Bölüm 5: Çizelgeleme

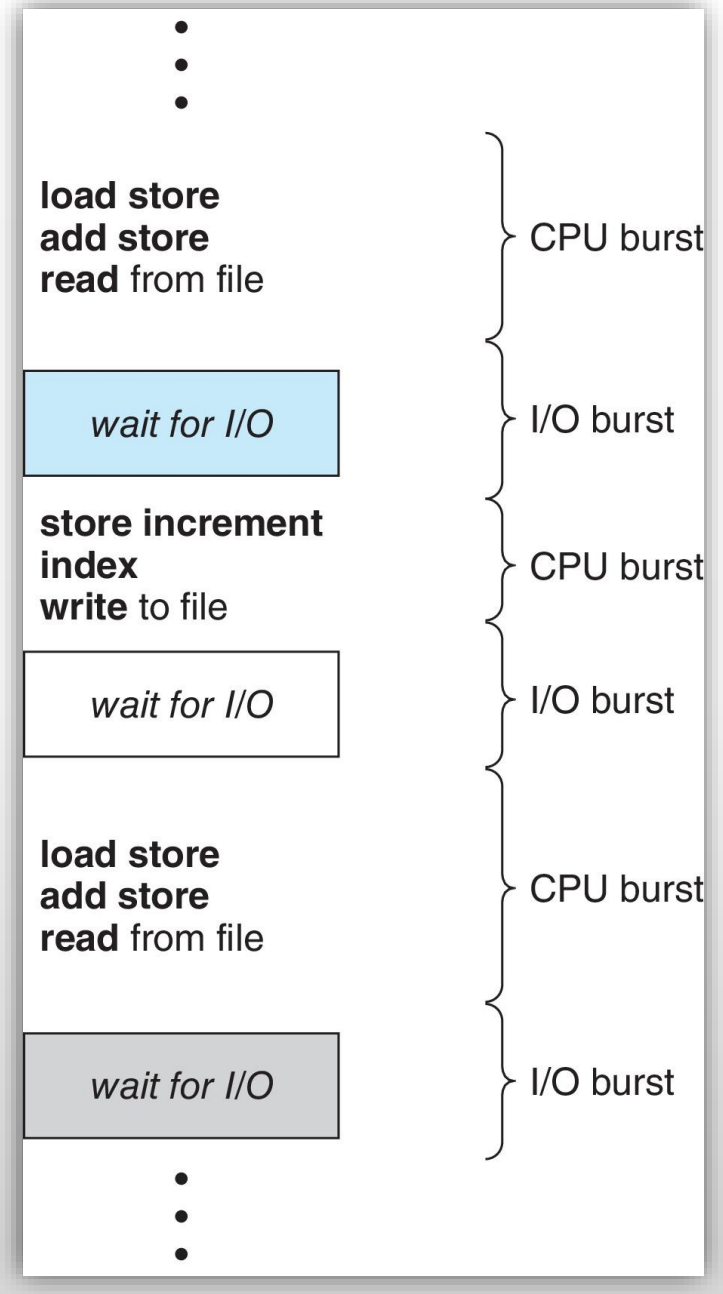
İşletim Sistemleri

Çizelgeleme (Scheduling)

- İşletim sisteminin etkin bir şekilde kaynakları yönetmesini sağlar.
- Belirlenen bir algoritma ile işlemciye işlemlerin sırayla atanması sürecidir.
- Hazır Kuyruğu (*Ready Queue*): Atanmayı bekleyen süreçler.
- Bekleme (*Waiting*): Giriş/Çıkış işlemleri tamamlanmayı bekleyen süreçler.
- Çalışan (*Running*): CPU üzerinde yürütülen aktif süreç.

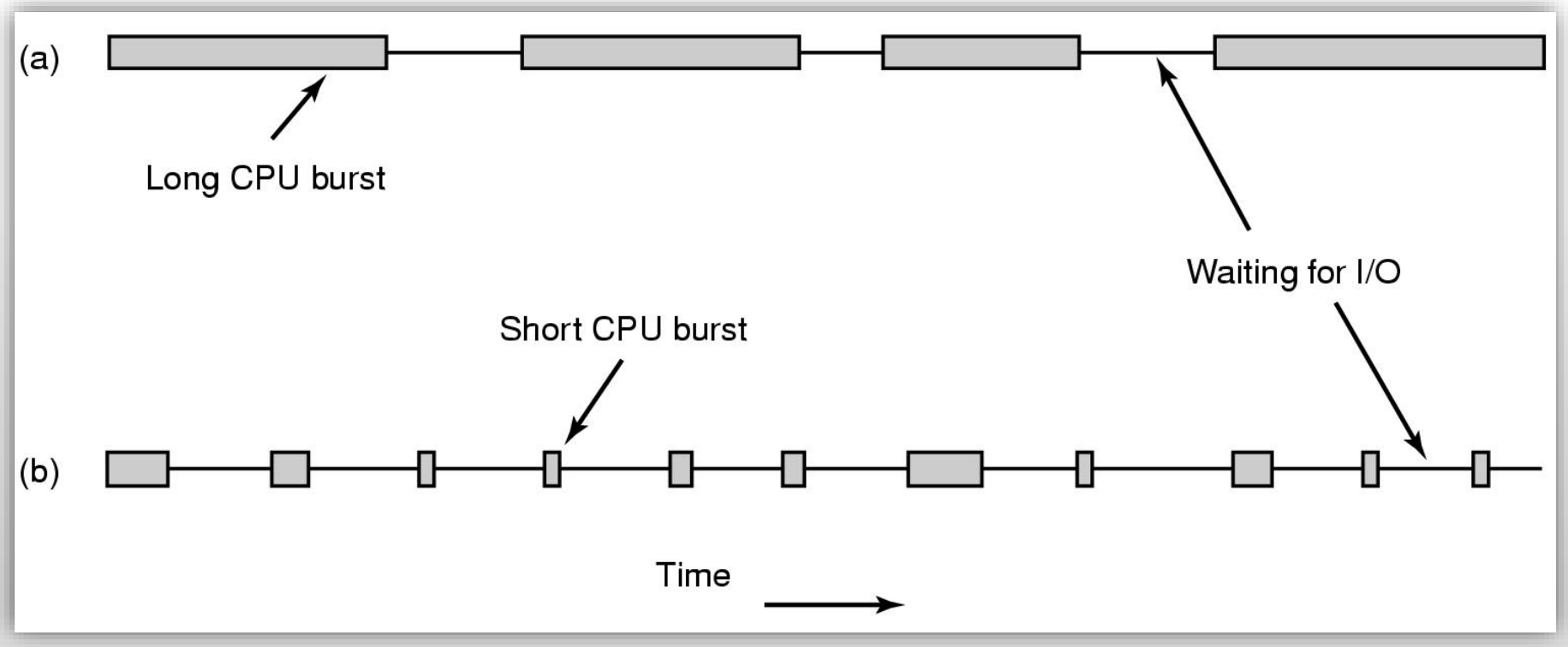
Çizelgeleme

- Çoklu programlama ile
  - İşlemci maksimum düzeyde kullanılır.
- CPU yürütme döngüsünü,
  - G/Ç beklemesi takip eder.
- CPU kaynak dağıtımı önemli!



İşlemci Kullanımı

(a) CPU'ya bağlı (*bound*) bir işlem. (b) G/Ç'ye bağlı bir işlem.



İşlemci Çizelgeleyici

- Hazır kuyruğunda bekleyen süreçler arasından seçim yapar.
  - Öğeler kuyrukta çeşitli şekillerde sıralanmış olabilir.
- Çizelgeleme kararları; bir süreç,
  - Çalışır durumundan bekleme durumuna geçtiğinde,
  - Çalışır durumundan hazır durumuna geçtiğinde,
  - Bekleme durumundan hazır durumuna geçtiğinde, ve
  - Sonlandığında, alınır.

Görev Dağıtıcı (Dispatcher)

- Çizelgeleyici tarafından seçilen sürece, işlemcinin kontrolünü verir.
  - Bağlam anahtarlama,
  - Kullanıcı moduna geçiş,
  - Kullanıcı programında uygun konuma atlama.
- **Görev dağıtıcı gecikmesi**:
  - Bir sürecin durdurulup başka sürecin başlatılmasına kadar geçen süre.

Kriterler

- **CPU kullanımı** – CPU mümkün olduğunca meşgul tutulmalı.
- **Verim** – birim zamanda yürütülmesi tamamlanan süreç sayısı.
- **Geri dönüş süresi** - bir süreci sonlandırmak için geçen süre.
- **Bekleme süresi** - bir sürecin hazır kuyruğunda bekleme süresi.
- **Yanıt süresi** – verilen bir komuta, ilk yanıtın üretilme süresi.

Çizelgelemenin Hedefleri

- Tüm sistemler
  - **Adalet** - her sürece CPU'dan adil bir pay vermek.
  - **Politika** - belirtilen politikanın yürütüldüğünü görmek.
  - **Denge** - sistemin tüm parçalarını meşgul tutmak.
- Toplu sistemler
  - **Verim** – birim zamanda yapılan işi maksimize etmek.
  - **Geri dönüş süresi** – başlatma ve sonlanma arası süreyi azaltmak.
  - **CPU kullanımı** - CPU'yu olabildiğince meşgul tutmak.

Çizelgelemenin Hedefleri

- Etkileşimli sistemler
  - **Yanıt süresi** - isteklere hızlı yanıt vermeli.
  - **Orantılılık** - kullanıcıların beklentilerini karşılamalı.
- Gerçek zamanlı sistemler
  - **Son teslim zamanı** (*deadline*) - veri kaybı olmamalı.
  - **Öngörülebilirlik** - multimedya sistemlerinde kalite düşmemeli.

Çizelgeleme

- Bir sonraki adımda hangi süreç çalıştırılacak?
- Bir süreç çalışırken işlemci,
  - süreç sonlanana kadar çalıştırmalı mı?
  - farklı süreçler arasında geçiş yapmalı mı?
- Süreç değiştirme pahalı
  - Kullanıcı modu ve çekirdek modu arasında geçiş.
  - Geçerli süreç bilgileri kaydedilir.
  - Bellek haritası (*memory map*) kaydedilir.
  - Önbellek temizlenir ve yeniden yüklenir.

Kavramlar

- **Önleyici** (*preemptive*) algoritma
  - Bir süreç, belirli bir zaman aralığının sonunda hala çalışıyor ise, askıya alınır ve başka bir süreç çalıştırılır.
- **Önleyici olmayan** (*non-preemptive*) algoritma
  - Çalıştırmak için bir süreç seçilir ve bloke olana kadar veya gönüllü olarak işlemciyi serbest bırakana kadar çalışmasına izin verilir.

Çizelgeleme Kategorileri

- Farklı ortamlar farklı çizelgeleme algoritmalarına ihtiyaç duyar.
- **Toplu** (*batch*)
  - Hala yaygın olarak kullanılıyor.
  - Önleyici olmayan algoritmalar süreç geçişlerini azaltır.
- **Etkileşimli**
  - Önleyici (*preemptive*) algoritmalar gerekli.
- **Gerçek zamanlı**
  - Süreçler hızlı çalışır ve bloke olur.

Toplu Sistemlerde Çizelgeleme

- **İlk gelen ilk hizmet alır** (*first-come first-served*)
  - Süreçler, kuyruğa girdikleri sırada yürütülür. Uzun süren süreçlerde düşük performansa, kısa işler için uzun bekleme süresine neden olur.
- **Önce en kısa iş** (*shortest job first*)
  - Kısa süren süreçler, uzun olanlara göre önceliklidir. CPU kullanımını üst düzeye çıkarır. Süreçlerin yürütme süresini tahmin etmek *zor*.
- **Önce en kısa süresi kalan** (*shortest remaining time next*)
  - Sonlanması için kısa süre gereken süreçler önce yürütülür. Dinamik bir yaklaşım, yürütme süresi değiştikçe kendini ayarlar. Her süreç için kalan süreyi takip etme ve kuyrukta sık değişiklik yapma maliyeti var.

FCFS Çizelgeleme

- Ağırlıklı geri dönüş süresi = geri dönüş süresi / servis süresi

Süreç	Gelme zamanı	Servis süresi	Başlama zamanı	Sonlanma zamanı	Geri dönüş süresi	Ağırlıklı geri dönüş süresi
A	0	1	0	1	1	1
B	1	100	1	101	100	1
C	2	1	101	102	100	100
D	3	100	102	202	199	1.99

FCFS Çizelgeleme

- İlk gelen süreç, CPU'yu ilk kullanma hakkına sahiptir.
- Süreçler arasında adil bir kaynak dağılımı sağlar.
- Uzun süren süreçler kısa süren süreçlerin beklemesine neden olur.
  - Konvoy etkisi. (*convoy effect*)
  - Bekleme süreleri değişkenlik gösterebilir.
- G/Ç bağlı süreçler az CPU ihtiyacına rağmen,
  - Sonlanana kadar diğer süreçlere şans vermezler.
- Tahmin edilebilir (*deterministic*) değil.

FCFS Çizelgeleme

```
// FCFS Stratejisi
void fcfs(struct Process processes[], int n) {
    int waitingTime = 0;
    // Süreçleri sırayla işle
    for (int i = 0; i < n; i++) {
        waitingTime += processes[i].burstTime;
        printf("Süreç %d için Bekleme Süresi: %d\n",
                        processes[i].processID, waitingTime);
    }
}
```

FCFS Örnek Senaryo

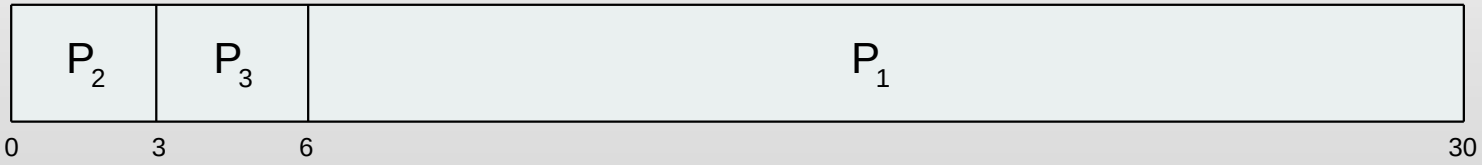
Süreç	Servis süresi
P1	24
P2	3
P3	3

- Süreçlerin çalışma sırası: **P1 -> P2 -> P3**
- Bekleme süresi: P1 = 0; P2 = 24; P3 = 27
- Ortalama bekleme süresi: (0 + 24 + 27) / 3 = 17

FCFS Örnek Senaryo

Süreç	Servis süresi
P1	24
P2	3
P3	3

- Süreçlerin çalışma sırası: **P2 -> P3 -> P1**
- Bekleme süresi: P1 = 6; P2 = 0; P3 = 3
- Ortalama bekleme süresi: (6 + 0 + 3) / 3 = 3



SJF Çizelgeleme

- Her süreç, servis süresi ile ilişkilendirilir.
- Çizelgeleme aşamasında servis süreleri göz önüne alınır.
- En kısa süren süreç, diğer süreçlerden önce çalıştırılır.
- Optimal bir algoritmadır.
  - Ortalama bekleme süresi minimumdur.
  - Kısa süren süreçler, diğerlerinin tamamlanmasını beklemez.
- Servis süresi nasıl hesaplanacak?
  - Kullanıcıya sorulabilir.
  - Tahmin edilebilir.

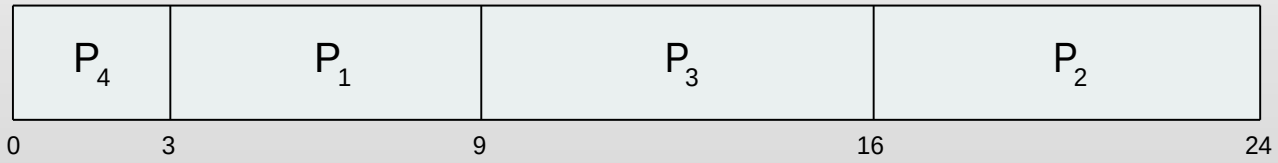
SJF Çizelgeleme

```
void sjf(struct Process proc[], int n) {
    int waitingTime = 0;
    for (int i = 0; i < n - 1; i++) {  // Süreçleri sırala
        for (int j = 0; j < n - i - 1; j++) {
            if (proc[j].burstTime > proc[j + 1].burstTime) {
                // Swap
        }
    }
    for (int i = 0; i < n; i++) {  // Süreçleri sırayla işle
        waitingTime += proc[i].burstTime;
    }
}
```

SJF Örnek Senaryo

Süreç	Servis süresi
P1	6
P2	8
P3	7
P4	3

- Bekleme süresi: P1 = 3; P2 = 16; P3 = 9; P4 = 0
- Ortalama bekleme süresi: (3 + 16 + 9 + 0) / 4 = 7



SJF Çizelgeleme

(a) Orijinal sıra.

8	4	4	4
A	B	C	D

(b) En kısa süreci önce yürütme sırası.

4	4	4	8
B	C	D	A

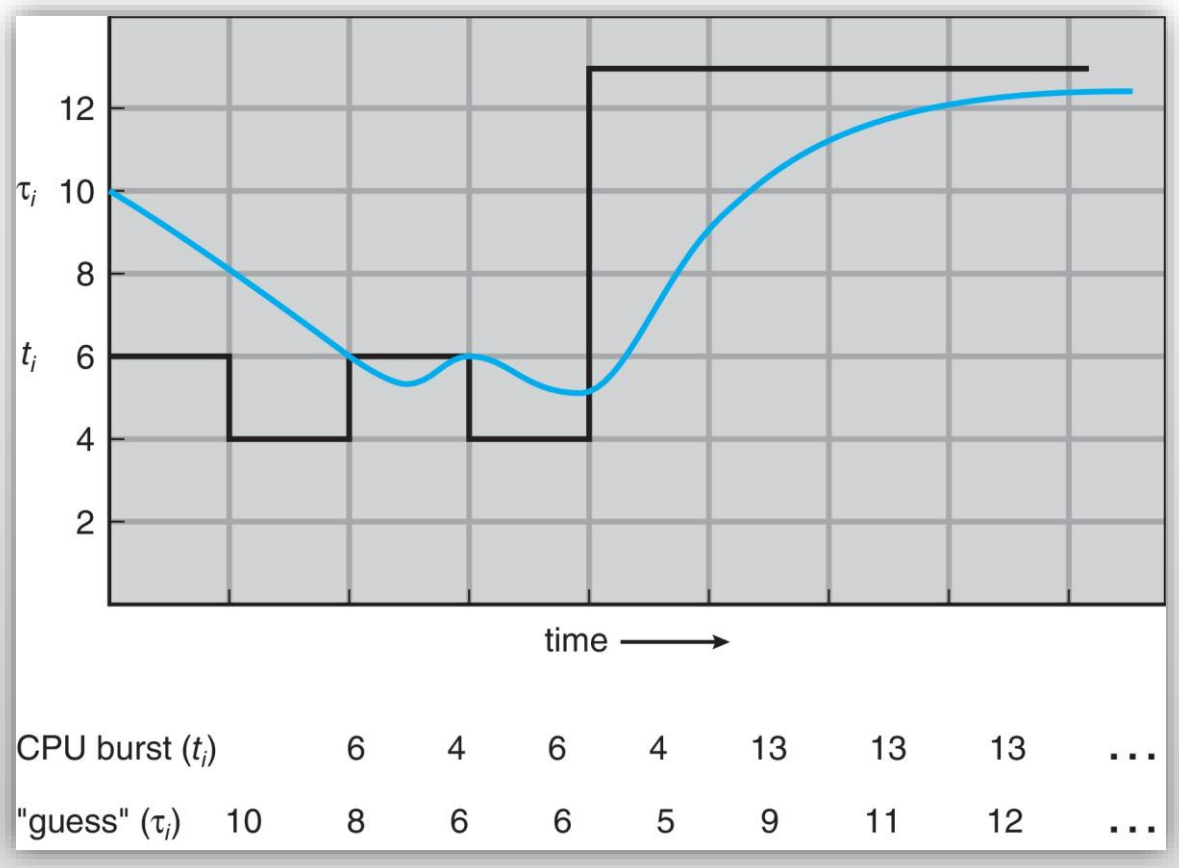
FCFS ve SJF Karşılaştırma

	Süreç	A	B	C	D	E	ortalama
	Geliş	0	1	2	3	4	
	Servis	4	3	5	2	4	
SJF	**Bitiş**	4	9	**18**	6	13	
	Dönüş	4	8	16	3	9	8
	Ağırlıklı	1	2.67	3.1	1.5	2.25	2.1
FCFS	**Bitiş**	4	7	12	14	**18**	
	Dönüş	4	6	10	11	14	9
	Ağırlıklı	1	2	2	5.5	3.5	2.8

SRTN Çizelgeleme

- SJF algoritmasının kesmeli (*preemptive*) versiyonu.
- Sıradaki yürütme için en kısa süreye sahip süreç seçilir.
- **Kesmeli (preemptive)**: yeni sürecin çalışma süresini, aktif sürecin sonlanmasına kalan süresiyle karşılaştırır.
  - Süreçlerin çalışma sürelerinin önceden bilinmesi gerekir.
  - Sürecin geçmişteki servis sürelerinden tahmin edilebilir.
  - Üstel ortalama (*Exponential averaging*) yöntemi kullanılabilir.
    - $s_0 = x_0$
    - $s_t = ax_t + (1 - a)\, s_{t-1},\ t > 0$
    - $0 < a < 1$

Üstel Ortalama



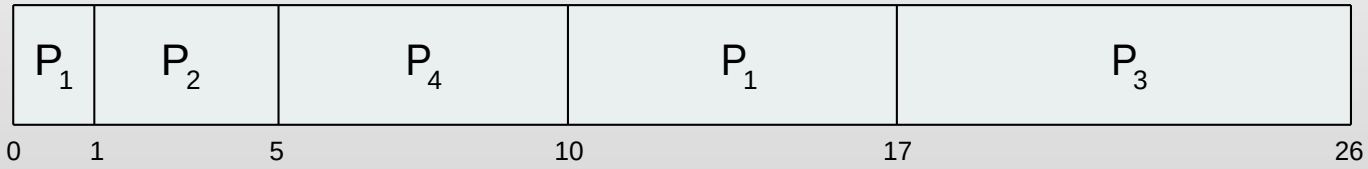
SRTN Çizelgeleme

```
void srtn(struct Process processes[], int n) {
    int remain[n];
    while (minIndex == -1) { // Süreçleri sırayla işle
        minIndex = -1;
        for (int i = 0; i < n; i++) { // En kısa süreci bul
            if (remain[i] > 0 && remain[i] < minTime) {
                minTime = remain[i];
                minIndex = i;
            }
        }
        remain[minIndex]--;
}
```

SRTN Örnek Senaryo

Süreç	Geliş süresi	Servis süresi
P1	0	8
P2	1	4
P3	2	9
P4	3	5

- Başlama süresi: P1 = 10; P2 = 1; P3 = 17; P4 = 5
- Ortalama bekleme süresi: [(10-1)+(1-1)+(17-2)+(5-3)] / 4 = 6.5



Sonuç

- Her algoritmada, CPU tahsisi farklı kriterlere göre belirlenir.
- FCFS ve SJF kesmeli değildir, (non-*preemptive*)
  - Bir süreç başlatıldıktan sonra tamamlanıncaya kadar devam eder.
- SRTN, kalan yürütme süreleri değiştikçe,
  - Süreçler arasında geçiş yapılmasına izin verir. (*preemptive*)
- Algoritma seçimi,
  - İşletim sistemine ve
  - Üzerinde çalıştığı sistemin gereksinimlerine bağlıdır.

İnteraktif Sistemlerde Çizelgeleme

- Sıralı (*Round-robin scheduling*)
- Önceliğe göre (*Priority scheduling*)
- Çoklu kuyruk (*Multiple queues*)
- En kısa süreç önce (*Shortest process next*)
- Garantili (*Guaranteed scheduling*)
- Piyango (*Lottery scheduling*)
- Adil paylaşım (*Fair-share scheduling*)

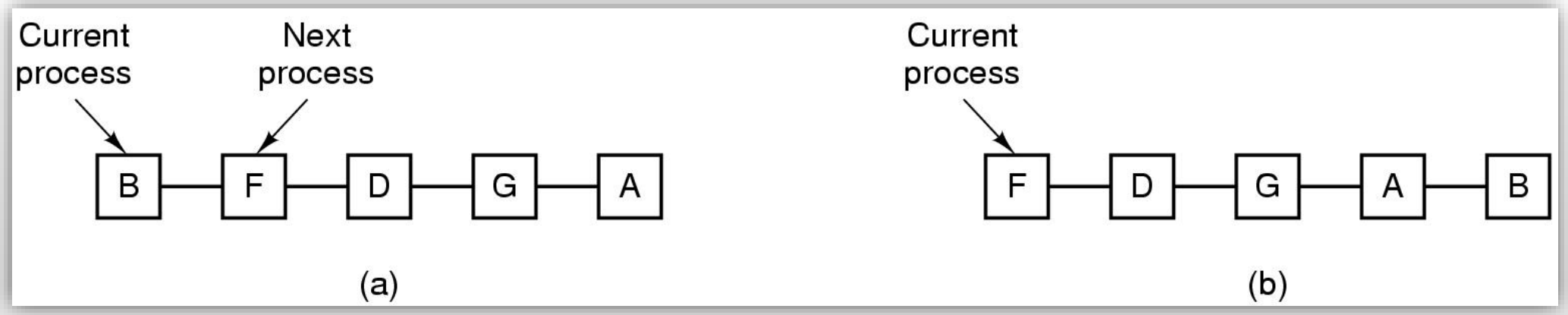
Sıralı Çizelgeleme

- Süreçler döngüsel bir sırada yürütülür.
- Sırası gelen sabit bir zaman diliminde (*time quantum*) işlemciyi kullanır.
- Uygulaması basit.
- Hem G/Ç'ye bağlı hem de CPU'ya bağlı süreçleri etkili bir şekilde yönetir.
- Zaman diliminin, bağlam anahtarlama süresinden büyük olması gerekir.
- Zaman diliminin çok küçük olması durumunda,
  - Yüksek ek maliyet ve uzun yanıt süresine neden olabilir.

Sıralı Çizelgeleme

(a) Çalıştırılabilir süreçlerin listesi.

(b) B süreci, zaman dilimini (*quantum*) kullandıktan sonra süreçlerin listesi.



Sıralı Çizelgeleme Örnek Senaryo

- Zaman dilimi = 20 *ms*

Süreç	P1	P2	P3	P4
Süre	53	17	68	24

- Gantt çizelgesi

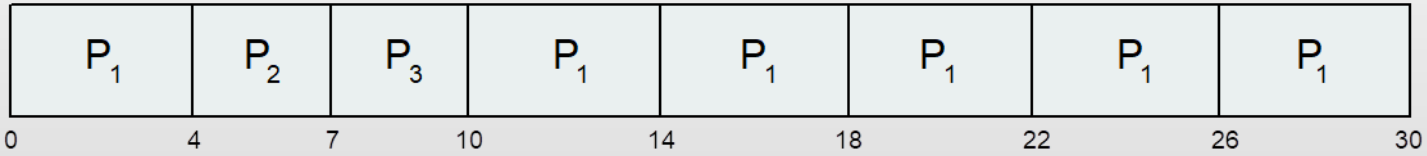
P1	**P2**	**P3**	**P4**	**P1**	**P3**	**P4**	**P1**	**P3**	**P3**
20	37	57	77	97	117	121	134	154	162

- SJF'den yüksek geri dönüş süresi, daha iyi yanıt (*response*) süresi

Sıralı Çizelgeleme Örnek Senaryo

Süreç	Servis süresi
P1	24
P2	3
P3	3

- Zaman dilimi = 4 *ms*
- Gantt çizelgesi



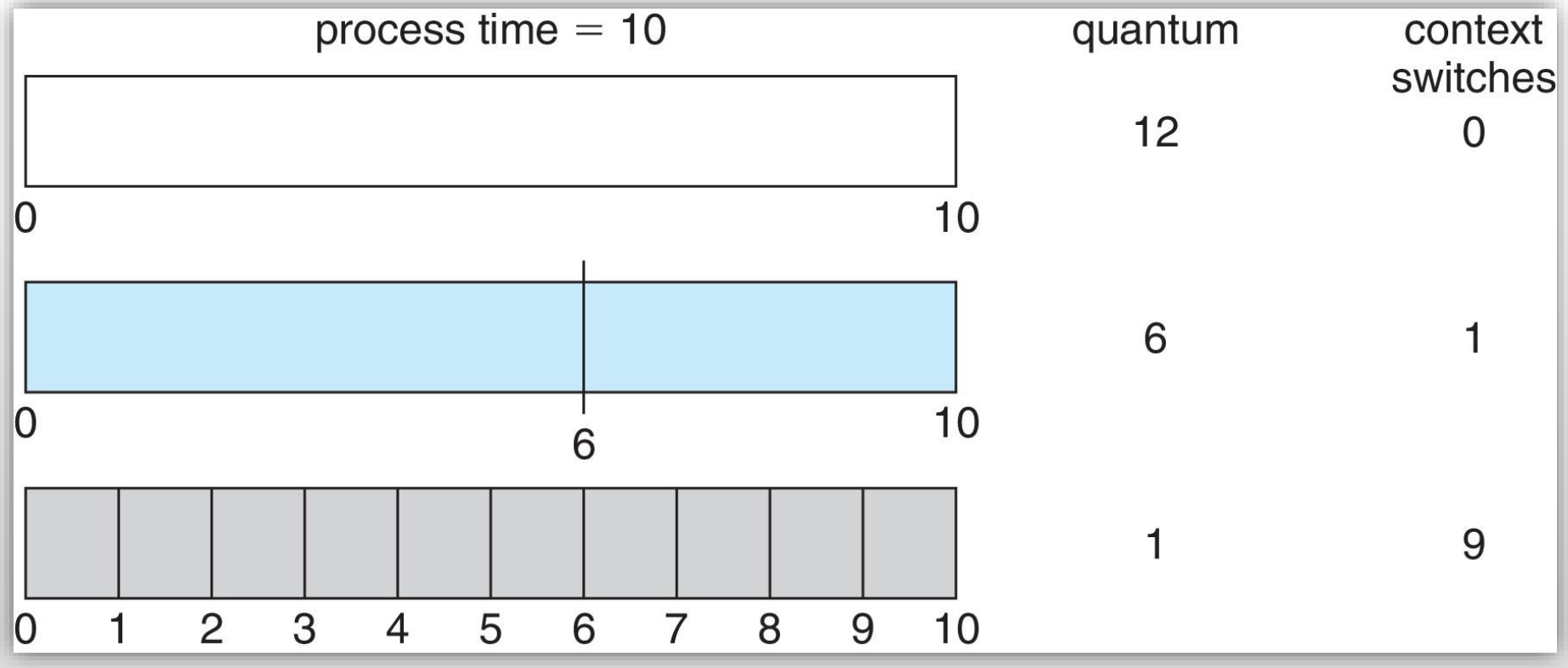
Bağlam Anahtarlama (context switch)

- Sıralı çizelgelemenin performansı seçilen zaman dilimine bağlıdır.
- Zaman dilimi, çok büyük ise,
  - FCFS algoritmasına benzer.
- Çok küçük ise
  - Bağlam anahtarlama sık olur,
  - Yüksek ek maliyet,
  - Verimsiz CPU kullanımı
- Bağlam anahtarlama maliyetinden büyük olmalıdır.

Bağlam Anahtarlama

- Zaman dilimi 4 *ms,* bağlam anahtarlama 1 *ms* ise,
  - İşlemcinin %20 zamanı boşa gider.
- Zaman dilimi 100 *ms* ise, boşa harcanan zaman %1'dir,
  - Ancak sistem daha az duyarlı olur.
- Zaman dilimini 20-50 *ms* civarında seçmek uygundur.

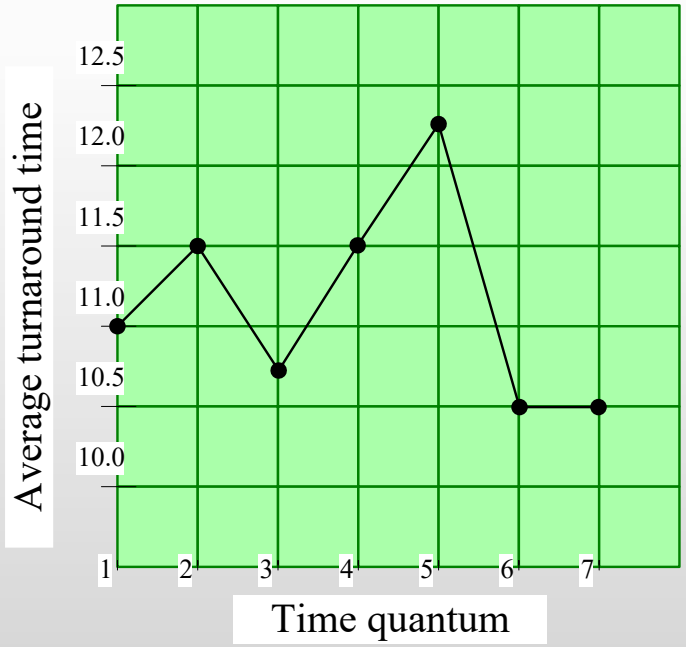
Bağlam Anahtarlama ve Zaman Dilimi (quantum)

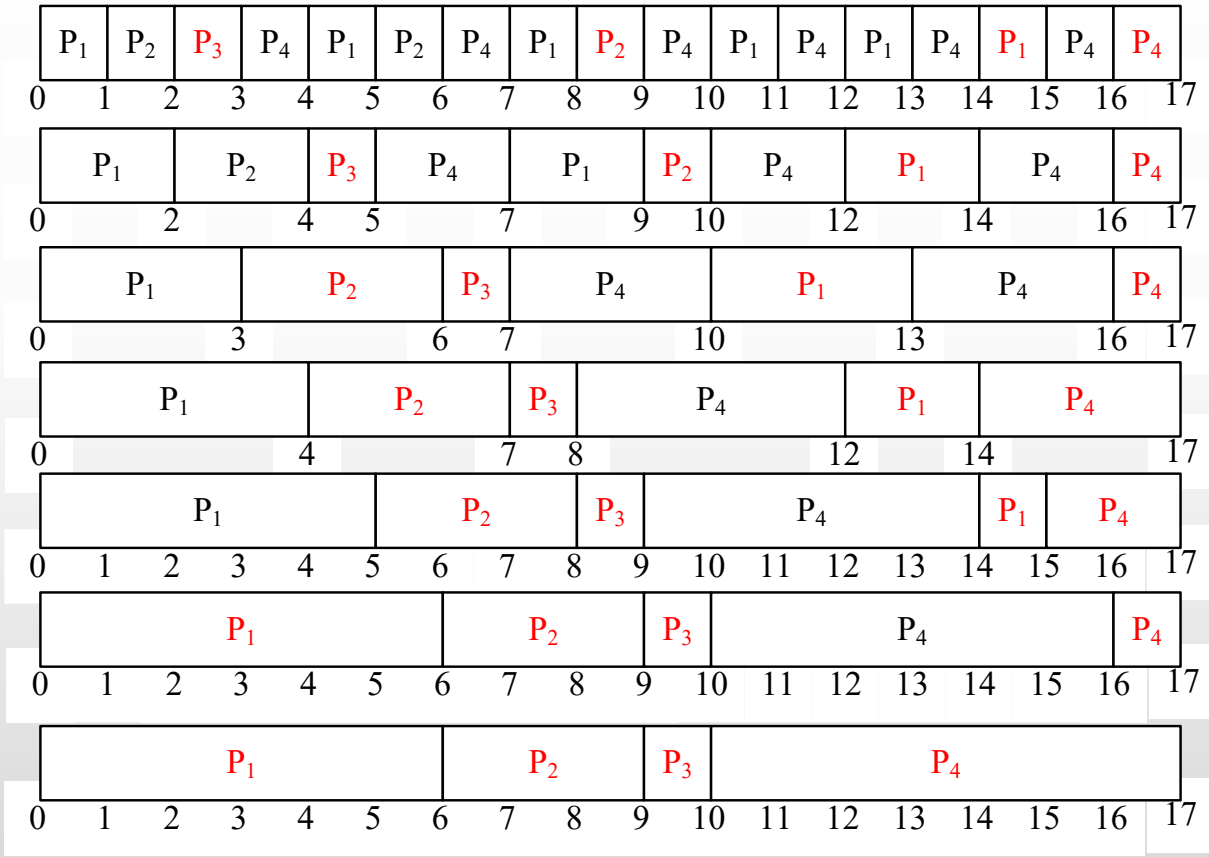


Kuantuma Göre Dönüş Süreleri

- q arttıkça dönüş süresi iyileşir mi?

Process	Time
P_1	6
P_2	3
P_3	1
P_4	7



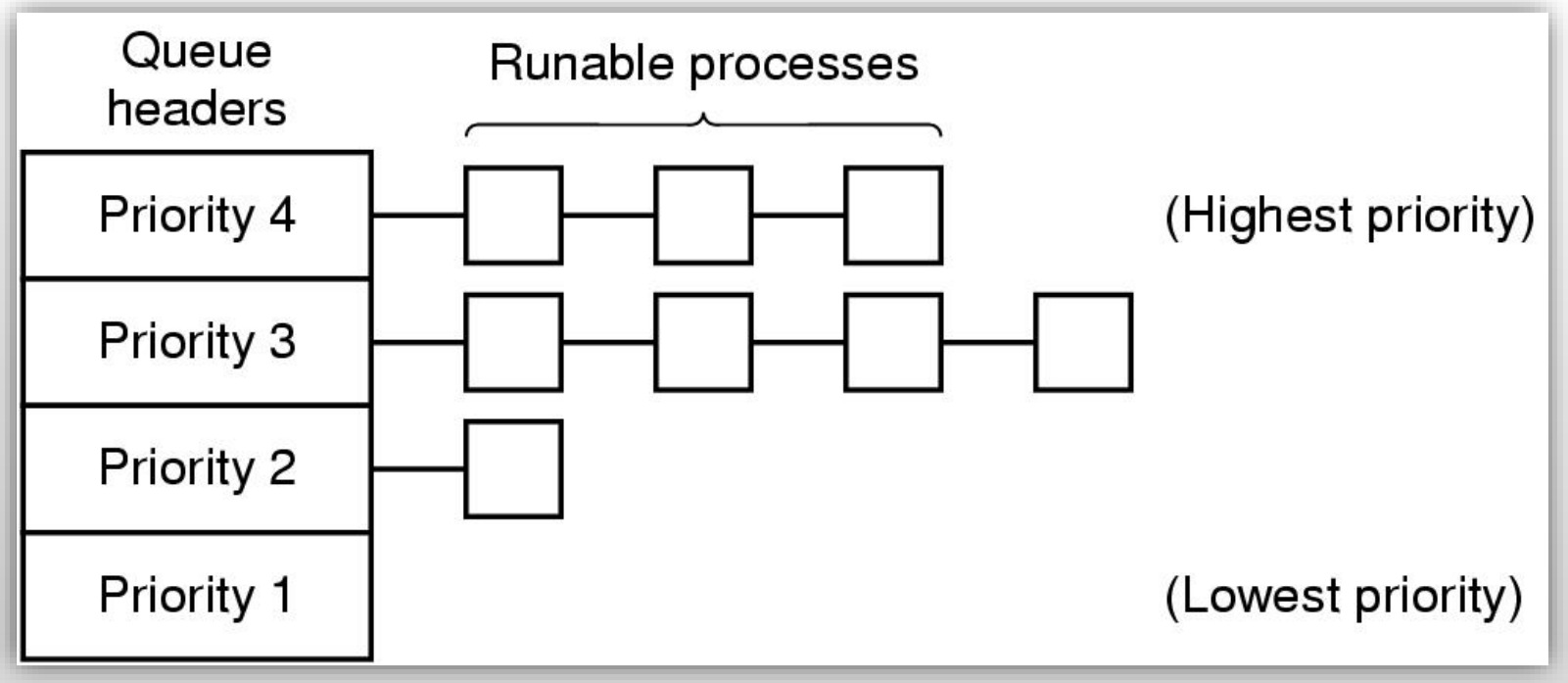


Önceliğe Göre Çizelgeleme

- Süreçler önceliğine göre sayısal değerler ile ilişkilendirilir.
- Süreçler önceliğe göre sırayla yürütülür,
  - Yüksek öncelikli süreç daha önce.
- Önemli süreçlere öncelik sağlar, genel sistem yanıt verebilirliğini artırır.
- Açlığa (*starvation*) neden olabilir.
  - Düşük öncelikli süreçlere hiç sıra gelmeyebilir.
- Bunu önlemek için yaşlanma (*aging*) mekanizması.
  - Bekleyen süreçlerin önceliği zamanla arttırılır.

Önceliğe Göre Çizelgeleme

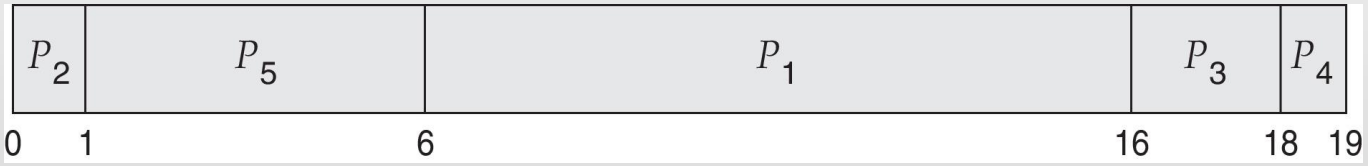
- Her sürecin önceliğinin sayısal bir karşılığı vardır.
- Tüm süreçlerin öncelikleri eşit ise, FCFS gibi davranır.



Önceliğe Göre Çizelgeleme Örnek Senaryo

Süreç	Servis süresi	Öncelik
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

- Gantt çizelgesi

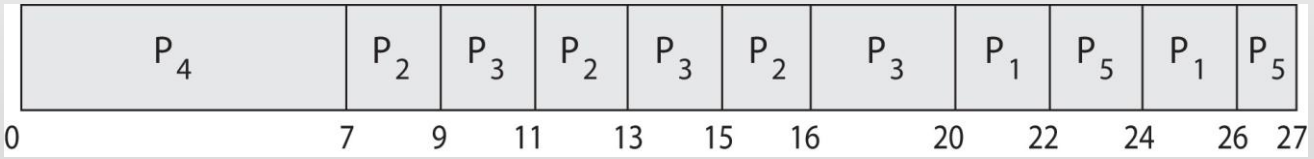


- Ortalama bekleme süresi: 8.2

Önceliğe Göre ve Sıralı Çizelgeleme

Süreç	Servis süresi	Öncelik
P1	4	3
P2	5	2
P3	8	2
P4	7	1
P5	3	3

- Gantt çizelgesi (kuantum = 2)



Çoklu Kuyrukta Çizelgeleme

- Süreçler önceliklerine göre ayrı kuyruklara yerleştirilir.
- Önceliği ve yaşlanmayı yönetir,
  - Ek yükü azaltarak sistem performansını artırır.
- Uygulaması karmaşıktır,
  - Sıra değişiklikleri nedeniyle yüksek yanıt süresine neden olabilir.
- CPU'ya bağlı sürece büyük zaman dilimi (*quantum*) verilir.
- Etkileşimli sürece küçük zaman dilimi (*quantum*) verilir.
- Yüksek öncelikli kuyruktakiler bir zaman dilimi koşar,
  - Bir sonraki kuyruktakiler iki zaman dilimi koşar...
  - Bir süreç koştuktan sonra, sonraki kuyruğa taşınır.

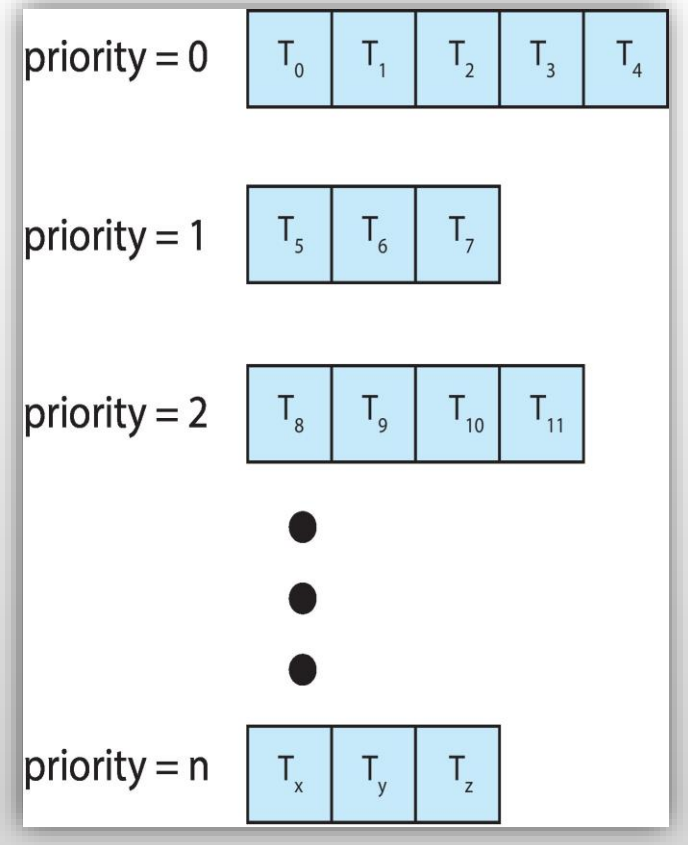
Çoklu Kuyrukta Çizelgeleme Örnek Senaryo

- Üç kuyruk:
  - **Queue0** – zaman dilimi 8 *ms*, FCFS
  - **Queue1** – zaman dilimi 16 *ms*, FCFS
  - **Queue2** – FCFS

Çoklu Kuyrukta Çizelgeleme

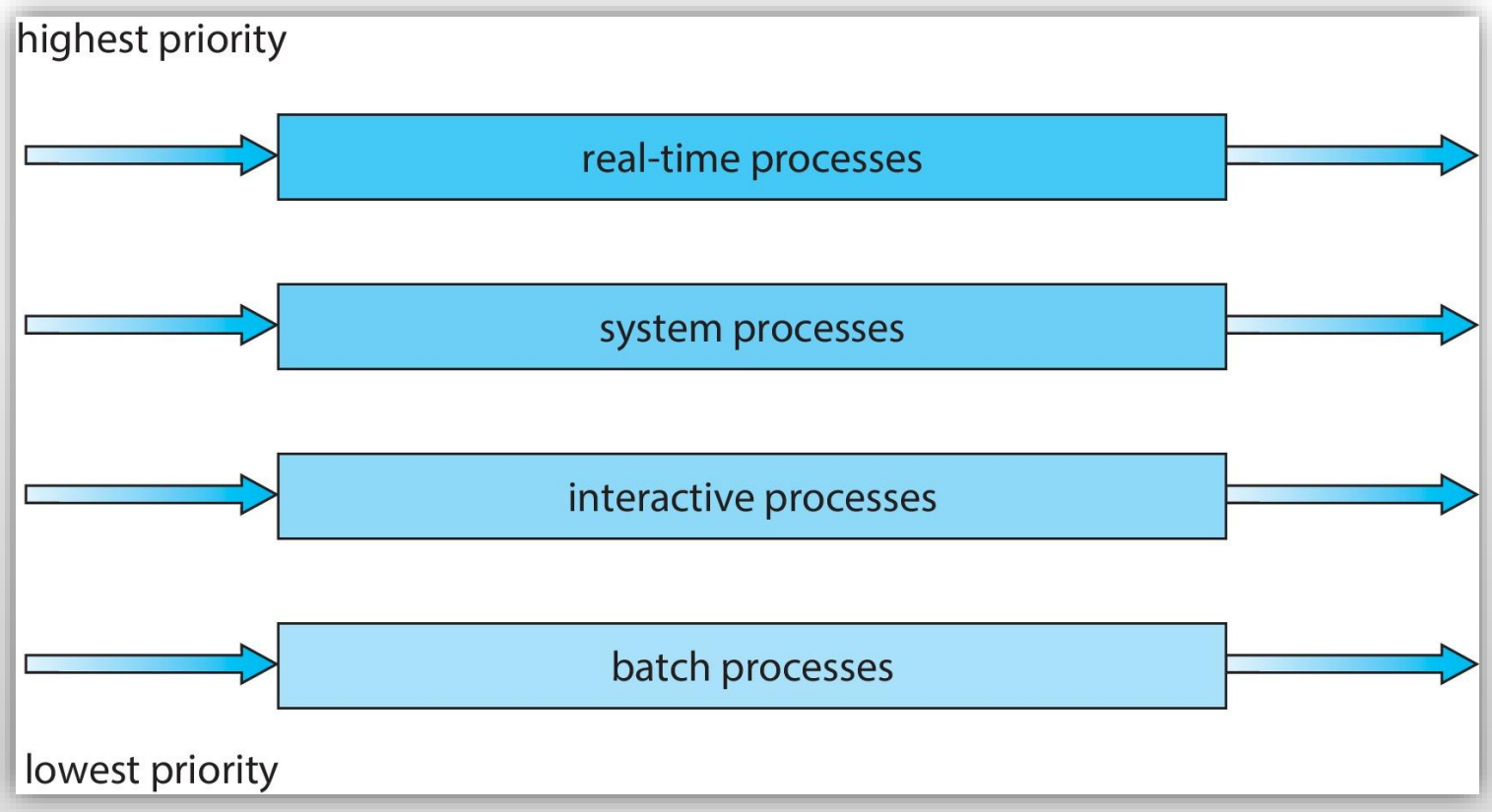
- Her öncelik düzeyi için farklı kuyruklar var.
- Süreç koştuktan sonra farklı kuyruğa atanabilir.
- Her düzeyde kullanılan algoritma değişebilir.
- Terminalde her *enter tuşu*na basıldığında

terminale ait süreç en yüksek öncelik kuyruğa taşınır.



Çoklu Kuyrukta Çizelgeleme

- .



En Kısa Süreç Önce Çizelgeleme

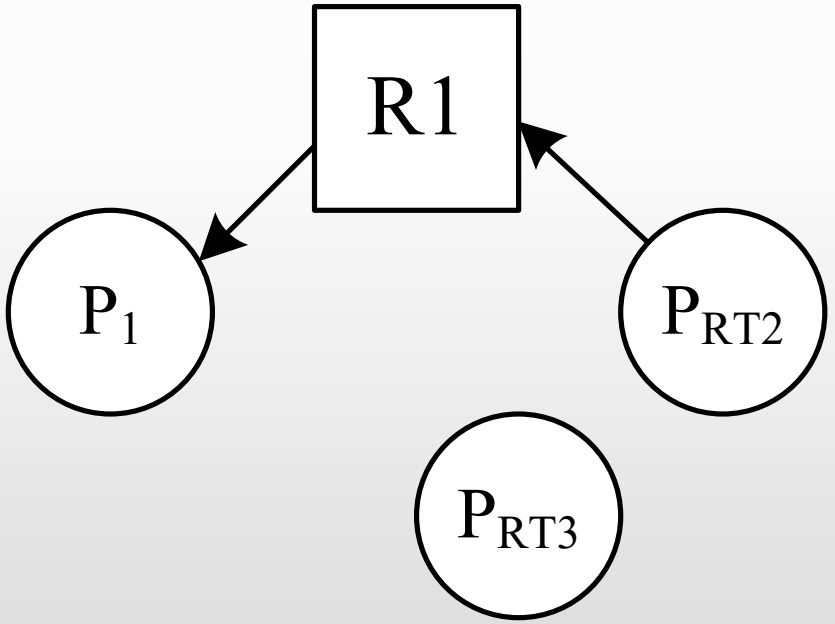
- Bir sonraki adımda yürütülecek süreç, kalan sürelerine göre çizelgelenir.
- CPU kullanımını en üst düzeye çıkarır.
- Ortalama bekleme süresini azaltır.
- Etkileşimli sistemlerde bir sürecin kalan süresini tahmin etmek zordur.
  - Geçmiş davranışlara dayalı olarak tahminde bulunulabilir.
    - T0, T0 / 2 + T1 / 2, T0 / 4 + T1 / 4 + T2 / 2..
  - Mevcut ve önceki tahminin ağırlıklı ortalaması alınabilir.

Öncelik Ters Çevirme (Priority Inversion)

- Düşük öncelikli bir süreç, bir kaynağı işgal etmişse, yüksek öncelikli bir süreç o kaynağı kullanamaz.
- Yüksek öncelikli süreç, düşük öncelikli sürecin sonlanmasını bekler.
  - Düşük öncelikli süreç sistem kaynağına erişmiş olabilir.
  - Kritik bölgede olabilir.

Öncelik Ters Çevirme

- Öncelik sıralaması: P1 < PRT3 < PRT2
- PRT3, P1'i önler, önceliği yüksek.
- PRT2, P1'i bekler, kaynağı P1 ele geçirmiş.
- PRT2, PRT3'ü bekler.



Garantili Çizelgeleme

- Önceden belirlenmiş bir programa göre çizelgeleme yapar.
- Gerçek zamanlı sistemler için uygun, öngörülebilir bir davranış sağlar.
- Sınırlı esneklik, öngörülemeyen durumlarda düşük performans.
- Katı gerçek zamanlıya karşı yumuşak gerçek zamanlı.
  - Katı: Bir fabrikada robot kontrolü (*hard real time*)
  - Yumuşak: CD çalar (*soft real time*)
- Algoritmalar,
  - statik (çalışma süreleri önceden bilinen) veya
  - dinamik (çalışma zamanı kararları) olabilir.

Piyango Çizelgeleme

- Süreçlere bilet atanır ve yürütme için rastgele bir bilet seçilir.
- İşlemci ataması için periyodik olarak çekiliş yapılır.
- Önceliği ve yaşlandırmayı yönetir.
- Ek maliyeti azaltır ve sistem performansını artırır.
- Uygulanması karmaşık.
- Dengesiz bilet dağılımı durumunda öngörülemez davranır.
- *Önemli* süreçler için fazla çekiliş hakkı tanınır.
  - Öncelikler bu şekilde değiştirilebilir.

Adil Paylaşım Çizelgeleme

- Süreçler son kullanım geçmişlerine göre çizelgelenir.
- Süreçler arasında kaynakları dengeler.
- Genel sistem yanıt verebilirliğini artırır.
- Kullanım geçmişlerinin izlenmesi, ek yüke neden olabilir.

Değerlendirme

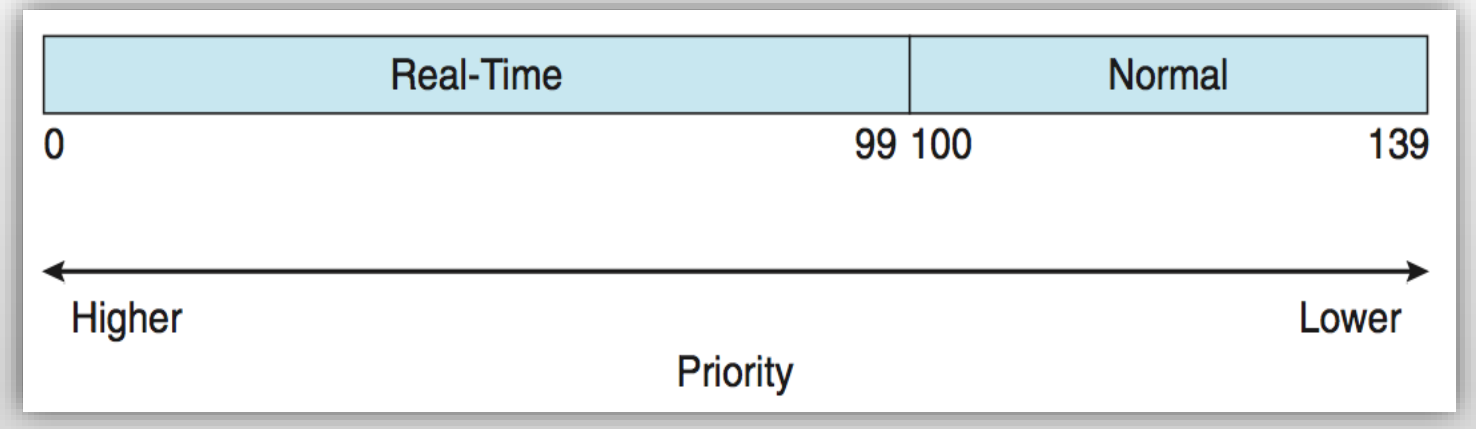
- Her algoritmanın kendi güçlü ve zayıf yönleri vardır.
- *Round-Robin*, *Priority*, *Multiple Queues* ve *Fair-Share* kesmeli algoritmalardır (*preemptive*).
- *Shortest Process Next*, *Guaranteed*, *Lottery* önceliği dikkate almaz ve CPU kullanımını üst düzeye çıkarmayı amaçlar.
- Algoritma seçimi, işletim sisteminin
  - yanıt verebilirlik,
  - öngörülebilirlik,
  - kaynak tahsisi ve
  - adalet gibi hedeflerine bağlıdır.

POSIX Gerçek Zamanlı Çizelgeleme

- POSIX.1b standardı.
- API, gerçek zamanlı iş parçacıklarını yönetmek için fonksiyonlar sağlar.
- Gerçek zamanlı iş parçacıkları için iki seçenek tanımlar:
  - SCHED_FIFO - FIFO kuyruğuna sahip FCFS stratejisi kullanır. Eşit önceliğe sahip iş parçacıkları için zaman dilimleme yoktur.
  - SCHED_RR - Eşit önceliğe sahip iş parçacıkları için zaman dilimleme vardır.
- Çizelgeleme kuralını öğrenmek ve atamak için:
- `pthread_attr_getsched_policy(pthread_attr_t *attr, int *policy)`
- `pthread_attr_setsched_policy(pthread_attr_t *attr, int policy)`

Linux Çizelgeleme

- *Completely fair scheduler* (CFS) kullanır.
- Her bir sürecin belirli bir önceliği vardır.
- En yüksek çizelgeleme sınıfındaki en yüksek öncelikli görevi seçer.
- Sabit zaman dilimleri atamak yerine CPU kullanım oranına dayalıdır.
- İki çizelgeleme sınıfı vardır, genişletilebilir.
  - varsayılan,
  - gerçek zamanlı.
- Gerçek zamanlı süreçlerin statik öncelikleri vardır.



Windows Çizelgeleme

- Önceliğe dayalı kesmeli (*priority based preemptive*) çizelgeleme kullanır.
- Sonraki adımda en yüksek öncelikli iş parçacığı çalışır.
- İş parçacığı (1) bloke olana kadar, (2) zaman dilimi bitene kadar, (3) daha yüksek öncelikli bir iş parçacığı gelene kadar çalışır.
- Gerçek zamanlı iş parçacıkları, diğerlerini bloke edebilir.
- 32 seviyeli öncelik şeması kullanır.
  - Değişken sınıf 1-15, gerçek zamanlı sınıf 16-31.
- Öncelik 0, bellek yönetimi iş parçacığıdır.
- Her öncelik için kuyruk tutulur.
- Çalıştırılabilir iş parçacığı yoksa, boş (*idle*) iş parçacığı çalıştırılır.

Windows Öncelikler

- .

	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

SON