## Noções de Programação Concorrente

---

O sistema operativo tem a responsabilidade de fornecer mecanismos de interação entre processos. Há normalmente dois padrões de interação entre processos:

- Cooperação : cooperam entre si para atingir um resultado em comum.
- Competição: competem por recurso (*CPU,* memória, impressora, *etc*).

Os casos anteriores são exemplos de sincronização, isto é, atraso deliberado de um processo até que determinado evento surja. Convém que este atraso seja passivo e não activo para não desperdiçar tempo de *CPU.*

Para haver interação tem que haver comunicação, que pode acontecer através de ficheiros, *pipas, sockets,* memória partilhada, *etc.*

- Comunicação : escrita/leitura de dados passados através de vários mecanismos (ficheiros, memória partilhada, …)
- Sincronização : coordenação de eventos entre processos (usando semáforos, variáveis de condição, *locks, etc*) para garantir ordem de eventos - “só leio depois de tu escreveres"

---

# Mecanismos de Sincronização

---

Exemplos de mecanismos de sincronização:

- Semáforos

- Exclusão mútua (*mutexes*, métodos *synchronized*)
- *Wait/signal/notify*

---

## Semáforos

Servem para resolver problemas de sincronização, exclusão mútua e controlo de capacidade e são definidos por três operações:

- Inicialização : s = cria_semaforo(valor_inicial)
- P(s) ou Down(s)
- V(s) ou Up(s)

---

Analogia: Imaginemos uma caixa cheia de bolas. Operação P : se há bolas na caixa, retiro uma e continuo, senão aguardo passivamente que alguém deposite uma; Operação V : devolvo a bola à caixa, se há alguém bloqueado à espera, acordo-o.

---

O pseudo-código de cada uma das operações é o seguinte:

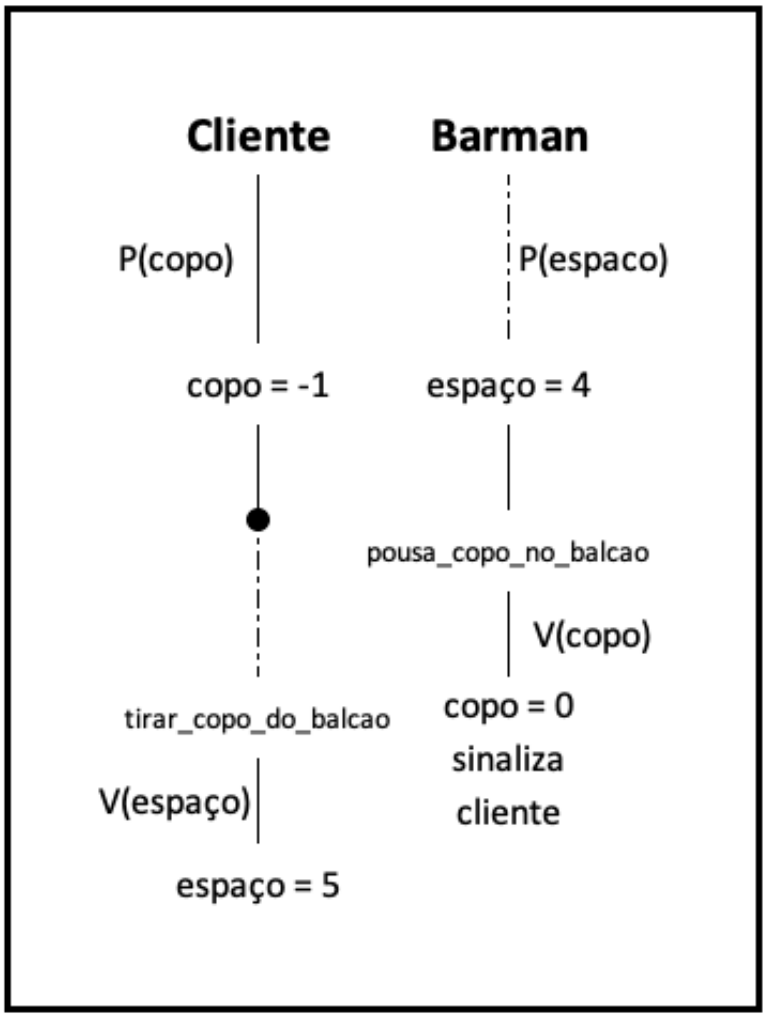
```
P(s){
    s = s - 1
    if( s < 0 )
        bloqueia(s)
}

V(s){
    s = s + 1
    if( s <= 0 )
        liberta(s)
}
```

Para exemplo, vai ser usado o problema de consumidor e cliente. Estão definidos dois semáforos para o exemplo, copo e espaço. O valor inicial do semáforo copo é de 0 e do espaço é a capacidade do balcão.

```
Cliente{
    P(copo)

    tira_copo_do_balcao()

    V(espaço)
}

Barman{
    P(espaço)

    pousar_copo_no_balcao()

    V(copo)
}
```

Assumindo que o espaço do balcão é 5, que corresponde ao valor inicial do semáforo espaço, e que no início não há copos, que corresponde ao valor inicial do semáforo copo, o comportamento pode ser o seguinte:

- Chega um cliente e pede um copo, baixando o valor do semáforo em 1, ficando a -1.
- Como é negativo, o cliente fica à espera.
- O barman retira um valor ao espaço do balcão ficando com o valor de 4.
- O barman coloca um copo no balcão.
- O barman indica que existe um copo, por isso o copo fica com o valor de 0 e liberta o cliente que estava à espera.
- O cliente prossegue e tira o copo do balcão.
- O cliente aumenta em 1 o espaço no balcão.




O algoritmo acima garante sincronização mas não exclusão mútua. Para isso é necessário criar um semáforo para o balcão, designado por *mutex*, com o valor inicial de 1, e variáveis *c* e *p* inicializadas a 0. Com isto, o código fica da seguinte maneira:

```
Cliente{
    P(copo)
        P(mutex)
            cx = buf[c++ % N]
        V(mutex)
    V(espaço)
}

Barman{
    P(espaço)
```

```
        P(mutex)
            buf[p++ % N] = px
        V(mutex)
    V(copo)
}
```

Ao começar o semáforo *mutex* com o valor de 1, é garantido que pelo menos um cliente ou barman acedem ao balcão. Se houver um segundo a querer aceder fica preso no *mutex*. Numa situação em que existem três copos no balcão, então o semáforo copo é igual a 3 e o semáforo espaço igual a 2. Quando um cliente pretende adquirir um copo, não fica preso no *P(copo)* pois existem copos disponíveis. Também não fica preso no semáforo *mutex* porque assumimos que neste momento é o único cliente. No momento que o cliente está a pegar no copo, entre *P(mutex) … V(mutex)* aparece outro cliente. Esse cliente não fica preso no semáforo copo pois ainda existem 2 copos. No entanto, vai ficar preso no *mutex* pois já existe alguém no balcão a pegar no copo. Quando o primeiro cliente liberta o balcão, o segundo é libertado e entra no balcão. Desta maneira garante-se que o balcão é mutuamente exclusivo.