Primitivas de IPC: Implementación de semáforos

Primitivas de IPC: Propósito

- Ayudar con los problemas asociados a la comunicación interprocesos
    - Tanto para procesos de usuario,
    - como para procesos del sistema.

¿Por qué es parte del núcleo?

- Deben estar disponibles para todos los procesos
- La invocación de wait() puede hacer que un proceso se bloquée.
    - => wait() debe tener acceso al despachador
- Una manera simple de hacer ejecutable a un proceso, es invocar signal() sobre un semáforo sobre el que el proceso está esperando
    - => signal() debe ser accesible para las rutinas de admin. de interrupción

Qué operaciones debemos implementar?

- wait(*sem*):
    - Hasta que *sem* > 0: decrementa *sem*

- signal(*sem*):
    - incrementa *sem*

- (*sem* es algún semáforo)

Características

- bloquea/desbloquea procesos
- encola/desencola procesos
- cambia estado de procesos: despachador
- indivisibilidad

bloquea/desbloquea procesos

- Todos los procesos que hacen un wait() sobre un semáforo con valor cero deben ser bloqueados

- Al hacer un signal sobre un semáforo nulo, algún proceso debe desbloquearse

- Cola de procesos bloqueados

Cola de procesos bloqueados

- Una por cada semáforo
- Contiene a los procesos bloqueados en ese semáforo
- Un wait() que bloquea:
    - cambia el estado a no ejecutable, y añade el proceso a la cola
- Un signal() sobre un semáforo sobre el que hay procesos durmiendo:
    - saca algún proceso de la cola, y lo hace ejecutable

bloquea/desbloquea procesos

- Todos los procesos que hacen un wait() sobre un semáforo con valor cero deben ser bloqueados

- Al hacer un signal sobre un semáforo nulo, algún proceso debe desbloquearse

- Cola de procesos bloqueados

Pseudocódigo

```
def wait_handler(sem):
        CLI()
        if sem != 0:
                sem--
        else:
                Este hilo añade su PCB a la cola de espera de sem, y se hace
                no-ejecutable
        RSI()

def signal_handler(sem):
        CLI()
        if cola vacía:
                sem++
        else:
                quita algún proceso de la cola, y hazlo ejecutable
        RSI()
```

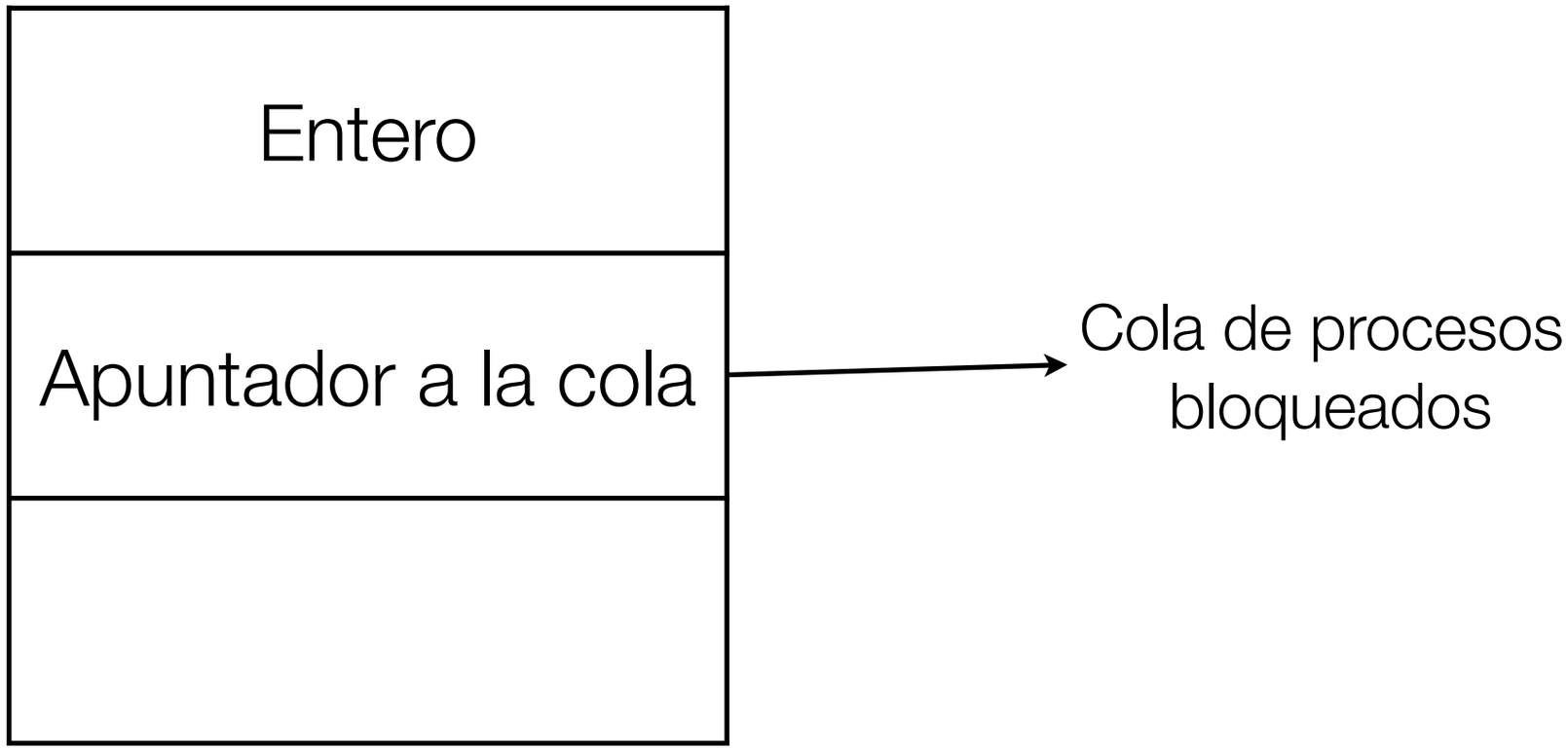
Organización de la cola

- A donde debe meterse un nuevo proceso: final, frente, enmedio...?

- Cuál proceso debe sacarse de la cola?

- Diferentes semáforos pueden tener diferente organización!

Estructura del semáforo



Características

- bloquea/desbloquea procesos
- encola/desencola procesos
- cambia estado de procesos: despachador
- indivisibilidad

Indivisibilidad

- Característica central de wait() y signal()

- No puede haber dos invocaciones sobre el mismo semáforo (en ejecución o suspendidas) *simultáneas*.

- Su ejecución debe comenzar con un tipo de bloqueo
    - y terminar con un desbloqueo

- Uniprocesador: podemos deshabilitar interrupciones.

- Multiprocesador: Necesitamos otras estrategias: test_and_set, etc.

Busy waiting: spinlocks

lock: variable booleana

spinlock: “girar alrededor de la variable, hasta que sea falsa”

tst: test and set

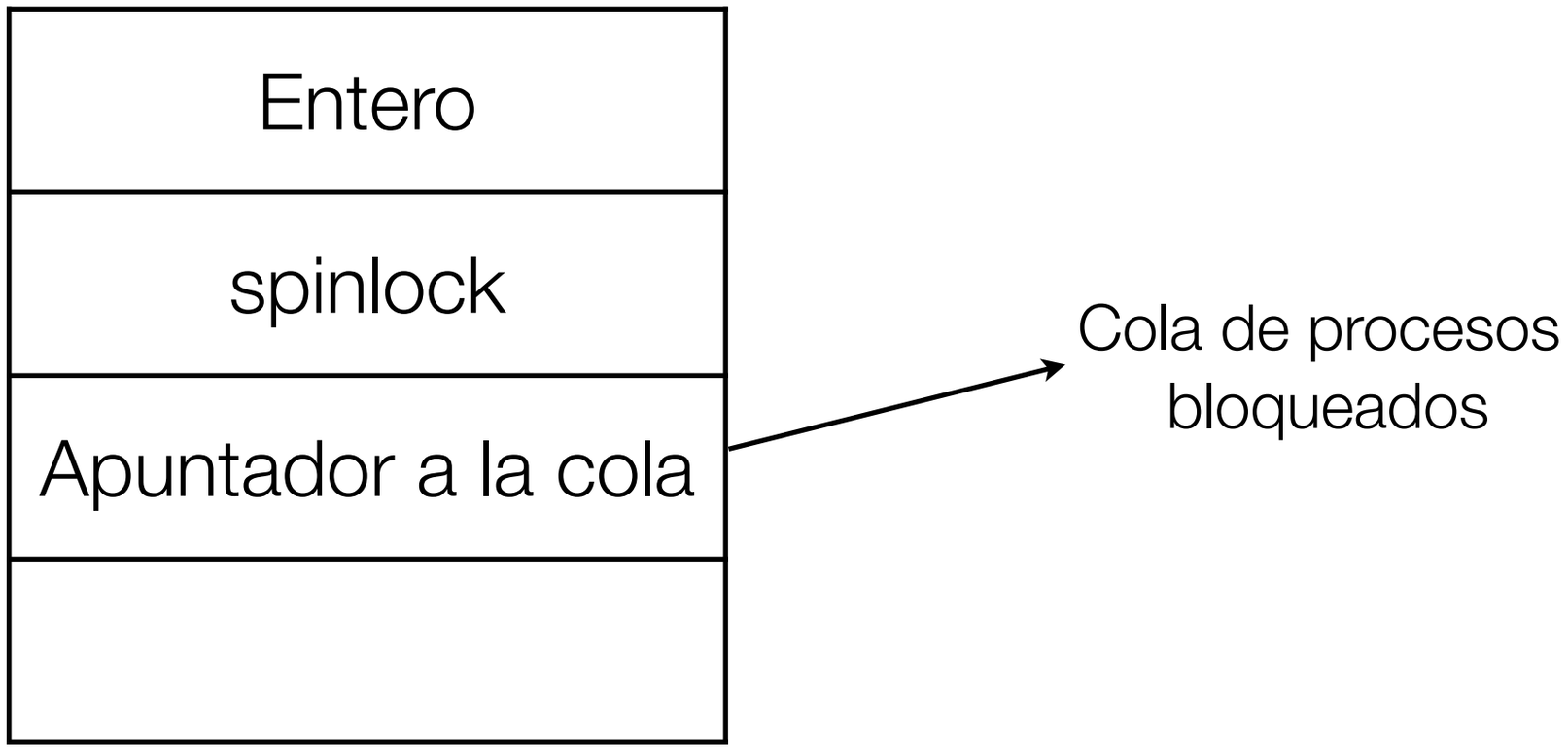
```
tst res, mem:
res := mem
        mem = true
```

```
xchange m1, m2:
        tmp := m1
        m1 := m2
        m2 := tmp
```

```
//init
mov locked, 0
-----------------------
        //spinlock
loop:tst status, locked
        cmp status, 1
        jeq loop
        //sección crítica
        //(actualización del semáforo)

        //fin sección crítica
        mov locked 0
```

Estructura del semáforo



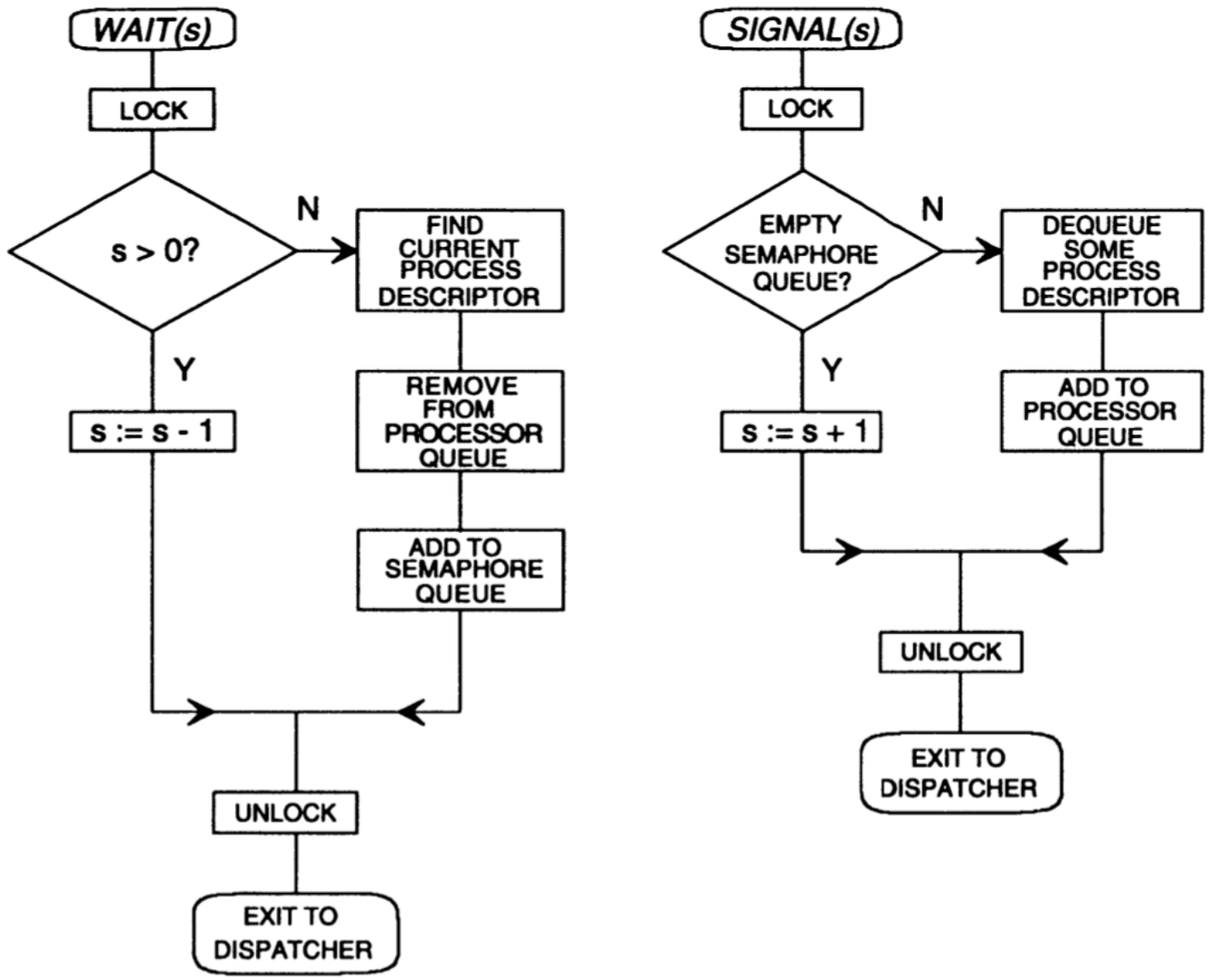


Figure 4.7 Implementation of wait and signal

Semáforos vs spinlocks/deshabilitar interrupciones

	Wait and *Signal*	*Lock* and *Unlock*
Purpose	General process synchronisation	Mutual exclusion of processes from *wait* and *signal* procedures
Implementation level	Software	Hardware
Delaying mechanism	Queueing	Busy waiting/interrupt inhibition
Typical delay time	Several seconds	Several microseconds