(Bad) Producers Consumers **using a semaphore**

```
void producer () {
  while(1){
     item := produce()
     sem_wait(mutex)
     sem_wait(not_full)
     write(buffer, item)
     sem_signal(not_empty)
     sem_signal(mutex)
  }
}
```

```
void consumer () {
  while(1){
      sem_wait(mutex)
      sem_wait(not_empty)
      item := read(buffer)
      sem_signal(not_full)
      sem_signal(mutex)
      consume(item)
  }
}
```

- **Deadlock** : the producer waits for the consumer to release `mutex` while the consumer waits for producer to release `not_empty` (or vice versa)

```
sem_init(not_full, n)
sem_init(not_empty, 0)
sem_init(mutex, 1)
```

(Bad) Producers Consumers **using a semaphore**

```
sem_init(not_full, n)
sem_init(not_empty, 0)
sem_init(mutex, 1)
```

```
void producer () {
  while(1){
     item := produce()
     sem_wait(mutex)
     sem_wait(not_full)
     write(buffer, item)
     sem_signal(not_empty)
     sem_signal(mutex)
  }
}
```

```
void consumer () {
  while(1){
     sem_wait(mutex)
     sem_wait(not_empty)
     item := read(buffer)
     sem_signal(not_full)
     sem_signal(mutex)
     consume(item)
  }
}
```

- **Deadlock** : the producer waits for the consumer to release `mutex` while the consumer waits for producer to release `not_empty` (or vice versa)

Deadlock

Deadlock when one thread tries to access a resource that a second holds, and vice-versa

- They can never make progress

```
void thread1 (){
      ...
      sem_wait(sem1)
      sem_wait(sem2)
      /* critical section */
      sem_signal(sem2)
      sem_ignal(sem1)
      ...
}
```

```
void thread2(){
      ...
      sem_wait(sem2)
      sem_wait(sem1)
      /* critical section */
      sem_signal(sem1)
      sem_signal(sem2)
...
}
```