— COMPITO CON SOLUZIONE —

Sistemi Operativi (modulo II / B)

16 Maggio 2012

Esercizio 1

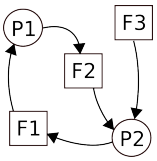
Si considerino 2 processi P1, P2 che necessitano di scrivere su 3 file F1, F2 e F3. La scrittura avviene in **mutua esclusione**: quando un file viene **aperto** (in scrittura) eventuali altri tentativi di aprire il file sono bloccanti. Considerare il seguente codice per P1 e P2:

```
process P1 {
    ...
    f1=open(F1,O_WRONLY); // apre in scrittura
    write(f1,data,datalen);
    f2=open(F2,O_WRONLY); // apre in scrittura
    write(f2,data,datalen);
    f3=open(F3,O_WRONLY); // apre in scrittura
    write(f1,data2,datalen);
    write(f3,data2,datalen);
    close(f1); close(f2); close(f3);
}

process P2 {
    ...
    f2=open(F2,O_WRONLY); // apre in scrittura
    write(f2,name,namelen);
    f3=open(F3,O_WRONLY); // apre in scrittura
    write(f3,name,namelen);
    f1=open(F1,O_WRONLY); // apre in scrittura
    write(f1,name2,namelen);
    write(f3,name2,namelen);
    close(f1); close(f2); close(f3);
}
```

1. Descrivere una possibile situazione di stallo disegnando il grafo di assegnamento delle risorse (spiegare brevemente)

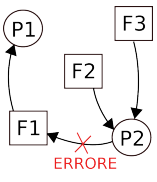
   Inizia P1 e viene interrotto dopo la prima open (F1). Parte P2 e apre F2 e F3 ma si blocca sulla open di F1, perché già aperto da P1. Quando P1 torna in esecuzione prova ad aprire F2 ma si blocca. I due processi sono in stallo in quanto ognuno attende un evento che può essere causato solo dall'altro processo. Il grafo di assegnamento mostra un ciclo tra P1 e P2 sulle risorse F1 e F2. Poiché c'è una sola istanza per risorsa il ciclo corrisponde a uno stallo.



2. Descrivere una tecnica a scelta per prevenire o controllare a run-time lo stallo e applicarla all'esempio precedente. Mostrare, in particolare, dove la tecnica interviene, perché viene evitato lo stallo e cosa accade ai processi P1 e P2.

   Si può utilizzare la tecnica dell'allocazione gerarchica: i file vengono ordinati come segue F1<F2<F3. Quando un processo apre un file tale file deve essere maggiore di tutti quelli già aperti da quel processo. Questo ordinamento stretto nella richiesta incrementale di risorse impedisce la formazione di attese circolari e di conseguenza previene lo stallo.

   Nel caso specifo quando P2 chiede F2, F3 e F1 si ha una violazione della politica e quindi l'ultima open restituisce un errore. Il processo P2 non esegue nessun controllo sul valore di ritorno della open e quindi prosegue. Anche la write successiva darà errore (anche in questo caso ignorato) mentre la scrittura su F3 avrà successo. Questo comportamento 'erroneo' di P2 non crea comunque una situazione di stallo in quanto i file verranno chiusi da P2 e lasciati a disposizione di P1. Illustriamo qui a lato il punto in cui avviene l'errore che impedisce la formazione della freccia di attesa da P2 a F1 evitando lo stallo.



Esercizio 2

Ci sono alcuni thread `automobile` che utilizzano un autolavaggio con `P` postazioni di lavaggio e `A` postazioni di aspirazione. Le auto si mettono in fila e attendono la prima postazione libera per il lavaggio e, successivamente, si mettono in fila e attendono la prima postazione libera per l'aspirazione, secondo il seguente schema:

```
thread automobile {
    autolavaggio.codaLavaggio();

    <lava l'auto>

    autolavaggio.codaAspirazione();

    <usa aspirapolvere>

    autolavaggio.esci();
}
```

Realizzare un monitor `autolavaggio` che implementi i metodi precedenti. Le code devono essere FIFO (First In First Out) evitare quindi di usare notifyAll. Spiegare brevemente la soluzione proposta.

Utilizziamo due condition per le due code. Utilizziamo inoltre la signal invece della notify che ci da un controllo più preciso su come i processi vengono sbloccati ed eseguiti realizzando un accesso FIFO. Questo ci permette, tra l'altro, di usare if invece che while per la condizione di attesa. (nota: l'uso di notify potrebbe non garantire FIFO nel caso un processo sbloccato venga superato da un altro che accede in quel momento al monitor. La soluzione con la notify e il while verrà comunque considerata corretta in quanto coerente con le esercitazioni in Java)
Per decidere se bloccarci usiamo due contatori `Npostazioni` e `Naspirazioni` inizializzati a `P` e `A` rispettivamente. Quando un'automobile utilizza una postazione o un aspirapolvere tali contatori vengono decrementati e i metodi diventano bloccanti quando i contatori valgono 0.

```
monitor autolavaggio {
    int Npostazioni = N, Naspirazioni = A;
    condition attendiPostazione, attendiAspirazione;

    codaLavaggio() {
        if (Npostazioni == 0) attendiPostazione.wait; // attende, se non ci sono postazioni
        Npostazioni - -; // decrementa il contatore
    }

    codaAspirazione() {
        Npostazioni ++; // libera una postazione
        attendiPostazione.signal; // sblocca la macchina successiva
        if (Naspirazioni == 0) attendiAspirazione.wait; // attende, se non ci sono aspirapolvere
        Naspirazioni - -; // decrementa il contatore
    }

    esci() {
        Naspirazioni ++; // libera un aspirapolvere
        attendiAspirazione.signal; // sblocca la macchina successiva
    }
}
```