Le risorse

Dicesi risorsa un qualunque oggetto, fisico o logico, di cui un processo necessita per portare a termine la sua evoluzione.

Poiché un processo evolve eseguendo istruzioni (procedure), una risorsa è sempre una entità astratta rappresentata da operandi di istruzioni (o procedure) e il suo uso consiste nell'esecuzione di una delle istruzioni (o procedure) applicabili a detti operandi

Una risorsa, sia essa fisica o logica, è quindi un oggetto costituito da procedure e da una struttura dati allocati in memoria.

Alcune definizioni

Allocare una risorsa ad un processo significa rendere visibile la risorsa al processo in modo che questi ne possa richiedere l'uso ossia richiedere l'esecuzione di una delle sue procedure.

L'allocazione può essere:

statica: la risorsa è allocata ad un processo alla sua creazione e deallocata alla sua terminazione

dinamica: la risorsa e allocata e deallocata ad un processo durante la sua evoluzione

Alcune definizioni

Una risorsa può essere:

- **dedicata**: se è allocata ad un solo processo
- **condivisa**: se è allocata a più di un processo.
- **privata o locale**: se dedicata e allocata staticamente
- **comune o globale**: se condivisa e/o allocata dinamicamente
- **ad uso esclusivo**: se deve essere usata da un processo alla volta.

Detta sezione critica una sequenza di istruzioni che deve essere eseguita in modo esclusivo dai processi che possono eseguirla nel proprio contesto, le procedure d'uso di una risorsa ad uso esclusivo costituiscono un insieme di sezioni critiche

Il gestore delle risorse

Al concetto di risorsa si associa sempre il concetto di gestore, ossia di una entità il cui compito è quello o di consentire agli stessi processi la corretta allocazione delle risorse comuni controllandone il corretto utilizzo o di utilizzare le risorsa per eseguire le operazioni richieste dai processi

Il gestore delle risorse fisiche è sempre un componente del S.O. mentre il gestore delle risorse logiche può anche essere un componente di programma utente o lo stesso programmatore

Compiti del gestore

Compiti del gestore sono:

- consentire ai processi che desiderano usare risorse comuni di risolvere il problema della ***mutua esclusione*** per garantire il corretto utilizzo delle risorse ad uso esclusivo
- eseguire l'operazione sulla risorsa per conto dei processi.

Il problema della mutua esclusione può essere risolto:

- con l'allocazione dinamica della risorsa ad un unico processo
- disciplinando le richieste di utilizzo delle risorse condivise allocate staticamente

Il gestore può essere

il programmatore che effettua l'allocazione statica delle risorse in fase di stesura di un programma utilizzando le regole di visibilità del linguaggio e che può risolvere il problema di uso esclusivo di una risorsa da parte di un processo rendendola privata

un componente dell'elaborazione che in alternativa:

A) consente ai processi che operano su risorse comuni di risolvere il problema della mutua esclusione o mediante l'allocazione dinamica della risorsa ad un unico processo o disciplinando l'accesso alle risorse condivise allocate staticamente dal programmatore

B) esegue l'operazione sulla risorsa per conto dei processi

Il componente l'elaborazione può essere

- **CASO A: la risorsa "gestore"** costituita dalla struttura dati di gestione e da procedure che operano su tale struttura. La risorsa gestore è una risorsa condivisa per i processi che intendono utilizzare la risorsa gestita. La risorsa gestore è ad uso non esclusivo ed è allocata staticamente dal programmatore. Le procedure che operano sulla risorsa gestore ne costituiscono le procedure d'uso
- **CASO B: un processo gestore** di cui la struttura dati di gestione costituisce una risorsa privata
- .

La risorsa gestore

- la struttura dati di gestione memorizza lo stato della risorsa gestita e gli identificatori dei processi che la utilizzano o che attendono di utilizzarla
- le procedure operando sulla struttura dati di gestione consentono ad un processo di :
  - verificare se la sua richiesta d'uso della risorsa può essere soddisfatta e in caso contrario di sospendersi di prendere nota del rilascio della risorsa al termine delle operazioni
  - rilasciare una risorsa dopo il suo uso e di sboccare eventuali altri processi in attesa

Il processo gestore

Il processo gestore deve includere:

- una struttura dati di gestione, che memorizza lo stato della risorsa gestita e gli identificatori dei processi che sta servendo e che attendono di essere serviti
- la struttura dati della risorsa gestita
- procedure che, operando sulla struttura dati della risorsa gestita ne consentono l'uso

Modelli di programmazione concorrente

Le interazioni tra processi possono essere gestite con riferimento a due distinti modelli di programmazione concorrente:

- **modello a memoria globale o a memoria comune**
- **modello a scambio di messaggio o a memoria locale**

Il nucleo di un S.O. rende disponibili ai processi le primitive per risolvere i problemi di interazione secondo entrambi i modelli

Il modello a memoria globale

Le risorse comuni sono oggetti memorizzati nella memoria comune ai processi che interagiscono.

Pertanto:

- la **competizione** si manifesta nell'accesso alle risorse comuni ad uso esclusivo o ad uso non esclusivo ma con un numero massimo di utilizzatori minore del numero di processi richiedenti
- la **cooperazione** si realizza mediante lo scambio di informazioni tramite risorse comuni

Il gestore di risorse nel modello a memoria globale

Nel modello un gestore di risorse può essere:

- una risorsa gestore residente nella memoria comune
- un processo gestore con cui cooperano i processi che competono scambiando informazioni tramite risorse comuni.

Poiché il processo gestore comunque richiede a sua volta un gestore per cooperare con i processi utenti, nel modello a memoria comune in pratica il gestore è sempre una risorsa

Il gestore di risorse nel modello a memoria locale

- nel modello a memoria locale il gestore di risorse e' sempre un processo a servizio dei processi utenti
- il processo gestore e' detto **server** mentre i processi utenti sono detti **client**

Virtualizzazione delle risorse

La virtualizzazione di una risorsa consiste nel fare apparire privata una risorsa comune.

Detti ***richiesta*** e ***rilascio*** le procedure della risorsa gestore è possibile virtualizzare per i processi utenti la risorsa gestita definendo, per ogni possibile procedura d'uso ***proc_uso_i*** della risorsa, una procedura ***uso_di_risorsa_virtuale***

Procedure *uso_di risorsa_virtuale*
.....
begin
  richiesta
  proc_uso_i
  rilascio
end

I semafori

Un semaforo è un oggetto costituito da una variabile ***sem*** di tipo semaforo e da due procedure ***wait*** e ***signal***.

Al livello di macchina virtuale definita dal nucleo il semaforo è costituito da una variabile di tipo intero non negativo (*sem*>=0) che ne esprime il valore ed è allocata nella memoria comune e dalle primitive:

wait(*sem*): **repeat until** *sem*> 0;
sem:= *sem* - 1

signal(*sem*):*sem*:= *sem* + 1;

che il processo chiamante esegue sul proprio processore virtuale

Realizzazione dei semafori

Il nucleo del S.O. realizza un semaforo in modo da trasformare l'attesa attiva del processo sul proprio processore virtuale, che si manifesta se il processo esegue una wait con sem=0, in una attesa passiva per liberare il processore reale su cui il processo è running

A tal fine il ***tipo semaforo*** viene implementato con un tipo:

record
contatore: *integer*;
primo: *coda*;
end;

ove *contatore* contiene il valore del semaforo e *primo* consente di implementare la coda dei processi in attesa sul semaforo come coda dei loro PCB

Realizzazione dei semafori

La primitiva ***wait*** diventa:

```
procedure wait(sem);
begin
  if sem.contatore=0 then
    begin
      <inserisci il PCB del processo running in sem.coda>
      context_switch;
    end;
  else
    sem.contatore=sem.contatore -1;
end;
```

Realizzazione dei semafori

La primitiva ***signal*** diventa:

```
procedure signal(sem)
begin
  if <la coda sem.coda non è vuota> then
    <estrai dalla coda il primo PCB e ponilo nella coda dei PCB
    dei processi ready>
  else
    sem.contatore:= sem.contatore + 1
end
```

Il gestore di una risorsa R ad uso esclusivo, condivisa ed allocata staticamente

La mutua esclusione nell'uso di R può essere garantita dal seguente gestore:

var *mutex*:*semaforo*

procedure *richiesta*
begin *wait*(*mutex*) **end**

procedure *rilascio*
begin *signal* (*mutex*) **end**

Il gestore di un insieme di risorse equivalenti e ad uso esclusivo,

Detto N il numero di risorse R_i equivalenti, la mutua esclusione nell'uso di una R_i è ottenuta allocandola dinamicamente con il gestore:

```
var mutex:semaforo initial(1);
    risorse:semaforo initial(N);
    libero array [1..N] of boolean initial (true);
procedure richiesta (var x:1..N);
var i: 0..N
begin
    wait (risorse); wait(mutex);
    i:=0; repeat i:=i+1 until libero[i];
    libero[i]:= false; x:=i;
    signal (mutex);
end;
procedure rilascio (x:1..N)
begin
    wait (mutex); libero[i]:= true; signal(mutex);
    signal (risorse);
end;
```

Sincronizzazione di processi cooperanti

La forma più elementare di cooperazione tra due processi è la loro sincronizzazione senza scambio di informazioni: Ciò è facilmente ottenibile con l'uso di semafori.

Esempio

Siano:

P_m un processo master avente funzione di controllo

P_s un processo slave che attende di essere attivato

var *sem*: *semaforo initial* (0)

Pm: **begin** ..... *signal* (*sem*); .....**end**;

Ps: **begin**.....*wait* (*sem)*;.....**end**;

Schema produttore/consumatore

La forma più evoluta di cooperazione tra due processi è quella dello schema ***produttore/consumatore*** in cui due o più processi scambiano informazioni attraverso uno o più buffer.

Produttore: processo che genera un messaggio e lo deposita in un buffer

Consumatore: processo che preleva un messaggio da un buffer e lo utilizza

Vincoli di cooperazione

Il produttore non può inserire un nuovo messaggio in un buffer pieno

Il consumatore non può prelevare un messaggio in un buffer vuoto

Schema produttore/consumatore

Lo schema può essere risolto costruendo un gestore che renda disponibili le procedure:

richiesta_buffer_vuoto(*buf*), *rilascio_buffer_pieno*(*buf*)
richiesta_buffer_pieno(*buf*), *rilascio_buffer_vuoto*(*buf*)

con cui è possibile costruire le procedure:

procedure *invio*(*x*:*messaggio*);
begin
richiesta_buffer_vuoto(*buf*); *buf*:=*x*; *rilascio_buffer_pieno*(*buf*);
end;

procedure *ricezione*(*x*:*messaggio*);
begin
richiesta_buffer_pieno(*buf*); *x*:=*buf*; *rilascio_buffer_vuoto*(*buf*);
end;

Schema produttore/consumatore

Possibili realizzazioni delle procedure di richiesta e rilascio

Caso A: n produttori, m consumatori, 1 buffer

var *buffer_disponibile*: *semaforo initial*(1);
messaggio_disponibile: *semaforo initial*(0);

procedure *richiesta_buffer_vuoto*(*buf*);
begin *wait (buffer_disponibile*); **end**;

procedure *rilascio_buffer_pieno*(*buf*);
begin *signal(messaggio_disponibile)*; **end**;

procedure *richiesta_buffer_pieno*(*buf*);
begin *wait (messaggio_disponibile*); **end**;

procedure *rilascio_buffer_vuoto*(*buf*);
begin *signal(buffer_disponibile)*; **end**;

Schema produttore/consumatore

procedure *richiesta_buffer_pieno*(*buf*);
begin
wait(*messaggio_disponibile*);
prelievo_buffer_pieno(*buf*);
end;

procedure *rilascio_buffer_vuoto*(*buf*);
begin
deposito_buffer_vuoto(buf)
signal(*buffer_disponibile*)
end;

Schema produttore/consumatore

Caso B: n produttori, m consumatori, k buffer

var *buffer_disponibile*: *semaforo initial*(k);
messaggio_disponibile: *semaforo initial*(0);

procedure *richiesta_buffer_vuoto*(*buf*);
begin
wait(*buffer_disponibile*);
prelievo_buffer_vuoto(*buf*);
end;

procedure *rilascio_buffer_pieno*(*buf*);
begin
deposito_buffer_pieno(buf)
signal(*messaggio_disponibile*)
end;

Schema produttore/consumatore

Le procedure di prelievo e deposito hanno una struttura che dipende dal tipo di struttura dati utilizzata. Ad esempio:

procedure *prelievo_buffer_vuoto*(*buf*);
begin
wait(*mutex1*);
<preleva un buffer dalla coda dei buffer liberi e restituisci il puntatore in *buf*>;
signal (mutex1)
end;

procedure *deposito_buffer_pieno*(*buf*);
begin
wait(*mutex2*);
<immetti il buffer *buf* nella coda dei buffer pieni >;
signal (mutex2)
end;

Schema produttore/consumatore

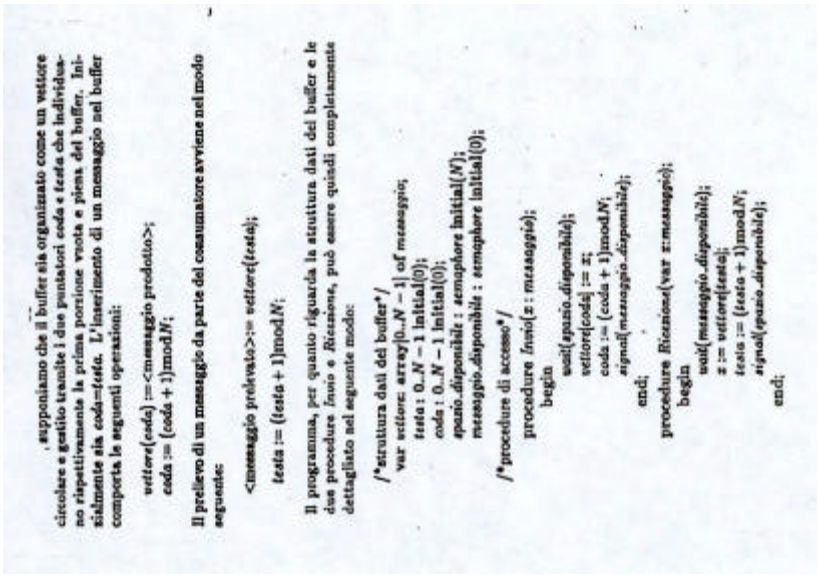
procedure *prelievo_buffer_pieno*(*buf*);
begin
wait(*mutex2*);
<preleva un buffer dalla coda dei buffer pieni e restituisci il puntatore in *buf*>;
signal (mutex2)
end;

procedure *deposito_buffer_vuoto*(*buf*);
begin
wait(*mutex1*);
<immetti il buffer *buf* nella coda dei buffer liberi >;
signal (mutex1)
end;

Uno schema produttore/consumatore semplificato

Nel seguito si riporta una possibile soluzione semplificata dello schema produttore/consumatore. La soluzione prevede di realizzare il pool di buffer con una coda circolare in cui un produttore immette gestendo un puntatore coda e un consumatore preleva gestendo un puntatore testa.

La soluzione e' semplificata in quanto, nel caso di piu' produttori e consumatori, non consente il parallelismo tra i produttori in fase di riempimento dei buffer e quello dei consumatori in fase di svuotamento.



, supponiamo che il buffer sia organizzato come un vettore circolare e gestito tramite i due puntatori *coda* e *testa* che individuano rispettivamente la prima posizione vuota e piena del buffer. Inizialmente sia *coda*=*testa*. L'inserimento di un messaggio nel buffer comporta le seguenti operazioni:

vettore(*coda*) :=<messaggio prodotto>;
coda := (*coda* + 1)mod*N*;

Il prelievo di un messaggio da parte del consumatore avviene nel modo seguente:

<messaggio prelevato>:= *vettore*(*testa*);
testa := (*testa* + 1)mod*N*;

Il programma, per quanto riguarda la struttura dati del buffer e le due procedure *Inserimento* e *Ricezione*, può essere quindi completamente dettagliato nel seguente modo:

```
/*struttura dati del buffer*/
var vettore: array[0..N – 1] of messaggio;
    testa : 0..N – 1 inizial(0);
    coda : 0..N – 1 inizial(0);
    spazio_disponibile : semaphore initial(N);
    messaggio_disponibile : semaphore inizial(0);
/*procedure di accesso*/
    procedure Inserimento(x : messaggio);
        begin
            wait(spazio_disponibile);
            vettore[coda] := x;
            coda := (coda + 1)mod N;
            signal(messaggio_disponibile);
        end;
    procedure Ricezione(var x:messaggio);
        begin
            wait(messaggio_disponibile);
            x := vettore[testa];
            testa := (testa + 1)mod N;
            signal(spazio_disponibile);
        end;
```

- PROBLEMA DEI LETTORI SCRITTORI

Più processi possono accedere contemporaneamente allo stesso insieme di dati (es. file).

LETTORI = Processi che possono solo leggere i dati

SCRITTORI = " " modificare i dati

VINCOLI DI USO DELLA RISORSA COMUNE:

1) I processi lettori possono accedere contemporaneamente alla risorsa

2) I processi scrittori accedono in mutua esclusione alla risorsa

3) Lettori e scrittori si escludono mutuamente nell'uso della risorsa

Varie soluzioni ... a seconda delle priorità dei lettori e scrittori

ad esempio una ipotesi è:

i processi lettori possono accedere alla risorsa anche se ci sono scrittori in attesa.

uno scrittore può accedere alla risorsa, solo se essa è libera.

Nell'ipotesi che più produttori e consumatori accedano allo stesso buffer, le operazioni di inserimento e prelievo devono essere eseguite rispettivamente in mutua esclusione ed essere quindi programmate come sezioni critiche. Ciò si ottiene introducendo due nuovi semafori *mutexI* e *mutexP*. Si ha pertanto:

```
var vettore: array[0..N − 1] of messaggio;
    testa : 0..N − 1 initial(0);
    coda : 0..N − 1 initial(0);
    mutexI: semaphore initial(1);

    mutexP: semaphore initial(1);
    spazio_disponibile: semaphore initial(N);
    messaggio_disponibile: semaphore initial(0);

procedure Invio(x : messaggio);
    begin
        wait(spazio_disponibile);
        wait(mutexI);
        vettore[coda] := x;
        coda := (coda + 1)mod N;
        signal(mutexI);
        signal(messaggio_disponibile);
    end;
procedure Ricezione(var x:messaggio);
    begin
        wait(messaggio_disponibile);
        wait(mutexP);
        x := vettore[testa];
        testa := (testa + 1)mod N;
        signal(mutexP);
        signal(spazio_disponibile);
    end;
```

Un comportamento analogo a quello dei processi lettori è ottenibile anche per i processi scrittori (con l'ovvia differenza che un solo processo alla volta può accedere alla risorsa) modificando nel modo seguente le procedure *Inizio_scrittura* e *Fine_scrittura*:

```
var mutex : semaphore initial(1);
    mutex1 : semaphore initial(1);
    mutex2 : semaphore initial(1);
    synch : semaphore initial(1);
    num_lettori : integer initial(0);
    num_scrittori : integer initial(0);

procedure Inizio_scrittura;
    begin
        wait(mutex2);
            num_scrittori := num_scrittori + 1;
            if num_scrittori = 1 then wait(synch);
        signal(mutex2);
        wait(mutex1);
    end;
procedure Fine_scrittura;
    begin
        signal(mutex1);
        wait(mutex2);
            num_scrittori := num_scrittori - 1;
            if num_scrittori = 0 then signal(synch);
        signal(mutex2);
    end;
```

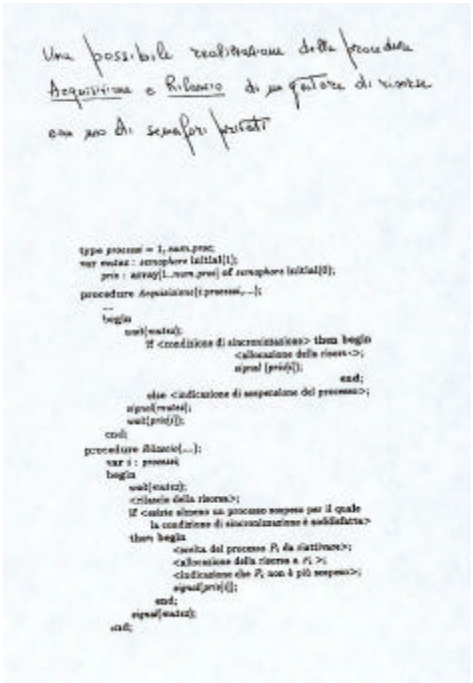
I semafori privati

un semaforo e' privato per un processo P se solo P può eseguire sul semaforo una primitiva *wait*

tutti i processi che condividono il semaforo possono eseguire la primitiva *signal*

il semaforo privato e'inizializzato a zero

i semafori privati consentono di realizzare gestori di risorse di tipo risorsa in cui la procedura di rilascio implementa una politica di scheduling del processo da sbloccare scelta in funzione della risorsa gestita e quindi differente a seconda dei casi



- l'operazione wait sul semaforo privato viene sempre eseguita anche quando il processo richiedente non deve essere bloccato

- il codice relativo all'assegnazione della risorsa viene duplicato nelle procedure *Acquisizione* e *Rilascio*

Lo schema seguente non ha questi inconvenienti:

```
var mutex : semaphore initial(1);
    priv : array[1..num_proc] of semaphore initial(0);
procedure Acquisizione(i:process);
    begin
        wait(mutex);
        if not <condizione di sincronizzazione> then
                begin
                    <indicazione di sospensione
                     del processo>;
                    signal(mutex);
                    wait(priv[i]);
                    <indicazione processo non
                      più sospeso>;
                end;
        <allocazione della risorsa>;
        signal(mutex);
    end;
procedure Rilascio(....);
    var i : process;
        begin
            wait(mutex);

    <rilascio della risorsa>;
    if <esiste almeno un processo sospeso
        per il quale la condizione di
        sincronizzazione è soddisfatta>
    then begin
                <scelta del processo Pi da riattivare>;
                signal(priv[i]);
            end;
    else signal(mutex);
end;
```

I semafori privati

si noti che pur essendo il primo schema di realizzazione delle procedure di acquisizione e rilascio meno elegante ed ottimizzato e' l'unico che consente, se e' tecnicamente possibile, di ripartire una risorsa rilasciata da un processo tra piu' processi in attesa ovvero di sbloccare con un rilascio piu' processi contemporaneamente

quanto sopra e' possibile in quanto e' possibile mettere in ciclo la verifica dell'esistenza di almeno un processo sospeso per il quale la condizione di sincronizzazione e' soddisfatta e, in caso affermativo, la scelta del processo da attivare e l'allocazione della risorsa richiesta con conseguente aggiornamento della quantità di risorsa disponibile.