Aula 7 – Sincronismo 3

Exclusão Mutua

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

Produtor x Consumidor

- Conjunto de processos que compartilham um mesmo buffer
  - produtores põem informação no buffer
  - consumidores retiram informação deste buffer

Produtor

- Exemplo (código em c)

```
#define N 100                                   /* number of slots in the buffer */
int count = 0;                                  /* number of items in the buffer */

void producer(void)
{
        int item;

        while (TRUE) {                          /* repeat forever */
                item = produce_item( );         /* generate next item */
                if (count == N) sleep( );       /* if buffer is full, go to sleep */
                insert_item(item);              /* put item in buffer */
                count = count + 1;              /* increment count of items in buffer */
                if (count == 1) wakeup(consumer);   /* was buffer empty? */
        }
}
```

Consumidor

- Exemplo (código em c)

```
void consumer(void)
{
     int item;

     while (TRUE) {                          /* repeat forever */
          if (count == 0) sleep( );          /* if buffer is empty, got to sleep */
          item = remove_item( );             /* take item out of buffer */
          count = count – 1;                 /* decrement count of items in buffer */
          if (count == N – 1) wakeup(producer);   /* was buffer full? */
          consume_item(item);                /* print item */
     }
}
```

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

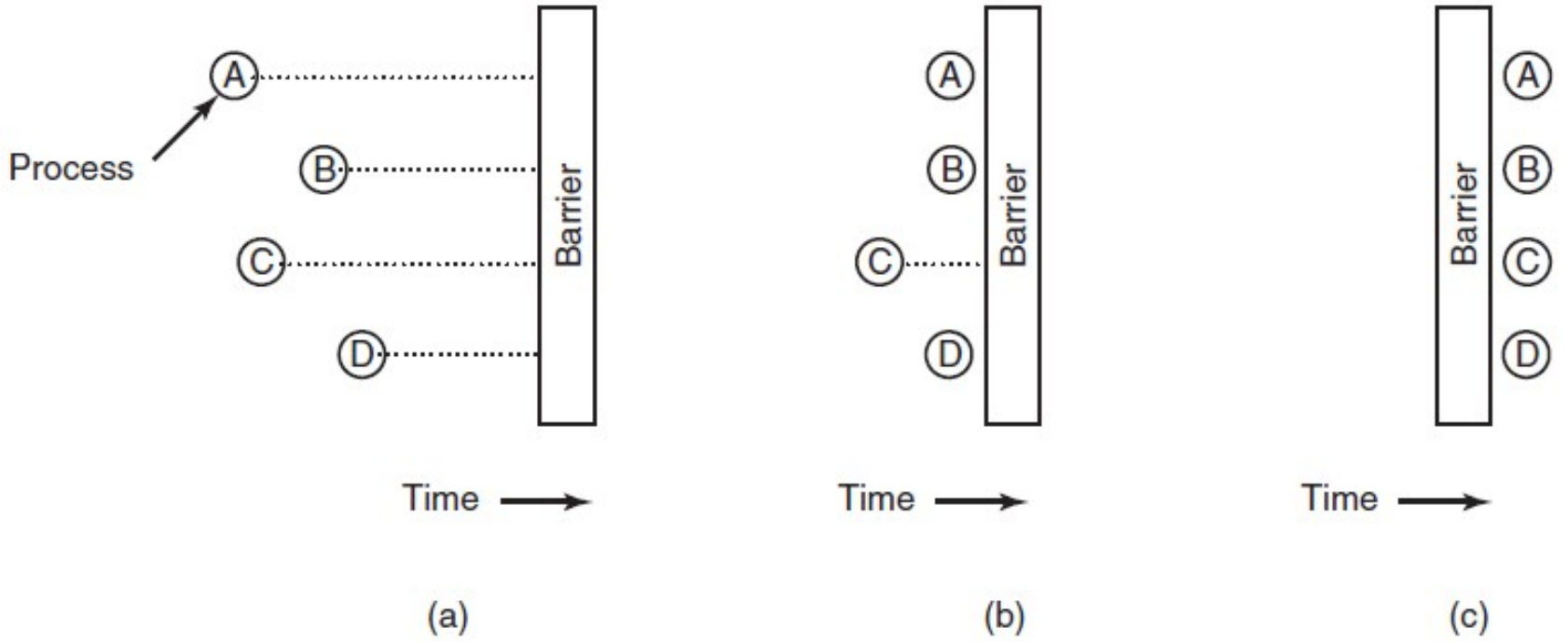
- Versão mais avançada de Evento
  - Thread tem que fazer um acquire() da condição antes do wait() um release() após
  - Para liberar as threads:
    - notify()
    - notifyAll()
  - Notificando uma ou mais threads
  - Também deve-se fazer acquire() e release()

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!")
```