda uzun** CPU burst ile exponential ya da hyperexponential karakterdedir.
- Yani process'lerin CPU üzerinde kaldıkları süreler genellikle çok sayıda kısa aralıklar şeklinde olmaktadır.



CPU Scheduler

- **CPU her boşta kaldığında**, işletim sisteminin ready queue'den bir process'i yürütmek üzere seçmesi gerekir.
- Bu seçme işlemi short-term scheduler veya diğer ismiyle CPU scheduler tarafından gerçekleştirilir.
- Ready queue, ilk gelen ilk çıkar (firs-in-first-out, FIFO) olmak zorunda değildir, priority queue, ağaç, sırasız bağlı liste şeklinde de oluşturulabilir.
- Ready queue hangi şekilde olursa olsun ready queue'de yer alan process'ler CPU'da çalışma şansı elde etmek üzere sıralanırlar.
- Kuyruk içindeki kayıtlarda, **process control block (PCB) tutulur.**

Preemptive and Nonpreemptive Scheduling

- CPU-scheduling kararı 4 durum altında gerçekleştirilir:
  1. Bir process **çalışma (running)** durumundan **bekleme (waiting)** durumuna geçtiğinde (örneğin process I/O isteğinde bulunduğunda),
  2. Bir process **running** durumundan **ready** durumuna geçtiğinde (örneğin kesme olduğunda),
  3. Bir process **waiting** durumundan **ready** durumuna geçtiğinde (örneğin beklenen I/O tamamlandığında),
  4. Bir process **sonlandırıldığında**.
- Eğer scheduling işlemi **1. ve 4. durumlarda gerçekleşmişse**, scheduling şeması nonpreemptive veya cooperative scheduling olur, bu durumda scheduling açısından bir seçim yoktur, hazır kuyruğunda sırada bekleyen process CPU'ya geçer.
  - **Nonpreemptive scheduling'te**, CPU bir process'e tahsis edildiğinde, process sonlanıncaya veya bekleme durumuna geçeneğe kadar process CPU'da kalır.

Preemptive and Nonpreemptive Scheduling

- Eğer scheduling işlemi 2. ve 3. durumlarda gerçekleşmişse scheduling şeması preemptive scheduling olur.
- Preemptive scheduling'de veriler process'ler arasında paylaşıldığında race condition söz konusu olabilir.
- Preemption, işletim sistemi çekirdeğinin tasarımını da etkiler. Çekirdek bir process adına önemli bir işi yürütürken (örneğin I/O kuyrukları gibi önemli çekirdek verilerini değiştirilirken), process bu değişikliklerin ortasında preempted olursa yani process CPU'dan çıkarılıp başka bir process CPU'ya geçerse ve çekirdeğin aynı yapıyı okuması veya değiştirmesi gerekirse kaos başlar.
- Preemptive kernel sistemlerde, paylaşılan çekirdek veri yapılarına erişilirken race condition'ı önlemek için mutex locks gibi mekanizmalar kullanılır.

Preemptive and Nonpreemptive Scheduling

- Nonpreemptive kernel yürütme modeli, görevlerin belirli bir zaman dilimi içinde tamamlaması gerektiği gerçek zamanlı hesaplamalar için zayıf bir modeldir.
- Çünkü nonpreemptive kernel sistem, context switch yapmadan önce bir sistem çağrısının tamamlanması için veya bir processin bloke olması için (I/O için beklemeye geçmesi için) veya sonlanması için bekler.
- Windows, macOS, Linux ve UNIX dahil hemen hemen tüm modern işletim sistemleri, **preemptive scheduling algoritmaları** kullanır.

Dispatcher

- Dispatcher, CPU scheduling işlevini gerçekleştiren bileşendir.
- **Dispatcher, short-term scheduler tarafından seçilen process'in CPU'ya geçmesini sağlayan modüldür.**
- Dispatcher aşağıdaki işlevleri içermektedir:
  - Context swichting
  - Kullanıcı moduna geçiş
  - Programı kaldığı yerden devam ettirmek için kullanıcı programında uygun konuma atlama
- Dispatcher'ın çok hızlı bir şekilde geçiş yapması zorunludur.
- Dispatcher'ın bir process'i durdurup diğer process'i başlatıncaya kadar geçen süreye dispatch latency denir.

Konular

- Temel kavramlar
- Scheduling kriterleri
- Scheduling algoritmaları
- Thread Scheduling
- Multiple-Processor Scheduling
- Gerçek zamanlı (Real-Time) CPU scheduling

Scheduling kriterleri

- CPU scheduling algoritmaları çok sayıda farklı kritere göre karşılaştırılır:
  - CPU utilization: CPU'nun olabildiği kadar kullanımda olması istenir. Kavramsal olarak CPU kullanım oranı %0 - %100 arasındadır. Gerçek sistemlerde bu oran %40 ile %90 arasındadır.
  - Throughput: Belirli bir zaman biriminde yürütülmesi tamamlanan process sayısıdır.
  - Turnaround time: Bir processin arz edilmesinden tamamlanmasına kadar geçen toplam süredir. Bir process'in hazır kuyruğunda bekleme süresi, CPU'da yürütülme süresi ve I/O için geçen sürelerin toplamıdır.
  - Waiting time: Bir process'in hazır kuyruğunda beklediği süredir. CPU scheduling algoritması, bir processin yürütülme süresini ve I/O gerçekleştirme süresini etkilemez. Ancak bir process'in hazır kuyruğunda beklemek için harcadığı süreyi etkiler.
  - Response time: Bir process'e bir istekte bulunulmasından process'ten ilk yanıtın üretilmesine kadar geçen süredir. (cevabın tamamını vermek için geçen süre değil, cevap vermeye ilk başlanılan zamana kadar geçen süredir.)
- CPU utilization'ı ve throughput'u maksimum yapmak; turnaround time, waiting time ve response time'ı minimum yapmak amaçlanır.

Konular

- Temel kavramlar
- Scheduling kriterleri
- Scheduling algoritmaları
- Thread Scheduling
- Multiple-Processor Scheduling
- Gerçek zamanlı (Real-Time) CPU scheduling

Scheduling algoritmaları

- CPU scheduling algoritmaları, **hazır kuyruğunda bekleyen process'lerden hangisinin CPU'ya geçeceğini belirlerler.**
- **Bu bölümde verilen scheduling algoritmaları tek bir işlemci çekirdeğine sahip tek bir CPU için ele alınmıştır, bu nedenle sistem bir seferde yalnızca bir process'i çalıştırabilir.**
  - First-Come, First-Served Scheduling
  - Shortest-Job-First Scheduling
  - Priority Scheduling
  - Round-Robin Scheduling
  - Multilevel Queue Scheduling
  - Multilevel Feedback Queue Scheduling

First-Come, First-Served Scheduling

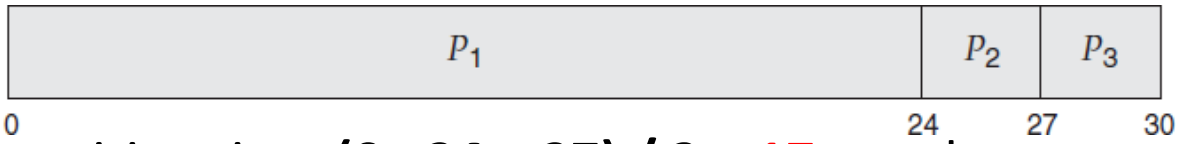
- En basit CPU scheduling algoritmasıdır ve first-come first served (FCFS) şeklinde çalışır.
- CPU'ya ilk istek yapan process, CPU'ya ilk geçen process olur.
- FIFO kuyruk yapısıyla yönetilebilir.
- Bu algoritmada ortalama waiting time genellikle yüksektir.
- **Waiting time process'lerin kuyruğa geliş sırasına göre çok değişmektedir.**

First-Come, First-Served Scheduling

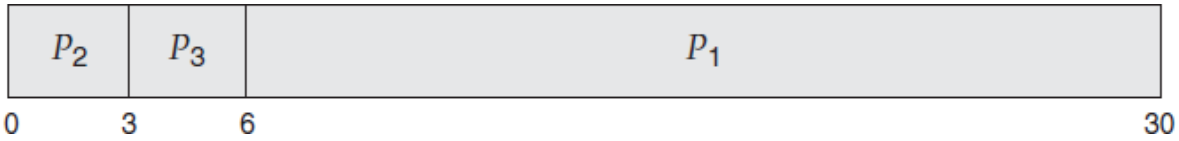
- Aşağıdaki 3 process için CPU'da çalışma süreleri ms olarak verilmiştir.

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Process'ler **P1, P2, P3** sırasıyla gelirse Gantt şeması aşağıdaki gibidir.



- Ortalama waiting time **(0+ 24 + 27) / 3 = 17 ms** olur.
- **P2, P3, P1** sırasıyla gelirse Gantt şeması aşağıdaki gibidir.



- Ortalama waiting time **(0+ 3 + 6) / 3 = 3 ms** olur.

First-Come, First-Served Scheduling

- FCFS algoritmasında, process'lerin çalışma süreleri çok farklıysa ortalama bekleme süreleri çok değişken olur.
- Çok sayıda kısa süreli process'in uzun süreli bir process'in CPU'dan çıkmasını beklemesine convoy effect denilmektedir. Bir tane CPU-bound ve çok sayıda I/O bound process'in olduğu bir durumda CPU'da kısa süreliğine çalıştırılacak I/O bound process'ler CPU-bound processin işini bitirmesini bekler.
- FCFS Scheduling algoritması nonpreemtive'dir, bir process'e CPU tahsis edildiğinde process sonlanana veya I/O isteği yapana kadar CPU'yu elinde tutar.
- FCFS algoritması, her process'in düzenli aralıklarla CPU'da yer almasının önemli olduğu interaktif sistemler için uygun değildir. Bir process'in CPU'yu uzun süre tutmasına izin vermek interaktif sistemin çalışmasını olumsuz etkiler.

Shortest-Job-First Scheduling

- Shortest-Job-First Scheduling (SJF) algoritmasında, CPU'ya bir sonraki **işlem süresi en kısa olan** (shortest-next-CPU-burst) process **atanır.**

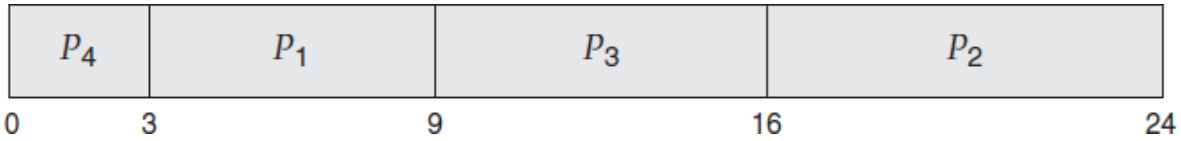
Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- Gant şeması ve ortalama waiting time ?

Shortest-Job-First Scheduling

- Shortest-Job-First Scheduling (SJF) algoritmasında, CPU'ya bir sonraki **işlem süresi en kısa olan** (shortest-next-CPU-burst) process **atanır.**

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3



- Ortalama waiting time, **(0 + 3 + 9 + 16) / 4 = 7 ms'**dir. **FCFS kullanılsaydı 10,25 ms olurdu** ((0 + 6 + 14 + 21) / 4)**.**
- **SJF algoritması minimum ortalama waiting time'yi elde eder.**

Shortest-Job-First Scheduling

- **SJF algoritmasındaki en büyük zorluk, sonraki çalışma süresini (burst time) tahmin etmektir.**
- Long-term (job) scheduling için kullanıcının belirlediği süre alınabilir.
- SJF algoritması genellikle long-term scheduling için kullanılır, short-term scheduling seviyesinde kullanılamaz.
- Short-term scheduling'te CPU'da sonraki çalışma süresini bilmek mümkün değildir, sonraki çalışma süresi tahmin edilmeye çalışılır.
    - Sonraki çalışma süresinin önceki çalışma süresine benzer olacağı beklenir.

Sonraki CPU Burst Süresinin Belirlenmesi

- Sonraki CPU burst süresi, **önceki** CPU burst sürelerinin **üstel ortalama** (exponential average) **değeri ile tahmin edilebilir:**

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n$$

  - t_n ,ise n. CPU burst süresini,
  - τ_{n+1} , tahmin edilecek sonraki CPU burst süresini,
  - α, $0 < \alpha < 1$ aralığında sabit bir değeri gösterir. (α genellikle ½ dir)

- t_n en son bilgiyi içerirken τ_n o ana kadar olan tüm geçmiş bilgiyi saklar.*

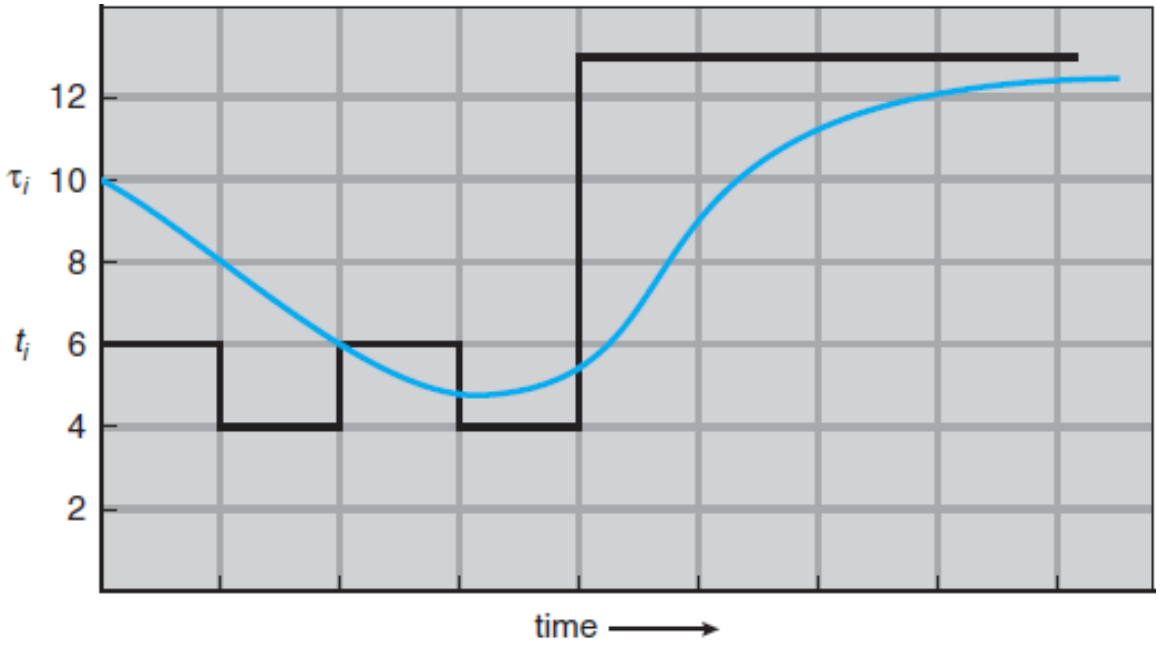
Shortest-Job-First Scheduling

- SJF algoritması preemptive veya nonpreemptive olabilir.
- Processler arasında seçim, önceki bir process devam ederken hazır kuyruğa yeni bir process geldiğinde ortaya çıkar.
- Yeni gelen process'in bir sonraki CPU burst süresi, şu anda yürütülen process'in **geriye kalan süresinden** daha kısa olabilir, bu durumda:
  - preemptive SJF çalışmakta olan process'i askıya alır,
  - nonpreemptive SJF çalışmakta olan process'in sonlanmasına izin verir.
- Preemptive SJF, shortest-remaining-time-first scheduling olarak adlandırılır.

Shortest-Job-First Scheduling

- $\alpha = 1/2$ *ve* $\tau_0 = 10$ için bir ortalama eksponansiyel görülmektedir.

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n$$



CPU burst (t_i)		6	4	6	4	13	13	13 ...
"guess" (τ_i)	10	8	6	6	5	9	11	12 ...

Example of Shortest-remaining-time-first

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

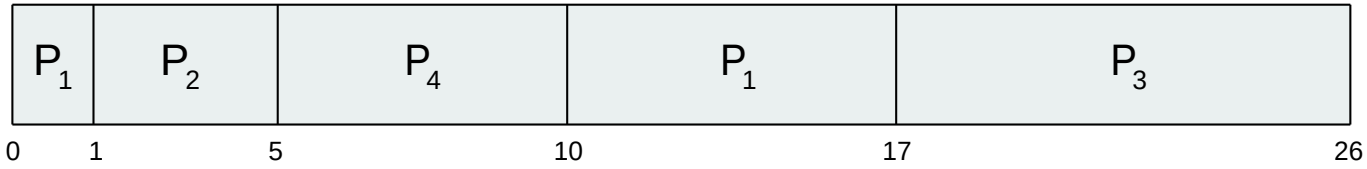
- Process'ler gösterilen zamanlarda hazır kuyruğuna ulaşırsa ve belirtilen CPU burst sürelerine ihtiyaç duyarsa, sonuçta ortaya çıkan **preemtive SJF scheduling** için

  Gantt şeması ve ortalama bekleme süresi ?

Example of Shortest-remaining-time-first

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- Process'ler gösterilen zamanlarda hazır kuyruğuna ulaşırsa ve belirtilen CPU burst sürelerine ihtiyaç duyarsa, sonuçta ortaya çıkan preemtive SJF scheduling için Gantt şeması şöyledir:



Bekleme süresi	P1	P2	P3	P4
	(10-1)	(1-1)	(17-2)	(5-3)
Ortalama bekleme süresi	(10-1)+(1-1)+(17-2)+(5-3)]/4 = 6.5 ms			

Priority Scheduling

- Her process bir öncelikle ilişkilendirilir ve **CPU en yüksek önceliğe sahip process'e tahsis edilir**.
- Eşit önceliğe sahip olanlar ise FCFS sırasıyla atanır.
- Shortest-job-first (SJF) algoritması, aynı zamanda önceliğin (p) bir sonraki CPU burst (tahmin edilen) süresinin tersi olduğu bir priority scheduling algoritmasıdır.
  - SJF algoritmasında CPU burst süresi ne kadar küçükse öncelik o kadar yüksek olur.

Priority Scheduling

- Aşağıda 5 process için öncelik değerlerine göre gantt şeması verilmiştir. (küçük sayı yüksek öncelikli olarak kabul edilmiştir)

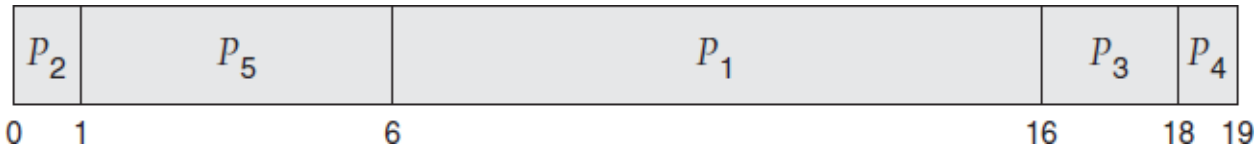
Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

  - Gantt şeması ve ortalama bekleme süresi ?

Priority Scheduling

- Aşağıda 5 process için öncelik değerlerine göre gantt şeması verilmiştir. (küçük sayı yüksek öncelikli olarak kabul edilmiştir)

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



P5 P1 P3 P4

- Ortalama waiting time **(1+ 6 + 16 + 18) / 5 = 8,2 ms** olur.

Priority Scheduling

- Priority scheduling preemptive veya nonpreemptive olabilir.
- Preemptive olunca, çalışmakta olan process'ten daha öncelikli bir process hazır kuyruğuna geldiğinde, çalışmakta olan process preempted olur CPU'dan çıkarılır, yeni gelen yüksek öncelikli process CPU'ya geçer.
- Nonpreemptive olunca, bir process hazır kuyruğuna geldiğinde çalışmakta olan process'ten daha öncelikli bile olsa, çalışmakta olan process durum değiştirene kadar CPU'da işini yürütmeye devam eder.

Priority Scheduling

- Priority scheduling algoritmasında, CPU sürekli yüksek öncelikli process'leri çalıştırabilir ve bazı processler sürekli hazır kuyruğunda bekleyebilir (indefinite blocking, starvation).
- Genellikle iki şeyden biri olur. Ya process en sonunda çalıştırılır ya da bilgisayar sistemi kesintiye uğrayınca veya çökünce tüm yürütülmesi tamamlanmamış düşük öncelikli process'ler kaybedilir.
- Starvation'ı engellemek için bir çözüm aging'dir, düşük öncelikli process'ler kuyrukta beklerken öncelik seviyesi belli zaman aralıklarında artırılır (Örn. her 15 dakikada 1 artırılır), böylece en düşük önceliğe sahip process'in bile belirli bir süre sonunda çalışması sağlanır.

Round-Robin Scheduling

- Round-robin (RR) scheduling, genellikle time-sharing sistemlerde kullanılır, FCFS scheduling algoritmasına benzer ancak RR algoritmasında preemption ile process'ler arasında switch'ing işlemi de yapılır.
- Hazır kuyruğundaki process'ler belirli bir zaman aralığında (time quantum, time slice q) CPU'ya dairesel bir sırayla atanır. Zaman aralığı genellikle 10 ms ile 100 ms aralığında seçilir.
- Hazır kuyruğu dairesel kuyruk şeklinde FIFO sırasına göre uygulanır. Yeni gelen process kuyruğun sonuna eklenir.
- q'dan daha kısa CPU burst süresine sahip process q dolmadan CPU'yu serbest bırakır, sonraki process CPU'ya geçer.
- CPU'da çalışan process'in CPU burst süresi q'dan büyükse q kadar CPU'da çalışır daha sonra context switch ile kuyruğun sonuna geçer.
- Round-robin scheduling ile ortalama waiting time genellikle uzundur.

Round-Robin Scheduling

- Aşağıda 3 process için CPU-burst time verilmiştir.

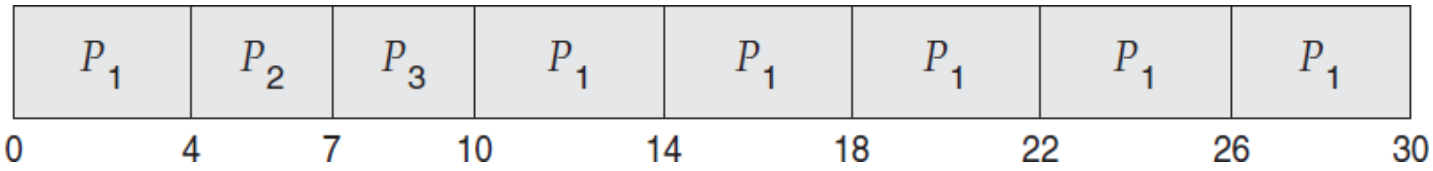
Process	Burst Time
P_1	24
P_2	3
P_3	3

- Round-Robin Scheduling algoritmasına göre Gantt şeması ve **ortalama bekleme süresi** ? q **= 4 ms**

Round-Robin Scheduling

- Aşağıda 3 process için CPU-burst time ve gantt şeması verilmiştir.
- Örnekte q **= 4 ms** olarak alınmıştır.

Process	Burst Time
P_1	24
P_2	3
P_3	3



- P_1 için = 10-4 = **6**, P_2 için **4**, P_3 için **7 ms waiting time vardır.**
- **Ortalama waiting time ise** 17 / 3 = **5,66 ms'dir.**
- q **time slice** süresiyle **n process** çalışan sistemde, bir process'in için **en fazla waiting time** $(n - 1) * q$ olur.

Round-Robin Scheduling

- Aşağıda 4 process için CPU-burst time verilmiştir.

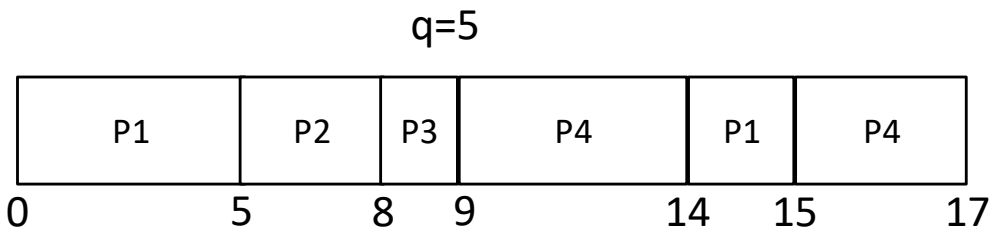
process	time
P_1	6
P_2	3
P_3	1
P_4	7

- Round-Robin Scheduling algoritmasına göre Gantt şeması ve **turaround süresi** ?
  q = 5 ms

Round-Robin Scheduling

- Aşağıda 4 process için CPU-burst time verilmiştir.

- Round-Robin Scheduling algoritmasına göre Gantt şeması ve **turaround süresi** ?
  q = 5 ms



P1 P2 P3 P4 turnaround time

15 + 8 + 9 + 17 = 49/4 = 12,25 ms

Round-Robin Scheduling

- Aşağıda 4 process için CPU-burst time verilmiştir.

process	time
P_1	6
P_2	3
P_3	1
P_4	7

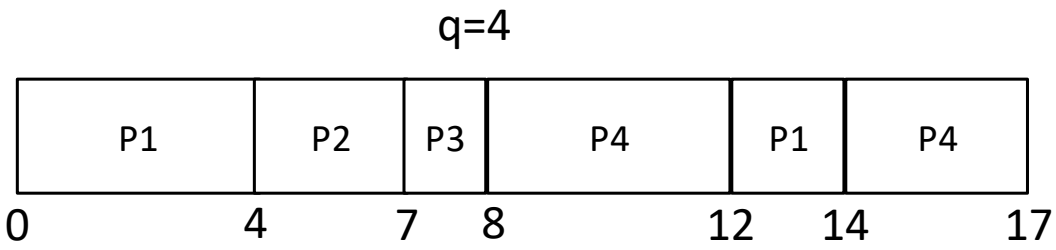
- Round-Robin Scheduling algoritmasına göre Gantt şeması ve **turaround süresi** ?
  q = 4 ms

Round-Robin Scheduling

- Aşağıda 4 process için CPU-burst time verilmiştir.

process	time
P_1	6
P_2	3
P_3	1
P_4	7

- Round-Robin Scheduling algoritmasına göre Gantt şeması ve **turaround süresi** ?
  q = 4 ms



P1 P2 P3 P4 turaround time

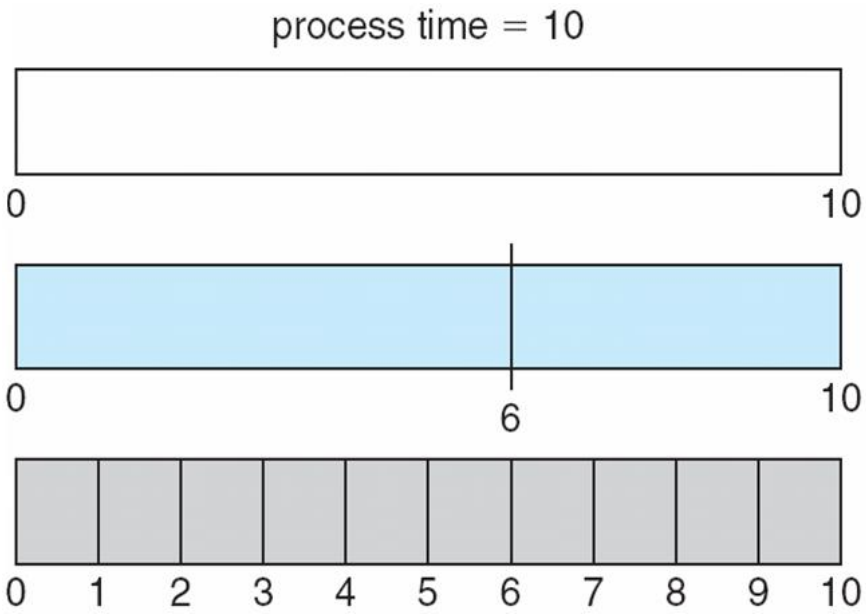
14 + 7 + 8 + 17 = 46/4 = 11,5 ms

Round-Robin Scheduling

- RR algoritmasının performansı büyük ölçüde time quantum (q) büyüklüğüne bağlıdır.
- q çok büyük olursa çalışma FCFS işleyişine benzer.
- q çok küçük olursa context switch işlemi çok fazla yapılır. Context switch işleminin çok fazla olması CPU'nun boşta kaldığı zamanın fazla olması demektir.
  - Bu nedenle, q değerinin context switch zamanına göre büyük olması istenir. context switch süresi q'nun yaklaşık yüzde 10'u ise, o zaman CPU zamanının yaklaşık yüzde 10'u context switch için harcanacaktır.

Time Quantum and Context Switch Time

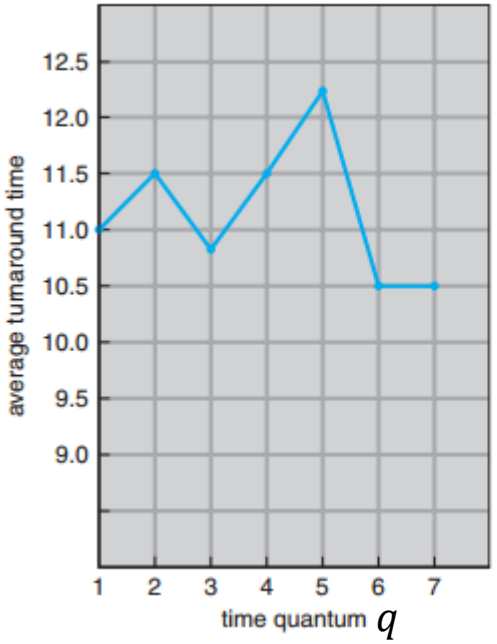
- Örnekte CPU burst süresi 10ms olan bir process için farklı quantum değerlerinde kaç kez context switch yapıldığı gösterilmiştir.



quantum	context switches
12	0
6	1
1	9

Time Quantum and Turnaround Time

- Turnaround süresi de time quantum büyüklüğüne bağlıdır. Şekilde görüldüğü gibi, bir dizi işlemin Turnaround süresi, time quantum büyüklüğü arttıkça mutlaka iyileşmez.
- Genel olarak, çoğu process'in bir sonraki CPU burst'leri tek bir quantum'da tamamlanırsa, ortalama turnaround süresi iyileştirilebilir.
- Temel bir kural: CPU burst'lerinin yüzde 80'inin time quantum'dan daha kısa olması gerektiğidir.



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Multilevel Queue Scheduling

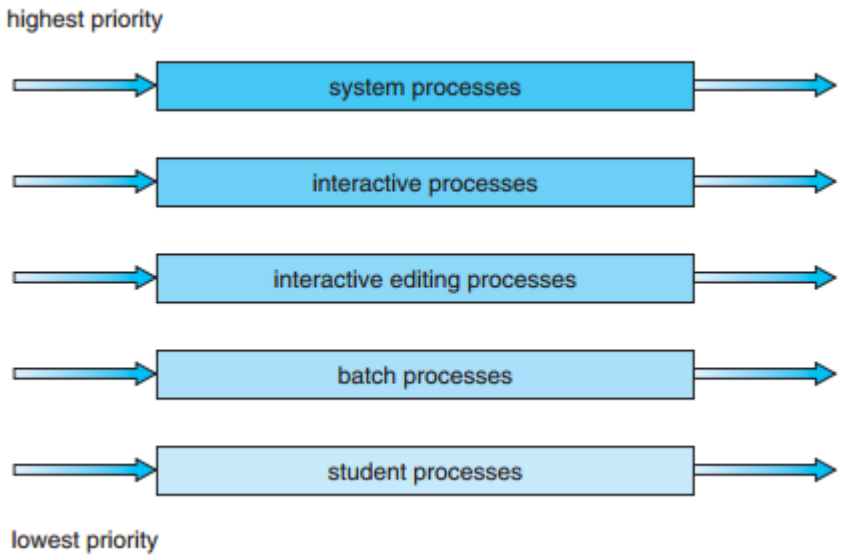
- **Multilevel Queue Scheduling (MQS) algoritmasında, process'ler farklı gruplar halinde sınıflandırılır.**
- Örneğin process'ler foreground (interaktif) ve background (batch) olarak 2 gruba ayrılabilir. Response-time kısa olması gereken process'ler foreground grubuna alınabilir ve bunlar background process'lerden daha öncelikli yürütülebilir.
- **Multilevel queue scheduling** algoritması **hazır kuyruğunu parçalara böler** ve kendi aralarında önceliklendirir.
- Process'ler bazı özelliklerine göre (hafıza boyutu, öncelik, process türü, …) bir kuyruğa atanır.
- Her kuyruk kendi scheduling algoritmasına sahiptir.

Multilevel Queue Scheduling

- Kuyruklar **arasında da** scheduling yapılması lazımdır, genellikle belli bir priority preemptive scheduling uygulanır. Örneğin, foreground kuyruğu, background kuyruğuna göre mutlak önceliğe sahip olabilir, ancak bu durumda starvation olabilir.
- Bir başka olasılık da kuyruklar arasında time-slice belirlemektir. Burada, her kuyruk CPU zamanının belirli bir bölümünü alır ve daha sonra kendi üzerindeki process'ler arasında scheduling yapabilir.
- Örneğin, foregorund-backgorund kuyruğu için, foregorund kuyruğuna process'ler arasında RR scheduling için CPU zamanının yüzde 80'i verilir, backgorund kuyruğu process'lerine FCFS uygulamak üzere CPU zamanının yüzde 20'si verilir.

Multilevel Queue Scheduling

- Her kuyruğa diğerlerine göre **mutlak öncelik tanımlanabilir, böyle bir durumda yüksek öncelikli kuyrukta process varken düşük öncelikli kuyruğa geçilmez.**
- Şekilde yüksek öncelikten düşük önceliğe doğru kuyruklar verilmiştir.
- Örneğin, sistem process'leri, interactive process'ler ve interactive editing process'ler için kuyrukların tümü boş olmadığı sürece batch processler kuyruğundaki hiçbir process çalışamaz.

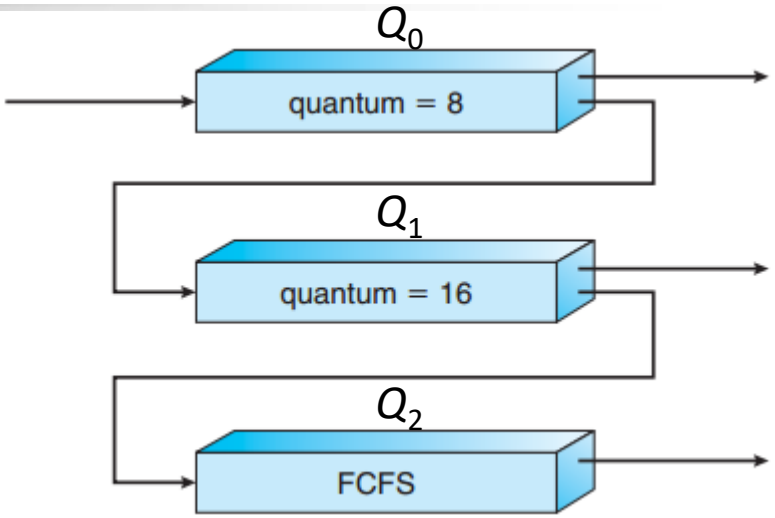


Multilevel Feedback Queue Scheduling

- **Multilevel Feedback Queue Scheduling algoritmasında, process'ler farklı kuyruklar arasında geçiş yapabilirler.**
- Bu yöntemde, I/O bound ve interaktif process'ler yüksek öncelikli kuyruğa, CPU'yu çok kullanan process'ler düşük öncelikli kuyruğa atanır.
- **Düşük öncelikli kuyrukta çok uzun süre bekleyen process'ler yüksek öncelikli kuyruğa aktarılabilir** (starvation engellenir).
- Hazır kuyruğuna gelen process öncelikle en yüksek öncelikli kuyruğa alınır.
- En yüksek öncelikli kuyruk tamamen boşalırsa ikinci öncelikli kuyruğa geçilir.

Multilevel Feedback Queue Scheduling

- Şekildeki örnekte en yüksek öncelikten en düşük önceliğe doğru 3 kuyruk verilmiştir:
  - Q_0 – RR (time quantum = 8 ms)
  - Q_1 – RR (time quantum = 16 ms)
  - Q_2 – FCFS
- Schedular öncelikle Q_0'daki process'leri yürütür, Q_0'da process kalmayınca Q_1'deki process'leri yürütür. Q_1'e bir process'in gelmesi Q_2'deki bir process'i preempted yapar.



Multilevel Feedback Queue Scheduling

- Genel olarak, multilevel feedback queue scheduler aşağıdaki parametrelerle tanımlanır:
  1. Kuyruk sayısı
  2. Her kuyruk için scheduling algoritması
  3. Bir process'in ne zaman daha yüksek öncelikli bir kuyruğa yükselteceğini belirlemek için kullanılan yöntem
  4. Bir process'in ne zaman daha düşük öncelikli bir kuyruğa indirileceğini belirlemek için kullanılan yöntem
  5. Bir process'in hizmete ihtiyacı olduğunda process'in hangi kuyruğa gireceğini belirlemek için kullanılan yöntem