Il nucleo di un sistema multiprogrammato (modello a memoria comune)

Nucleo di un Sistema a Processi

In un sistema multiprogrammato (o «a processi») vengono offerte tante unità di elaborazione astratte (macchine virtuali) quanti sono i processi; ogni macchina possiede come set di istruzioni elementari quelle corrispondenti all'unità centrale reale più le istruzioni relative alla creazione ed eliminazione dei processi, al meccanismo di comunicazione e sincronizzazione (compresa la comunicazione con i dispositivi di I/O visti come processori esterni).

Questo modello consente di mettere in evidenza le proprietà logiche di comunicazione e sincronizzazione tra processi senza doversi occupare degli aspetti implementativi legati alle particolari caratteristiche del processore fisico (es. gestione delle interruzioni).

Def: Si chiama nucleo (**kernel**) il modulo (o insieme di funzioni) realizzato in software, hardware o firmware che supporta il concetto di processo e realizza gli strumenti necessari per la gestione dei processi.

Il nucleo costituisce il livello più interno di un qualunque sistema basato su processi concorrenti. Ad esempio:

- il livello più elementare di un sistema operativo multiprogrammato;
- il supporto a tempo di esecuzione di un linguaggio per la programmazione concorrente.

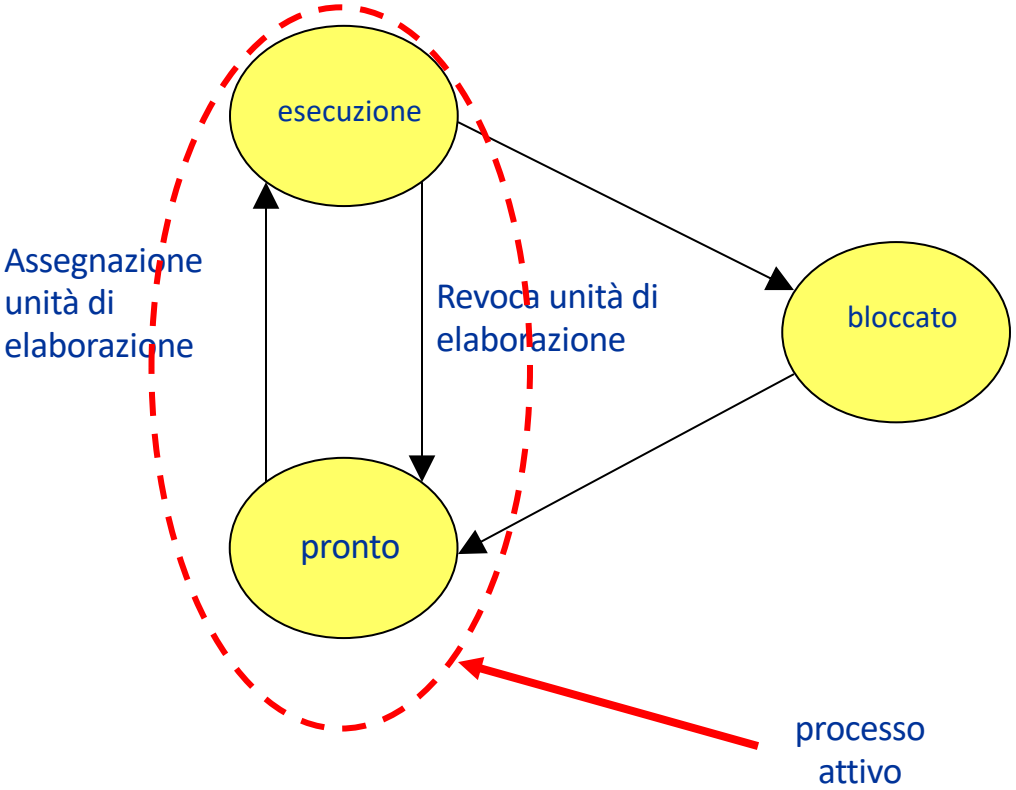
- Il nucleo è il solo modulo che è *conscio* dell'esistenza delle **interruzioni:**
  - ogni processo che richiede un'operazione ad un dispositivo utilizza un'opportune primitiva del nucleo che provvede a **sospenderlo** in attesa del completamento dell'azione richiesta.
  - Quando l'azione è completata, un **segnale di interruzione** inviato dal dispositivo alla CPU viene catturato e gestito dal nucleo, che provvede a risvegliare il processo sospeso.

- La gestione delle interruzioni è quindi invisibile ai processi ed ha come unico effetto rilevabile di rallentare la loro esecuzione sulle rispettive macchine virtuali.

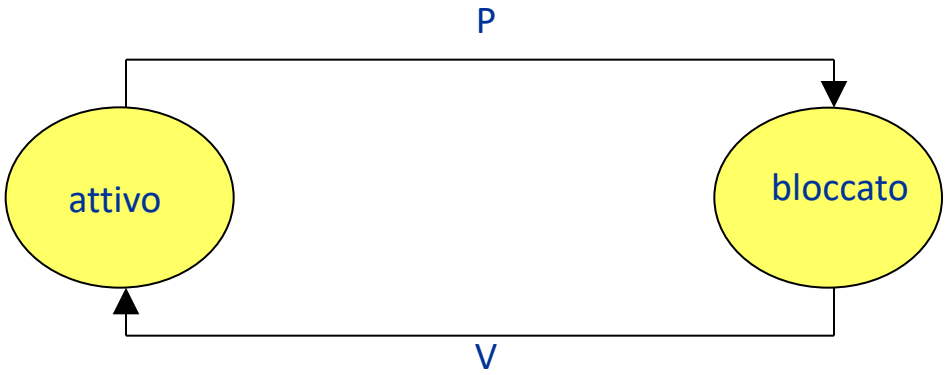
- Nel seguito verranno descritte le strutture dati e le procedure di un nucleo per sistemi monoprocessore e multiprocessore.

Obiettivo: realizzazione dei processi e loro sincronizzazione.

Stati di un processo:



Stati di un processo



Le transizioni tra i due stati sono implementate dai meccanismi di sincronizzazione realizzati dal nucleo.

Es: semaforo:

p per sospensione

v per risveglio.

Contesto di un processo: è l'insieme delle informazioni contenute nei registri del processore, quando esso opera sotto il controllo del processo.

Salvataggio di contesto: Quando un processo perde il controllo del processore, il contenuto dei registri del processore (contesto) viene salvato in una struttura dati associata al processo, chiamata **descrittore**.

Ripristino del contesto: Quando un processo viene schedulato, i valori salvati nel suo descrittore vengono caricati nei registri del processore.

Funzioni del Nucleo

Il compito fondamentale del nucleo di un sistema a processi **è gestire le transizioni di stato** dei processi. In particolare:

a) Gestire il **salvataggio** ed il **ripristino** dei **contesti** dei processi:

per ogni avvicendamento tra 2 processi P1 e P2 nell'uso della CPU, va salvato il contesto del processo deschedulato P1 (registri -> descrittore) e ripristinato il contesto del processo schedulato P2 (descrittore->registri).

b) Scegliere a quale tra i processi pronti assegnare l'unità di elaborazione (scheduling della CPU):

Quando un processo abbandona il controllo dell'unità di elaborazione, il nucleo deve **scegliere** tra tutti i processi pronti quello da mettere in esecuzione. La scelta è dettata dalla **politica di scheduling** adottata (es: FIFO, SJF, Priorità ecc.).

c) Gestire le interruzioni dei dispositivi esterni

traducendole eventualmente in attivazione di processi da bloccato a pronto.

d) Realizzare i meccanismi di sincronizzazione dei processi

gestendo il passaggio dei processi dallo stato di esecuzione allo stato di bloccato e da bloccato a pronto (es. primitive p e v).

Caratteristiche desiderabili del nucleo

- **Efficienza**: condiziona l'intera struttura a processi. Per questo motivo esistono sistemi in cui alcune o tutte le operazioni del nucleo sono realizzate in hardware o mediante microprogrammi.
- **Dimensioni**: la semplicità delle funzioni richieste al nucleo fa si che la sua dimensione risulti estremamente limitata
- **Separazione tra meccanismi e politiche**: il nucleo deve, per quanto possibile, contenere solo meccanismi consentendo così, a livello di processi, di utilizzare tali meccanismi per la realizzazione di diverse politiche di gestione a seconda del tipo di applicazione

Realizzazione del Nucleo

Modello a Memoria comune
Architettura Monoprocessore

STRUTTURE DATI del Nucleo

Descrittore del processo. Contiene le seguenti informazioni:

- **Identificatore del processo:**

  Ad ogni processo è associato un valore che lo identifica univocamente durante il suo tempo di vita.
- **Stato del processo:** pronto, esecuzione, bloccato ecc.
- **Modalità di servizio** dei processi: contiene parametri di scheduling. Ad esempio:
  - FIFO
  - Priorità (fissa o variabile)
  - Deadline. Il descrittore contiene un valore che sommato all'istante di richiesta di servizio da parte del processo determina il tempo massimo entro il quale la richiesta può essere soddisfatta (es. sistemi in tempo reale).
  - Quanto di tempo. (es. Sistemi time sharing).

- **Contesto del processo:**

  contatore di programma, registro di stato, registri generali, indirizzo dell'area di memoria privata del processo.

- **Riferimenti a code :**

  a seconda dello stato del processo, il suo descrittore può essere inserito in apposite code (es. coda dei processi bloccati su un dispositivo, ready queue, ecc). Ogni descrittore contiene, pertanto, il riferimento all'elemento successivo nella stessa coda.

Descrittore del Processo

Esempio di realizzazione:

```
typedef struct
{   int indice_priorità;
      int delta_t;
}modalità_di_servizio;

typedef struct
{     int nome;
      ...
      modalità_di_servizio servizio;
      tipo_contesto contesto;
      tipo_stato  stato; //esecuzione, pronto, bloccato, ecc.
      int   successivo;  // riferimento al prossimo elemento nella coda
}descrittore_processo;

/* array di tutti i descrittori:*/
descrittore_processo descrittori[num_max_proc];
```

HP: scheduling pre-emptive a divisione di tempo con priorità

Coda dei processi pronti

Esistono una o più (caso di scheduling con priorità) **code di processi pronti.** Quando un processo e` riattivato per effetto di una v, viene inserito al fondo della coda corrispondente alla sua priorità.

Non è detto che vi sia sempre almeno un processo pronto:

- La coda dei processi pronti contiene sempre almeno un processo fittizio (***dummy process***) che va in esecuzione quando tutte le altre code sono vuote. Il processo dummy ha la priorità più bassa ed è sempre nello stato di pronto.

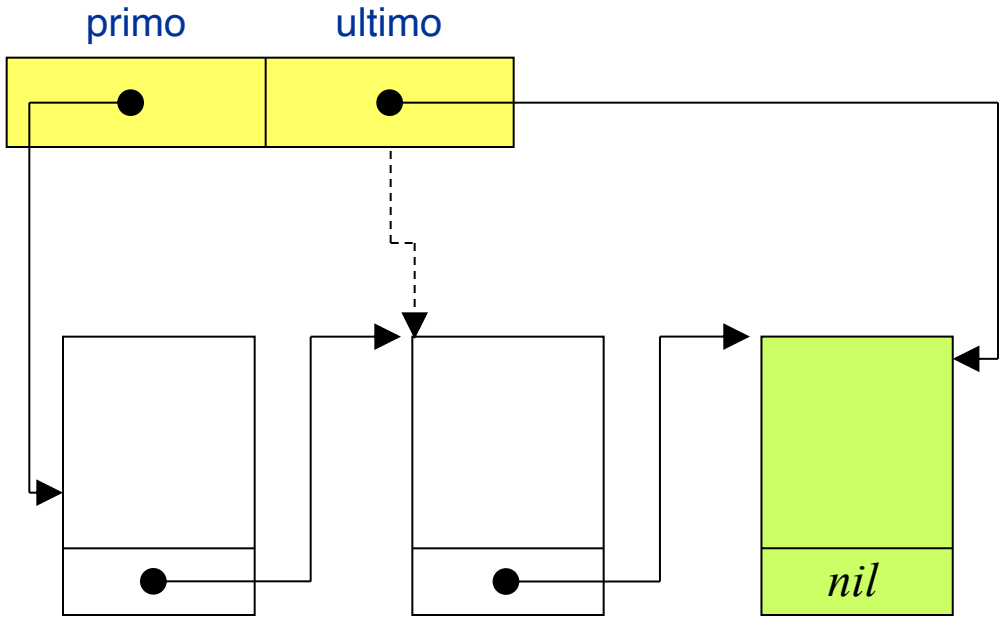
- Il processo dummy rimane in esecuzione fino a quando qualche altro processo diventa pronto, eseguendo un ciclo senza fine.

Coda dei processi pronti

Esempio di realizzazione:

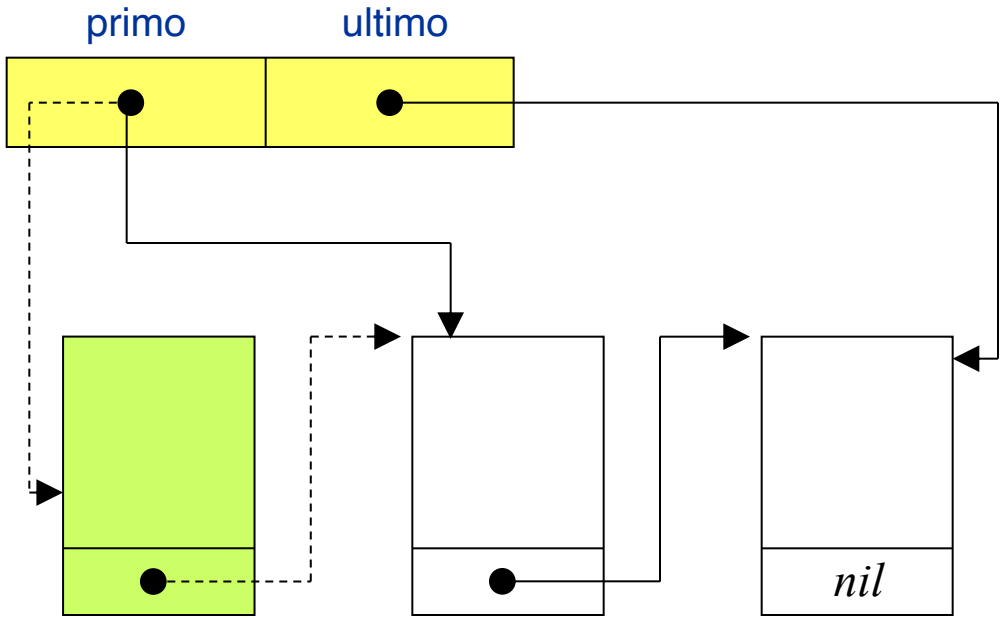
```
typedef struct
{int primo,ultimo;}descrittore_coda;

typedef descrittore_coda coda_a_livelli[Npriorità];

coda_a_livelli  coda_processi_pronti;
```

Inserimento di un descrittore in coda:




```
void Inserimento(int P, descrittore_coda C){..}
/* inserisce il processo di indice P nella coda C */
```

Prelievo di un descrittore da una coda:




```
int Prelievo(descrittore_coda C){..}

/* estrae il primo processo dalla coda C e restituisce
il suo indice */
```

Coda dei descrittori liberi:

- Coda nella quale sono concatenati i descrittori *disponibili* per la creazione di nuovi processi e nella quale *sono re-inseriti* i descrittori dei processi terminati:

```
descrittore_coda descrittori_liberi;
```

Processo in esecuzione:

Il nucleo necessita di conoscere quale processo è in esecuzione. Questa informazione, rappresentata dall'indice del descrittore del processo, viene contenuta in una particolare variabile del nucleo (spesso, un registro del processore) :

```
int  processo_in_esecuzione; /* indice del processo running*/
```

Quando il nucleo è inizializzato (il che avviene durante l'operazione di bootstrap dell'elaboratore), viene creato un processo e l'indice del processo viene assegnato a **`processo_in_esecuzione`**.

Funzioni del Nucleo

- Le funzioni del nucleo realizzano le operazioni di **transizione di stato** per i singoli processi. Ogni transizione prevede il prelievo, da una coda, del descrittore del processo coinvolto ed il suo inserimento in un'altra coda.
- Si utilizzano a questo scopo due procedure: **Inserimento** e **Prelievo** di un descrittore da una coda. Se la coda è vuota: valore -1 (NIL).
- Analogamente verrà assegnato il valore -1 al campo successivo dell'ultimo descrittore nella coda.

Struttura del nucleo

- La struttura del nucleo è articolata in due livelli:
  - **Livello superiore**. Contiene tutte le **funzioni direttamente utilizzabili dai processi** sia interni sia esterni (dispositivi di I/O); in particolare, le primitive per la creazione, eliminazione e sincronizzazione dei processi e le funzioni di risposta ai segnali di interruzione.
  - **Livello inferiore**: realizza le funzionalità di **cambio di contesto**: salvataggio del contesto del processo deschedulato, scelta di un nuovo processo da mettere in esecuzione tra quelli pronti e ripristino del suo contesto.

Esecuzione del kernel

Le funzioni del nucleo, per motivi di protezione, sono le sole che:

- possono operare sulle strutture dati che rappresentano lo stato del sistema (descrittori, code di descrittori, semafori…)
- possono utilizzare istruzioni privilegiate (abilitazione e disabilitazione delle interruzioni, invio di comandi ai dispositivi).

Le funzioni del nucleo devono essere eseguite in modo mutuamente esclusivo.

👉 Pertanto **nucleo e processi eseguono in due ambienti separati**:

- Il nucleo esegue nel modo di **massimo privilegio** (**modo kernel**, o ring 0).
- I processi di utente eseguono a un **minore livello di privilegio** (**modo user**, o ring >0).

I due ambienti di esecuzione corrispondono a modi (ring) diversi di esecuzione della CPU (kernel e user). Il meccanismo di passaggio da uno all'altro è basato sul meccanismo delle interruzioni.

In particolare:

- Nel caso di funzioni chiamate da **processi esterni** (**dispositivi**), il passaggio all'ambiente del nucleo è ottenuto tramite il meccanismo di risposta al segnale di **interruzione** (interruzioni esterne)

- Nel caso di funzioni chiamate da **processi interni**, il passaggio è ottenuto tramite l'esecuzione di **system calls** (chiamate al supervisore, SVC -> interruzioni interne).

- In entrambi i casi, al completamento della funzione richiesta, il trasferimento all'ambiente user avviene tramite il meccanismo di **ritorno da interruzione** (**RTI**).

Funzioni del livello inferiore: CAMBIO DI CONTESTO

- Salvataggio del contesto del processo in esecuzione nel suo descrittore e inserimento del descrittore nella coda dei processi bloccati o dei processi pronti. (**Salvataggio_stato**)
- Rimozione del processo a maggior priorità dalla coda dei pronti e caricamento dell'identificatore di tale processo nel registro processo in esecuzione (**Assegnazione_CPU**)
- Caricamento del contesto del nuovo processo nei registri di macchina.(**Ripristino_stato**)

Realizzazione

```
void Salvataggio_stato( )
{       int j;
        j = processo_ in_esecuzione;
        descrittori[j].contesto= <valori dei registri CPU>;
}

void Ripristino_stato( )
{ int j;
  j = processo_ in_esecuzione;
  <registro-temp>=descrittori[j].servizio.delta_t;
  <registri-CPU>= descrittori[j].contesto ;
}

void Assegnazione_CPU( ) // scheduling: hp algoritmo con priorità
{ int  k=0,j;
        while (coda_processi_pronti[k].primo)==-1)
                k++;
  j=Prelievo (coda_processi_pronti [k]);
  processo_in_esecuzione=j;
}
```

Gestione del temporizzatore

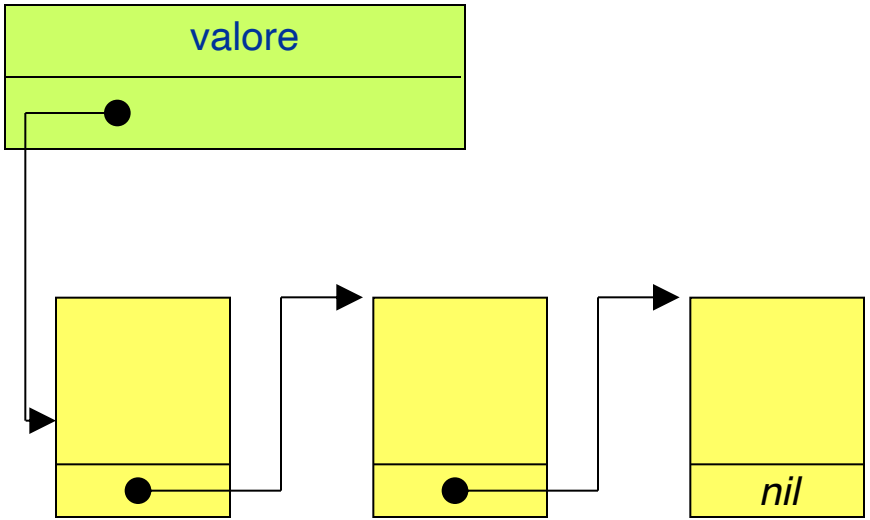
Per consentire la modalità di servizio a divisione di tempo è necessario che il nucleo gestisca un **dispositivo temporizzatore** tramite un'apposita procedura che ad intervalli di tempo fissati, provveda a sospendere il processo in esecuzione ed assegnare l'unità di elaborazione ad un altro processo.

```
void Cambio_di_Contesto()
{               int j,  k ;
        Salvataggio_stato();
        j = processo_in esecuzione;
        k=descrittori[j].servizio.priorità;
        Inserimento (j,coda_processi_pronti[k]);
        Assegnazione_CPU ( );
        Ripristino_stato ( );
}
```

Realizzazione del semaforo (caso monoprocessore)

Semafori (caso monoprocessore)

- Nel nucleo di un sistema monoprocessore il semaforo può essere implementato tramite:
  - Una **variabile intera** che rappresenta il suo valore(>=0)
  - una coda di descrittori dei processi in attesa sul semaforo (bloccati)



Se non ci sono processi in coda, il puntatore contiene la costante nil (ad esempio, -1).

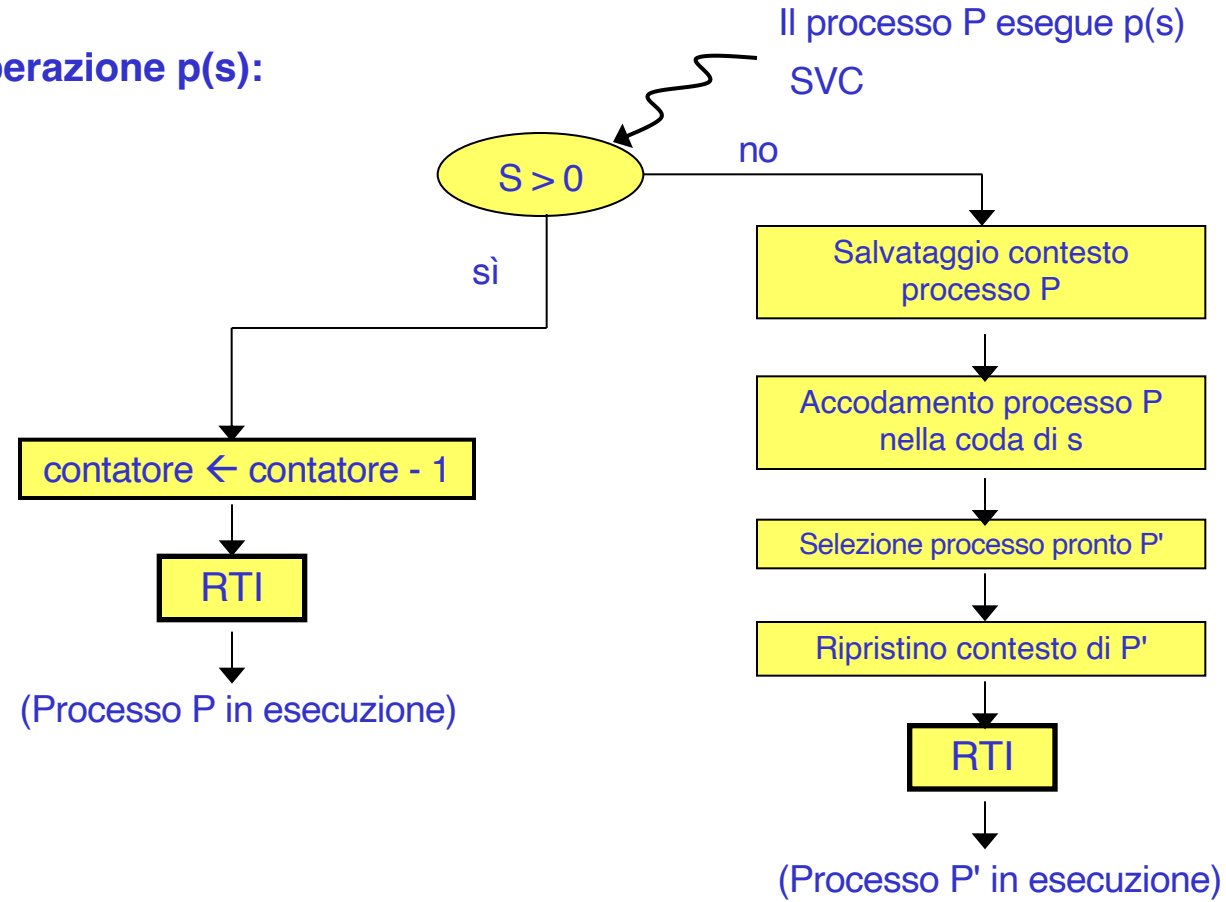
La coda viene gestita con politica FIFO: i processi risultano ordinati secondo il loro tempo di arrivo nella coda associata al semaforo.

Il descrittore di un processo viene inserito nella coda del semaforo come conseguenza di una primitiva p non passante; viene prelevato per effetto di una v.

```
typedef struct
{   int contatore;
    descrittore_coda coda;
} descr_semaforo;
```

Semafori: operazione p

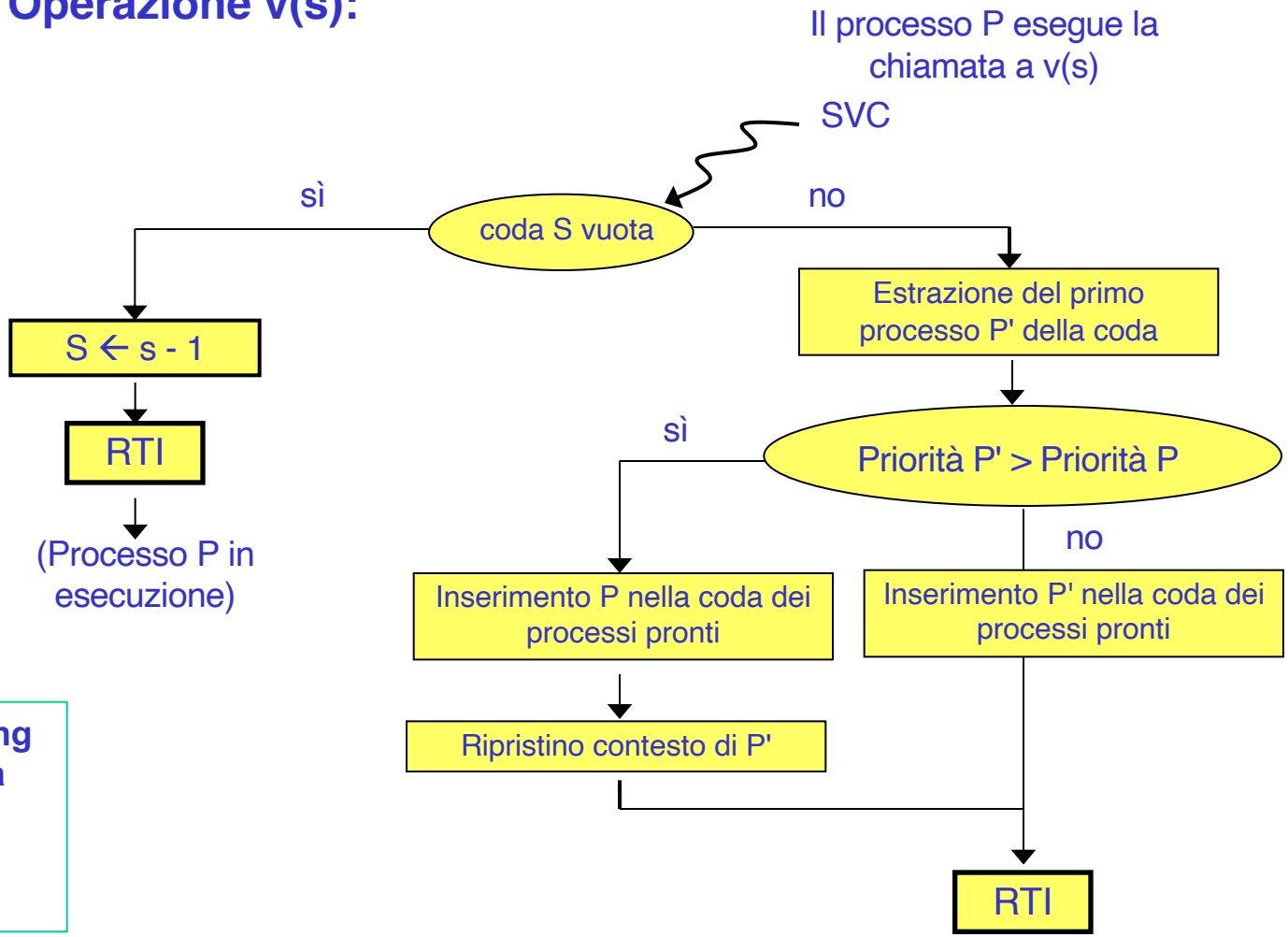
Operazione p(s):



Realizzazione di P

```
/* insieme di tutti i semafori: */
descr_semaforo semafori[num_max_sem];

/*ogni semaforo e` rappresentato dall'indice che lo individua nel
  vettore semafori: */

typedef int semaforo;

void P (semaforo s)
{       int j;int k;
        if (semafori[s].contatore ==0)
        {       Salvataggio_stato();
                j=processo_in_esecuzione;
                Inserimento(j,semafori[s].coda);
                Assegnazione CPU();// scheduling
                Ripristino_stato();
        }
        else contatore--;
}
```

Operazione v(s):



HP: scheduling pre-emptive a divisione di tempo con priorità

Realizzazione V

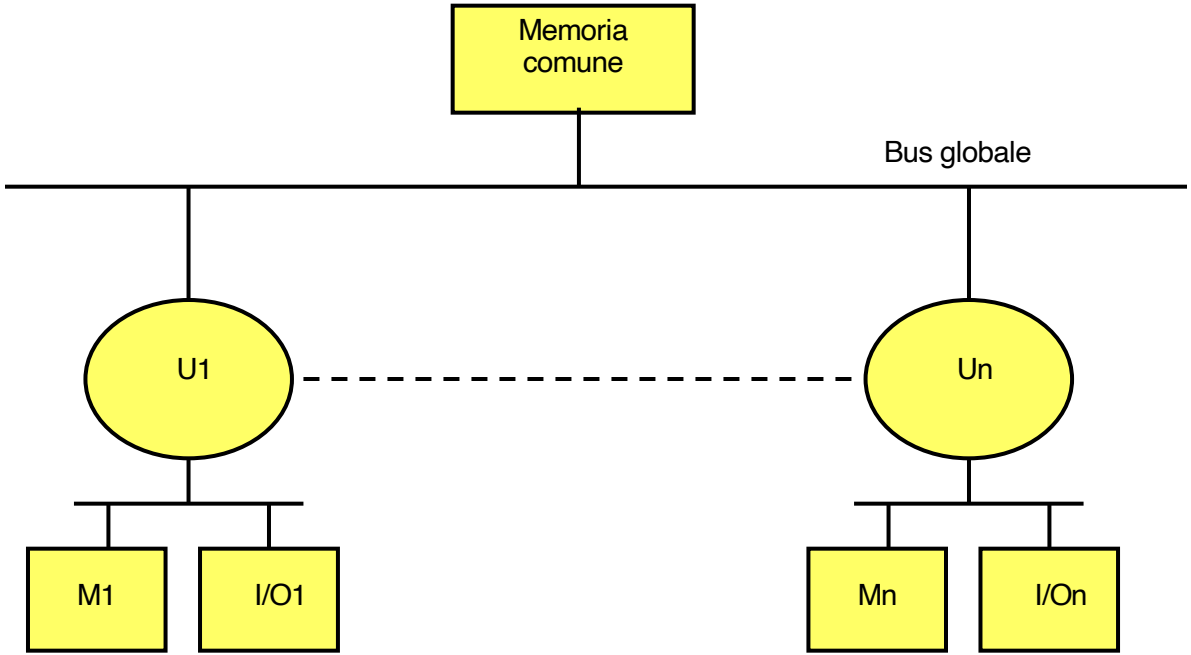
```
void V (semaforo s)
{ int j,k,p,q; /*j,k: processi; p,q indici priorità*/

  if(semafori[s].coda.primo!=-1) // c'è qualcuno in coda
  {     k=Prelievo (semafori[s].coda);
        j= processo_in_ esecuzione;
        p= descrittori[j].servizio.priorità;//priorità del proc. running
        q= descrittori[k].servizio.priorità; //priorità proc. risvegl.
        if( p > q) //il processo in esecuzione è prioritario
                Inserimento (k,coda_processi_pronti[q]);
        else { //preemption: cambio di contesto
                Salvataggio_stato();
                Inserimento(j,coda_processi_pronti[p]);
                processo_in_esecuzione=k;
                Ripristino_stato();
        }
  }
  else semafori[s].contatore ++; //se non ci sono processi in coda
}
```

Realizzazione del Nucleo

Architettura Multiprocessore

Sistemi operativi multiprocessore

Architettura:



Sistemi operativi multiprocessore: organizzazione interna

Il sistema operativo che esegue su un'architettura multiprocessore deve gestire una **molteplicità di CPU**, ognuna delle quali può accedere alla stessa **memoria condivisa**.

Due modelli:

1. **Modello SMP**: unica copia del nucleo condivisa tra tutte le CPU

2. **Modello a nuclei distinti**: più istanze di nucleo concorrenti

Nucleo in sistemi multi-processore

Modello SMP: Simmetric Multi Processing

Nel modello SMP, c'è un'unica copia del nucleo del sistema operativo allocata nella **memoria comune** che si occupa della gestione di tutte le risorse disponibili, comprese le CPU.

Caratteristiche:

- Ogni processo può essere **allocato su una qualunque CPU**.

- E' possibile che processi che eseguono su CPU diverse richiedano contemporaneamente funzioni del nucleo (es. System Call): poichè, in generale, ogni funzione comporta un accesso alle strutture dati interne al nucleo, occorre fare in modo che gli accessi al nucleo avvengano in modo **sincronizzato**:

  competizione tra CPU nell'esecuzione del nucleo → **necessità di sincronizzazione**

SMP: sincronizzazione nell'accesso al nucleo

Soluzione ad un solo lock:

Viene associato al nucleo **un lock L,** per garantire la mutua esclusione nell'esecuzione di funzioni del nucleo da parte di processi diversi: l'accesso esclusivo alle sue strutture dati può essere ottenuto delimitando il corpo di ogni richiesta per il nucleo (es. system call) con le primitive **lock** e **unlock** applicate all'**unico lock L**.

Problema fondamentale: limitazione del grado di parallelismo, escludendo a priori ogni possibilità di esecuzione contemporanea di funzioni del nucleo che operano su strutture dati distinte (ad esempio: due P su due semafori diversi non possono essere eseguite in parallelo).

SMP: sincronizzazione nell'accesso al nucleo

Soluzione a più lock:

Un maggior grado di parallelismo può essere ottenuto individuando all'interno del nucleo diverse classi di sezioni critiche, ognuna associata a una struttura dati separata e sufficientemente indipendente dalle altre.
Ad ogni struttura dati viene associato **un lock distinto**.
Ad esempio:

- la coda dei processi pronti
- i singoli semafori
- ecc.

vengono protetti tramite lock distinti.
Ogni operazione del nucleo che richiederà l'accesso a una particolare struttura S_i protetta dal lock L_i, conterrà una sezione critica il cui prologo ed epilogo saranno rispettivamente lock(L_i) e unlock(L_i).

Modello SMP: scheduling dei processi

Il modello SMP consente la schedulazione di ogni processo su uno qualunque dei processori: possibilità di attuare politiche di distribuzione equa del carico sui processori (**load balancing**).

Tuttavia, in alcuni casi può risultare più conveniente **assegnare un processo ad un determinato processore.** Ad esempio:

- I processori possono accedere più rapidamente alla loro memoria privata piuttosto che a quella remota: potrebbe convenire schedulare il processo sul processore la cui memoria privata già contiene il suo codice.
- in sistemi NUMA l'accesso alla memoria «più vicina» è più rapido: conviene schedulare il processo sul processore più vicino alla memoria ove è allocato il suo spazio di indirizzamento.
- I processori hanno memorie cache (es. TLB memoria virtuale). Per ridurre l'overhead dovuto a TLB flush e cache miss conviene schedulare un processo nel processore sul quale era stato precedentemente eseguito.

La scelta della politica di scheduling ha un impatto sul numero di code da gestire:

1 ready queue vs. **1 ready queue/nodo**

Nucleo in sistemi multi-processore

Modello a Nuclei distinti

In questo modello la struttura interna del sistema operativo è articolata su più nuclei, ognuno dedicato alla gestione di una diversa CPU.

L'assunzione di base è che l'insieme dei processi che eseguiranno nel sistema sia partizionabile in tanti sottoinsiemi (**nodi virtuali**) lascamente connessi, cioè con un ridotto numero di interazioni reciproche:

- Ciascun nodo virtuale è assegnato ad un **nodo fisico**: esso è gestito da un nucleo distinto e pertanto tutte le strutture dati del nucleo relative al nodo virtuale (es: descrittori dei processi, semafori locali, code dei processi pronti, ecc.) vengono allocate sulla **memoria privata** del nodo fisico.

- In questo modo tutte le interazioni locali al nodo virtuale possono avvenire **indipendentemente** e **concorrentemente** a quelle di altri nodi virtuali, facendo riferimento al nucleo del nodo.

Solo le interazioni tra processi appartenenti a nodi virtuali diversi utilizzano la memoria comune.

Nucleo in sistemi multi-processore

Organizzazione SMP vs. nuclei distinti:

- **Grado di parallelismo tra CPU:**
  - il modello a nuclei distinti è più vantaggioso, in quanto il grado di accoppiamento tra CPU è più basso. Maggiore scalabilità.

- **Gestione ottimale delle risorse computazionali:**
  - iI modello SMP fornisce i presupposti per un migliore bilanciamento del carico tra le CPU (load balancing) perché lo scheduler può decidere di allocare ogni processo su qualunque CPU.
  - Il secondo modello, invece, vincola ogni processo ad essere schedulato sempre sullo stesso nodo.

Realizzazione dei semafori nel modello SMP

Tutte le CPU condividono lo stesso nucleo: per sincronizzare gli accessi al nucleo, le struttura dati del nucleo vengono protette tramite lock.

In particolare:

- i singoli **semafori**
- la **coda dei processi pronti**

vengono protetti tramite **lock distinti**.

In questo caso due operazioni P su semafori diversi

- possono operare in modo **contemporaneo** se non risultano sospensive.
- In caso contrario, vengono **sequenzializzati** solo gli accessi alla coda dei processi pronti.

Esempio: scheduling **pre-emptive basato su priorità** ->

L'esecuzione della V può portare al cambio di contesto tra il processo risvegliato (P_r) e uno tra i processi in esecuzione (P_i), se la priorità di P_r è più alta della priorità di P_i : in questo caso il nucleo deve provvedere a revocare l'unità di elaborazione al processo P_i con priorità più bassa ed assegnarla al processo P_r riattivato dalla V.

- Occorre che il nucleo mantenga l'informazione del processo in esecuzione a più bassa priorità e del nodo fisico su cui opera.

- Inoltre è necessario un meccanismo di **segnalazione** tra le unità di elaborazione.

SMP: segnalazione inter-processore

Ad esempio, siano:

sem un semaforo, sul quale assumiamo sia inizialmente sospeso un processo (P_j)

P_i il processo che esegue la V(sem) eseguendo sul nodo U_i

P_j il processo riattivato per effetto della V(sem)

P_k il processo a più bassa priorità che esegue su U_k

1. **P_i chiama V(sem):** Il nucleo, attualmente eseguito da U_i con la funzione V(sem), invia un segnale di interruzione a U_k.

2. **U_k gestisce l'interruzione** utilizzando le funzioni del nucleo: inserisce P_k nella coda dei processi pronti e mette in esecuzione il processo Pj.

Realizzazione dei semafori nel modello a nuclei distinti

Solo le interazioni tra processi appartenenti a nodi virtuali diversi utilizzano la memoria comune.

Distinzione tra:

- **Semafori privati** di un nodo U, cioè utilizzati da processi appartenenti al nodo virtuale U. Poichè hanno visibilità confinata al nodo U, vengono realizzati come nel caso monoprocessore.

- **Semafori condivisi** tra nodi, cioè utilizzati da processi appartenenti a nodi virtuali differenti.

La memoria comune dovrà contenere tutte le informazioni relative ai **semafori condivisi.**

Realizzazione dei semafori condivisi

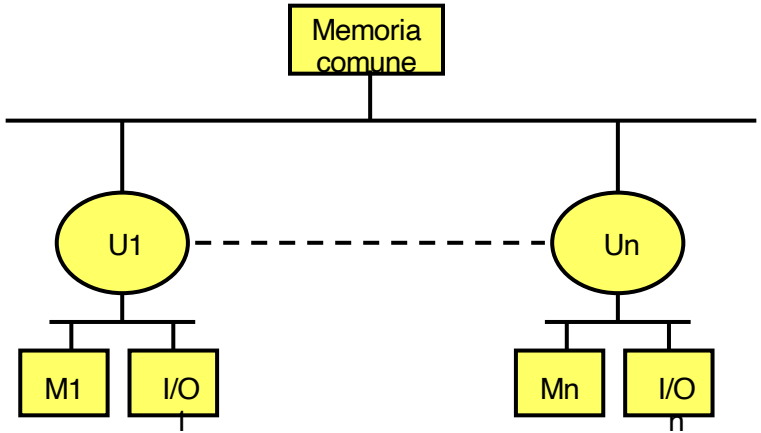
Ogni semaforo condiviso viene rappresentato nella memoria comune da **un intero non negativo**; l'accesso al semaforo sarà protetto da un lock x, anch'esso memorizzato in memoria comune.

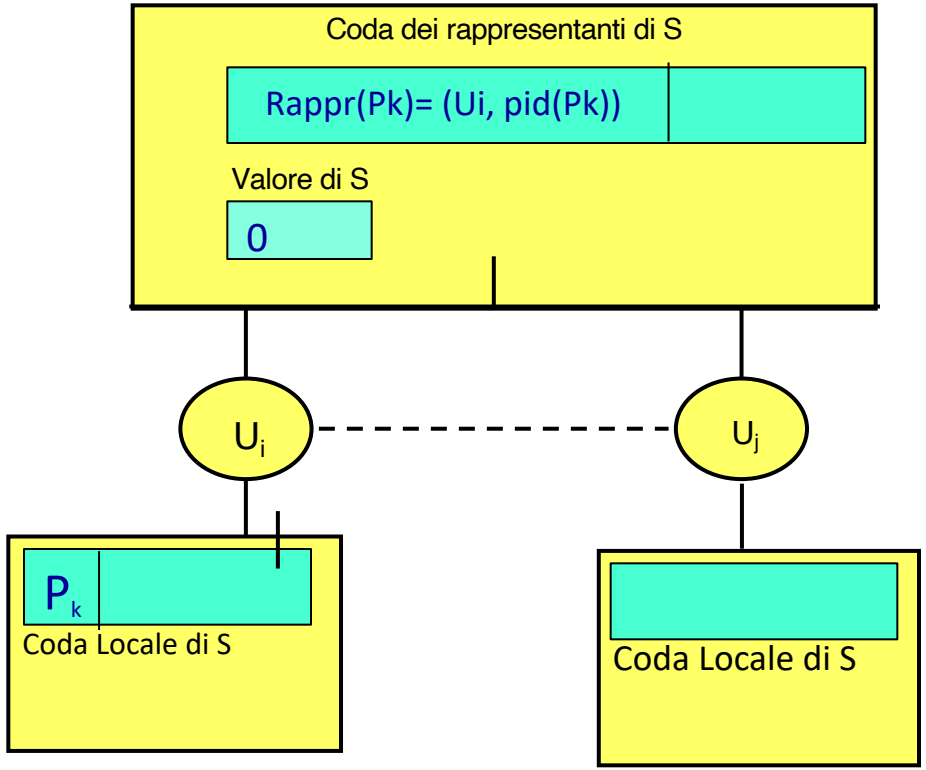
Rappresentante del processo: Insieme minimo di informazioni sufficienti per identificare sia il nodo fisico su cui il processo opera, sia il descrittore contenuto nella memoria privata del processore.

Realizzazione dei semafori condivisi

Per ogni semaforo condiviso S vengono mantenute varie code:

- **Su ogni nodo Ui:** una coda locale a Ui contenente i descrittori dei processi locali sospesi su S; la coda risiede nella memoria privata del nodo e viene gestita esclusivamente dal nucleo del nodo.
- **Una coda globale dei rappresentanti di tutti i processi sospesi su S**, accessibile da ogni nucleo.





Rappresentazione del semaforo condiviso S nel caso in cui vi sia un solo processo in attesa Pk, appartenente al nodo Ui.

Esecuzione di una P sospensiva su un semaforo S (Val_S=0) condiviso tra nodi virtuali:

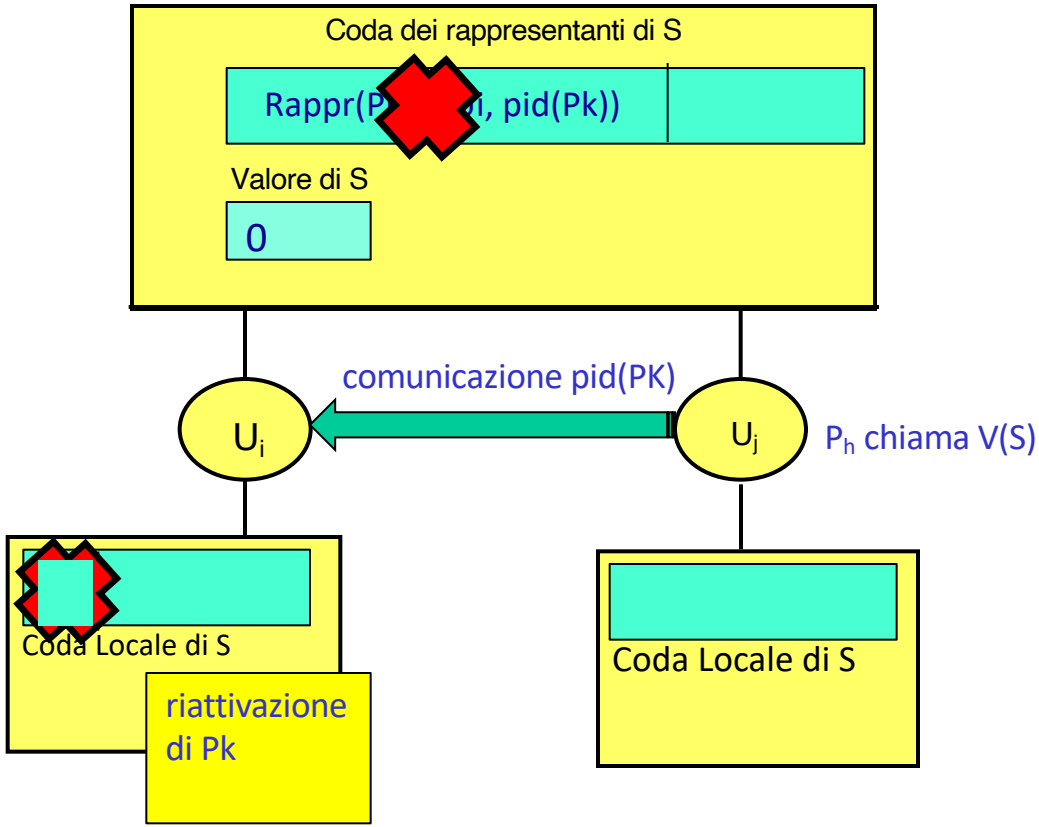
il nucleo del nodo Ui sul quale opera il processo Pk che ha chiamato la P(s) , provvede a:

- inserire il rappresentante di Pk nella coda globale dei rappresentanti associata a S.
- inserire il descrittore di Pi nella coda locale a Ui associata a S.

Esecuzione di una V su un semaforo S (con rappresentanti in coda):

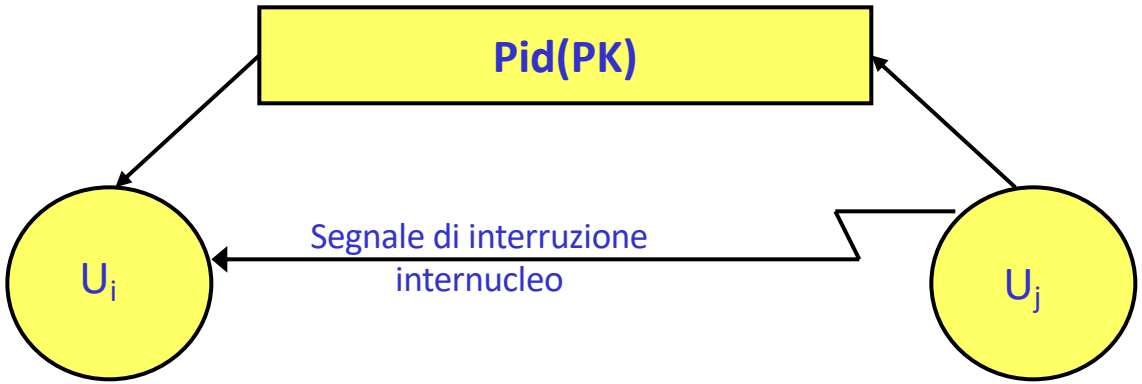
Assumendo che la V(S) sia chiamata dal processo P_h appartenente al nodo Uj:

1. Viene verificata le coda dei rappresentanti di S: il primo elemento è il rappresentante del processo $P_k$che appartiene al nodo U_i , che viene eliminato dalla coda.
2. il nucleo di Uj provvede a comunicare **tempestivamente** l'identità del processo Pk risvegliato al nucleo di Ui;
3. Il nucleo di Ui provvede a riattivare il processo Pk, estraendo il suo descrittore dalla coda locale.


Coda dei rappresentanti di S
Rappr(P
i, pid(Pk))
Valore di S
0
comunicazione pid(PK)
U_i
U_j
P_h chiama V(S)
Coda Locale di S
riattivazione
di Pk
Coda Locale di S

Comunicazione tra nuclei

La comunicazione tra Uj e Ui deve essere ***tempestiva***, cioè: qualunque attività si stia svolgendo in Ui, occorre interromperla per notificare al nucleo di Ui che c'è un processo (Pk) locale a Ui da riattivare => **segnale di interruzione** da Uj a Ui.



Il nucleo di Uj gestisce l'interruzione accedendo a un buffer in memoria comune dove Ui ha preventivamente scritto il pid del processo da risvegliare (Pk) e quindi estrae il descrittore di Pk dalla coda locale di S, portando il processo risvegliato nello stato ready.

Realizzazione P (modello a nuclei distinti)

```
void P(semaforo sem):
{       if (<sem appartiene alla memoria privata>)
                <come nel caso monoprocessore>
        else
        {       lock(x);
                < esecuzione della P con eventuale
                sospensione del rappresentante del
                processo  nella coda di sem>;
                unlock(x);
        }
}
```

Realizzazione V (modello a nuclei distinti)

```
void V (semaforo sem)
{ if (<sem appartiene alla memoria privata>)
          <caso monoprocessore>
  else {
          lock(x);
          if  (<la coda non è vuota>)
          if (<il processo appartiene al nodo del segnalante>)
                <caso monoprocessore>;
          else { //semaforo condiviso
                <estraz. processo dalla coda dei rappresentanti>;
            <si determina il buffer di comunicazione con il nodo
                cui il processo appartiene>;
              if (<area occupata>) <si aspetta>;
          else <si inserisce nel buffer l'identificatore del
                processo riattivato>;
                <si invia interrupt al nodo cui appartiene il
                processo>;}
          else <si incrementa il valore del semaforo>;
          unlock(x);
  }
}
```