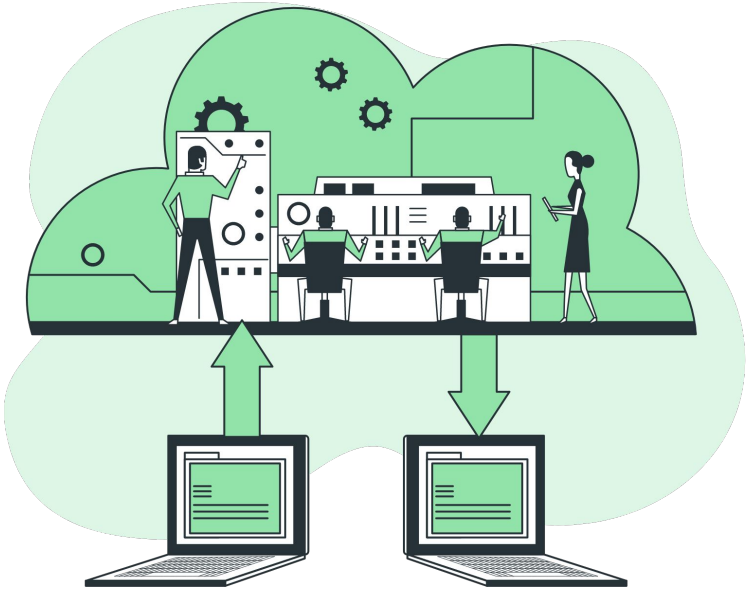
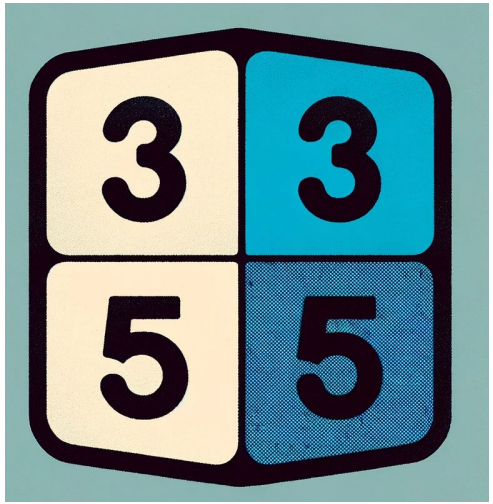
04
Senkronizasyon



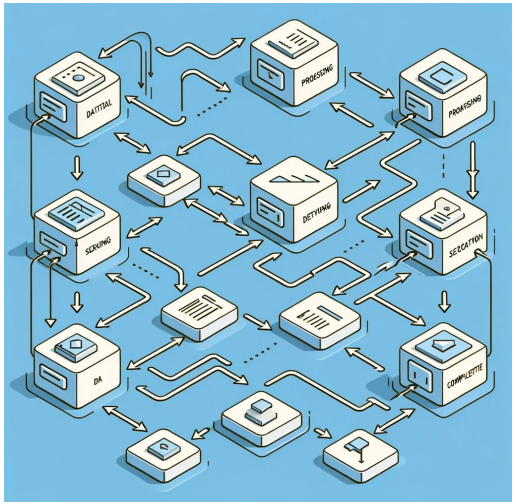
Senkronizasyon Nedir?

Senkronizasyon, *genellikle iki veya daha fazla işlemin veya olayın birbiriyle* ***uyum*** *içinde, belirli bir* ***sıra*** *veya* ***zamanlama*** *ile gerçekleşmesini sağlama sürecidir.*

Serileştirme ve Karşılıklı Dışlama



Karşılıklı Dışlama



Serileştirme

Eşzamanlama Problemi

Mesajlaşma

```
Eat breakfast
Work
Eat lunch
Call Bob
```

```
Eat breakfast
Wait for a call
Eat lunch
```

Mesajlaşma

```
1 Eat breakfast
2 Work
3 Eat lunch
4 Call Bob
```

Sequentially ?

```
1 Eat breakfast
2 Wait for a call
3 Eat lunch
```

Concurrently ?

Determinizm

Thread A

```
1 print "yes"
```

Thread B

```
1 print "no"
```

Paylaşılan Değişken

Aşağıdaki örnekte, x iki thread tarafından erişilen paylaşılan bir değişkendir.

İş Parçacığı A	**İş Parçacığı B**
x = 5	x = 7
print (x)	

Atomik İşlemler

Thread A

```
1 count = count + 1
```

Thread B

```
1 count = count + 1
```

Atomik İşlemler

Thread A

```
1 count = count + 1
```

Thread B

```
1 count = count + 1
```

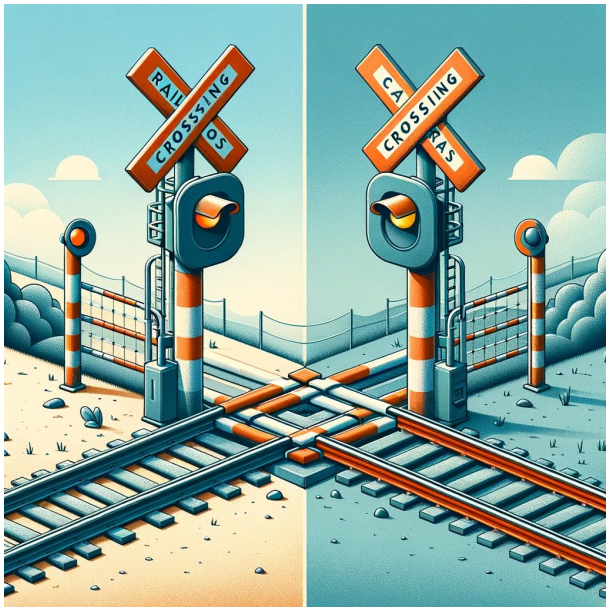
Thread A

```
1 temp = count
2 count = temp + 1
```

Thread B

```
1 temp = count
2 count = temp + 1
```

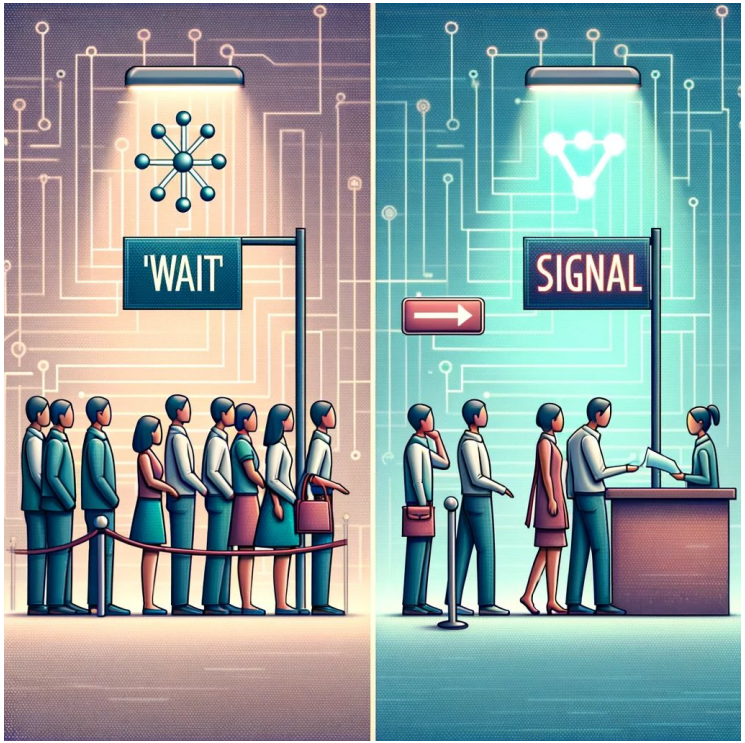
Semafor



Semafor

```
sem = Semaphore(1)

sem.wait()
sem.signal()
```



Semafor Faydaları

Hata Önleme

Anlaşılabilirlik

Taşınabilirlik

02

Senkronizasyon Paternleri

Signaling

Thread A

```
1 statement a1
2 sem.signal()
```

Thread B

```
1 sem.wait()
2 statement b1
```

Rendezvous

Thread A

```
1 statement a1
2 statement a2
```

Thread B

```
1 statement b1
2 statement b2
```

Rendezvous

Thread A

```
1 statement a1
2 aArrived.signal()
3 bArrived.wait()
4 statement a2
```

Thread B

```
1 statement b1
2 bArrived.signal()
3 aArrived.wait()
4 statement b2
```

Thread A

```
1 statement a1
2 bArrived.wait()
3 aArrived.signal()
4 statement a2
```

Thread B

```
1 statement b1
2 bArrived.signal()
3 aArrived.wait()
4 statement b2
```

Deadlock

Thread A

```
1 statement a1
2 bArrived.wait()
3 aArrived.signal()
4 statement a2
```

Thread B

```
1 statement b1
2 aArrived.wait()
3 bArrived.signal()
4 statement b2
```

Mutex

Thread A

```
1 count = count + 1
```

Thread B

```
1 count = count + 1
```

Mutex

Thread A

```
1 mutex.wait()
2     # critical section
3     count = count + 1
4 mutex.signal()
```

Thread B

```
1 mutex.wait()
2     # critical section
3     count = count + 1
4 mutex.signal()
```

Multiplex

Multiplex solution

```
1 multiplex.wait()
2     critical section
3 multiplex.signal()
```

Barrier

```
1 rendezvous
2 critical point
```

Barrier hint

```
1 n = the number of threads
2 count = 0
3 mutex = Semaphore(1)
4 barrier = Semaphore(0)
```

Barrier

```
rendezvous

mutex.wait()
    count = count + 1
mutex.signal()

if count == n: barrier.signal()

barrier.wait()

critical point
```

Çalışır mı?

Barrier

```
1  rendezvous
2
3  mutex.wait()
4      count = count + 1
5  mutex.signal()
6
7  if count == n: barrier.signal()
8
9  barrier.wait()
10 barrier.signal()
11
12 critical point
```

Çalışır mı?

Barrier (Deadlock) 😵

```
1  rendezvous
2
3  mutex.wait()
4      count = count + 1
5      if count == n: barrier.signal()
6
7      barrier.wait()
8      barrier.signal()
9  mutex.signal()
10
11 critical point
```

Reusable Barrier ? ♻️

```
1  rendezvous
2
3  mutex.wait()
4      count += 1
5  mutex.signal()
6
7  if count == n: turnstile.signal()
8
9  turnstile.wait()
10 turnstile.signal()
11
12 critical point
13
14 mutex.wait()
15     count -= 1
16 mutex.signal()
17
18 if count == 0: turnstile.wait()
```

Çalışır mı?

Reusable Barrier ? ♻️

```
1  rendezvous
2
3  mutex.wait()
4      count += 1
5      if count == n: turnstile.signal()
6  mutex.signal()
7
8  turnstile.wait()
9  turnstile.signal()
10
11 critical point
12
13 mutex.wait()
14     count -= 1
15     if count == 0: turnstile.wait()
16 mutex.signal()
```

Reusable Barrier ☑️ ♻️

```
# rendezvous

mutex.wait()
    count += 1
    if count == n:
        turnstile2.wait()        # lock the second
        turnstile.signal()       # unlock the first
mutex.signal()

turnstile.wait()                 # first turnstile
turnstile.signal()

# critical point

mutex.wait()
    count -= 1
    if count == 0:
        turnstile.wait()          # lock the first
        turnstile2.signal()       # unlock the second
mutex.signal()

turnstile2.wait()                 # second turnstile
turnstile2.signal()
```

Preload Barrier 🥞

```
# rendezvous

mutex.wait()
    count += 1
    if count == n:
        turnstile.signal(n)       # unlock the first
mutex.signal()

turnstile.wait()                  # first turnstile

# critical point

mutex.wait()
    count -= 1
    if count == 0:
        turnstile2.signal(n)      # unlock the second
mutex.signal()

turnstile2.wait()                 # second turnstile
```

Queue

Queue solution (leaders)

```
1 followerQueue.signal()
2 leaderQueue.wait()
3 dance()
```

Queue solution (followers)

```
1 leaderQueue.signal()
2 followerQueue.wait()
3 dance()
```

Queue

Queue solution (leaders)

```
mutex.wait()
if followers > 0:
    followers--
    followerQueue.signal()
else:
    leaders++
    mutex.signal()
    leaderQueue.wait()

dance()
rendezvous.wait()
mutex.signal()
```

Queue solution (followers)

```
mutex.wait()
if leaders > 0:
    leaders--
    leaderQueue.signal()
else:
    followers++
    mutex.signal()
    followerQueue.wait()

dance()
rendezvous.signal()
```