Nesneye Yönelimli Programlama – BLM 205

Hafta 14: Eş Zamanlılık ve Çoklu İşparçacıkları

Fenerbahçe Üniversitesi

Öğretim Elemanları

Öğretim Üyesi: Dr. Vecdi Emre Levent

Ofis: 311

Email: emre.levent@fbu.edu.tr

Asistan: Arş. Gör. Uğur Özbalkan

Ofis: 307

Email: ugur.ozbalkan@fbu.edu.tr

Asistan: Arş. Gör. Ecenur Alioğulları

Ofis: 307

Email: ecenur.aliogullari@fbu.edu.tr

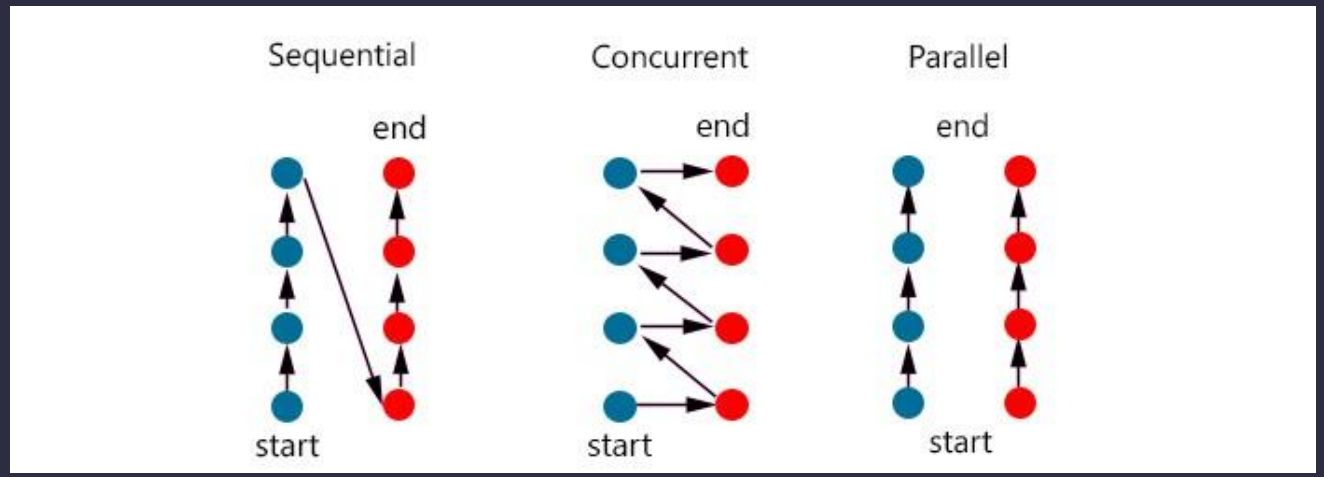
Ders Planı

- Eş Zamanlılık ve Çoklu İşparçacıları
  - Thread vs Process
  - Çoklu thread yazılım geliştirme

Eş Zamanlılık ve Çoklu İşparçacıkları

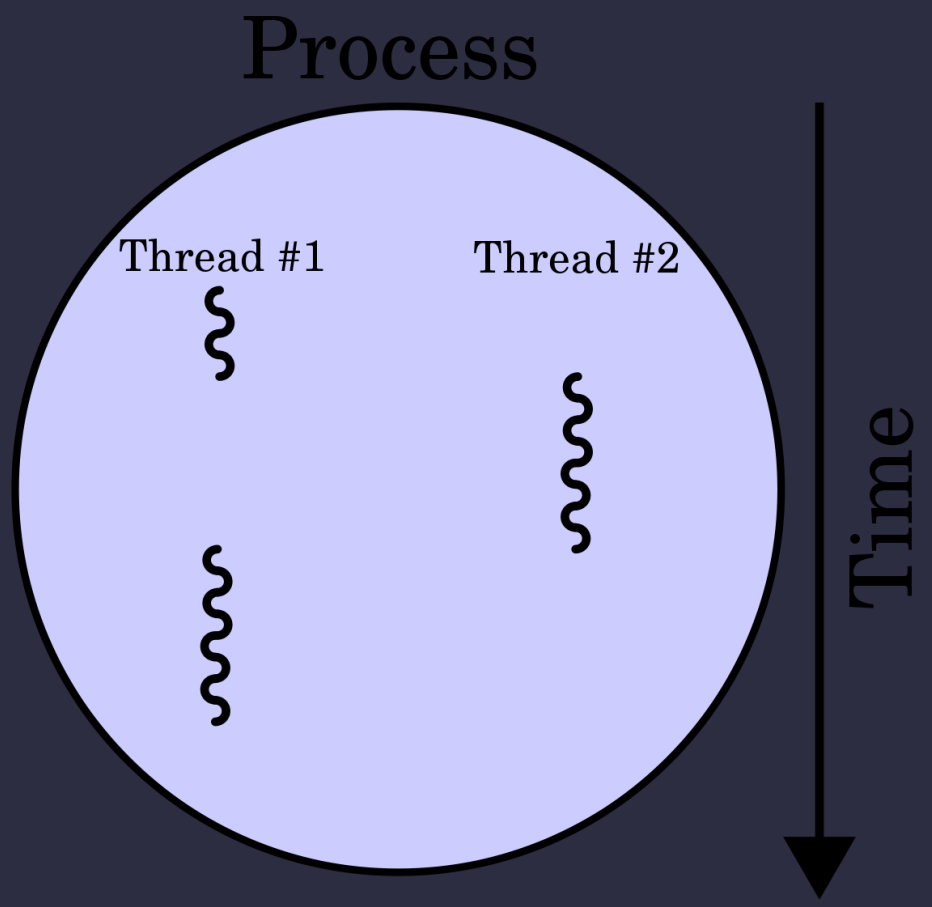
Eş zamanlılık (Concurrency) birden çok işin birlikte yürütülmesine denmektedir.

Çoklu İşparçacıkları (Multi thread) ise, bir işin, küçük iş parçacılarına (thread) bölünerek, paralel olarak yürütülmesidir.



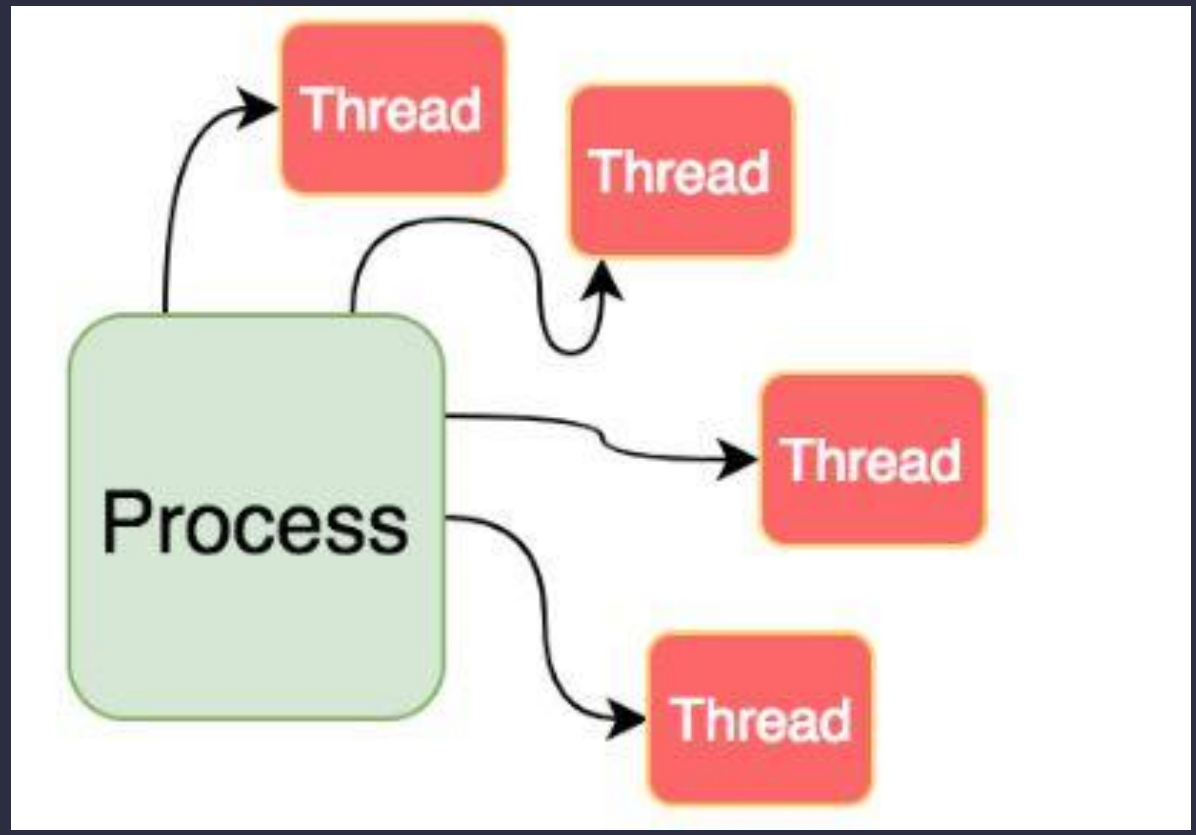
Eş Zamanlılık ve Çoklu İşparçacıkları

Geliştirilen bir uygulama (Process), bir arada çalışabilecek küçük işparçacılarından (thread) oluşabilecek şekilde yazıldıysa, işlemci üzerinde paralel olarak koşabilirler.



Eş Zamanlılık ve Çoklu İşparçacıkları

Process'lerden küçük iş parçacıkları oluşturulabilir.



Eş Zamanlılık ve Çoklu İşparçacıkları

Geliştirilen uygulamalar, özellikle threading kullanılmıyorsa, tek thread olarak çalışmaktadırlar.

Yani paralel olarak koşmazlar

Eş Zamanlılık ve Çoklu İşparçacıkları

Thread'lerin paralel olarak koşması ile, uygulamanın tamamlanma zamanı azaltılabilir.

Thread'ler oluşturulduğunda, tüm thread'ler aynı bellek bölgesini kullanır.

Process'ler arasında bellek ortak kullanılmaz. Her bir process için ayrı bellek bölgesi bulunur.

Eş Zamanlılık ve Çoklu İşparçacıkları

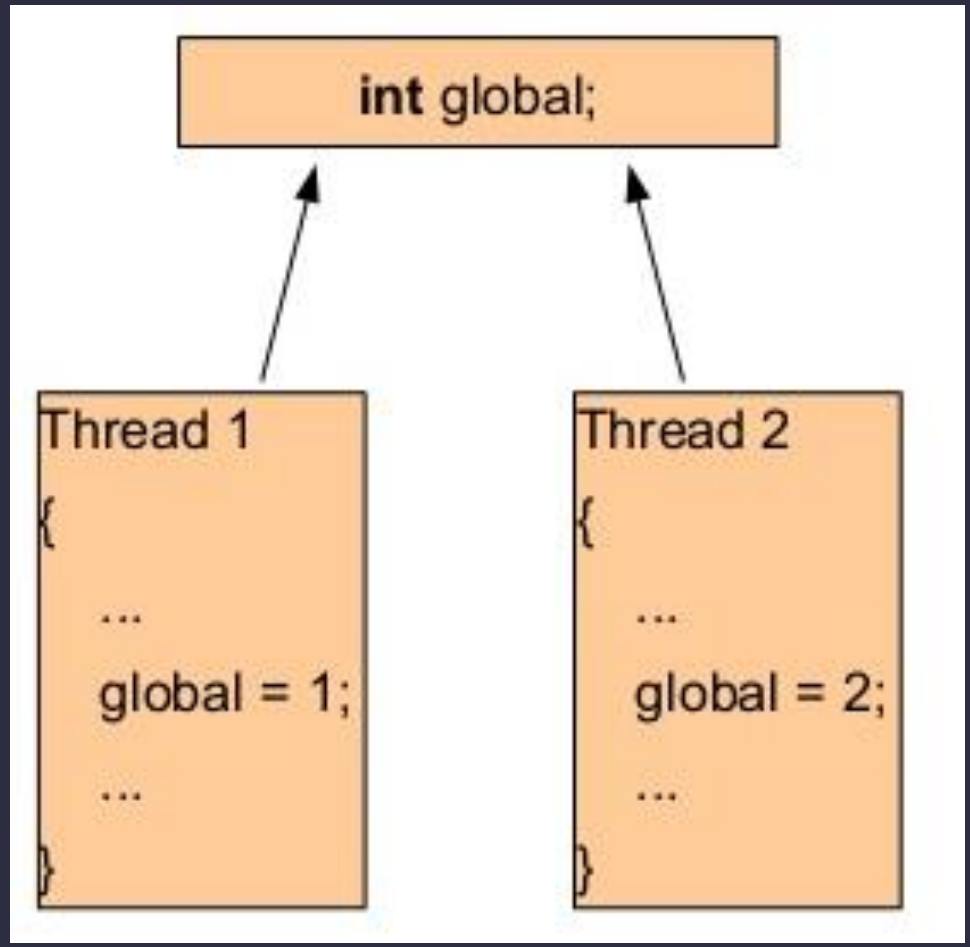
Python dilinde, GIL (Global Interpreter Lock) isminde bir yapı, aynı anda birden çok thread'in aktif olmasına izin vermez.

Bu yapının olmasının nedeni, Python'un çoklu thread mekanizmasının "thread-safe" olmamasıdır.

Eş Zamanlılık ve Çoklu İşparçacıkları

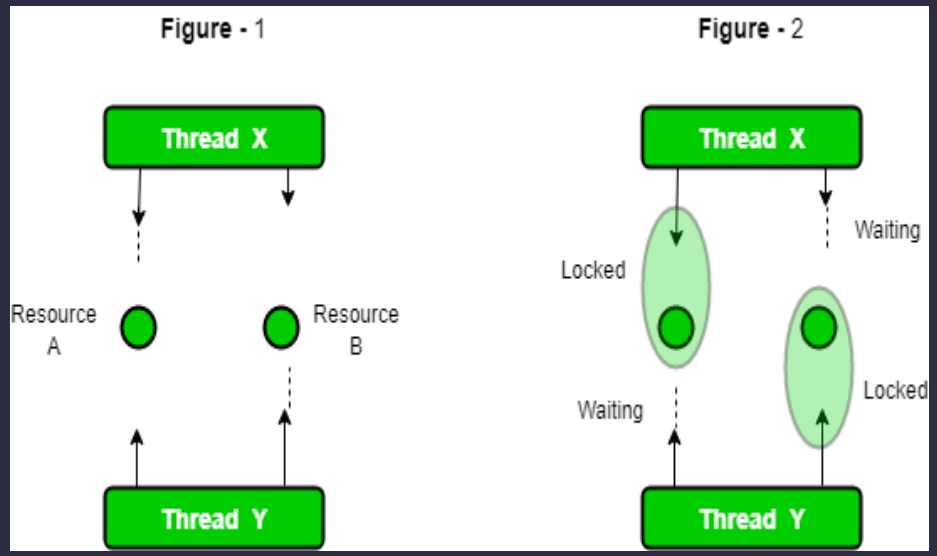
Race condition: Birden çok threadin, aynı veriye erişmeye ve/veya değiştirmeye çalışmasıdır.

Bir yapının thread-safe olması, yaratılan thread'lerin birbirleri ile karışmasını engelleyen yapıların barındırmasıdır.



Eş Zamanlılık ve Çoklu İşparçacıkları

Deadlock: Birden çok thread, aynı kaynağa erişmeye çalışıp, bir kaynağın bir thread tarafından kitlenip, diğer thread'in onu beklemesi nedeniyle yaşanabilir.



Eş Zamanlılık ve Çoklu İşparçacıkları

Yaratılmış tüm thread'ler aynı bellek bölgesini kullanmaktadır.

Örneğin,

a = 2

Değeri bulunsun

Eş Zamanlılık ve Çoklu İşparçacıkları

Yaratılmış tüm thread'ler aynı bellek bölgesini kullanmaktadır.

Örneğin,

Bu satırdan sonra, 2 thread başlatılıp;

Thread 1: a = a + 2
Thread 2: a = a * 3

Yapmak isterse

Eş Zamanlılık ve Çoklu İşparçacıkları

Yaratılmış tüm thread'ler aynı bellek bölgesini kullanmaktadır.

Örneğin,

İşletim sistemi thread'leri kendi algoritmasına göre sıraya dizecek ve ilk olarak thread 2 biterse,

a = 2 * 3, 6 olacak

Sonra thread 1 çalışacak

a = 6 + 2, 8 olacaktır.

Eş Zamanlılık ve Çoklu İşparçacıkları

Yaratılmış tüm thread'ler aynı bellek bölgesini kullanmaktadır.

Örneğin,

ilk olarak thread 1 biterse,

a = 2 + 2, 4 olacak

Sonra thread 2 çalışacak

a = 4 * 3, 12 olacaktır.

Eş Zamanlılık ve Çoklu İşparçacıkları

Yaratılmış tüm thread'ler aynı bellek bölgesini kullanmaktadır.

Örneğin,

Dolayısıyla herhangi bir kontrol mekanizması olmadığı için, tutarsız sonuçlar elde edilecektir.

Eş Zamanlılık ve Çoklu İşparçacıkları

Ardışık çalışma örneği

Örnek Kod Parçacığı

```
import time

def yorucuFonksiyon(n, id):
    for x in range(1, n):
        for y in range(1, n):
            x**y
    print(id, " tamam")

def ardisik(n):
    for i in range(n):
        yorucuFonksiyon(500, i)

if __name__ == "__main__":
    baslangic = time.time()
    ardisik(10)
    bitis = time.time()
    print("Gecen Sure: ", bitis - baslangic)
```

Çıktı

```
0 tamam
1 tamam
2 tamam
3 tamam
4 tamam
5 tamam
6 tamam
7 tamam
8 tamam
9 tamam
Gecen Sure:  33.7322199344635
```

Eş Zamanlılık ve Çoklu İşparçacıkları

Çoklu Thread çalışma örneği

Örnek Kod Parçacığı

```
import threading
import time

def yorucuFonksiyon(n, id):
    for x in range(1, n):
        for y in range(1, n):
            x**y
    print(id, " tamam")

def threaded(n):
    threads = []

    for i in range(n):
        t = threading.Thread(target=yorucuFonksiyon, args=(500,i,))
        threads.append(t)
        t.start()

    for t in threads:
        t.join()

baslangic = time.time()
threaded(10)
bitis = time.time()
print("Gecen Sure: ", bitis - baslangic)
```

Çıktı

```
92  tamam
7  tamam
3  tamam
1 8  tamam
 tamam46  tamam
 tamam tamam
05  tamam  tamam
Gecen Sure:  32.90734529495239
```

Thread oluşturuldu

Thread başlatıldı

Thread join ifadesi, diğer thread'lerin bitmesini bekletmek için kullanılır

Aynı anda sadece 1 thread aktif oldu, thread'ler azar azar çalıştırılarak koştular. (Concurrent ancak paralel değil)

Eş Zamanlılık ve Çoklu İşparçacıkları

Python’da GIL mekanizması olmasa idi çoklu thread kod versiyonu çok daha hızlı çalışması beklenmektedir.

Eş Zamanlılık ve Çoklu İşparçacıkları

Çoklu Thread çalışma örneği

Örnek Kod Parçacığı

```
import threading
import time

def yorucuFonksiyon(n, id):
    time.sleep(2)
    print(id, "tamam")

def threaded(n):
    threads = []

    for i in range(n):
        t = threading.Thread(target=yorucuFonksiyon, args=(500,i,))
        threads.append(t)
        t.start()

    for t in threads:
        t.join()


baslangic = time.time()
threaded(10)
bitis = time.time()
print("Gecen Sure: ", bitis - baslangic)
```

Çıktı

```
0 tamam
4 tamam
215 tamam
3   6 tamamtamamtamamtamam
7



 tamam
8 9tamam
tamam
Gecen Sure:  2.0310635566711426
```

Oldukça hızlı çalıştı çünkü, yorucu fonksiyonda sleep kullanıldı. Python yapılacak işleri schedule ettiği için GIL olmasına rağmen daha hızlı çalıştı.

Eş Zamanlılık ve Çoklu İşparçacıkları

Python’da gerçek anlamda paralel çalışma yapmak için multiprocessing kütüphanesi kullanılmalıdır.

Eş Zamanlılık ve Çoklu İşparçacıkları

Çoklu Process çalışma örneği

Örnek Kod Parçacığı

```
import time
import multiprocessing

def yorucuFonksiyon(n, myid):
    for x in range(1, n):
        for y in range(1, n):
            x**y
    print(myid, " tamam")

def multiproc(n):
    processes = []

    for i in range(n):
        p = multiprocessing.Process(target=yorucuFonksiyon,
args=(500,i,))
        processes.append(p)
        p.start()

    for p in processes:
        p.join()

if __name__ == "__main__":
    baslangic = time.time()
    multiproc(10)
    bitis = time.time()
    print("Gecen Sure: ", bitis - baslangic)
```

Çıktı

```
3  tamam
1  tamam
2  tamam
5  tamam
0  tamam
8  tamam
9  tamam
4  tamam
7  tamam
6  tamam
Gecen Sure:  15.62522292137146
```

Eş Zamanlılık ve Çoklu İşparçacıkları

Çoklu Process çalışma örneği

Örnek Kod Parçacığı

```
import time
from timeit import default_timer as timer
from multiprocessing import Pool, cpu_count

def ustAlma(n):
    time.sleep(2)
    return n * n

if __name__ == '__main__':

    baslangic = timer()
    print(f'Hesaplamalar {cpu_count()} cekirdek ile basladi')
    degerler = (2, 4, 6, 8)

    with Pool() as pool:
        res = pool.map(ustAlma, degerler)
        print(res)

    bitis = timer()
    print(f'Gecen Sure: {bitis - baslangic}')
```

Çıktı

```
Hesaplamalar 4 cekirdek ile basladi
[4, 16, 36, 64]
Gecen Sure: 2.4957844
```

Process havuzu oluşturuldu, kullanılacak çekirdek sayısı mevcut CPU çekirdek sayısı seçilmektedir.

Eş Zamanlılık ve Çoklu İşparçacıkları

Çoklu Process çalışma örneği

Örnek Kod Parçacığı

```
import time
from timeit import default_timer as timer
from multiprocessing import Pool, cpu_count

def kuvvetAlma(x, n):
    time.sleep(1)
    return x ** n

if __name__ == '__main__':
    baslangic = timer()
    print(f'Hesaplamalar {cpu_count()} cekirdek ile yapiliyor')
    degerler = ((2, 2), (4, 3), (5, 5))

    with Pool() as pool:
        sonuc = pool.starmap(kuvvetAlma, degerler)
        print(sonuc)

    bitis = timer()
    print(f'Gecen Sure: {bitis - baslangic}')
```

Çıktı

```
Hesaplamalar 4 cekirdek ile yapiliyor
[4, 64, 3125]
Gecen Sure: 1.4737055000000001
```

Birden çok argüman verildiğindeki syntax

Eş Zamanlılık ve Çoklu İşparçacıkları

Tek çekirdek ile PI hesaplama

Örnek Kod Parçacığı

```
from decimal import Decimal, getcontext
from timeit import default_timer as timer

def pi(precision):

    getcontext().prec = precision

    return sum(1/Decimal(16)**k *
        (Decimal(4)/(8*k+1) -
         Decimal(2)/(8*k+4) -
         Decimal(1)/(8*k+5) -
         Decimal(1)/(8*k+6)) for k in range (precision))

baslangic = timer()
degerler = (1000, 1500, 2000)
data = list(map(pi, degerler))
print(data)
bitis = timer()
print(f'Ardisik hesaplama gecen sure: {bitis - baslangic}')
```

Çıktı

Ardisik hesaplama gecen sure:
6.162239899999999

Ardışık olarak hesaplama

Eş Zamanlılık ve Çoklu İşparçacıkları

Çok çekirdek ile PI hesaplama

Örnek Kod Parçacığı

```
from decimal import Decimal, getcontext
from timeit import default_timer as timer
from multiprocessing import Pool, current_process
import time


def pi(precision):
    getcontext().prec=precision

    return sum(1/Decimal(16)**k *
        (Decimal(4)/(8*k+1) -
         Decimal(2)/(8*k+4) -
         Decimal(1)/(8*k+5) -
         Decimal(1)/(8*k+6)) for k in range (precision))

if __name__ == '__main__':
    baslangic = timer()

    with Pool(3) as pool:

        values = (1000, 1500, 2000)
        data = pool.map(pi, values)
        print(data)

    bitis = timer()
    print(f'Paralel calisma gecen sure: {bitis - baslangic}')
```

Çıktı

```
Paralel calisma gecen sure: 4.2036388
```

Paralel çalışma

Eş Zamanlılık ve Çoklu İşparçacıkları

Process kendi bellek örneği

Örnek Kod Parçacığı

```
from multiprocessing import Process, current_process

data = [1, 2]

def denemeFunc():

    data.extend((3, 4, 5))
    print("Processteki veri: ", data)

if __name__ == '__main__':

    worker = Process(target=denemeFunc)
    worker.start()
    worker.join()

    print("Maindeki data:", data)
```

Çıktı

```
Processteki veri:  [1, 2, 3, 4, 5]
Maindeki data: [1, 2]
```

Process'lerdeki kullanılan bellekler farklı olduğu için data mainde değişmedi.