Esercitazioni di Sistemi Operativi

19.2.2002

- **Esercizio 1**

  Dati P ($P > 2$) processi che comunicano per mezzo di un buffer circolare, definire lo schema di un generico processo produttore in cui l'uso coordinato del buffer è regolato per mezzo di semafori.

  Soluzione

  Sia buffer un array contenente N celle in cui vengono depositati i messaggi; siano vuoto e pieno due semafori inizializzati rispettivamente a N e 0; infine sia mutex un semaforo di mutua esclusione inizializzato a 1 e T l'indice della prima cella libera nel buffer.

  processo produttore

```
type item = ...
var N: integer;
    buffer: array[0..N − 1] of item;
    pieno: semaphore(initial 0);
    vuoto: semaphore(initial N);
    mutex: semaphore(initial 1);
    T: integer;

repeat
    < prepara messaggio mess >
    wait(vuoto);
       wait(mutex);
          buffer[T] := mess;
          T := (T + 1) mod N;
          signal(mutex);
       signal(pieno)
until(halt)
```

- **Esercizio 2**
  Due processi P e Q comunicano tramite un buffer di capacità un messaggio. P invia messaggi che sono stringhe con dimensione massima di 80 caratteri. Q estrae i caratteri uno alla volta. Definire i dati condivisi e scrivere le procedure *invio(x:A;y:B)* (y denota la dimensione di x) e *ricezione(var ch:char)*, con

```
type A = array[1..80] of char;
     B = 1..80;
```

Soluzione

```
var pieno: semaphore(initial 1);
    vuoto: semaphore(initial 0);
    index: integer;
    buff_msg: A;
    dim_msg: B;

Procedure invio(x:A;y:B)
begin
    wait(pieno);
    buff_msg := x;
    dim_msg := y;
    index := 1;
    signal(vuoto);
end;

Procedure ricezione(var ch: char)
begin
    wait(vuoto);
    ch := buff_msg[index];
    index := index+1;
    if index > dim_msg
        then signal(pieno)
        else signal(vuoto);
end;
```

- **Esercizio 3**

  Un conto corrente bancario è condiviso da diverse persone (processi). Ciascuno può depositare o prelevare soldi dal conto. Il saldo non deve mai diventare negativo.

  a) Scrivere un monitor per risolvere questo problema. Il monitor deve contenere due procedure entry *deposito(cifra)* e *prelievo(cifra)* (n.b. assumere *cifra* sempre positivo).

  b) Modificare il monitor del punto a) in modo che i prelievi vengano eseguiti con politica FIFO. Assumere che esista una funzione magica *quanto(sosp)* che ritorna la cifra richiesta dal primo processo in attesa sulla condition *sosp* (cioè in attesa del prelievo).

  c) Suppore che la funzione magica *quanto(sosp)* non esista. Modificare il monitor ottenuto in b) per simulare tale funzione nella soluzione.

  Soluzione

  a)

```
type contos = Monitor;
var saldo: integer;
    sosp: condition;
    n_sosp: integer;
Procedure entry deposito(cifra: integer)
begin
    saldo := saldo + cifra;
    if n_sosp > 0 then begin
        n_sosp := n_sosp − 1;
        sosp.signal;
    end;
end;
Procedure entry prelievo(cifra: integer)
var presi: boolean;
begin
    presi := false;
    repeat
        if saldo > cifra then begin
            saldo := saldo − cifra;
            presi := true;
        end;
        else begin
            n_sosp := n_sosp + 1;
            sosp.wait;
            if n_sosp > 0 then begin
                n_sosp := n_sosp − 1;
                sosp.signal;
            end;
        end;
    until presi;
end;
begin
    saldo := 0;
    n_sosp := 0;
end;
```

b)

```
type contom1 = Monitor;
var saldo: integer;
    sosp: condition;
    n_sosp: integer;
Procedure entry deposito(cifra: integer)
begin
    saldo := saldo + cifra;
    if (n_sosp > 0) and (quanto(sosp) < saldo) then begin
        n_sosp := n_sosp − 1;
        sosp.signal;
    end;
end;
Procedure entry prelievo(cifra: integer)
begin
    if (n_sosp > 0) or (cifra > saldo) then begin
        n_sosp := n_sosp + 1;
        sosp.wait;
    end
    saldo := saldo − cifra;
    if (n_sosp > 0) and (quanto(sosp) < saldo) then begin
        n_sosp := n_sosp − 1;
        sosp.signal;
    end;
end;
begin
    saldo := 0;
    n_sosp := 0;
end;
```

c)

```
type contom2 = Monitor;
var saldo: integer;
    sosp: condition;
    n_sosp: integer;
    c: coda;
```

Si suppone di usare la variabile *c* di tipo *coda*; *coda* è un tipo di dato astratto su cui sono definite le procedure *insert(coda,valore)* e *remove(coda)* che, rispettivamente, inseriscono e tolgono elementi dalla coda e una funzione *first(coda)* che restituisce il valore del primo elemento che verrà estratto dalla coda. Infine, la coda viene inizializzata per mezzo della funzione *crea(coda)*.

```
Procedure entry deposito(cifra: integer)
begin
    saldo := saldo + cifra;
    if n_sosp > 0 then begin
        if first(c) < saldo then begin
            n_sosp := n_sosp − 1;
            remove(c);
            sosp.signal;
        end;
    end;
end;
Procedure entry prelievo(cifra: integer)
begin
    if n_sosp > 0 then begin
        n_sosp := n_sosp + 1;
        insert(c,cifra);
        sosp.wait;
    end;
    else if cifra > saldo then begin
        n_sosp := n_sosp + 1;
        insert(c,cifra);
        sosp.wait;
        if first(c) < saldo then begin
            n_sosp := n_sosp − 1;
            remove(c);
            sosp.signal;
        end;
    end;
    else saldo := saldo − cifra;
end;
begin
    saldo := 0;
    n_sosp := 0;
    crea(c);
end;
```

- **Esercizio 4**

  Un semaforo si dice *privato* per un processo P_i, quando solo il processo P_i può eseguire sul semaforo la primitiva *wait*. La primitiva *signal* sul semaforo può essere invece eseguita anche da altri processi. Il semaforo privato viene inizializzato al valore 0.

  Si consideri un insieme di processi P_1, P_2, ..., P_n che utilizza un insieme di risorse comuni R_1, R_2, ..., R_m. Ogni processo può utilizzare una qualunque delle risorse; la condizione si riduce quindi a valutare se tra tutte le risorse ne esiste una libera. A ciascun processo è assegnata una priorità per cui in fase di rilascio di una risorsa, tra tutti i processi in attesa, deve essere scelto quello cui corrisponde la massima priorità.

  Soluzione

  Indichiamo con:

  PS[**i**] : una variabile binaria che assume il valore 1 se il processo P_i è sospeso, il valore 0 altrimenti.

  R[**j**] : una variabile binaria che assume il valore 1 se la risorsa j–esima è occupata, il valore 0 altrimenti.

  RA[**i**] : una variabile intera che rappresenta l'indice della risorsa assegnata al processo i–esimo; il valore 0 indica nessuna risorsa.

  DISP : una variabile intera che indica il numero delle risorse non occupate.

  SOSP : una variabile intera che indica il numero dei processi sospesi.

  mutex : un semaforo di mutua eslusione.

  priv$_i$: il semaforo privato del processo P_i.

  Lo schema seguito dal processo P_i per acquisire una risorsa è il seguente:

```
begin
    wait(mutex);
    if DISP > 0 then begin
        <Seleziona una risorsa K fra quelle disponibili
        utilizzando il vettore R >;
        DISP := DISP − 1;
        R[K] := 1;
        RA[i] := K;
        signal(priv_i);
    end;
    else begin
        SOSP := SOSP + 1;
        PS[i] := 1;
    end;
    signal(mutex);
    wait(priv_i);
    <Uso della risorsa RA[i] >;
end;
```

Lo schema seguito da un processo P_j per riattivare un processo sospeso dopo aver rilasciato la risorsa m–esima è il seguente:

```
begin
    wait(mutex);
    if SOSP ≠ 0 then begin
        < Seleziona il processo P_i di priorità maggiore tra
        quelli sospesi utilizzando il vettore PS >;
        SOSP := SOSP − 1;
        PS[i] := 0;
        RA[i] := m;
        signal(priv_i);
    end;
    else begin
        R[m] := 0;
        DISP := DISP + 1;
    end;
    signal(mutex);
end;
```

- **Esercizio 5**: *Selvaggi a cena!*
  Una tribù di N selvaggi mangia in comune da una pentola che può contener fino a M ($M < N$) porzioni di stufato di missionario. Quando un selvaggio ha fame si serve, a meno che la pentola sia vuota; in questo caso sveglia il cuoco ed aspetta che questo abbia riempito di nuovo la pentola. La struttura dei processi selvaggio(i) e cuoco è la seguente:

```
Selvaggio(i)          Cuoco
...                   ...
begin                 begin
    repeat                repeat
       serviti                riempi
       <mangia>           until false
       ...              end
    until false
end.
```

  Scrivere un monitor (con le procedure entry *serviti* e *riempi*) che controlla tale interazione. Il cuoco deve essere svegliato solo quando la pentola è vuota.

  Soluzione

```
type cm = Monitor;
var pasti: 0..M;
    cb: boolean;
    sb: integer;
    selv, cu: condition;
Procedure entry serviti
begin
    if pasti = 0 then
        if cb then begin
            cb := false;
            cu.signal
        end
        else begin
            sb := sb + 1;
            selv.wait
        end;
pasti := pasti − 1;
end;
Procedure entry riempi
begin
    if pasti > 0 then begin
        cb := true;
        cu.wait
    end;
    pasti := M;
    while sb > 0 do begin
        sb := sb − 1;
        selv.signal
        if pasti = 0 then pasti := M;
    end;
end;
```

```
begin
    pasti := 0;
    cb := false;
    sb := 0;
end;
```

Esercizi da svolgere

1. N processi P_1, P_2, P_3, ..., P_N condividono 3 stampanti; prima di usare una stampante un processo P_i invoca una *request(stamp)*, che ritorna l'identificatore della stampante libera da usare. Dopo avere utilizzato la stampante, il processo la rilascia con una *release(stamp)*.

   (a) Sviluppare un monitor che implementa *request* e *release*.

   (b) Assumendo che i processi abbiano associato delle priorità che vengono passate come parametro addizionale alla *request*, modificare le procedure in modo che le stampanti vengano assegnate ai processi in attesa con priorità maggiore (**NB**: le strutture dati già definite nel monitor non vanno modificate).

2. Consideriamo il *processo produttore* che produce pagine di stampa e le deposita nel buffer di una stampante laser. Il *processo consumatore* corrispondente procede al prelevamento dell'informazione dal buffer ed alla successiva fase di stampa.

   (a) Il produttore non può inserire una nuova pagina se il buffer della stampante è pieno;

   (b) Il consumatore non può prelevare una pagina per stamparla se il buffer è vuoto.

   Se N è il numero massimo di pagine che possono essere contenute nel buffer, d è il numero di pagine attualmente depositate dal produttore ed e il numero di pagine estratte dal consumatore, deve essere sempre verificata la condizione:

   $$0 \leq d - e \leq N.$$

   Descrivere la soluzione del problema, utilizzando due semafori, "*spazio–disponibile*" e "*pagina–disponibile*", i cui valori iniziali sono rispettivamente N e 0.

3. Si vuole modellare un sistema composto da 1 macchinetta distributrice di aranciata, 1 fornitore di aranciata e N=20 consumatori di aranciata.
   Requisiti:

   (a) La macchinetta fornisce due operazioni:
   - caricamento, che permette al fornitore di rifornire la macchinetta con M=10 aranciate;
   - prelievo, che permette ad un consumatore di prelevare una aranciata.

   (b) Il fornitore inizialmente carica la macchinetta, poi si sospende in attesa di essere chiamato a rifornire nuovamente la macchinetta vuota.

   (c) Ogni consumatore ha accesso alla macchinetta per prelevare aranciate.

   (d) Il primo consumatore che trova la macchinetta scarica deve chiamare il fornitore e attendere una nuova disponibilità di aranciata. Finché la macchinetta è vuota, i consumatori si devono sospendere in attesa di aranciata.

   Scrivere le procedure *carica* e *preleva* che simulano la situazione utilizzando il costrutto semaforico.