Bounded Buffer (Hoare)

```
get() {
    lock.acquire();
    while (nelem == 0)
        empty.wait(&lock);
    item = buf[front];
    front = front++ % size;
    nelem--;
    full.signal();
    lock.release();
    return item;
}
```

```
put(item) {
    lock.acquire();
    while (nelem == size)
        full.wait(&lock);
    buf[last] = item;
    last = last++ % size;
    nelem++;
    empty.signal();
    lock.release();
}
```

Bounded Buffer (Mesa)

```
get() {
    lock.acquire();
    if(nelem == 0 || !nextGet.empty()) {
        self = createCondition();
        nextGet.append(self);
        do self.wait(&lock);
        while(nelem == 0);
        nextGet.remove(self);
        destroyCondition(self);
    }
    ...
    item = buf[front];
    front = front++ % size;
    nelem--;
    if(!nextPut.empty())
        nextPut.first()->signal();
    lock.release();
    return item;
}
```

Bounded Buffer (Semafori e lock)

```
get() {
    empty.P();
    mutex.P();
    item = buf[front];
    front = front++ %. size;
    mutex.V();
    full.V();
    return item;
}
```

```
put(item) {
    full.P();
    mutex.P();
    buf[last] = item;
    last = last++ %. size;
    mutex.V();
    empty.V();
}
```

Condition Variables (Semafori)

```
wait(lock) {
    sem = createSemaphore();
    waiting.append(sem);
    lock.release();
    sem.P();
    destroySemaphore(sem);
    lock.acquire();
}
```

```
signal() {
    if(! waiting.empty()) {
        sem = waiting.remove();
        sem.V();
    }
}
```

Lock

```
LockAcquire() {
    disableInterrupts();
    if (val == BUSY) {
        waiting.add(myTCB);
        suspend();
    }
    else
        val = BUSY;
    enableInterrupts();
}
```

```
LockRelease() {
    disableInterrupts();
    if (!waiting.empty()){
        thTCB = waiting.remove();
        readyList.append(thTCB);
    }
    else
        val = FREE;
    enableInterrupts();
}
```

Spinlock

```
spinlockAcquire(&spinlockvalue){
    LOOP: TSL R3, &spinlock value
          CYP R3, #BUSY
          BEQ LOOP
          END
}

spinlockRelease(&spinlockvalue){
    MOV #FREE, &spinlockvalue
    MFENCE // memory barrier
}
```

Semafori con spinlock

```
P(Sem) {
    disableInterrupts();
    spinlockAcquire(&spinlock);
    if(sem.val == 0) {
        waiting.add(myTCB);
        suspend(&spinlock);
    }
    else
        sem.val--;
    spinlockRelease(&spinlock);
    enableInterrupts();
}
```

```
V(Sem) {
    disableInterrupts();
    spinlockAcquire(&spinlock);
    if(!waiting.empty()) {
        thTCB = waiting.remove();
        readyList.append(thTCB);
    }
    else
        sem.val++;
    spinlockRelease(&spinlock);
    enableInterrupts();
}
```