

İŞLETİM SİSTEMLERİ

İş Sıralama (CPU Zamanlama /
Planlama / Çizelgeleme)



Windows 11



macOS
Sonoma

Öğrenme Hedefleri

Bu konuyu çalıştıktan sonra:

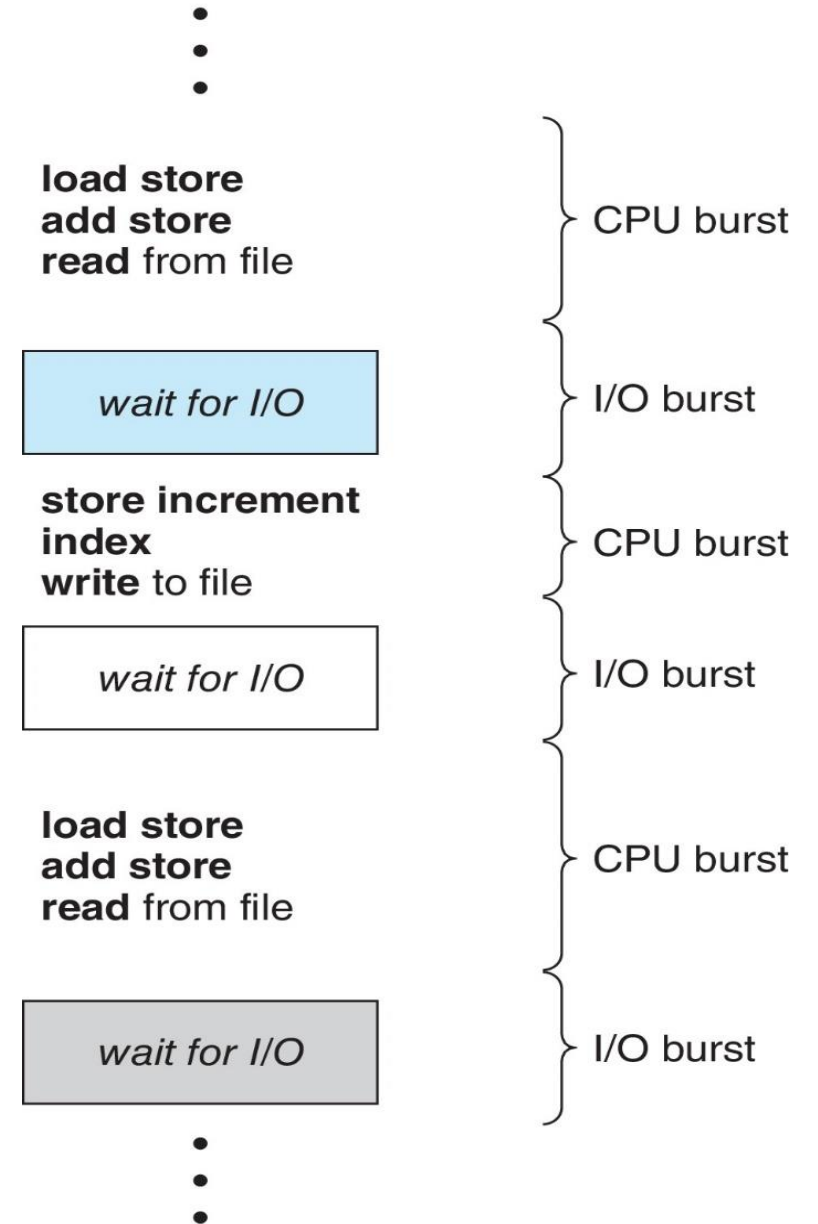
- 📖 Temel Kavramlar
- 📖 Sıralama Kriterleri
- 📖 İş Sıralama Algoritmaları

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.

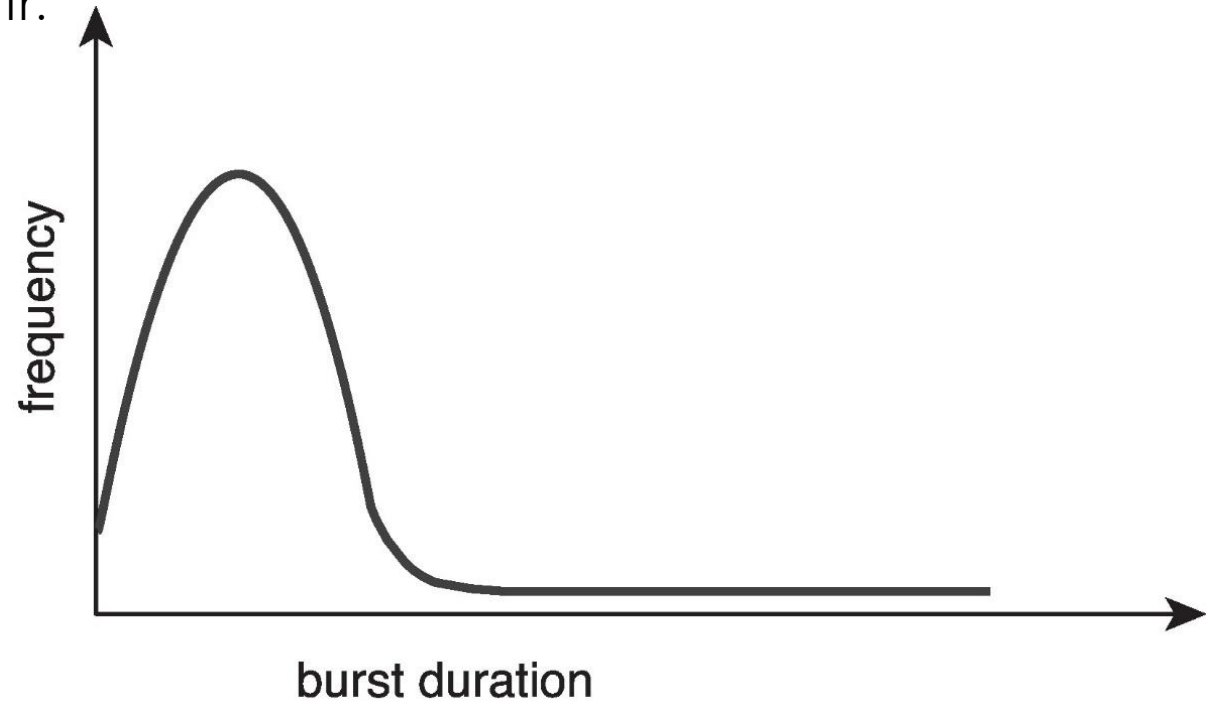
Temel Kavramlar

- Çoklu programlama ile elde edilen maksimum CPU kullanımı
- CPU-I/O Patlama (Burst) Çevrimi – Proses çalışması CPU çalışması **çevrimi** ve I/O beklemesinden oluşur
- **CPU patlamasını I/O patlaması** takip eder
- **CPU patlama** dağılımı ana ilgi noktasıdır.



CPU Patlama Zamanları Histogramı

- CPU patlamalarının süreleri genellikle işlemde işleme ve bilgisayardan bilgisayara büyük ölçüde değişse de, genellikle Şekil'de gösterilen frekans eğrisine benzer bir eğilime sahiptir.
- Eğri genellikle üstel veya hiperüstel olarak karakterize edilir, yani birçok kısa CPU patlaması ve az sayıda uzun CPU patlaması bulunur.
- Giriş/çıkışa bağlı bir program genellikle birçok kısa CPU patlaması içerirken, CPU'ya bağlı bir program birkaç uzun CPU patlaması içerebilir.



CPU İş Sıralayıcı (Scheduler)

- CPU hafızada hazır kuyruğunda bekleyen prosesler arasından seçim yapar ve CPU çekirdeğini bir tanesine tahsis eder.
 - Kuyruk çeşitli şekillerde sıralanabilir. (a FIFO queue, a priority queue, a tree, or unordered linked list)
- CPU iş sıralama kararları aşağıdaki durumlarda verilir:
 1. Çalışıyor durumundan bekleme durumuna geçerken
 2. Çalışıyor durumundan hazır durumuna geçerken
 3. Bekleme durumunda hazır durumuna geçerken
 4. Proses sonlanınca
- 1 ve 4 durumları için iş sıralama bakımından başka seçenek yoktur. Yeni bir proses (eğer hazır kuyruğunda bulunuyorsa) çalışması için seçilir.
- 2 ve 3 nolu durumlarda bir seçenek vardır.

Kesintili (Preemptive) ve Kesintisiz(Nonpreemptive) İş Sıralama

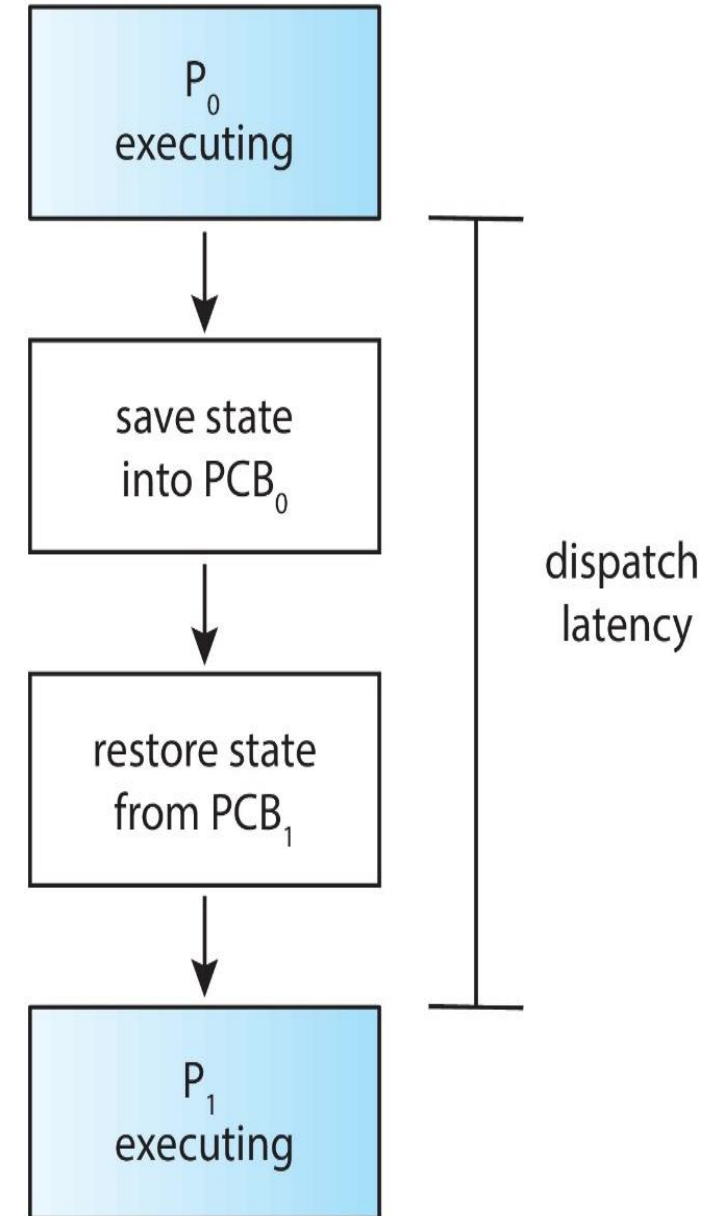
- 1 ve 4 durumları **kesintisizdir**
- Diğer tüm iş sıralama işlemleri **kesintilidir**
- **Kesintisiz** iş sıralamada, CPU bir prosese tahsis edildikten sonra, proses CPU'yu sonlanana veya bekleme durumuna geçene kadar elinde tutar.
- Windows, MacOS, Linux ve UNIX dahil olmak üzere neredeyse tüm modern işletim sistemleri **kesintili** iş sıralama algoritmaları kullanır.

Kesintili İş Sıralama ve Yarış Koşulları

- Kesintili iş sıralama, verilerin çeşitli prosesler arasında paylaşılması durumunda yarış koşullarına neden olabilir.
- Verileri paylaşan iki proses düşünün.
 - Bir proses verileri güncellerken, diğer bekleyen proses çalışabilsin diye çalışması yarıda kesilir.
 - İkinci proses daha sonra tutarsız bir durumda olan verileri okumaya çalışır.

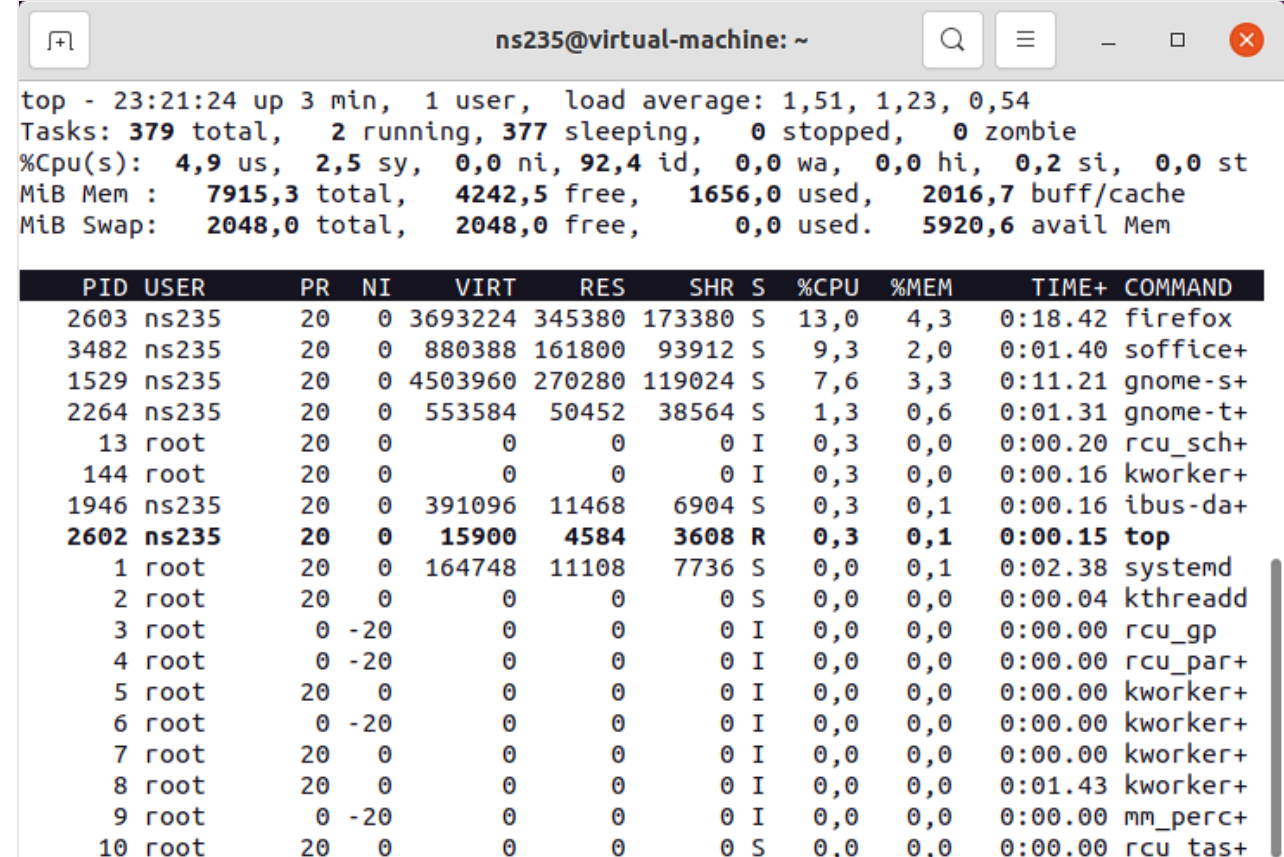
Görevlendirici - Dispatcher

- Görevlendirici modülü CPU'nun kontrolünü iş sıralayıcı tarafından seçilen prosese verir.
- Bu aşağıdaki işlemleri içerir:
 - Bağlam anahtarlama
 - Kullanıcı moduna değişim
 - Kullanıcı programını yeniden başlatmak için programdaki uygun bir konuma dallanma
- **Görevlendirme Gecikmesi** – bir prosesi sonlandırmak ve bir başkasını çalıştırmak için geçen süre.



İş Sıralama Kriterleri (Scheduling Criteria)

- **CPU kullanım oranı (utilization)**– CPU'nun mümkün olduğunca meşgul olmasını istiyoruz.
- Kavramsal olarak, CPU kullanımı %0 ile %100 arasında değişebilir.
- Gerçek bir sistemde,
 - hafif yüklü bir sistem için %40'tan
 - ağır yüklü bir sistem için %90'a kadar değişebilir.



```
top - 23:21:24 up 3 min, 1 user, load average: 1,51, 1,23, 0,54
Tasks: 379 total, 2 running, 377 sleeping, 0 stopped, 0 zombie
%Cpu(s): 4,9 us, 2,5 sy, 0,0 ni, 92,4 id, 0,0 wa, 0,0 hi, 0,2 si, 0,0 st
MiB Mem : 7915,3 total, 4242,5 free, 1656,0 used, 2016,7 buff/cache
MiB Swap: 2048,0 total, 2048,0 free, 0,0 used. 5920,6 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2603	ns235	20	0	3693224	345380	173380	S	13,0	4,3	0:18.42	firefox
3482	ns235	20	0	880388	161800	93912	S	9,3	2,0	0:01.40	soffice+
1529	ns235	20	0	4503960	270280	119024	S	7,6	3,3	0:11.21	gnome-s+
2264	ns235	20	0	553584	50452	38564	S	1,3	0,6	0:01.31	gnome-t+
13	root	20	0	0	0	0	I	0,3	0,0	0:00.20	rcu_sch+
144	root	20	0	0	0	0	I	0,3	0,0	0:00.16	kworker+
1946	ns235	20	0	391096	11468	6904	S	0,3	0,1	0:00.16	ibus-da+
2602	ns235	20	0	15900	4584	3608	R	0,3	0,1	0:00.15	top
1	root	20	0	164748	11108	7736	S	0,0	0,1	0:02.38	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.04	kthreadd
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par+
5	root	20	0	0	0	0	I	0,0	0,0	0:00.00	kworker+
6	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker+
7	root	20	0	0	0	0	I	0,0	0,0	0:00.00	kworker+
8	root	20	0	0	0	0	I	0,0	0,0	0:01.43	kworker+
9	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_perc+
10	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tas+

(CPU kullanımı, Linux, macOS ve UNIX sistemlerinde top komutunu kullanarak elde edilebilir.)

İş Sıralama Kriterleri (**Scheduling Criteria**)

- **Çıkış (Throughput)** – CPU işlemleri yürütüyorsa, iş yapılıyor demektir. İşin bir ölçüsü işlem başına zaman birimi olarak tamamlanan işlem sayısıdır ve buna "throughput" denir. Uzun işlemler için bu oran birkaç saniyede bir işlem olabilir; kısa işlemler için ise bu oran saniyede onlarca işlem olabilir.

Örneğin, bir veritabanı sunucusu düşünelim. Bu sunucu, kullanıcıların veri sorgularını işlemek için CPU'yu kullanır. Kullanıcılar birçok farklı türde sorgu gönderebilir: bazıları basit ve hızlı sonuçlar üretirken, diğerleri daha karmaşık ve daha uzun sürebilir.

Eğer sunucu CPU'su meşgul bir şekilde çalışıyorsa, örneğin bir saniyede ortalama **50** sorguyu işleyebiliyorsa, bu, sunucunun throughput'unun **50 sorgu/saniye** olduğunu gösterir. Bu, sunucunun ne kadar etkin bir şekilde çalıştığını ve kullanıcı taleplerini ne kadar hızlı işlediğini gösterir.

Öte yandan, eğer sunucu CPU'su daha az yoğun bir şekilde çalışıyorsa ve bir saniyede sadece **10** sorguyu işleyebiliyorsa, bu, sunucunun daha düşük bir throughput'a sahip olduğunu ve kullanıcı taleplerini işlemekte daha yavaş olduğunu gösterir.

Bu örnekte, CPU'nun yoğunluğu ve sorgu işleme hızı (throughput), sunucunun performansını ve kullanıcı deneyimini etkileyen önemli faktörlerdir.

İş Sıralama Kriterleri (**Scheduling Criteria**)

- **Tamamlanma (Turnaround) zamanı** – Turnaround süresi, bir işlemin yürütülmesi için ne kadar zaman gerektiğini belirleyen önemli bir ölçüttür. Bir işlemin gönderilmesinden tamamlanmasına kadar geçen süredir.
- Turnaround süresi,
 - hazır kuyrukta beklenen süre,
 - CPU'da yürütme süresi ve
 - Giriş/Çıkış işlemleri için harcanan sürenin toplamıdır.

Bu süre, bir işlemin başlatılmasından tamamlanmasına kadar olan toplam süreyi ifade eder. Özetle, bir işin tamamlanması için geçen süredir.

İş Sıralama Kriterleri (**Scheduling Criteria**)

- **Bekleme (Waiting) zamanı** – Bekleme süresi, bir işlemin hazır kuyrukta bekleyerek geçirdiği sürenin toplamıdır.
- CPU zamanlama algoritması, bir işlemin yürütme veya Giriş/Çıkış işlemleri süresini etkilemez. Yalnızca bir işlemin hazır kuyrukta beklediği süreyi etkiler. Bekleme süresi, hazır kuyrukta geçirilen sürelerin toplamıdır. Bu süre, bir işlemin CPU'ya atanmak için beklediği toplam süreyi ifade eder.
- Özetle, bir işin hazır kuyrukta beklemek için harcadığı süredir.

İş Sıralama Kriterleri (**Scheduling Criteria**)

- **Cevap (Response) zamanı**– Tepki süresi, bir kullanıcının bir işlem başlatma isteği gönderdiği andan, sistem ilk yanıtı üretmeye başlayıncaya kadar geçen süredir. Bu süre, kullanıcıdan gelen bir talebin karşılanmaya başladığı anı ifade eder.
- Örneğin, bir kullanıcı bir dosya açma isteği gönderdiğinde, tepki süresi, sistem dosyayı açmaya başladığı andır, dosyanın tamamen açılması veya işlem tamamlanması değildir.
- Tepki süresi, kullanıcı deneyimini değerlendirmek için önemli bir ölçüttür çünkü kullanıcıların hızlı bir yanıt alması, sistemin etkinliğini ve kullanılabilirliğini gösterir.

İş Sıralama Algoritması Optimizasyon Kriterleri

CPU zamanlama, hazır kuyruğundaki işlemler arasından CPU çekirdeğinin hangisine tahsis edileceğine karar verme problemiyle ilgilenir.

- Max CPU kullanımı
- Max çıkış
- Min tamamlanma zamanı
- Min bekleme zamanı
- Min cevap zamanı

İş Sıralama Algoritmaları

CPU scheduling algoritmaları, hazır kuyruğunda bekleyen process'lerden hangisinin CPU'ya atanacağını belirlerler.

- First-Come, First-Served Scheduling
- Shortest-Job-First Scheduling
- Priority Scheduling
- Round-Robin Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

İlk Gelen İlk Çalışır Algoritması (**First-Come, First-Served - FCFS**)

<u>Proses</u>	<u>Patlama (Burst) Zamanı</u>
P_1	24
P_2	3
P_3	3

- Proseslerin P_1, P_2, P_3 sırasında geldiğini varsayalım

Gantt Diyagramı :



Bekleme zamanları: $P_1 = 0$; $P_2 = 24$; $P_3 = 27$;

Ortalama bekleme zamanı: $(0 + 24 + 27) / 3 = 17$ milisaniye

FCFS İş Sıralama (Devam)

Proseslerin aşağıdaki sırada geldiğini varsayalım:

P_2, P_3, P_1

Gantt diyagramı :



Bekleme zamanı $P_1 = 6; P_2 = 0; P_3 = 3$

Ortalama bekleme zamanı: $(6 + 0 + 3)/3 = 3$ milisaniye < Bir önceki örnek 17 olduğundan daha performanslıdır.

Az önceki durumdan daha iyi

Konvoy etkisi – **uzun** proseslerin **kısa** proseslerden önce gelmesi

Bir adet CPU-bağımlı proses ve birden fazla I/O-bağımlı prosesler durumu gibi

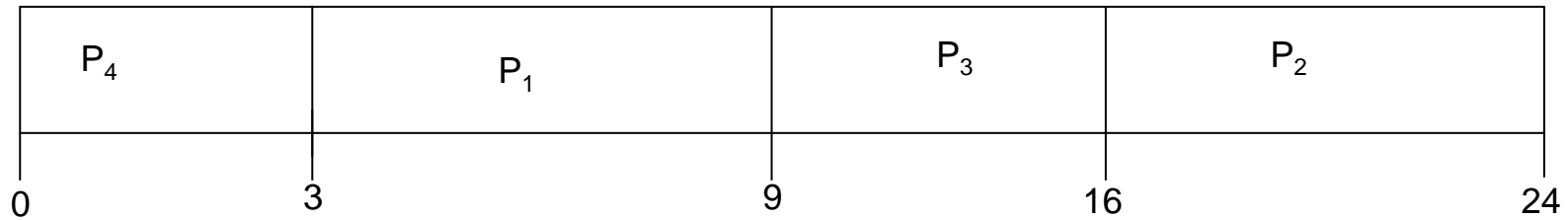
En Kısa İş Önce (Shortest-Job-First - SJF) Algoritması

- Her bir proses bir sonraki CPU patlama süresiyle ilişkilendirir
 - En kısa zamanlı prosesini tespit etmek için bu zaman değerlerini kullan
- SJF en uygundur – verilen bir grup proses için minimum ortalama bekleme zamanına neden olur.
- **Zorluk bir sonraki CPU isteğinin patlama süresinin hesaplanmasındadır.**
 - Kullanıcıdan istenir
 - veya hesaplanır

En Kısa İş Önce (SJF) Örneği

<u>Proses</u>	<u>Patlama (Burst) Zamanı</u>
P_1	6
P_2	8
P_3	7
P_4	3

- SJF iş sıralama diyagramı



- Ortalama bekleme zamanı = $(3 + 16 + 9 + 0) / 4 = 7$

Bir Sonraki CPU Patlama Zamanının Belirlenmesi

- Süre sadece tahmin edilebilir – bir öncekine benzer olur.
 - Bir sonraki CPU patlaması tahmin edilen en kısa süreli prosesi al.
- Üstel ortalama ve bir önceki CPU patlama süresi kullanılarak hesaplanabilir.

1. t_n = actual length of n^{th} CPU burst

2. τ_{n+1} = predicted value for the next CPU burst

3. $\alpha, 0 \leq \alpha \leq 1$

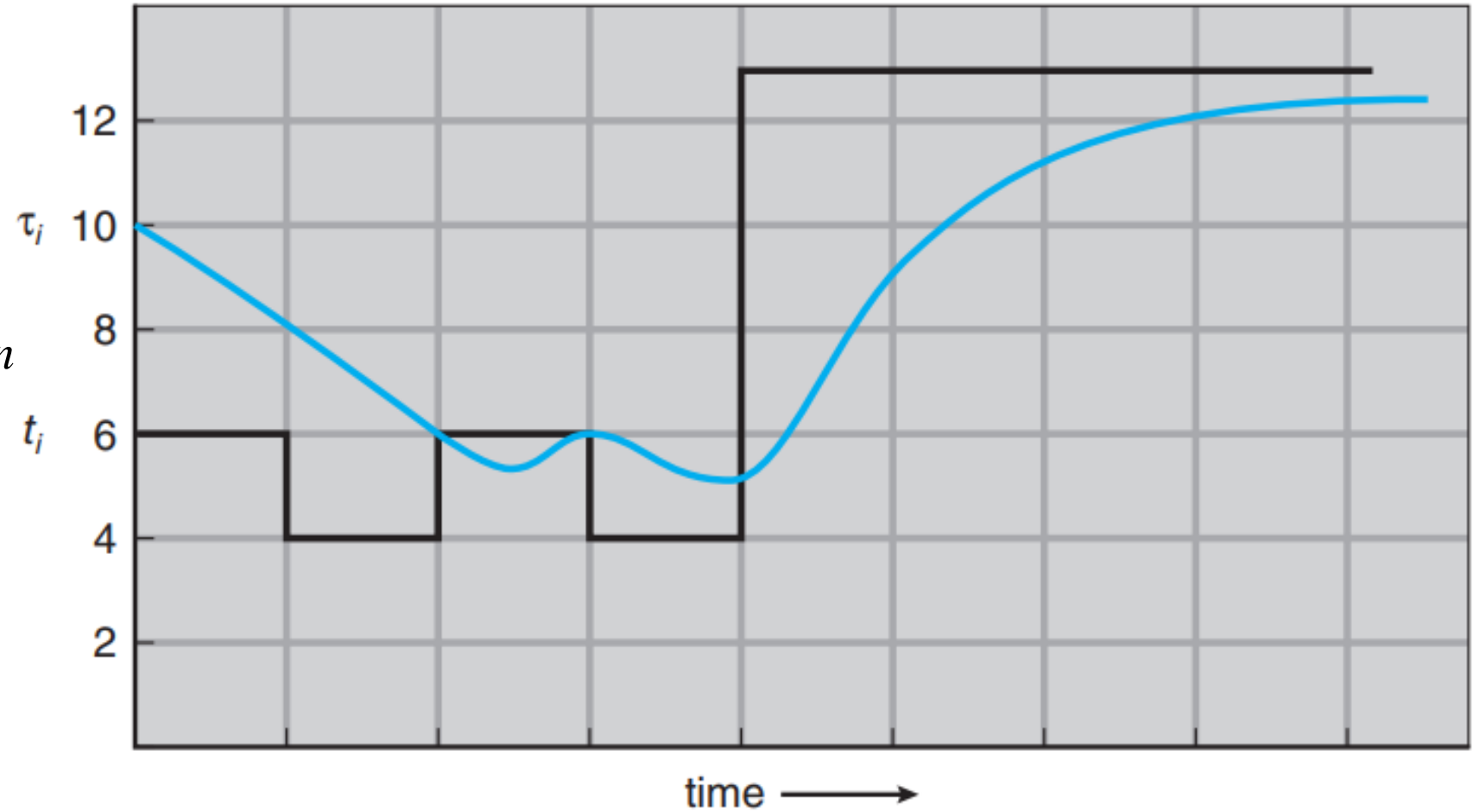
4. Define :

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

- Genelde , α ½ seçilir

Bir Sonraki CPU Patlamasının Süresinin Tahmini

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



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

Üssel Ortalama Örnekleri

- $\alpha = 0$

- $\tau_{n+1} = \tau_n$
- Yakın zaman bilgisi kullanılmıyor.

- $\alpha = 1$

- $\tau_{n+1} = \alpha t_n$
- Sadece son CPU kullanım bilgisi hesaba katılıyor.

- Eğer formülü genişletirsek:

$$\begin{aligned}\tau_{n+1} = & \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \dots \\ & + (1 - \alpha)^j \alpha t_{n-j} + \dots \\ & + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

- Hem α hem de $(1 - \alpha)$ 1 veya birden küçük olduğundan, her bir takip eden terim, kendinden önce gelen terimden daha az ağırlığa sahiptir.

En Kısa Kalan Zaman Önce Sıralama - Shortest Remaining Time First (SRTF)

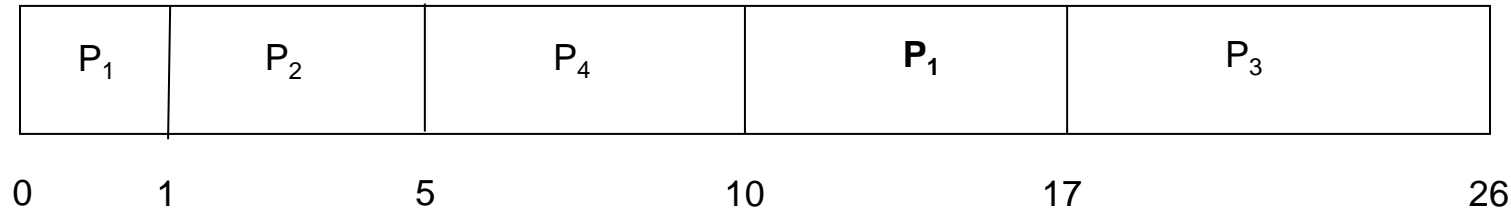
- SJF'nin **kesintili sürümü**
- Hazır kuyruğuna yeni bir proses geldiğinde, bundan sonra hangi prosesin seçileceği kararı, SJF algoritması kullanılarak yeniden yapılır.
- SRTF, belirli bir proses kümesi için minimum ortalama bekleme süresi açısından SJF'den daha "optimal" midir?

En Kısa Kalan Zaman Önce Örneği

- Bu örnekte farklı varış zamanı ve kesintili olan prosesleri analiz ederiz.

<u>Proses</u>	<u>Varış Zamanı</u>	<u>Patlama Zamanı</u>
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- Kesintili* SJF Gantt Diyagramı



- İşlem P1 zaman 0'da başlatılır, çünkü bu işlem kuyruktaki tek işlemdir. İşlem P2 zaman 1'de gelir. İşlem P1 için kalan süre (7 milisaniye), işlem P2 için gereken süreden (4 milisaniye) daha fazladır, bu nedenle işlem P1 kesintiye uğrar ve işlem P2 planlanır.
- Ortalama Bekleme zamanı = $[(10-1)+(1-1)+(17-2)+5-3]/4 = 26/4 = 6.5$ msec
- Kesintisiz SJF zamanlaması kullanılsaydı, ortalama bekleme süresi kaç milisaniye olurdu?

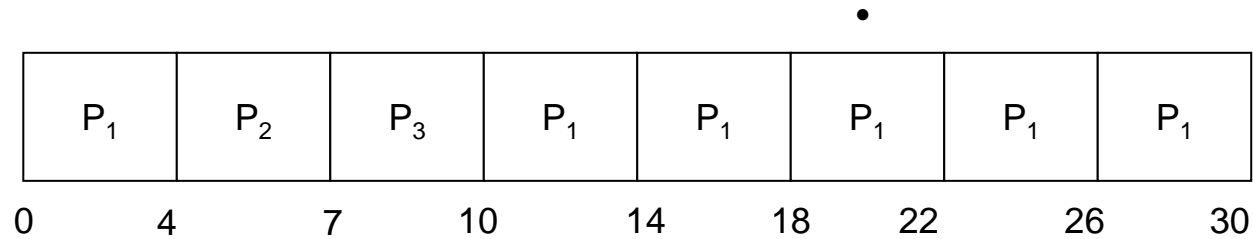
Çevrimsel Sıralı (Round Robin - RR)

- Her bir proses genellikle 10-100 milisaniye arası küçük bir zaman süresince CPU'ya sahip olur .
- Bu zaman değerine **kuantum zamanı** denir ve q ile gösterilir.
- Bu süre tamamlandıktan sonra proses kesilir ve hazır kuyruğunun sonuna eklenir.
- Eğer hazır kuyruğunda n adet proses varsa ve kuantum zamanı değeri q ise her bir proses $1/n$ kadar CPU 'yu elde eder. Hiçbir proses $(n-1)q$ süresinden daha fazla beklemez
- Zamanlayıcı bir sonraki prosesi devreye almak için her bir kuantum süresi sonunda keser.
- Performans
 - q büyük \Rightarrow FIFO
 - q küçük $\Rightarrow q$ bağlam değişimine göre büyük olmalıdır aksi halde sistem kasılır

Round Robin Örneği ($q = 4$)

<u>Process</u>	<u>Patlama Zamanı</u>
P_1	24
P_2	3
P_3	3

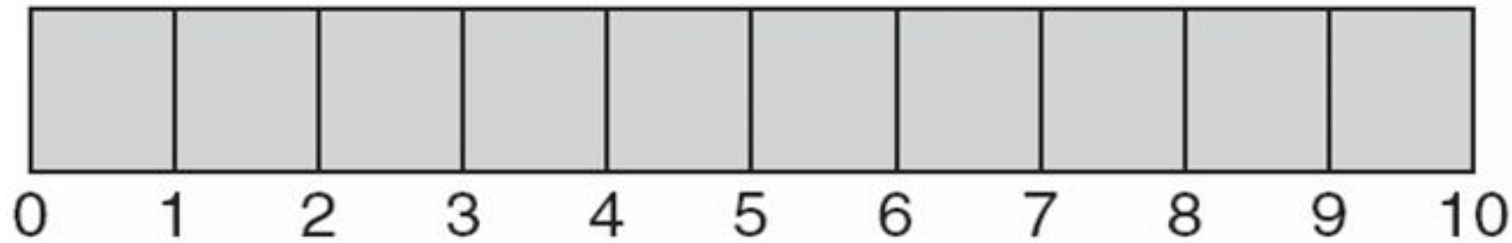
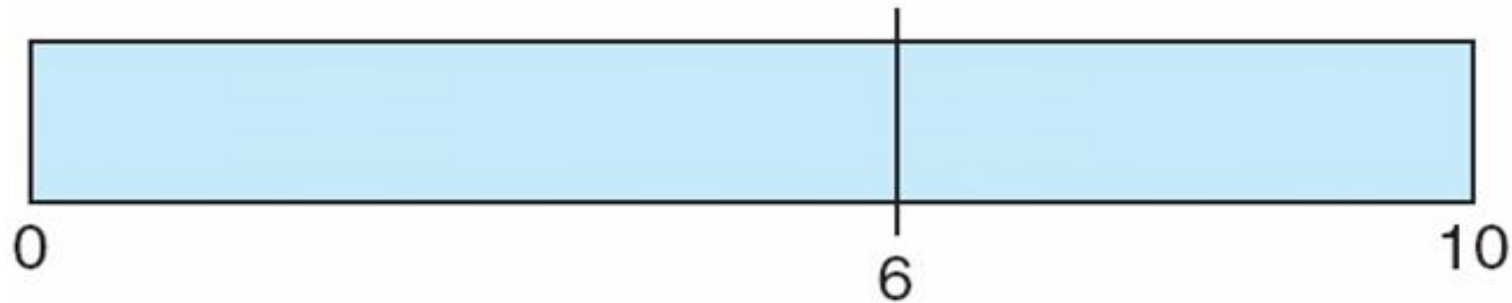
- Gantt diyagramı:



- Tipik olarak, SJF'den daha yüksek bir tamamlanma zamanı, ancak daha iyi bir cevap zamanına sahiptir.
- q bağlam değişimi zamanına göre büyük olmalıdır
 - q genellikle 10ms ila 100ms,
 - bağlam değişimi < 10 mikrosaniye

Kuantum ve Bağlam Değişim Zamanı

process time = 10



quantum

12

6

1

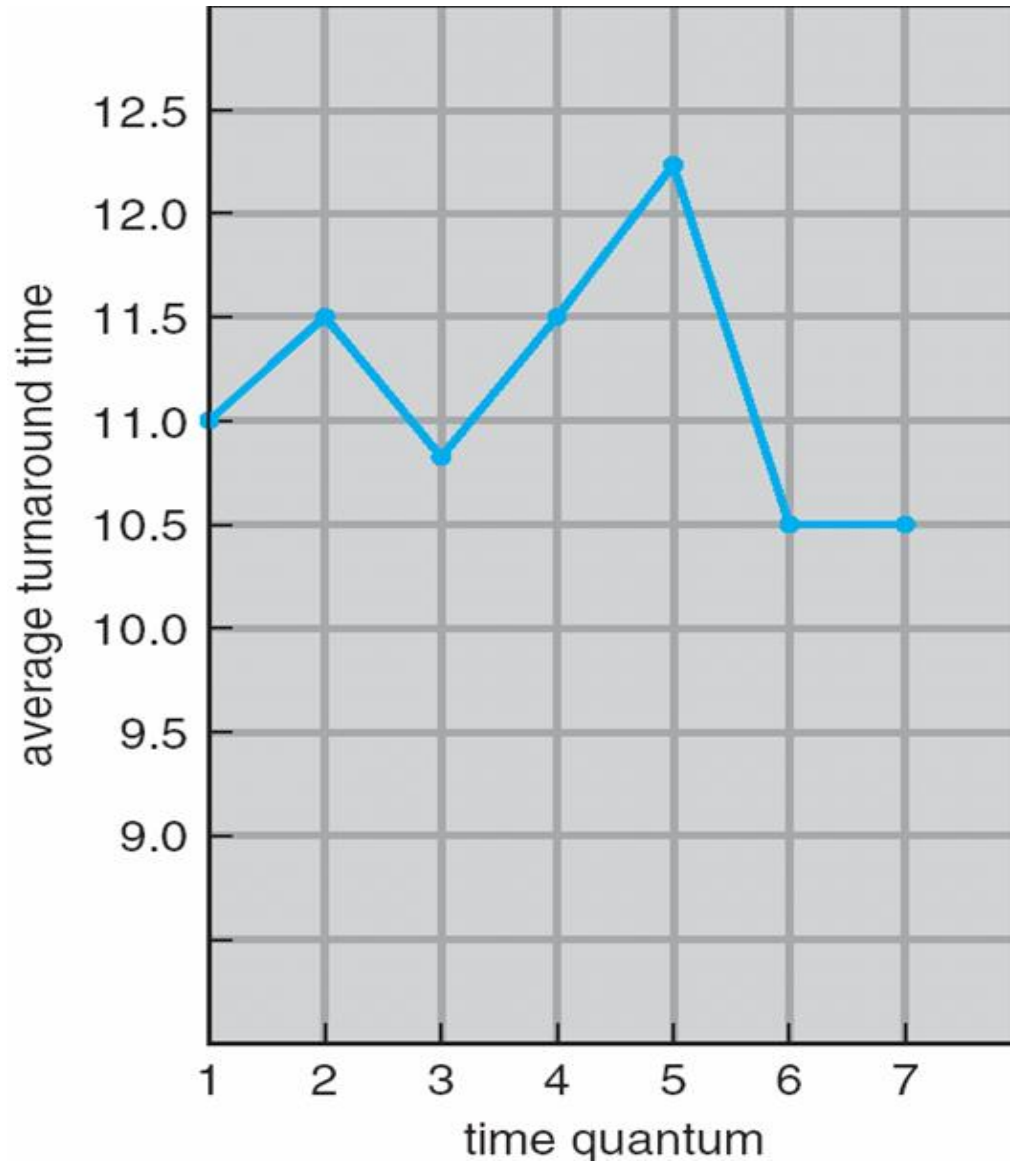
context
switches

0

1

9

Tamamlanma Zamanının Kuantum Zamanıyla Değişimi



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Zaman dilimi, bağlam değiştirme zamanı ile karşılaştırıldığında büyük olmalıdır, ancak çok büyük olmamalıdır. Zaman dilimi çok büyükse, RR zamanlaması FCFS politikasına dönüşür.

Kural olarak, CPU patlamalarının %80'inin zaman diliminden daha kısa olması gerektiği söylenebilir.

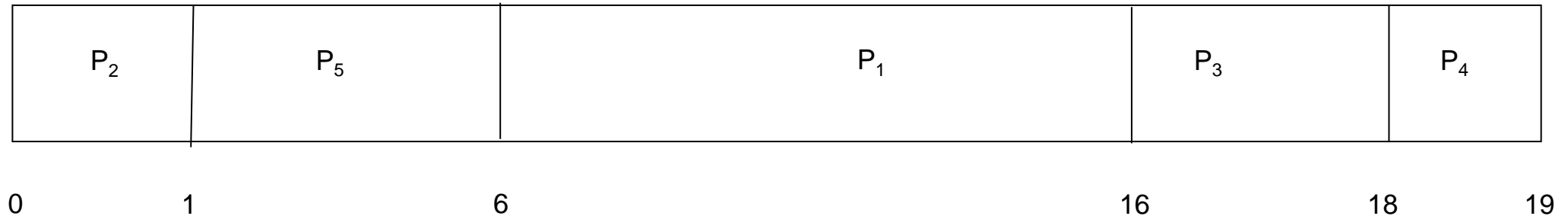
Öncelikli İş Sıralama

- Bir öncelik sayısı (tamsayısı) her bir prosese atanır
- CPU en yüksek öncelik değerine sahip prosese tayin edilir (**en küçük tamsayı = en yüksek öncelik**)
 - Kesintili
 - Kesintisiz
- SJF öncelik değerinin tahmini bir sonraki CPU patlama zamanının tersinin olduğu bir öncelikli iş sıralama yaklaşımıdır
- Problem = **Açlıktan Ölme (Starvation)** – düşük öncelikli prosesler hiç çalışmayabilir.
- Çözüm = **Yaşlandırma** – zaman ilerlerken proses önceliğini artır.
- Not: SJF, önceliğin tahmin edilen bir sonraki CPU patlama süresinin tersi olduğu öncelikli bir sıralama algoritmasıdır.

Öncelikli İş Sıralama Örneği

<u>Proses</u>	<u>Patlama Zamanı</u>	<u>Öncelik</u>
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Öncelikli İş Sıralama Gantt Diyagramı

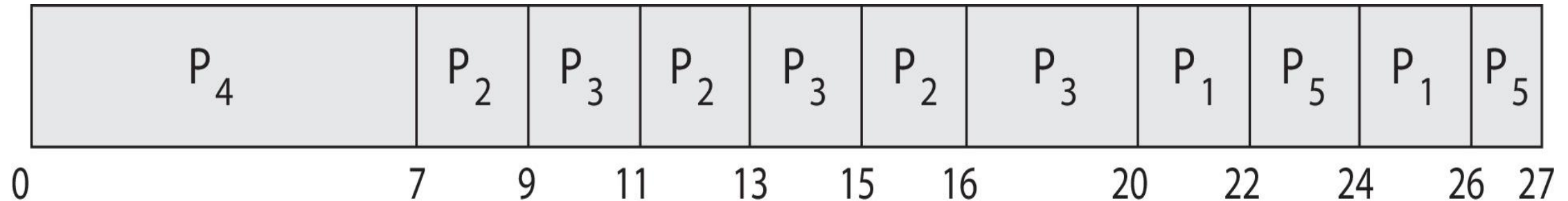


- Ortalama bekleme zamanı = 8.2 msec

Çevrimsel Sıralı + Öncelikli Sıralama

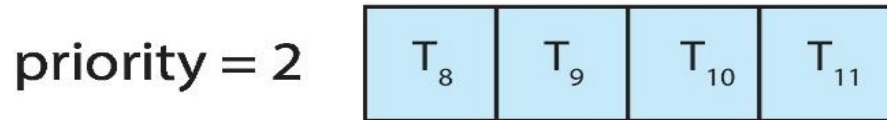
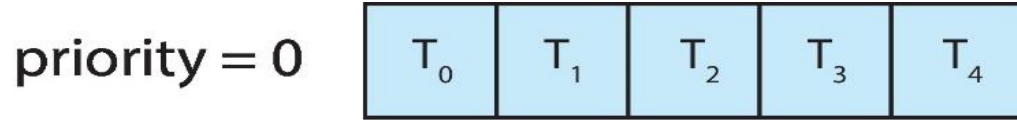
<u>Process</u>	<u>Patlama Zamanı</u>	<u>Öncelik</u>
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- En yüksek öncelikli prosesi çalıştır. Öncelik aynı ise çevrimsel sıralı
- Gantt Diyagramı, quantum = 2



Çok Seviyeli Kuyruk

- Her öncelik için ayrı kuyruğun olduğu öncelikli sıralama
- En yüksek öncelikli kuyruktaki prosesler en önce sıralanır

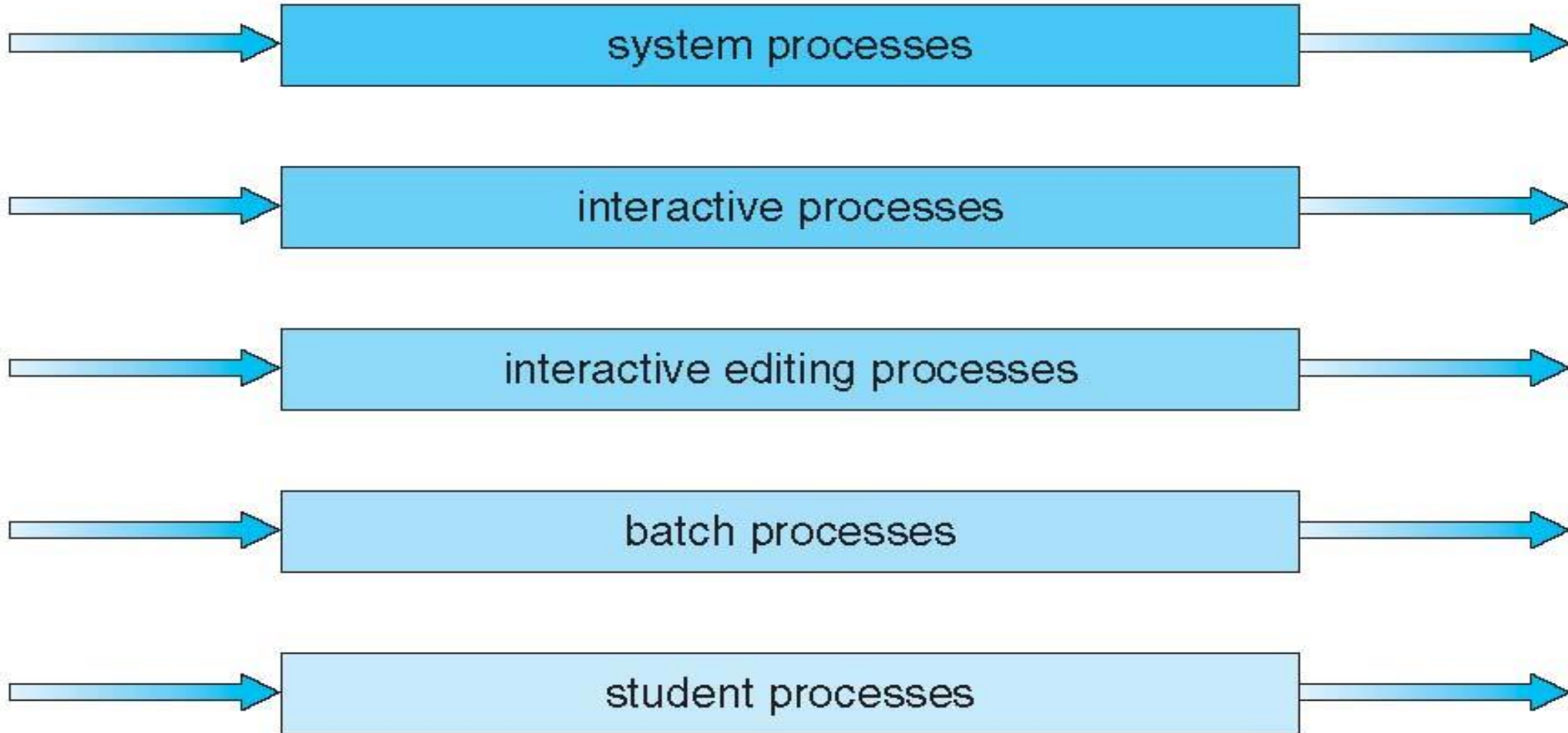


Çok Seviyeli Kuyruk

- Hazır kuyruğu ayrı kuyruklar halinde düzenlenir, ör:
 - Ön plan (interaktif)
 - Arkaplan (toplu iş - batch)
- Proses sürekli bir kuyruktadır
- Her kuyruk kendi iş sıralama algoritmasına sahiptir.
- ön plan – çevrimsel sıralı - RR
- arka plan – FCFS
- İş sıralama kuyruklar arasında yapılmalıdır:
 - Sabit öncelikli iş sıralama (ön plandakilerin tümü bittikten sonra arka plandakilere hizmet edilir). Ölüm olasılığı.
 - Zaman dilimli iş sıralama– her kuyruk belirli bir CPU zamanını elde eder ve prosesleri arasında paylaştırır, RR’ de ön plana kadar 80%
 - FCFS’de arka plan için 20%

Çok Seviyeli Kuyruk

highest priority



lowest priority

Çok Seviyeli Geri Beslemeli Kuyruk

- Bir proses çeşitli kuyruklar arasında hareket edebilir;
- Bir tür yaşlandırma - aging uygulamasıdır
- Çok seviyeli-geri besleme kuyruğu iş sıralama işlemi aşağıdaki parametreler ile tanımlanır :
 - Kuyruk sayısı
 - Her kuyruğun iş sıralama algoritması
 - Bir prosesin ne zaman bir üst kuyruğa geçeceğini belirleme yöntemi
 - Bir prosesin ne zaman bir alt kuyruğa geçeceğini belirleme yöntemi
 - Bir prosesin çalışmaya ihtiyaç duyduğunda hangi kuyruğa ekleneceğini belirleme yöntemi

Çok Seviyeli Geri Beslemeli Kuyruk Örneği

- Üç adet kuyruk:
 - Q_0 – 8 milisaniye kuantum değerine sahip RR
 - Q_1 – 16 milisaniye kuantumlu RR
 - Q_2 – FCFS
- İş Sıralama
 - Yeni bir görev Q_0 kuyruğuna girer ve FCFS olarak hizmet görür
 - CPU'yu ele geçirdiğinde 8 milisaniyesi vardır
 - 8 milisaniyede işini bitiremezse, Q_1 kuyruğuna geçer
 - Q_1 kuyruğunda görev yine FCFS gibi işlenir ve ilave 16 milisaniye alır
 - hala işini tamamlayamazsa kesilir ve Q_2 ye iletilir.

