Paralel Hesaplama

Öğr. Gör. Zafer SERİN

PROCESS-THREAD KIYASLAMA

Thread	Process
Aynı bellek alanında çalışırlar.	Farklı bellek alanında çalışırlar
Daha az sistem kaynağı tüketirler.	Daha fazla sistem kaynağı tüketirler.
İzolasyonları yoktur.	İzolasyonları vardır.
GIL vardır.	GIL yoktur.

PROCESS OLUŞTURMA

- Pek çok programlama dilinde bir Process başlatıldığında parent(ebeveyn) Process'in bellek alanı tamamen kopyalanır ve yeni bir Process oluşturulmuş olur. Python programlama dilinde bu durum biraz daha özelleştirilebilir.
- Python'da Process başlangıç metodu olarak 3 yöntem kullanılabilir. Bunlar: spawn, fork ve forkserver olarak adlandırılır. Spawn yöntemi Windows ve UNIX işletim sistemlerinde çalışırken fork ve forkserver yöntemleri sadece UNIX işletim sistemlerinde çalışır.

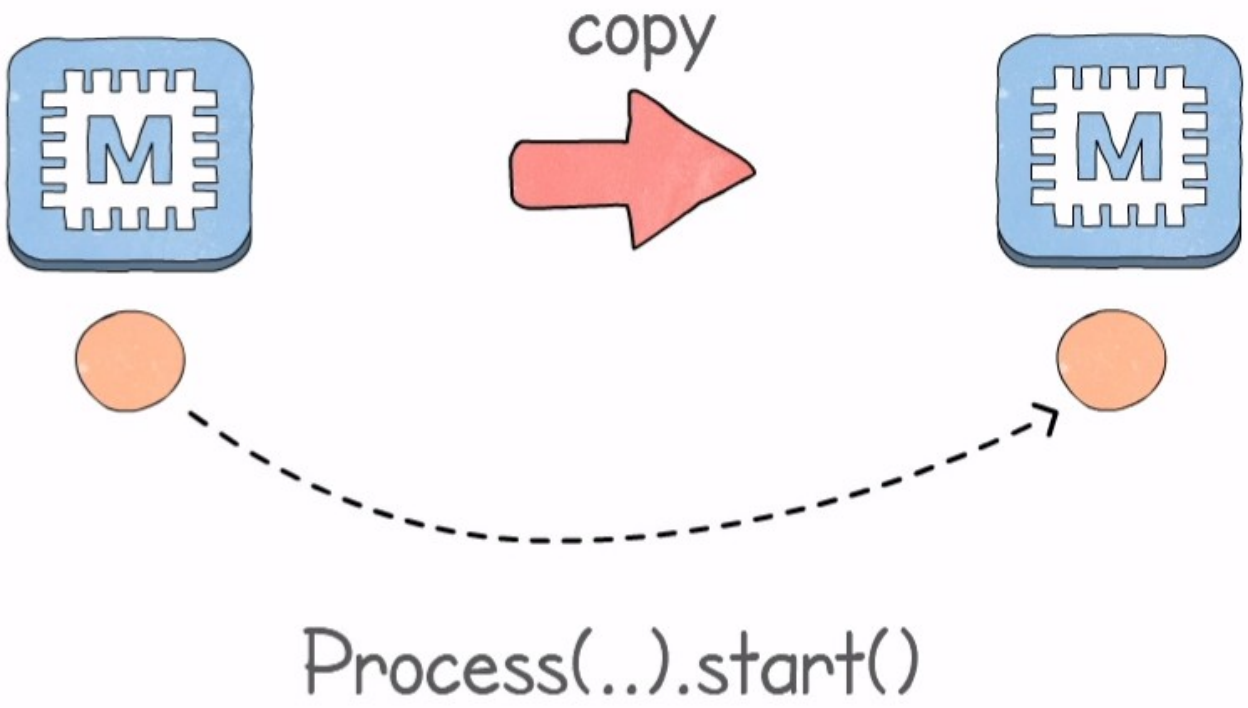
PROCESS OLUŞTURMA

- Fork yöntemi ile Process başlatıldığında tüm bellek alanı kopyalanırken Spawn ile Process başlatıldığında dosya tanımlayıcıları gibi bazı kısımlar kopyalanmaz. Bu sayede bellek alanından tasarruf edilmiş olur ama Process'i başlatmak daha uzun süre alır. Fork ile Process başlatıldığında ise tüm bellek alanı kopyalandığı için daha çok bellek kullanılır ama ayıklama ile uğraşmadığı için Process'i oldukça hızlı bir şekilde başlatır. Windows ve macOS işletim sistemleri için Spawn metodu varsayılan iken Linux işletim sistemi için Fork metodu varsayılandır.

PROCESS OLUŞTURMA

- ForkServer yöntemi ise diğer 2 yöntemin avantajlarını birlikte kullanmaya çalışan hibrit bir yaklaşımdır. Bu yöntemde ilk kez parent Process'ten bir başka Process oluşturulurken tüm bellek alanı kopyalanarak bir ForkServer oluşturulur ve bu ForkServer'ın belleği kullanılarak Process oluşturulur. 2. bir Process oluşturulmak istendiğinde bu sefer tüm işlemler yeniden yapılmak yerine 1. Process oluşturulurken kullanılan ForkServer belleği yeniden kullanılarak 2. Process oluşturulur.
- Spawn ve ForkServer yeni bir Python Interpreter'ı başlatırken Fork başlatmaz!

PROCESS OLUŞTURMA



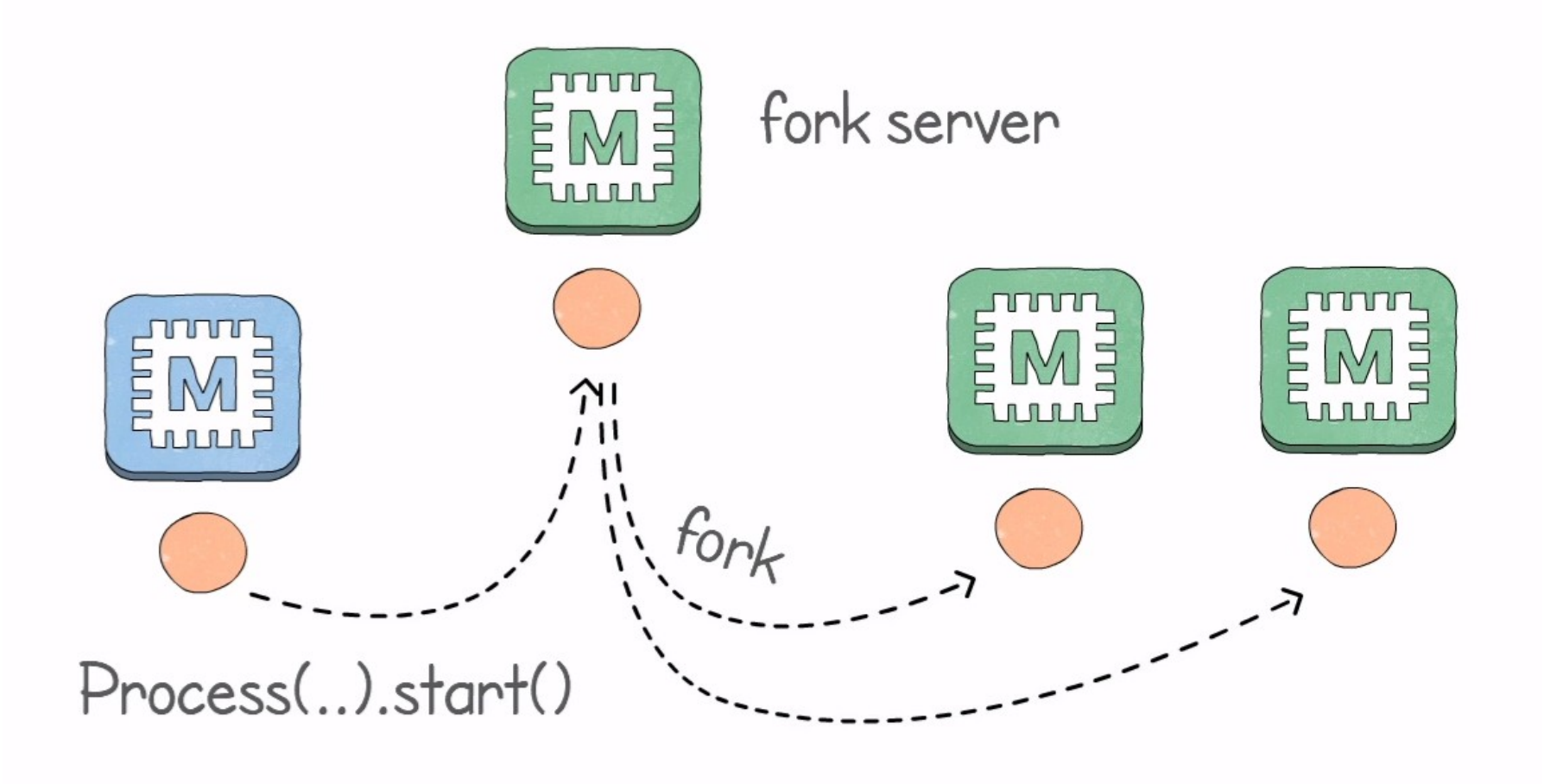
START METHOD:

'spawn'

'fork'

'forkserver'

PROCESS OLUŞTURMA



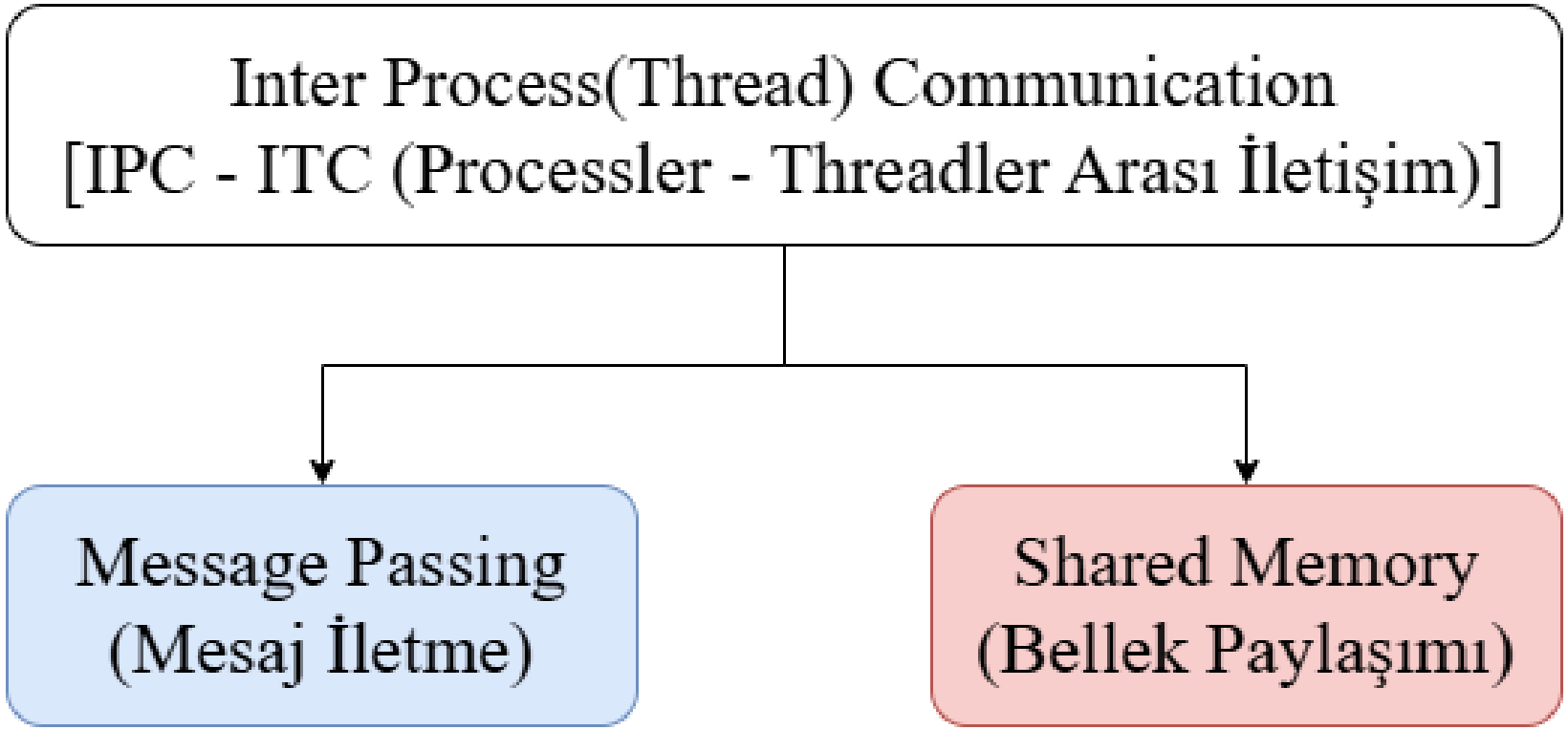
THREADLER VE PROCESSLER ARASI İLETİŞİM

- Thread’ler ve Process’ler yaptıkları işlem ile ilgili diğer Thread veya Process’leri bilgilendirme, şu anki durum ile ilgili bilgi vermelidir. Aksi taktirde yapılacak işlemler hatalı olacak ve sonuçlar problemli olacaktır. Thread ve Process’ler arası iletişim için Inter-Process(Thread)-Communication(IPC - ITC) terimi kullanılır. Bu iletişim temel olarak 2 şekilde gerçekleşebilir. Bunlar: Message Passingg(Mesaj Gönderme) ve Shared Memory(Bellek Paylaşımı) olarak adlandırılır.

THREADLER VE PROCESSLER ARASI İLETİŞİM

- Message Passing yöntemi aynı insanların iletişim yöntemine benzer şekilde gerçekleşir. Thread veya Process yaptığı işlem ile ilgili bilgiyi diğerlerine iletir. Sanki konuşuyormuş veya mesaj atıyormuş gibi düşünülebilir.
- Shared Memory yönteminde ise ortak bir bellek alanı tanımlanır ve bu alana diğer Thread veya Process'ler bilgiyi yazarak diğerlerinin de öğrenmesini sağlar. Bir tahtaya yazı yazılması ve herkesin bu yazıyı okuyabilmesi gibi düşünülebilir. Thread'ler zaten ortak bellek alanı kullandıkları için bu yöntem Thread'lerde daha kolay uygulanabilir.

THREADLER VE PROCESSLER ARASI İLETİŞİM



NEDEN SENKRONİZASYONA İHTİYACIMIZ VAR?

- Bir örnek ile anlatmak gerekirse Kazanan ve Harcayan isminde 2 kişi olduğunu düşünelim.
- Kazanan kişi 10 TL kazanmakta ve Harcayan kişi 10 TL harcamakta olsun
- Kazanan ve Harcayanın ortak bir banka hesabında başlangıç için 100 TL paraları olsun.
- Bu durumda Kazanan kişi banka hesabında ki 100 TL değerini okur ve 10 TL kazandığı zaman ilgili banka hesabına 110 TL olarak yeni değeri yazar.
- Harcayan kişi ise banka hesabındaki yeni 110 TL değerini okur

NEDEN SENKRONİZASYONA İHTİYACIMIZ VAR?

- Banka hesabındaki değerin sırayla okunması paralel değil seri hesaplamaya bir örnek olarak verilir.
- Paralel hesaplama söz konusu olduğunda banka hesabında ki 100 TL değeri hem Kazanan hem de Harcayan kişi tarafından aynı anda okunabilir. Bu durumda kazanan kişi 100 TL değerini ve Harcayan kişi de 100 TL değerini okuyacaktır.
- Kazanan kişi 100 TL değerini okuduğu için ilgili bellek alanına 10 TL ile birlikte 110 TL yazar ama Harcayan kişi de 100 TL başlangıç değerini okuduğı için o da ilgili bellek alanına 90 TL yazar.

NEDEN SENKRONİZASYONA İHTİYACIMIZ VAR?

- Bu işlem sonucunda başlangıçta 100 TL olan banka hesabında Kazanan kişi 10 TL kazanmış, Harcayan kişi ise 10 TL harcamış olmasına rağmen hesapta 90 TL kalmış olacak ve 10 TL kayıp olacaktır. İşte bu gibi bir problemin önlenmesi için Thread Senkronizasyonuna ihtiyacımız vardır. Yaşanan bu olaya race condition(yarış durumu) adı verilir.
- Burada Kazanan kişi 1.Thread, Harcayan kişi 2.Thread, ortak banka hesabı ve para değeri ise ortak kullanılan bellek alanını ifade etmektedir.
- Donanım özelliklerine, işletim sistemine, RAM miktarına vb. sebeplere bağlı olarak senkronizasyon olmasa da doğru sonuçlar elde edilebilir. Buna karşın doğru kodlama yapmak önemlidir.

NEDEN SENKRONİZASYONA İHTİYACIMIZ VAR?

- Yaşanan bu durum Process kullanırken de ortaya çıkabilir.

MUTEX İLE SENKRONİZASYON

- Mutex kullanımı en basit haliyle ilgili kodun belirli bir bölümüne aynı anda sadece 1 Thread veya Process'in müdahale edebileceği bir yaklaşımdır.
- İlgili kod bölümünün korunduğu, kilitlendiği düşünülebilir.

```
from threading import Lock
mutex = Lock()
mutex.acquire()
# Bu aralıktaki kodlara aynı anda sadece 1 Thread veya Process erişebilir
mutex.release()
```

Join Kullanımı

- Python ile Thread veya Process kullanırken join() metodu çağrılarak ilgili Thread veya Process'in bitmesi beklenebilir. Burada dikkat edilmesi gereken nokta join() metodu hangi Thread veya Process üzerinden çağrılıyorsa o Thread veya Process bitmeden kodun geri kalanının işlenmeyeceğidir!