YZM 3102
İşletim Sistemleri

Yrd. Doç. Dr. Deniz KILINÇ
Celal Bayar Üniversitesi
Hasan Ferdi Turgutlu Teknoloji Fakültesi
Yazılım Mühendisliği

BÖLÜM – 7

Bu bölümde,

- Temel Kavramlar
- Planlama Kriterleri
- Planlama Algoritmaları
- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-First (SJF) Scheduling
- Priority Scheduling
- Round-Robin Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling
- Multi-processor Scheduling

konularına değinilecektir.

Temel Kavramlar

- CPU scheduling (planlama), multiprogramming çalışan işletim sistemlerinin temelini oluşturur.
- CPU, **prosesler arasında geçiş yaparak** bilgisayarı daha verimli kullanılır hale getirir. **Neydi bu iş?**
- Her *zaman aralığında*, bir prosesin çalıştırılması amaçlanır.
- *Tek işlemcili sistemlerde*, **bir t anında sadece bir proses çalıştırılabilir**.
- Birden fazla proses olduğunda, bunlar CPU'nun işinin bitmesi için bekleyeceklerdir.

Temel Kavramlar (devam…)

- CPU'nun proseslerde ortaya çıkacak bekleme sürelerinde, başka prosesleri çalıştırılması sağlanır.
- Hafızada çok sayıda proses bulundurulur (**Nerede bu prosesler?**).
- Bir proses herhangi bir şekilde (**Hangi şekilde?**) **beklemeye geçtiğinde**, CPU başka bir prosese geçiş yapar.
- Bilgisayardaki **tüm kaynaklar**, kullanımdan önce zamana göre planlanır.

Temel Kavramlar (devam…)

- CPU **planlama algoritmasına göre** sırası gelen proses,
  - Bekleme sırasından (Ready - Hazır Kuyruk) alınarak,
  - Dağıtıcı (Dispatcher) ismi verilen bir işlem tarafından CPU'ya gönderilir.
- CPU'da, yine proses planlama algoritmasının izin verdiği kadar (*ya **bitene** ya da **belirli bir zaman** geçene kadar*) çalışan program
  - ya biter ve hafızadan kaldırılır
  - ya da **tekrar bekleme sırasına** bir sonraki çalışma için yerleştirilir.

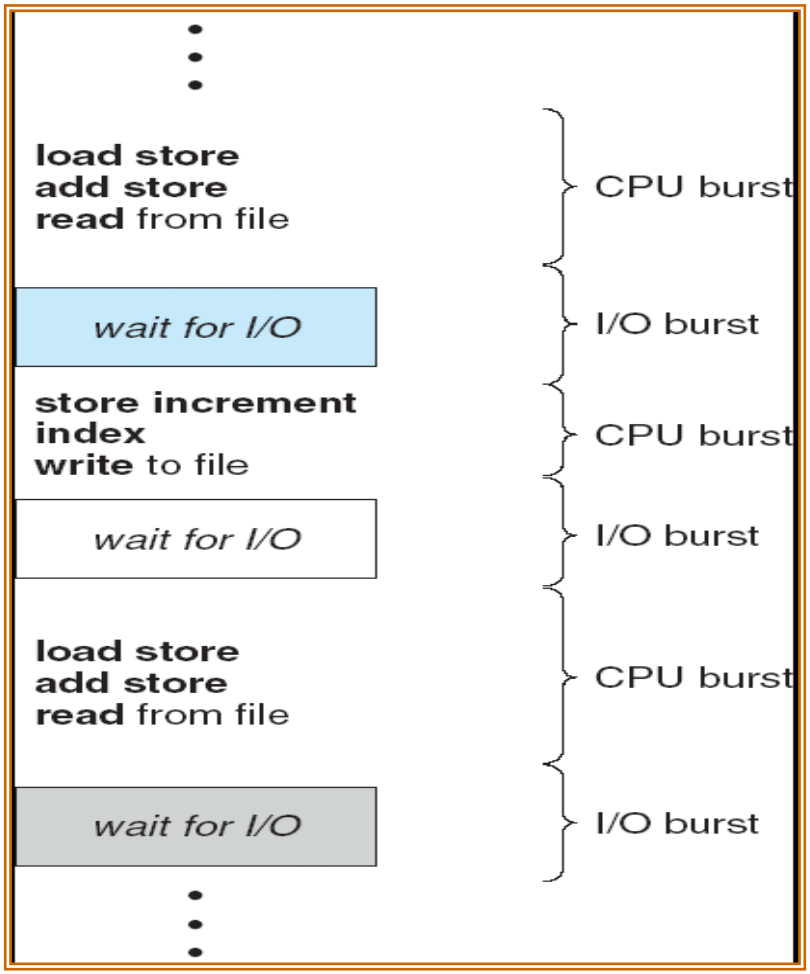
CPU -I/O Burst Cycle

- CPU'nun bir prosesi işlemek için **I/O Wait gelene kadar** *ihtiyacı olan zaman aralığı*.
- Başka deyişle CPU'nun tek proses için *tek bir seferde* harcadığı vakittir.
- Proses işletimi:
  - CPU burst ile başlar ve **sonra** I/O burst gelir bunu başka bir CPU burst ve arkasından başka bir I/O burst bunu takip eder.
  - **Son CPU burst**, sistemden terminate isteğinde bulunarak **programı sona erdirir**.

CPU -I/O Burst Cycle (devam…)

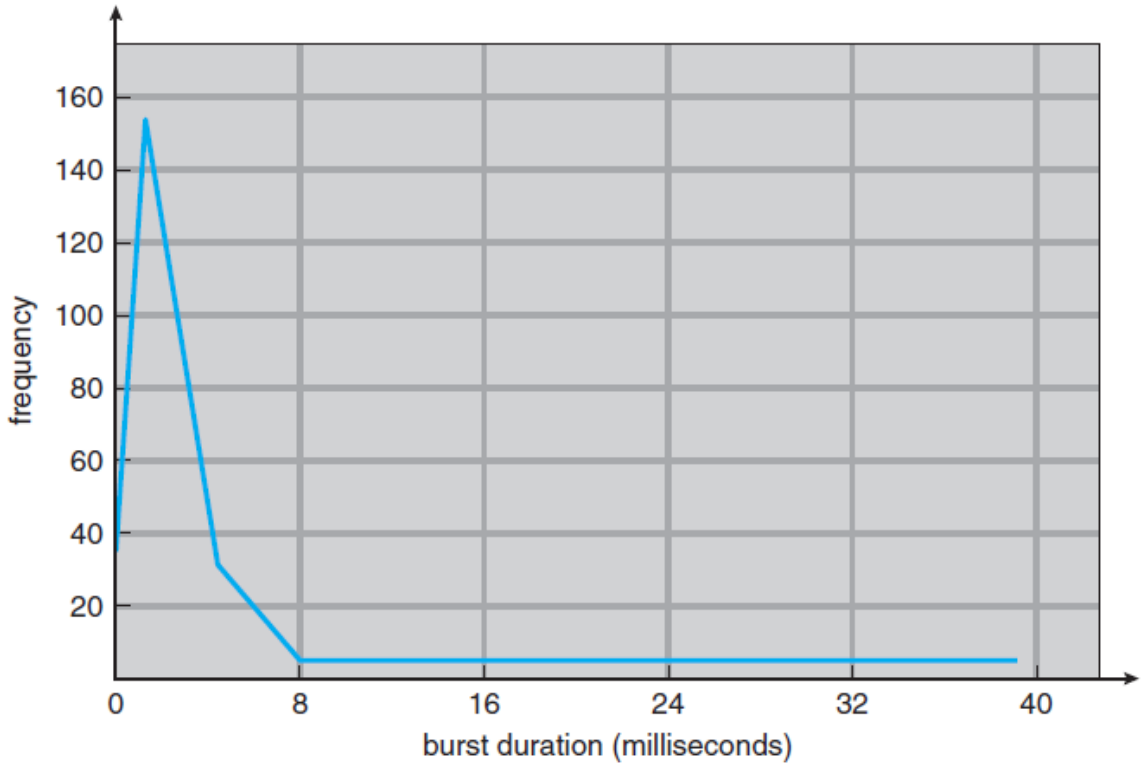
- I/O ve CPU burst <u>arasındaki kullanım</u> **sıklığı ve bekleme**, işin
  - **CPU bound** veya
  - **I/O bound**

olması ile önemli oranda ilgilidir.



CPU -I/O Burst Cycle (devam…)

- CPU burst süresi, **prosesten prosese** ve **bilgisayardan bilgisayara** *çok farklı olabilmektedir*. CPU burst süresi **kısa olanlar** **çok sık**, CPU burst süresi **uzun olanlar** ise **çok seyrek** çalışmaktadır.

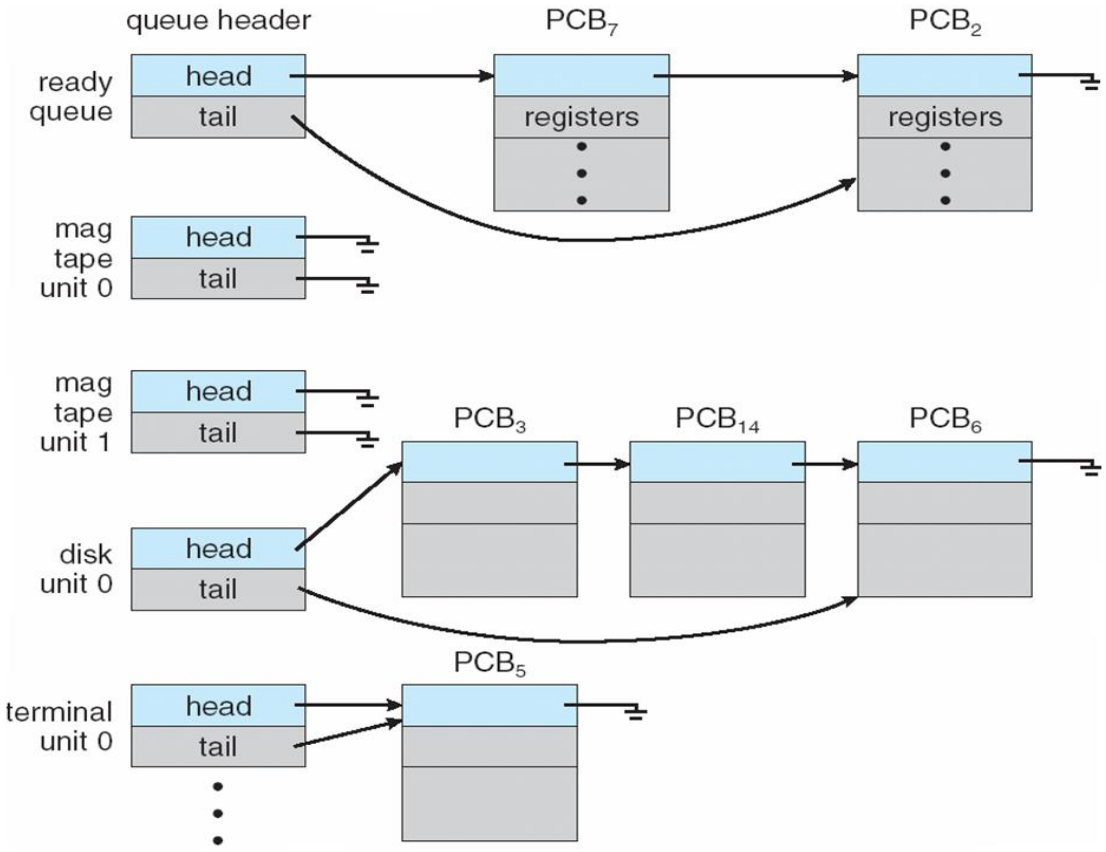


CPU Planlayıcı (Scheduler)

- CPU **bekleme durumuna geçtiğinde** (**Idle**), işletim sistemi hazır kuyruğundan (ready queue) bir prosesi çalıştırılmak üzere **seçmek zorundadır**.
- Seçme işlemi, kısa dönem planlayıcı (**short-term scheduler** veya CPU scheduler) tarafından gerçekleştirilir.
- Hazır kuyruğu, **ilk gelen ilk çıkar** (firs-in-first-out, FIFO) **olmak zorunda değildir**.
- Hazır kuyruğu, FIFO, Priority Queue, Ağaç, Sırasız Bağlı Liste şeklinde gerçekleştirilmiş olabilir.

CPU Planlayıcı (Scheduler) (devam…)

- Hazır kuyruğunda bekleyen **tüm proseslerin** CPU tarafından çalıştırılma durumları vardır.
- Kuyruk içindeki kayıtlarda, Process Control Block (PCB) tutulur.



Kesintili (preemptive) ve Kesmeyen (non-preemptive) Kavramı

- Hazır kuyruğu ile CPU arasında planlama ilişkisini kuran CPU planlama algoritmaları (CPU scheduling algorithms) temel olarak 2 grupta incelenebilir:
  - **Kesintili algoritmalar (preemptive):** Yürütülen yani **çalışan bir prosesin** CPU'dan kaldırılması ve **istenilen başka bir prosesin (öncelikli)** CPU'da yürütülmesi sağlanabilir.
  - **Kesmeyen algoritmalar (nonpreemptive):** Proses CPU'da çalışmaya başladıktan sonra;
    - Proses **tamamlanıncaya veya durana kadar** CPU'yu kullanır.
    - Kendi kodunda bulunan bir I/O isteği ile bloklanıncaya kadar ya da kendi isteği ile işlemciden çıkıncaya kadar çalışır.

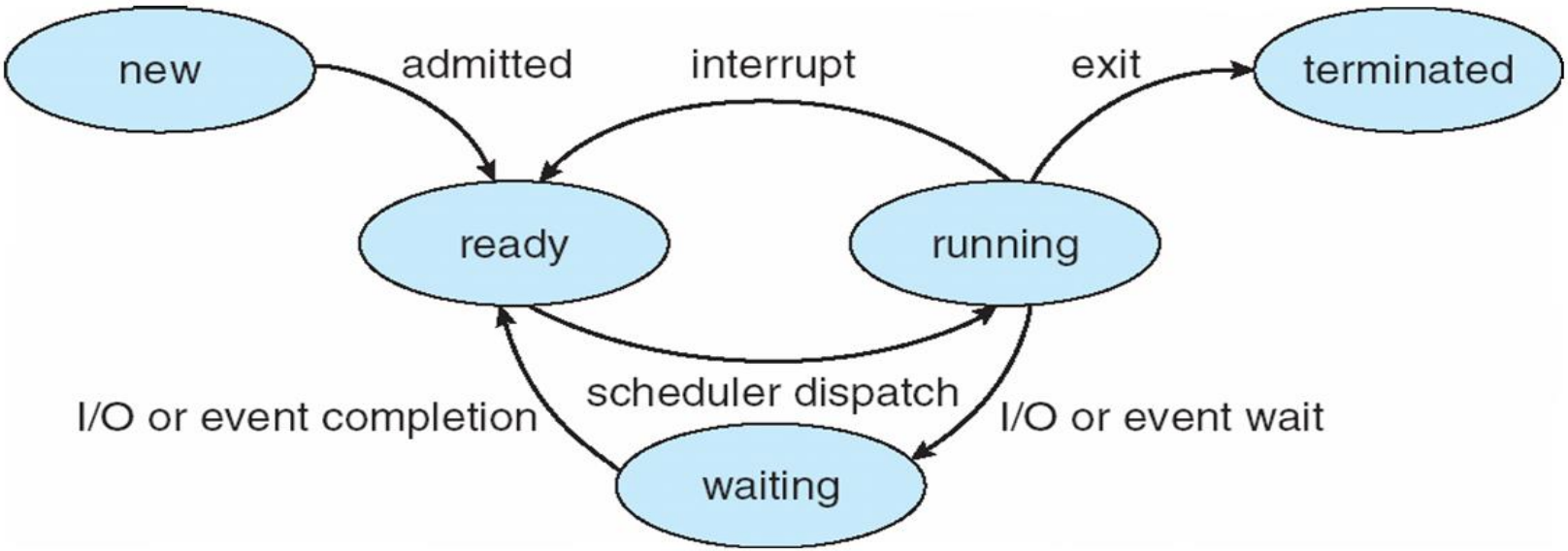
Kesintili (preemptive) ve Kesmeyen (non-preemptive) Kavramı (devam…)

- Windows 3.1, nonpreemptive planlama kullanmıştır.
- **Diğer tüm Windows** versiyonları preemptive planlama kullanmıştır.
- Mac OS X işletim sistemi de preemptive planlama kullanmaktadır.
- Preemptive planlama veri paylaşımı yaptığında **race condition** gerçekleşir.
- Bir prosesin kernel verisi üzerinde değişiklik yaparken ***yarıda kesilerek*** başka bir prosese geçilmesi ve aynı veriye erişim yapılması halinde **karışıklık meydana gelir**.

CPU Planlama

- CPU planlama kararı **4 durum altında** gerçekleştirilir:
  1. Bir proses çalışma durumundan bekleme durumuna geçtiğinde (***I/O isteği***),
  2. Bir proses çalışma durumundan hazır durumuna geçtiğinde (***interrupt***),
  3. Bir proses bekleme durumundan hazır durumuna geçtiğinde (***I/O tamamlanması***),
  4. Bir proses **sonlandırıldığında**.
- 1. ve 4. durumlarda **başka seçenek yoktur** ve **yeni bir proses** ***hazır kuyruğundan seçilerek*** çalıştırılmaya başlanır.
- 2. ve 3. durumlarda farklı duruma geçme seçenekleri bulunmaktadır.
- Eğer planlama işlemi 1. ve 4. durumlarda gerçekleşmişse, buna **nonpreemptive** veya **cooperative** planlama denir.
- 2. ve 3. durumlarda gerçekleşmişse **preemptive** planlama denir.

Proses Durumları - Hatırlatma



Dağıtıcı/Görevlendirici (Dispatcher)

- İşletim sistemi tasarımında kullanılan görevlendirici, **CPU planlama algoritmasına göre** beklemekte olan proseslerden sıradakini alıp, ***CPU'ya yollayan*** programın ismidir.
- Görevlendirici bu proseslerden **sırası gelenin** hazır kuyruğundan (ready queue) alınarak CPU'ya gönderilmesi işlemini yerine getirir.
- Bu fonksiyon şunları kapsamaktadır:
  - Bağlam değişimi (**Context Switch**)
  - Kullanıcı moduna geçiş
  - Programı tekrar başlatmak için kullanıcı programında uygun bölgeye geçişin sağlanması
- Görevlendiricinin çok hızlı bir şekilde **geçiş yapması zorunludur**.
- Prosesler arasında geçiş süresine dispatch latency (görevlendirici gecikme süresi) denilmektedir.

Planlama (Scheduling) Kriterleri

- CPU planlama algoritmaları çok sayıda farklı kritere göre karşılaştırılır:

1. **CPU kullanım (Utilization):** CPU'nun olabildiği kadar kullanımda olması istenir. CPU kullanım oranı **%0 - %100** arasındadır. Gerçek sistemlerde bu oran **%40 (normal) ile %90 (yoğun)** arasındadır.
2. **Üretilen iş (Throughput):** Birim zamanda tamamlanan proses sayısıdır (saniyede, saatte).
3. **Dönüş süresi (Turnaround time):** Bir prosesin
   - Hafızaya alınmak için bekleme süresi,
   - Hazır kuyruğunda bekleme süresi,
   - CPU'da çalıştırılması ve
   - I/O işlemi yapması için **geçen sürelerin toplamıdır**.

Planlama (Scheduling) Kriterleri (devam…)

4. **Bekleme süresi (Waiting time):** Bir prosesin hazır kuyruğunda beklediği süredir.
5. **Cevap süresi (Response time):** Bir prosese istek gönderildikten cevap dönünceye kadar geçen süredir.

Explain the difference between response time and turnaround time. These times are both used to measure the effectiveness of scheduling schemes.

Ans:

Turnaround time is the sum of the periods that a process is spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O. Turnaround time essentially measures the amount of time it takes to execute a process. Response time, on the other hand, is a measure of the time that elapses between a request and the first response produced.

Planlama (Scheduling) Kriterleri (devam…)

Optimizasyon için ne istiyoruz?

- Maksimum CPU Kullanım (utilization)
- Maksimum Üretilen İş (throughput)
- Minimum Dönüş Süresi (turnaround time)
- Minimum Bekleme Süresi (waiting time)
- Minimum Cevap Süresi (response time)

Planlama (Scheduling) Algoritmaları

- CPU planlama algoritmaları, hazır kuyruğunda bekleyen proseslerden hangisinin CPU'ya atanacağını belirlerler.
  1. **First-Come, First-Served (FCFS) Scheduling**
  2. **Shortest-Job-First (SJF) Scheduling**
  3. **Priority Scheduling**
  4. **Round-Robin Scheduling**
  5. **Multilevel Queue Scheduling**
  6. **Multilevel Feedback Queue Scheduling**

1.First-Come, First-Served (FCFS)

- **En basit** CPU planlama algoritmasıdır ve first-come first served (FCFS) şeklinde çalışır.
- CPU'ya **ilk istek yapan proses**, CPU'ya **ilk atanan prosestir**.
- FIFO kuyruk yapısıyla yönetilebilir.
- FCFS algoritmasıyla **ortalama bekleme süresi** genellikle **yüksektir**.
- Bekleme süreleri, proseslerin kuyruğa geliş sırasına göre ***çok değişmektedir***.

1.First-Come, First-Served (FCFS) (devam…)

Proses	Burst Zamanı
P_1	24
P_2	3
P_3	3

- Proseslerin **geliş sırası**: P_1 , P_2 , P_3 bu durumda **Gantt Chart**

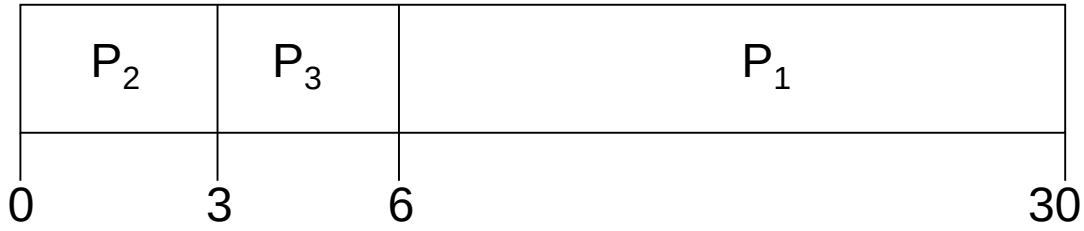
- **Bekleme zamanları:** P_1 = 0; P_2 = 24; P_3 = 27
- **Ortalama bekleme zamanı**: (0 + 24 + 27)/3 = 17 ms

1.First-Come, First-Served (FCFS) (devam…)

Proseslerin sırası **aşağıdaki gibi değişirse**

$$P_2 , P_3 , P_1$$

- Planlama için Gantt chart



- **Bekleme zamanları:** $P_1 = 6; P_2 = 0, P_3 = 3$
- **Ortalama bekleme zamanı:** $(6 + 0 + 3)/3 = 3$

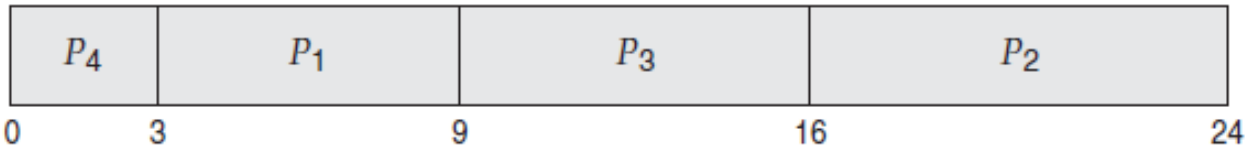
1.First-Come, First-Served (FCFS) (devam…)

- FCFS algoritmasında, proseslerin çalışma süreleri **çok farklıysa** ortalama bekleme süreleri çok değişken olur.
- Bir CPU-bound proses ile çok sayıda I/O bound proses varsa, CPU-bound proses ***CPU'da çalışırken*** tüm I/O bound prosesler ***hazır kuyruğunda bekler***, I/O cihazları boş kalır.
- **Çok sayıda küçük prosesin** büyük bir prosesin CPU'yu terk etmesini beklemesine **convoy effect** denilmektedir.
- Bir prosese CPU tahsis edildiğinde sonlanana veya I/O isteği gelene kadar CPU'yu elinde tutar.
- FCFS algoritması belirli zaman aralıklarıyla CPU'yu paylaşan **time-sharing sistemler için uygun değildir**.

2.Shortest-Job-First (SJF)

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

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3



- **Ortalama bekleme süresi**, (0 + 3 + 16 + 9) / 4 = 7 ms'dir. FCFS kullanılsaydı 10,25 ms olurdu.
- SJF algoritması **minimum ortalama bekleme süresini** elde eder.

2.Shortest-Job-First (SJF) (devam…)

- SJF algoritmasındaki **en büyük zorluk**, *sonraki çalışma süresini* **tahmin etmektir**.
- SJF algoritması genellikle **long-term scheduling** için kullanılır.
- Short-term scheduling'te CPU'da sonraki çalışma süresini **bilmek mümkün değildir**.
- Short-term scheduling'te sonraki *çalışma süresi* **tahmin edilmeye çalışılır**.
- Sonraki proses çalışma süresinin, ***önceki çalışma süresine*** **benzer olacağı** beklenir (***Approximate SJF scheduling***)

2.Shortest-Job-First (SJF) (devam…)

- Sonraki CPU burst genellikle **"exponential average of the measured lengths of previous CPU bursts"** şeklinde aşağıdaki formül ile kestirilir.

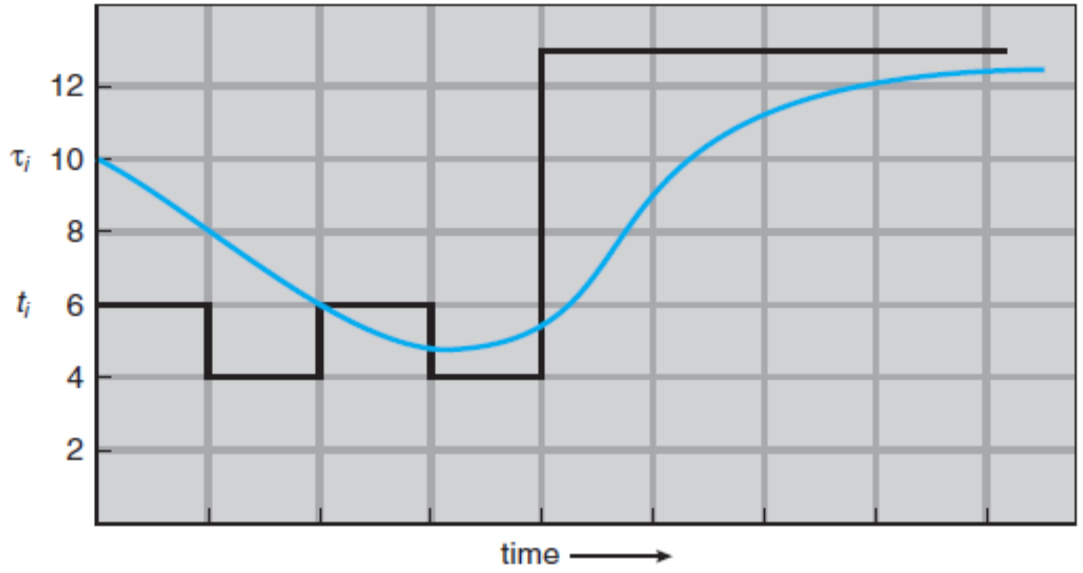
$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n$$

- **tn:** n. CPU burst süresi
- τ_{n+1} : sonraki CPU burst için kestirilen değer.
- τ_n : past history.
- α: recent ve past history arasındaki bağlantı ağrılığını kontrol eder. $0 \leq \alpha \leq 0$
- α =1/2 seçilir. Böylece recent ve past history eşit ağırlıklandırılır.
- $\tau_0 = 10$ ve α=1/2 için çözüm:

2.Shortest-Job-First (SJF) (devam…)

$\tau_0 = 10$ ve α=1/2 için çözüm:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \cdots + (1 - \alpha)^j \alpha t_{n-j} + \cdots + (1 - \alpha)^{n+1} \tau_0.$$



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

Figure 6.3 Prediction of the length of the next CPU burst.

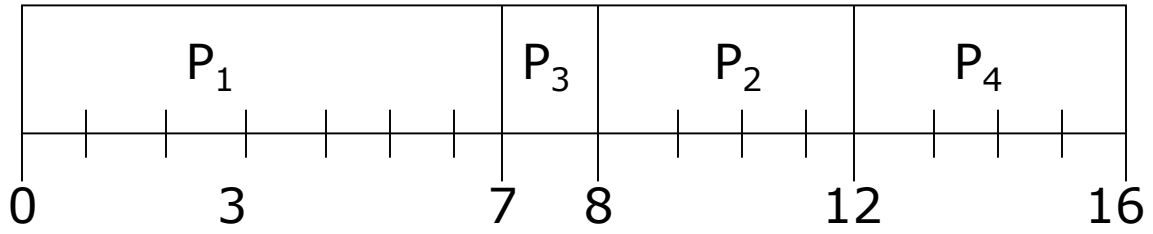
2.Shortest-Job-First (SJF) (devam…)

- SJF algoritması **preemptive** veya **nonpreemptive** olabilir.
- Çalışmakta olan prosesten
  - **Daha kısa süreye sahip yeni bir proses** geldiğinde,
  - Preemptive SJF **çalışmakta olan prosesi keser**,
  - Non-preemptive SJF çalışmakta olanın sonlanmasına izin verir.
- Preemptive SJF,
  - **Shortest-remaining-time-first** (**Kalan çalışma süresi en kısa olan ilk çalışsın**) planlama olarak adlandırılır.

2.Shortest-Job-First (SJF) (devam…)

Proses	Geliş Zamanı	Burst Zamanı
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (**non-preemptive**)

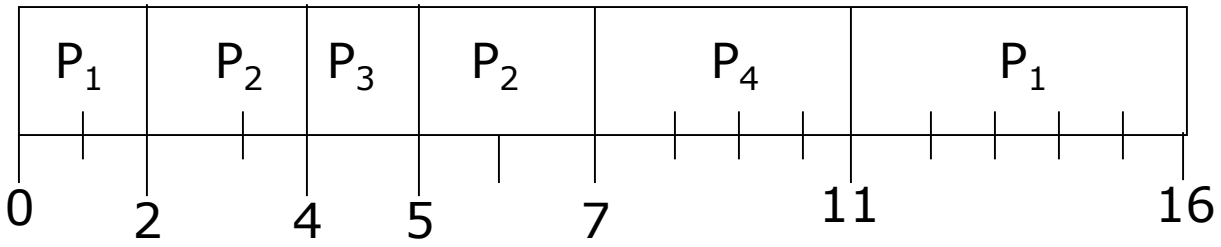


- Ortalama bekleme süresi = (0 + (8 - 2) + (7 - 4) + (12 - 5))/4
- Ortalama bekleme süresi = (0 + 6 + 3 + 7)/4 = 4

2.Shortest-Job-First (SJF) (devam…)

Proses	Geliş Zamanı	Burst Zamanı
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (**preemptive**)

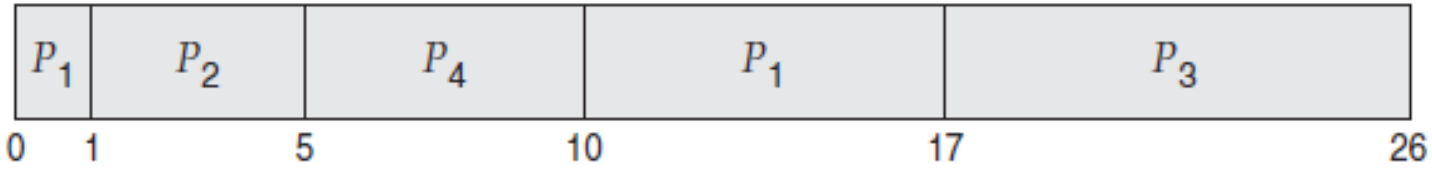


- Ortalama bekleme süresi = ((11-2) + (5-4) + (4-4) +(7-5))/4
- Ortalama bekleme süresi = (9 + 1 + 0 +2)/4 = 3

2.Shortest-Job-First (SJF) (devam…)

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

Preemptive SJF



Ortalama bekleme süresi = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

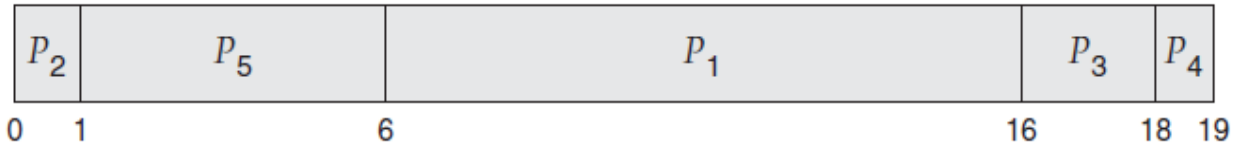
3. Önceliğe Göre Planlama (Priority Scheduling)

- CPU en yüksek önceliğe sahip prosese atanır.
- Eşit önceliğe sahip olanlar ise **FCFS sırasıyla** atanır.
- **Shortest-job-first (SJF)** algoritması, priority planlama algoritmalarının **özel bir durumudur**.
  - SJF algoritması tahmin edilen **CPU-burst süresine göre** önceliklendirme yapar.
  - SJF algoritmasında, **CPU burst süresi azaldıkça öncelik artar**, CPU burst süresi arttıkça **öncelik azalır**.

3. Önceliğe Göre Planlama (Priority Scheduling)

- Aşağıda 5 proses için öncelik değerine göre gantt şeması verilmiştir.

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



- Ortalama bekleme süresi **(1+ 6 + 16 + 18) / 4 = 8,2 ms** olur.

3. Önceliğe Göre Planlama (Priority Scheduling)

- Önceliklendirme kriterleri aşağıdakilerden bir veya birkaç tanesi olabilir:
  - Zaman sınırı
  - Hafıza gereksinimi
  - Açılan dosya sayısı
  - I/O burst ve CPU burst oranı
  - Prosesin önemi
- Priority planlama **preemptive** veya **nonpreemptive** olabilir.
  - Preemptive yönteminde, bir proses hazır kuyruğuna geldiğinde, çalışmakta olan prosesten daha öncelikli ise, çalışmakta olan kesilir.
  - Nonpreemptive yönteminde, bir process hazır kuyruğuna geldiğinde, çalışmakta olan prosesten daha öncelikli bile olsa, çalışmakta olan devam eder.

3. Önceliğe Göre Planlama (Priority Scheduling)

- Priority planlama algoritmasında,
  - CPU sürekli yüksek öncelikli prosesleri çalıştırabilir ve
  - Bazı prosesler sürekli hazır kuyruğunda bekleyebilir (**indefinite blocking**, **starvation**).
- Sınırsız beklemeyi **engellemek için**
  - Öncelik yaşlanması (**priority aging**) yöntemi kullanılır.
  - Düşük öncelikli prosesler kuyrukta beklerken **öncelik seviyesi artırılır** (Örn. her 15 dakikada 1 artırılır).
- **Öncelik değeri artırılarak** ***en düşük önceliğe sahip*** prosesin bile belirli bir süre **sonunda çalışması** sağlanır.

4. Round Robin

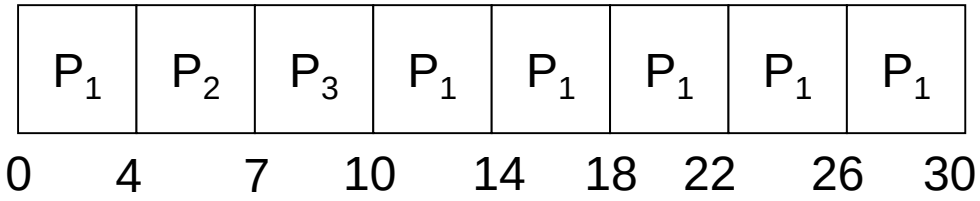
- Round-robin (RR) planlama, genellikle **time-sharing** (zaman paylaşımlı) sistemlerde kullanılır.
- Hazır kuyruğundaki prosesler **belirli bir zaman aralığında** (*time slice = quantum*) CPU'ya sıralı atanır.
- Zaman aralığı genellikle **«10 ms ile 100 ms»** aralığında seçilir.
  - (1 zaman aralığı = **time quantum**)
- Bu süreden ***daha kısa sürede sonlanan proses*** CPU'yu serbest bırakır.
- Round-robin planlama ile **ortalama bekleme süresi** genellikle uzundur.

4. Round Robin (devam…)

Proses	Süre (Burst Time)
P_1	24
P_2	3
P_3	3

Örnek1 RR:
Time Quantum = 4

- Gantt Şeması



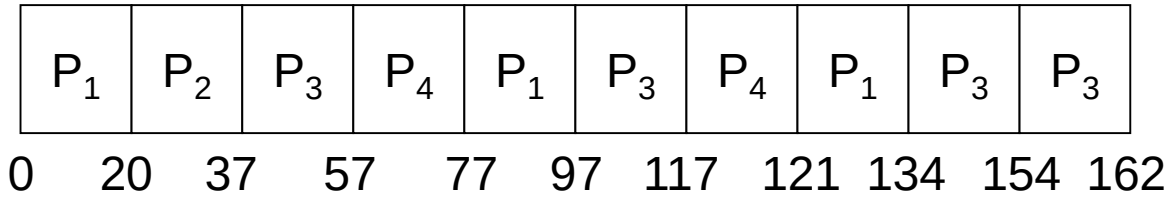
- P1 için bekleme (10-4)=6 , P2=4 , P3=7 bekleme süresine sahiptir.
- ***Ortalama bekleme zamanı***= 17/3=5.66

4. Round Robin (devam…)

Proses	Süre (Burst Time)
P_1	53
P_2	17
P_3	68
P_4	24

Örnek2 RR:
Time Quantum = 20

- Gantt chart:

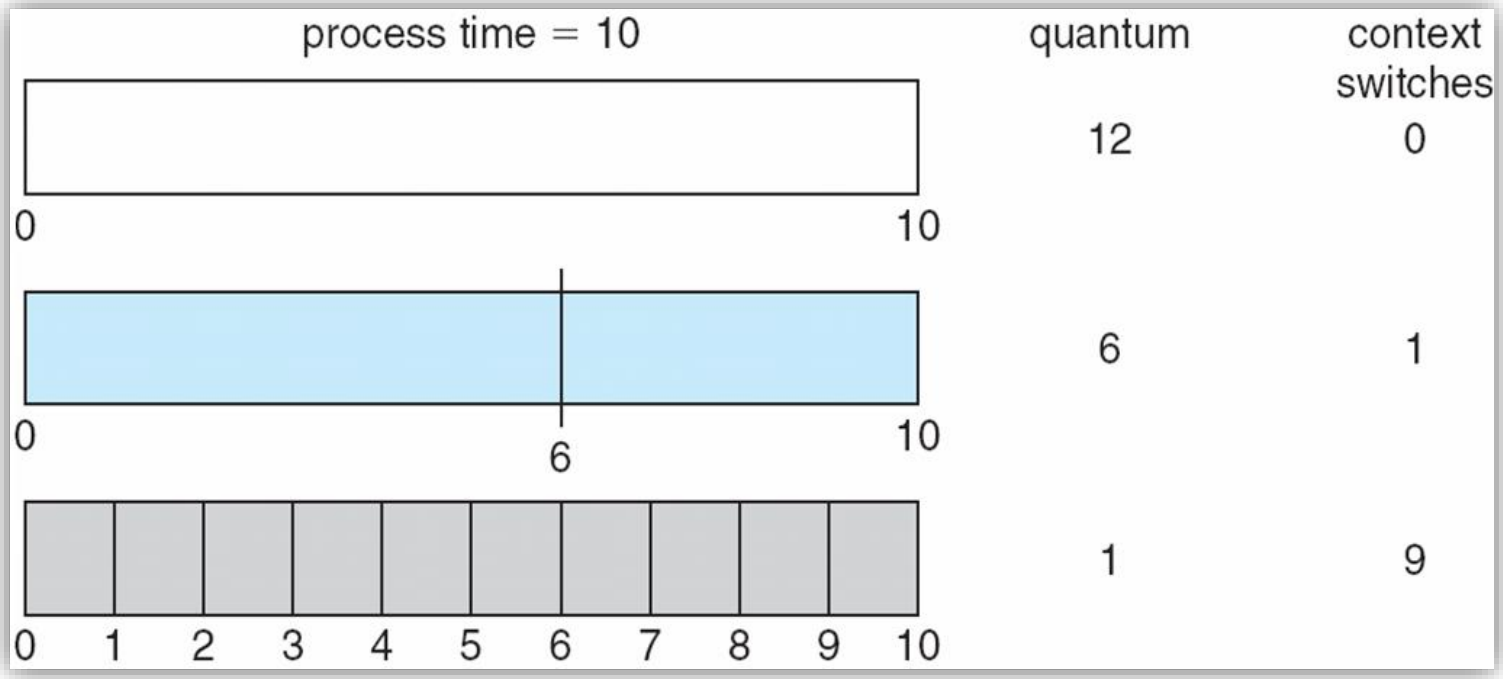


- ***Bekeleme zamanı***: P1:57+24=101; P2:20; P3:37+40+17=94; P4:57+40=97

4. Round Robin (devam…)

- Time slice süresi (*quantum*) **çok büyük olursa** çalışma **FCFS** yöntemine benzer.
- Time slice süresi **çok küçük olursa** **context switch** işlemi **çok fazla yapılır**.
  - Context switch süresi **overhead olur** ve çok fazla context switch yapılması ***istenmez***.
- Time slice süresi, context switch süresinin genellikle 10 katı alınır.
  - Modern sistemler quantum 10-100 ms arasında
  - Context switch time <10 ms
  - CPU'nun %10 süresi context switch için harcanır.

4. Round Robin (devam…)



5. Multi-level Queue

- Multilevel Queue Scheduling (MQS) algoritmasında, prosesler **farklı gruplar** halinde **sınıflandırılır**.
- Örneğin prosesler **foreground (interaktif)** ve **background (batch)** olarak **2 gruba** ayrılabilir.
- Foreground proseslerde ***response-time*** kısa olması gereklidir ve background proseslere göre ***önceliklidir***.
- Multilevel queue scheduling algoritması hazır kuyruğunu parçalara böler ve kendi aralarında önceliklendirir.
  - Prosesler bazı özelliklerine göre (***hafıza boyutu, öncelik, process türü***, …) bir kuyruğa atanır.
  - Her kuyruk ***kendi planlama algoritmasına*** sahiptir.

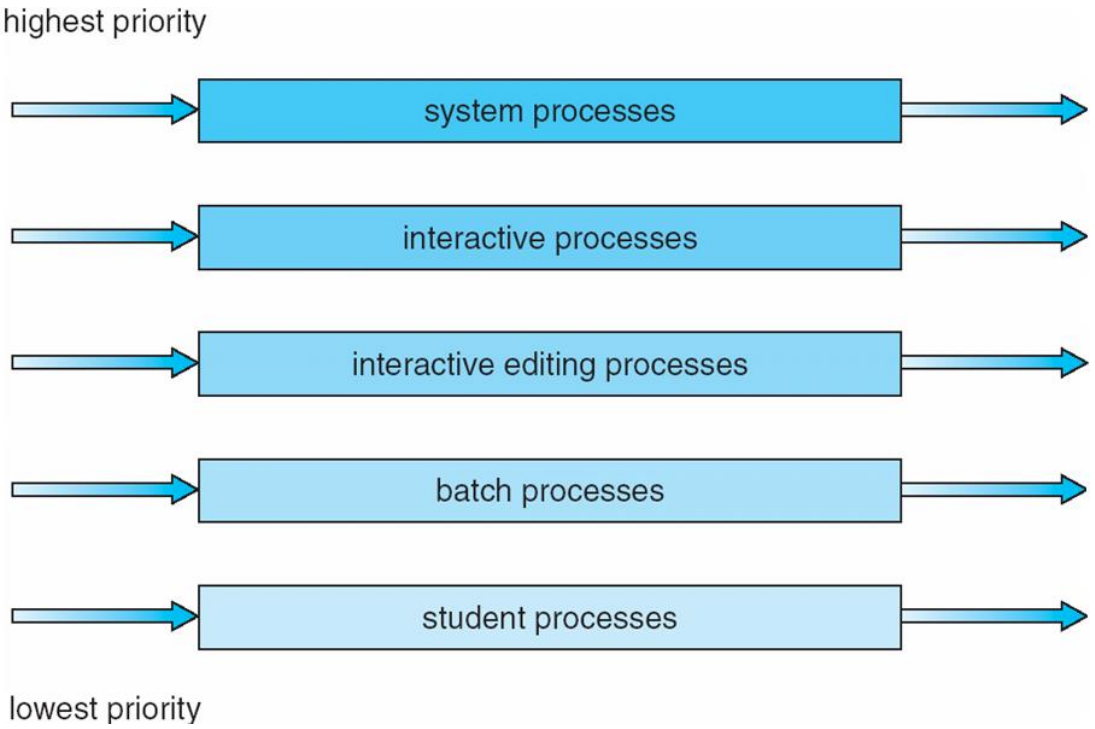
5. Multi-level Queue (devam…)

- Örneğin:
  - Ön plan ..foreground – **RR**
  - Arka plan..background – **FCFS**
- Kuyruklar **arasında da** planlama yapılmalıdır:
  - Sabit öncelikli planlama (Fixed priority scheduling)
  - Örneğin, önce arka plandaki işlemleri yap sonra ön plandakileri yap.
- Bu durumda **starvation (açlık)** problemi ortaya çıkabilir.
- **Alternatif 1: Time slice** – Her kuyruk işlemlerini planlamak için belli bir CPU time slice ataması yapılabilir.
  - Örneğin 80% foreground, 20% background

5. Multi-level Queue (devam…)

- **Alternatif 2:**
  Her kuyruğa diğerlerine göre **mutlak öncelik** tanımlanabilir ve **kendisinde proses varken** ***düşük öncelikli kuyruğa geçilmez***.

1. System processes
2. Interactive processes
3. Interactive editing processes
4. Batch processes
5. Student processes



6. Multi-level Feedback Queue

- Multilevel Queue Scheduling (MQS) algoritmasında, prosesler **farklı kuyruklar arasında** **geçiş yapabilirler**.
- Bu yöntemde, I/O bound ve interaktif prosesler ***yüksek öncelikli kuyruğa*** atanır.
- Düşük öncelikli kuyrukta **çok uzun süre bekleyen** prosesler *yüksek öncelikli kuyruğa aktarılır* (indefinite lock engellenir).
- *Hazır kuyruğuna* gelen proses, **öncelikle**, **en yüksek öncelikli kuyruğa** alınır.
- En **yüksek öncelikli kuyruk** **tamamen boşalırsa** ikinci öncelikli kuyruğa **geçilir**.

6. Multi-level Feedback Queue (devam…)

- En karmaşık algoritmadır.
- Aşağıdaki parametreler ile tanımlanır:
  - Kuyruk sayısı
  - Her kuyruk için bir düzenleme algoritması
  - Yüksek öncelikli kuyruğa terfi etmesini belirleyecek method
  - Düşük öncelikli kuyruğa geçirilmesini sağlayacak method
  - Servise ihtiyaç duyan bir prosesin hangi kuyruğa ekleneceğini belirleyen bir method

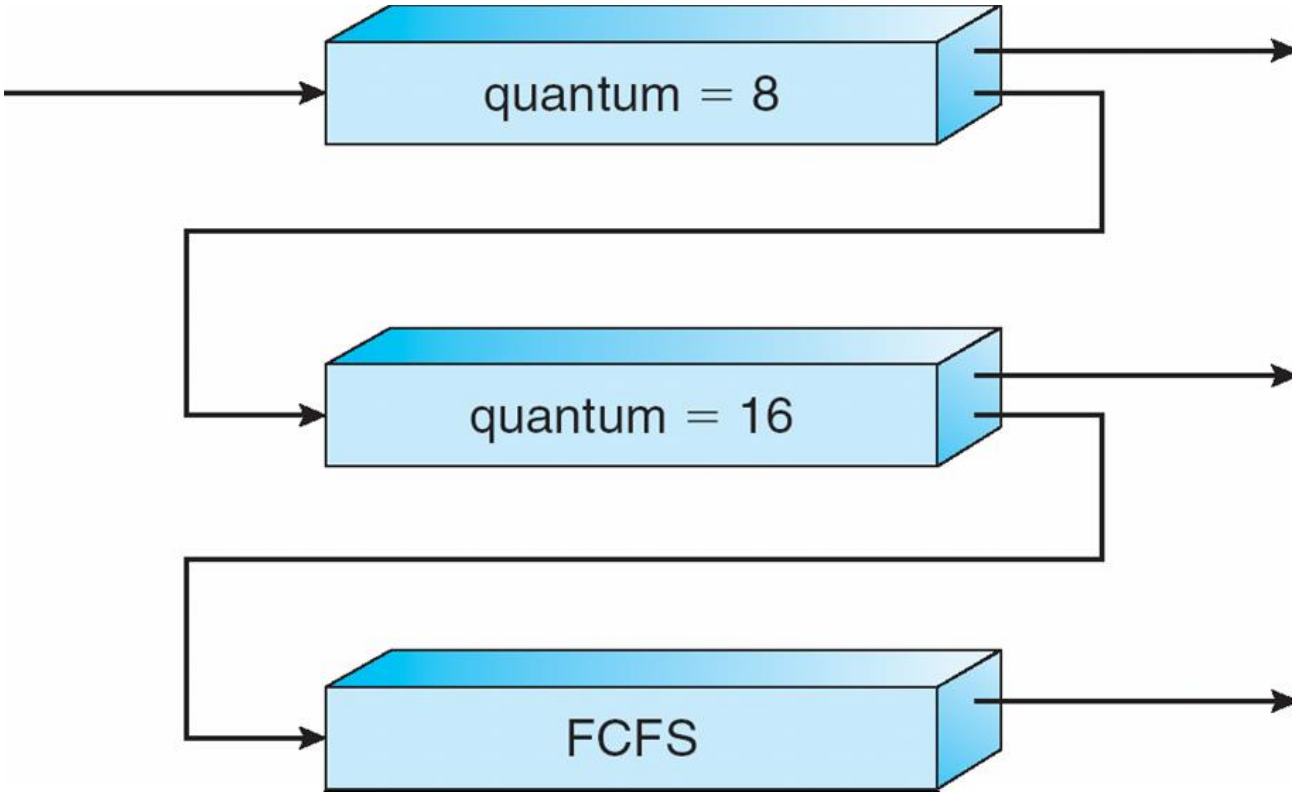
6. Multi-level Feedback Queue (devam…)

Örnek

- Üç kuyruk:
  - Q_0 – RR, quantum 8 ms
  - Q_1 – RR, quantum 16 ms
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

6. Multi-level Feedback Queue (devam…)

Örnek (devam…)



Multi-processor Scheduling

- Çok işlemci kullanılan sistemlerde yük paylaşımı (load sharing) yapılabilir, ancak scheduling çok daha karmaşık hale gelir.
- Asymmetric multi-processing yaklaşımında, CPU'larda birisi (master) scheduling algoritmaları, I/O işlemleri ve diğer sistem aktivitelerini yönetir. Diğer CPU'lar kullanıcı kodlarını çalıştırır.
- Symmetric multi-processing yaklaşımında, her CPU kendi scheduling algoritmasına sahiptir ve master CPU yoktur.
- Tüm CPU'lar **ortak** hazır kuyruğuna sahip olabilir veya ***ayrı ayrı hazır kuyruğu olabilir***.
- Birden fazla CPU'nun **paylaşılan veri yapısına erişimi engellenmelidir**.
- Birden fazla CPU'nun *aynı prosesi çalıştırması* **engellenmelidir**.
- Windows, Linux ve Mac OS X işletim sistemleri SMP desteğini sağlarlar.

Multi-processor Scheduling (devam…)

- Bir proses **başka bir işlemciye** **aktarıldığında**, *eski işlemcideki* **cache bellek** **bilgileri aktarılmaz**.
- Yeni aktarılan işlemcinin **cache bellek bilgileri oluşana kadar** *hit rate oranı* çok **düşük** kalır.
- Bir proses çalışmakta olduğu *işlemci ile ilişkilendirilir* (**processor affinity**) ve sonraki çalışacağı işlemci de aynı olur.
- Bazı sistemlerde, proses bir işlemciye atanır, ancak aynı işlemcide çalışmayı **garanti etmez** (soft affinity).
- Bazı sistemlerde, process bir işlemciye atanır ve her zaman aynı işlemcide çalışmayı **garanti eder** (hard affinity).
- **Linux işletim sistemi** *soft affinity* ve *hard affinity* desteğine sahiptir.

Multi-processor Scheduling (devam…)

- Yük dengeleme (load balancing), SMP sistemlerde tüm işlemciler üzerinde iş yükünü dağıtarak verimi artırmayı amaçlar.
- Her işlemcinin kendi kuyruğuna sahip olduğu sistemlerde, yük dengeleme **iyi yapılmazsa** bazı işlemciler *boş beklerken* diğer işlemciler *yoğun çalışabilir*.
- Ortak **kuyruk kullanan sistemlerde** *yük dengelemeye* **ihtiyaç olmaz**.
- Yük dağılımı için iki yöntem kullanılır: **push migration** ve **pull migration**.
- **Push migration yönteminde:** bir görev işlemcilerin iş yükünü kontrol eder ve boş olanlara dolu olan diğer işlemcilerdeki prosesleri aktarır.
- **Pull migration yönteminde:** boş kalan işlemci dolu olan diğer işlemcilerde bekleyen bir prosesi kendi üzerine alır.

İYİ ÇALIŞMALAR…

Yararlanılan Kaynaklar

- **Ders Kitabı:**
  - Operating System Concepts, Ninth Edition, Abraham Silberschatz, Peter Bear Galvin, Greg Gagne
- **Yardımcı Okumalar:**
  - İşletim Sistemleri, Ali Saatçi
  - Şirin Karadeniz, Ders Notları
  - İbrahim Türkoğlu, Ders Notları
- **M. Ali Akcayol, Gazi Üniversitesi Bilgisayar Mühendisliği Bölümü**