Aula 6 – Sincronismo 2

Exclusão Mútua

- Variável de impedimento
- Locks e Rlocks
  - Lock → qualquer thread que tentar adquiri-lo irá bloquear, mesmo se o mesmo segmento em si já estiver segurando o bloqueio.
  - Nesses casos, RLock (bloqueio de reentrada) é usado.

Exclusão Mutua

- Lock X Rlock

```python
import threading

num = 0
lock = Threading.Lock()

lock.acquire()
num += 1
lock.acquire() # This will block.
num += 2
lock.release()
```

```python
import threading

# With RLock, that problem doesn't happen.
lock = Threading.RLock()

lock.acquire()
num += 3
lock.acquire() # This won't block.
num += 4
lock.release()
lock.release()
```

Exclusão Mútua

- Com alternância estrita

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

Exclusão Mútua

- Solução de Peterson

```
#define N          2                    /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
      int other;                        /* number of the other process */

      other = 1 – process;              /* the opposite of process */
      interested[process] = TRUE;       /* show that you are interested */
      turn = process;                   /* set flag */
      while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
      interested[process] = FALSE;      /* indicate departure from critical region */
}
```

Exclusão Mútua

- Instrução TSL
  - Auxílio do hardware

```
enter_region:
        TSL REGISTER,LOCK       | copy lock to register and set lock to 1
        CMP REGISTER,#0         | was lock zero?
        JNE enter_region        | if it was nonzero, lock was set, so loop
        RET                     | return to caller; critical region entered

leave_region:
        MOVE LOCK,#0            | store a 0 in lock
        RET                     | return to caller
```

Exclusão Mutua

- Soluções anteriores
  - com espera ocupada
- Soluções ideais
  - Com sleep e wakeup

Semáforos

- E.W. Dijkstra (1965) → variável inteira
  - Operação **down**: se maior que zero, decrementa
  - Operação **up**: se menor que o máximo, incrementa
- Up e down são generalizações de acquire e release
- Ações atômicas

Mutex

- Mutex → Mutual Exclusion
  - Semáforo binário

```
mutex_lock:
        TSL REGISTER,MUTEX      | copy mutex to register and set mutex to 1
        CMP REGISTER,#0         | was mutex zero?
        JZE ok                  | if it was zero, mutex was unlocked, so return
        CALL thread_yield       | mutex is busy; schedule another thread
        JMP mutex_lock          | try again
ok:     RET                     | return to caller; critical region entered

mutex_unlock:
        MOVE MUTEX,#0           | store a 0 in mutex
        RET                     | return to caller
```

Eventos

- Comunicação entre threads
  - Baseado em uma flag interna
  - Operações
    - set()
    - wait()
    - clear()

Eventos

```
1  import random, time
2  from threading import Event, Thread
3
4  event = Event()
5
6  def waiter(event, nloops):
7      for i in range(nloops):
8      print("%s. Waiting for the flag to be set." % (i+1))
9      event.wait() # Blocks until the flag becomes true.
10     print("Wait complete at:", time.ctime())
11     event.clear() # Resets the flag.
12     print()
13
14 def setter(event, nloops):
15     for i in range(nloops):
16     time.sleep(random.randrange(2, 5)) # Sleeps for some time.
17     event.set()
```

Condição

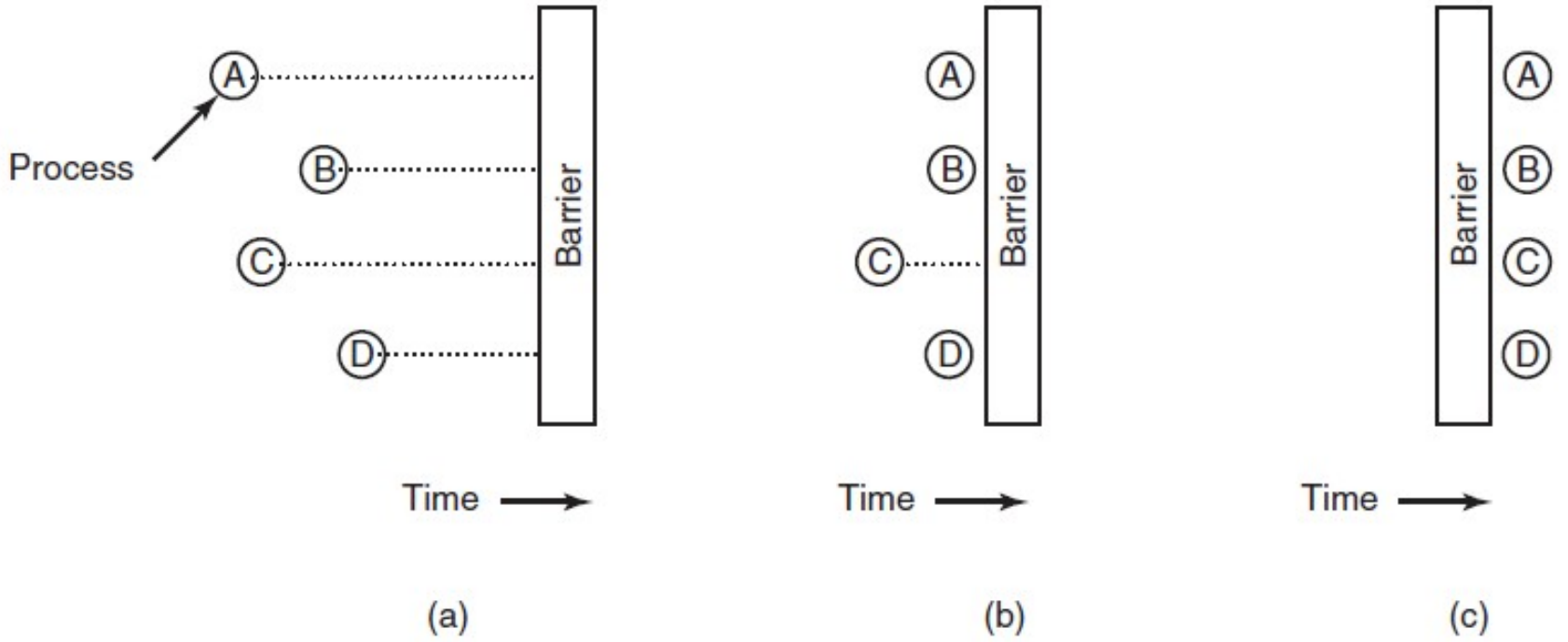
- Versão mais avançada de Evento
  - Thread tem que fazer um acquire() da condição antes do wait() um release() após
  - Para liberar as threads:
    - notify()
    - notifyAll()
  - Notificando uma ou mais threads
  - Também deve-se fazer acquire() e release()

Barreira

- Sincronização simples
  - Cada thread tenta ultrapassar uma barreira chamando o método wait()
  - Thread será bloqueada até que todas as demais threads tenham feito essa chamada
  - Assim que isso acontecer, as threads são liberados simultaneamente

Barreira

- Exemplo



Barreira

```
 1 from random import randrange
 2 from threading import Barrier, Thread, current_thread
 3 from time import ctime, sleep
 4
 5 b = Barrier(4)
 6
 7 def player():
 8     sleep(randrange(1, 6))
 9     print("%s reached the barrier at: %s" % (current_thread().name, ctime()))
10     b.wait()
11
12 threads = []
13 print("Race starts now…")
14 for i in range(4):
15     threads.append(Thread(target=player))
16     threads[-1].start()
17
18 for thread in threads:
19     thread.join()
20 print("Race over!")
```