Azzolini Riccardo 2019-11-20

# Implementazione dei semafori

## 1 Strutture dati necessarie

L'implementazione di un semaforo richiede:

- un intero, per memorizzare il valore del semaforo;
- una variabile lock, per garantire l'indivisibilità di *wait* e *signal*;
- una lista dei processi bloccati sul semaforo, che:
  - è una struttura dati dinamica, perché la sua dimensione varia in base a quanti processi bloccati ci sono;
  - viene solitamente gestita come coda, per assicurare che i processi vengano sbloccati secondo la politica FIFO.

## 2 Idea dell'implementazione di wait e signal

```
procedure wait(sem) {
    if sem.value > 0 {
        sem.value--;
    } else {
        insert PCB in sem.list;
        block_me();
    }
}

procedure signal(sem) {
    if sem.list not empty {
        remove first process from sem.list;
        wake up process;
    } else {
        sem.value++;
    }
}
```

- La wait decrementa l'intero corrispondente al semaforo, se esso ha valore maggiore di 0, altrimenti inserisce il PCB del processo corrente nella coda dei processi bloccati sul semaforo (che può essere realizzata mediante i PCB pointer, come la coda dei processi ready usata dallo scheduler), e mette il processo in waiting (`block_me`).
- La signal rimuove il primo processo bloccato dalla coda e lo sveglia, o, se la coda è vuota, incrementa l'intero contenente il valore del semaforo.

## 3 Implementazione a livello kernel

In un'implementazione dei semafori *kernel level*:

- I semafori sono strutture dati del kernel, e, in particolare, la lista dei processi bloccati può essere realizzata sfruttando i PCB pointer.
- Wait e signal sono system call, solitamente accompagnate da delle funzioni wrapper di libreria per invocarle dai linguaggi ad alto livello.
- Per bloccare/sbloccare i processi, wait e signal lavorano direttamente sui PCB e sulle strutture dati di scheduling.
- L'indivisibilità di wait e signal può essere ottenuta:
  - usando variabili lock e istruzioni indivisibili (`TS/SWAP`): ciò causa busy wait, ma solo per il breve tempo necessario ad eseguire la wait/signal;
  - su sistemi uniprocessore, disabilitando gli interrupt (che permette di evitare il busy waiting).

## 4 Implementazione a livello utente

È possibile anche un'implementazione *user level* dei semafori. In tal caso:

- I semafori devono essere strutture dati accessibili in modalità user, quindi non è possibile usare direttamente i PCB pointer per la coda dei processi bloccati, dato che essi sono accessibili solo in modalità kernel. Una esempio di soluzione è una lista di PID.
- Wait e signal sono procedure di libreria, che eseguono in modalità user.
- Per bloccare/sbloccare i processi, non potendo lavorare direttamente sui PCB e sulle strutture dati di scheduling, wait e signal devono invocare le apposite system call messe a disposizione dal SO.

- L'indivisibilità di wait e signal si ottiene con variabili lock e istruzioni indivisibili (anche su sistemi uniprocessore, non c'è la possibilità di sfruttare invece la disabilitazione degli interrupt: ciò si può fare solo in modalità kernel).