Capitolo 5

Il Nucleo

Dopo aver definito i principali concetti su processi e risorse e posto i fondamenti della teoria, è possibile pensare di costruire una realizzazione concreta di un Sistema Operativo. Questa realizzazione, se pur semplificata per scopi didattici, trova una buona corrispondenza nei sistemi reali o comunque può essere presa a riferimento in gran parte delle realizzazioni esistenti.

Si ricorda la convenienza di costruire un Sistema Operativo a strati; nell’ambito di tale realizzazione ciascuno strato fornirà un insieme di funzioni dipendenti solo dagli strati ad esso più interni.

Al centro sono disponibili le istruzioni fornite dall'hardware dell'elaboratore. Gli strati successivi del Sistema possono essere considerati come la realizzazione di diverse macchine virtuali; da questo punto di vista il Sistema Operativo nel suo insieme realizza la macchina virtuale richiesta dall'utente.

La principale interfaccia tra l'hardware della macchina di base e il Sistema Operativo è fornita dal Nucleo che ne costituisce lo strato più interno.

5.1 Caratteristiche essenziali dell'elaboratore

Lo scopo del **Nucleo** è di fornire un ambiente nel quale possano esistere i Processi, ciò implica il trattamento delle interruzioni, la commutazione del/dei processori tra i Processi, e la realizzazione di meccanismi di comunicazione tra gli stessi. Prima di descrivere in dettaglio queste funzioni è conveniente definire i prerequisiti hardware essenziali per supportare un Sistema Operativo.

A. MECCANISMO DI INTERRUZIONE

Per poter sovrapporre nel tempo le attività di ingresso/uscita al funzionamento della unità centrale, deve essere possibile interrompere il Processo in esecuzione al termine di un trasferimento di dati da o verso una unità periferica.

E' quindi necessario che l'elaboratore fornisca un meccanismo di interruzione che consenta almeno il salvataggio del contatore di programma (PC) del Processo interrotto e trasferisca il controllo ad una locazione fissa in memoria.

Questa locazione sarà usata come inizio di un segmento di programma conosciuto come **Procedura di interruzione** (**Interrupt Routine** o **Interrupt Handler**) il cui scopo è quello di determinare la causa dell'interruzione e di intraprendere le azioni più appropriate. L'Interrupt Handler sarà descritto in seguito con riferimento ad una delle possibili realizzazioni del meccanismo di interruzione.

B. PROTEZIONE DELLA MEMORIA

Quando più Processi sono in esecuzione contemporaneamente è necessario proteggere l'area di memoria del Processo dall'accesso non autorizzato da parte di altri Processi. I meccanismi di protezione sono realizzati nei dispositivi hardware di indirizzamento di memoria e saranno descritti in dettaglio in seguito. Nella trattazione si suppone garantita tale funzione.

C. ISTRUZIONI PRIVILEGIATE

Per impedire l'interferenza reciproca tra Processi concorrenti, un sottoinsieme delle istruzioni dell'elaboratore deve essere riservato all'uso esclusivo del Sistema Operativo.

Queste istruzioni privilegiate eseguono funzioni di:

- abilitazione e disabilitazione di interruzioni
- smistamento del processore tra i Processi
- accesso ai registri di protezione della memoria
- operazioni di ingresso e uscita
- arresto del processore centrale.

Per distinguere le fasi in cui sono o non sono lecite le istruzioni privilegiate, la maggior parte degli elaboratori può operare in due modi differenti, usualmente conosciuti come **modo Supervisore** e **modo Utente**.

Le istruzioni privilegiate possono essere utilizzate solo nel modo Supervisore. La commutazione da modo Utente a modo Supervisore avviene automaticamente in una delle seguenti circostanze:

- Si verifica una interruzione asincrona
- Un Processo Utente chiama il Sistema Operativo (interruzione sincrona) per eseguire funzioni richiedenti l'uso di una istruzione privilegiata. Una simile chiamata è definita ***Supervisor Call*** (SVC) o ***Extra Code***
- Si verifica una condizione di errore nel Processo Utente. La condizione può essere trattata come una interruzione interna, e servita in prima istanza da un procedura di interruzione
- Il Processo Utente cerca di eseguire una istruzione privilegiata. Il tentativo può essere considerato come un particolare tipo di errore e trattato come nel caso precedente.
- Il ritorno da modo Supervisore a modo Utente è effettuato tramite una istruzione che è di per se stessa privilegiata.

D. OROLOGIO IN TEMPO REALE

Un oscillatore, che effettua interruzioni a intervalli fissi di tempo, prende il nome di **orologio in tempo reale** e ciò perché il tempo è misurato in sincronismo col mondo esterno piuttosto che in base al tempo di esecuzione di un Processo. L'esistenza di un orologio in tempo reale è essenziale per la realizzazione di politiche di gestione di un Sistema Operativo, per la stima dell'uso di risorse da parte dei diversi Processi ed inoltre per il coordinamento con attività a tempo.

Nel seguito, per la realizzazione del Sistema Operativo, si supporranno disponibili le caratteristiche hardware appena descritte. Più precisamente, volendo considerare la possibilità di includere diversi processori centrali in una singola configurazione, si assumerà che ogni processore abbia le prime tre caratteristiche e che ci sia un singolo orologio in tempo reale che possa interrompere ogni processore.

Si esaminerà solo la configurazione nella quale i processori siano identici e condividano una memoria comune.

Questa ipotesi esclude l'esame delle Reti di elaboratori in cui i processori hanno memorie separate e comunicano con particolari strutture di trasmissione dati ed i Sistemi di elaborazione *load sharing* nei quali processori differenti condividono un carico di lavoro in modo da ottimizzare le prestazioni globali.

5.2 Principali caratteristiche del Nucleo

La relazione tra il Nucleo e la parte rimanente del Sistema Operativo è illustrata in fig. 5.1. La linea orizzontale alla base rappresenta l'hardware dell'elaboratore; la parte in grassetto di questa linea rappresenta l'insieme di istruzioni privilegiate, l'uso delle quali è stato limitato al Nucleo tranne restrizioni riguardanti la protezione della memoria e le operazioni di ingresso/uscita descritte nei capitoli successivi.

Fig. 5.1. Realizzazione a strati di una macchina virtuale

Il nucleo consiste di tre segmenti di programma:

- la procedura di primo intervento a fronte di interruzioni (**FLIH first-level interrupt handler**) che esegue la gestione iniziale di tutte le interruzioni (int)
- il **Dispatcher**, che smista i Processori Centrali tra i Processi (dis)
- le procedure **wait** e **signal** che realizzano le primitive di comunicazione tra Processi. Queste procedure sono attivate tramite extracodici (SVC) dai Processi interessati (w&s).

Fig. 5.2. Le Componenti del nucleo

Il Nucleo costruito direttamente sull'hardware dell'elaboratore risulta essere la parte del Sistema Operativo più profondamente dipendente dal tipo di elaboratore. Infatti esso è probabilmente la sola parte del Sistema Operativo che è necessario scrivere interamente in linguaggio Assemblatore.

Con poche eccezioni, limitate principalmente alla gestione dell'ingresso/uscita (***I/O drivers***), gli altri strati possono essere codificati assai più convenientemente in linguaggi ad alto livello. La limitazione dell'uso di un linguaggio assemblatore ad una piccola parte del Sistema Operativo (probabilmente non più di 400 istruzioni), resa possibile dalla struttura ipotizzata gerarchica, può contribuire ad una realizzazione priva di errori, comprensibile e di facile gestione e ad una maggior portabilità del sistema.

5.3 Rappresentazione dei Processi

Le procedure del Nucleo sono responsabili del mantenimento dell'ambiente in cui esistono i Processi ed operano su alcune strutture di dati che sono la rappresentazione fisica dei Processi nel Sistema Operativo. La natura di questa struttura di

dati è l'argomento di questo paragrafo.

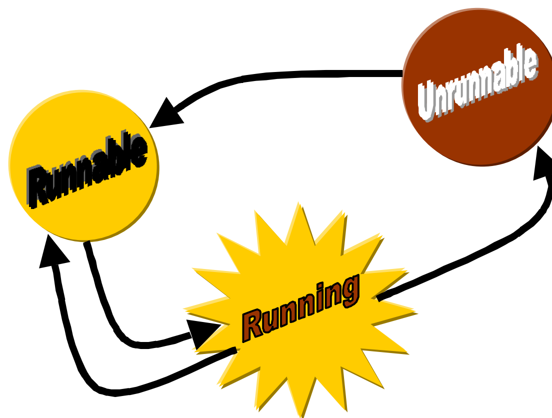
Ogni Processo può essere rappresentato da un **Descrittore di Processo** (**PD**), indicato anche come **Struttura di Controllo** o **Vettore di Stato** (**Control Block**, **State Vector**) che è l'area di memoria contenente quasi tutte le informazioni relative al processo. Il contenuto del Descrittore di Processo verrà chiarito in seguito.

Nel descrittore sono presenti l'identificatore (o nome) del Processo ed il relativo indicatore di stato.

Un Processo può trovarsi in uno dei tre stati principali: può essere **Running**, cioè essere in esecuzione su un processore, **Runnable**, cioè in grado di procedere non appena gli viene assegnato un processore, o **Unrunnable**, non è in grado di utilizzare un processore anche se gli fosse stato assegnato.

Il motivo più comune per cui un Processo è Unrunnable è quello di essere in attesa del completamento di un trasferimento di informazioni ad una unità periferica.

Fig. 5.3. Lo stato di un processo



Lo stato di un Processo è una informazione essenziale per la politica di assegnazione del processore realizzata dal Dispatcher. Il Processo che è Running su un processore centrale sarà indicato come quello corrente per quel processore; in qualunque momento il numero dei processi correnti nel Sistema Operativo è naturalmente minore o uguale al numero di processori disponibili.

Una parte del Descrittore di Processo è usata per contenere le informazioni che occorre salvare quando il Processo perde il controllo del processore. Queste informazioni, necessarie per la successiva ripresa del Processo, comprendono i valori di tutti i registri della macchina come: il contatore di programma, gli accumulatori, i registri indice che potrebbero essere modificati da un altro Processo. Tale parte del Descrittore comprende anche i valori di qualunque registro proprio del Processo e usato nell'indirizzare o proteggere la memoria.

Il complesso di queste informazioni è indicato come lo **Immagine del Processo** (altri nomi nella letteratura sono **context block** e **volatile environment**).

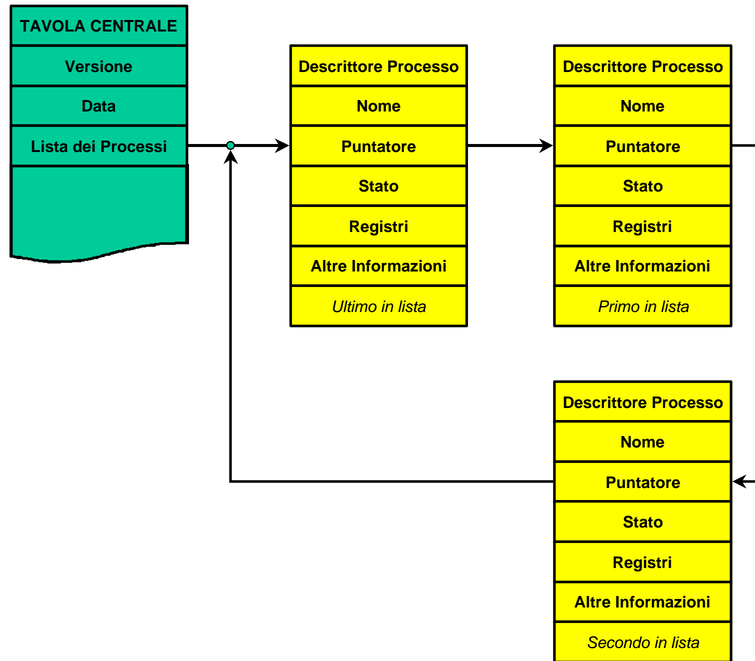
Più formalmente l'Immagine del Processo può essere definita come quel sottoinsie-

me di informazioni modificabili e condivise nel sistema che sono accessibili dal Processo.

Ogni Descrittore di Processo è collegato ad una **Struttura di Processi** che li lega insieme nel Sistema.

Si ritiene conveniente pensare tale Struttura di Processi come una lista nella quale sono collegati i Descrittori di Processo. Essa è la prima struttura di dati del Sistema Operativo; poiché essa non sarà affatto l'ultima, è conveniente introdurre una **Tabella Centrale**, con lo scopo di fornire un mezzo di accesso a tutte le strutture di informazione necessarie al corretto funzionamento del Sistema

Fig. 5.4. Tabella Centrale del Nucleo: link ai processi



La Tabella Centrale farà riferimento ad ogni struttura di dati tramite un puntatore, e può essere usata per contenere altre informazioni globali di supporto come data, ora e revisione del Sistema.

5.4 Procedura di interruzione di primo intervento

La **first-level interrupt handler** (**FLIH**) è la parte del sistema operativo interessata a rispondere ai segnali provenienti dal mondo esterno (interruzioni asincrone) ed a gestire gli eventi che si verificano all'interno del sistema di elaborazione (interruzioni sincrone, error trap, extracode, SVC).

Si farà riferimento ad entrambi i tipi di segnali con il termine di interruzioni e si useranno gli aggettivi esterno e interno per distinguerli tra loro dove sarà necessa-

rio.

La funzione della FLIH è doppia:

- determinare l'origine dell'interruzione
- gestire l'interruzione.

Nel paragrafo 5.1 si è notato che il meccanismo di interruzione proprio dell'elaboratore è responsabile del salvataggio almeno del contatore di programma del Processo interrotto. A fronte di una interruzione si deve comunque garantire che gli altri registri, richiesti dalla FLIH e che possono essere riusati dal processo interrotto, siano salvati per essere ripristinati successivamente. Se questa prestazione non è fornita dal meccanismo di interruzione hardware essa deve essere realizzata come prima operazione della stessa FLIH.

Fig. 5.5. FLIH - il primo componente del Nucleo

Poiché la procedura FLIH è relativamente semplice, i registri da essa utilizzati non saranno molti e sicuramente in numero più limitato di quelli che descrivono l'immagine del processo interrotto che perciò non occorre salvare nella sua totalità. Nella maggior parte dei casi il Processo può essere ripreso dopo la gestione dell'interruzione.

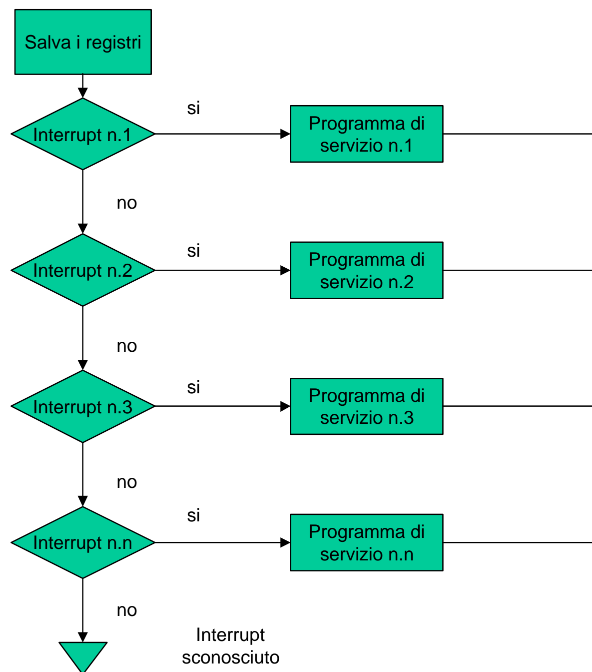
Una strategia alternativa per salvare i valori dei registri, adottato in alcuni microprocessori, è di fornire un set extra di registri per l'uso in modo Supervisore. La FLIH può usare questi registri e lasciare intatti quelli del Processo interrotto. La determinazione dell'origine dell'interruzione può essere realizzata più o meno facilmente facendo affidamento sull'hardware disponibile.

Nel caso di un hardware elementare, nel quale tutte le interruzioni trasferiscono il controllo alla stessa locazione, l'identificazione deve essere ottenuta tramite una sequenza di test sullo stato dei flag di tutte le possibili sorgenti di interruzioni. Questa sequenza di test, spesso chiamata **Skip Chain**, è definita in fig. 5.5 in modo che le sorgenti di interruzione più frequenti siano esaminate per prime.

Su alcuni elaboratori la Skip Chain è non necessaria per la presenza di un dispositivo hardware che distingue tra le possibili cause di interruzione trasferendo il controllo ad una locazione differente per ogni interruzione: si parla in tal caso di **gestione vettorizzata delle interruzioni**.

Ciò riduce il tempo necessario per identificare una interruzione.

Fig. 5.6. Skip Chain per identificare l'interruzione



Una soluzione di compromesso impiegata su molti elaboratori, è quella di fornire un numero limitato di posizioni di salto ognuno dei quali è condiviso da un gruppo di dispositivi. La prima fase di identificazione della interruzione è realizzata dall'-hardware ed è sufficiente completarla con una Skip Chain più breve, diversa per ogni posizione di interruzione. La distinzione tra interruzioni esterne, error traps e extracodes (SVC) è spesso fatta in questo modo. Il meccanismo di interruzione può dare un ulteriore aiuto memorizzando in locazioni fisse informazioni aggiuntive sulla causa di interruzione.

Le interruzioni al processore centrale sono normalmente inibite dopo una commutazione da modo utente a supervisore. Ciò assicura che i valori dei registri, salvati all'inizio dalla FLIH, non possano essere soprascritti da una successiva interruzione che avvenga prima che la FLIH sia terminata. Una interruzione che avviene mentre è disabilitato il meccanismo di interruzione viene tenuta in sospeso sino alla successiva riabilitazione che avviene alla ripresa del modo utente.

Questa soluzione diventa inaccettabile quando qualche periferica richiede risposte molto rapide ed è importante non perdere dati (tempo di risposta limitato). In questi casi è conveniente introdurre il concetto di priorità fra le varie cause di interruzioni, e realizzare procedure di interruzione che siano a loro volta interrompibili da una richiesta di servizio da parte di un dispositivo a più alta priorità.

Gli attuali microprocessori consentono di realizzare una disabilitazione selettiva delle interruzioni. Quando la FLIH sta servendo un'interruzione essa disabilita tutte le altre di priorità uguale o inferiore (fig. 5.7).

In questi casi deve essere prestata particolare attenzione nel salvataggio dei registri, utilizzando differenti locazioni in accordo al livello di priorità dell'interruzione.

I sistemi hardware più sofisticati di gestione delle interruzioni possono distinguere automaticamente i vari livelli di priorità, trasferendo il controllo e salvando i registri in locazioni differenti per ciascun livello ed inibendo automaticamente le interruzioni allo stesso od a minor livello.

Fig. 5.7. Livelli di interruzione

Modo utente	**Processi**
Livello 0	**Priorità minima**
Livello 1	codice interruzione
Livello 2	
Livello n	**Priorità massima**
Modo supervisore	**Codice ininterrompibile**

La seconda funzione della FLIH, e precisamente quella di servizio di un'interruzione è chiaramente dipendente dal tipo di interruzione interessata. I dettagli delle routine di servizio per i dispositivi di ingresso/uscita e per la gestione degli errori saranno descritti nei capitoli successivi.

Per quanto riguarda il Nucleo è sufficiente notare che la procedura di interruzione è eseguita in modo Supervisore, con le interruzioni totalmente o parzialmente disabilitate, ed è quindi desiderabile ridurre il più possibile il tempo di esecuzione in tali condizioni.

In generale ogni routine realizzerà azioni minime, per esempio il trasferimento di un carattere da un dispositivo di ingresso ad un buffer; la responsabilità per la azione successiva, in risposta al carattere ricevuto, sarà riservata al processo che avanza normalmente nel modo utente.

E' importante notare che una interruzione, sia interna che esterna, è da considerarsi un evento in grado di modificare, nella gran parte dei casi, lo stato dei processi.

Per esempio un Processo Unrunnable, avendo richiesto un trasferimento di dati da/verso unità periferica, sarà reso Runnable alla interruzione dovuta al completamento dell'operazione di trasferimento.

Risulta altrettanto ovvio che il Processo corrente sarà incapace di proseguire all'atto della esecuzione della maggior parte degli extracodici (SVC), come ad esempio nel caso di una operazione di wait su un semaforo bloccato o alla richiesta di una operazione di ingresso/uscita.

In ogni caso il cambio di stato del Processo interessato è effettuato da un cambia-

mento dello stato del suo Descrittore di Processo effettuato dalla procedura di gestione delle interruzioni.

Una conseguenza del cambiamento di stato di uno dei processi del sistema è che il processo che era Running sul processore interessato prima dell'interruzione può non essere il più adatto ad andare in esecuzione immediatamente dopo.

Può avvenire, per esempio, che l'interruzione renda Runnable un processo che ha una più alta priorità di quello che è nello stato di Running.

Il problema di quando smistare un processore centrale tra i Processi, e di come assegnarlo è esaminato nel paragrafo successivo.

5.5 Dispatcher

Il ***Dispatcher*** (a volte indicato come ***scheduler* a basso livello**) ha il compito di assegnare i processori centrali ad i vari Processi esistenti nel Sistema. Esso entra in funzione ogni qualvolta un Processo Running non può procedere nella sua esecuzione e ogniqualvolta c'è motivo di supporre che un processore possa essere utilizzato in modo più conveniente.

Le cause di attivazione del Dispatcher possono essere così riassunte:

- un'interruzione esterna che cambia lo stato di qualche Processo;
- un extracodice (SVC) nel Processo corrente per cui esso è temporaneamente incapace di procedere;
- un errore (trap) che causa la sospensione del Processo corrente;

queste eventualità sono tutte cause d'interruzione speciali poiché esse sono tutte interruzioni che modificano lo stato di alcuni Processi.

Per semplicità è preferibile non fare distinzione tra le interruzioni discriminando ad esempio quelle (come un'operazione di wait su un semaforo verde) che non hanno effetto sullo stato di alcun Processo. Si assumerà quindi che il Dispatcher venga attivato dopo ogni interruzione. Il maggior lavoro del Dispatcher, anche quando non viene modificato lo stato di alcun Processo, è più che compensato dai vantaggi che si ottengono da un trattamento uniforme delle interruzioni.

Le azioni del Dispatcher sono relativamente semplici da descrivere:

Fig. 5.8. Pseudocodice del Dispatcher

Dispatcher

```
Il processo corrente è ancora il più adatto ad avanzare?

Se è così:

Riprendi l'esecuzione del processo corrente restituendo il controllo
alla posizione indicata dal PC appena salvato dal dispositivo di
gestione delle interruzioni.

Altrimenti:

Salva l'Immagine del Processo corrente nel suo Descrittore;

Ripristina l'Immagine del Processo più adatto ad avanzare
prelevandolo dal relativo descrittore;

Riprendi l'esecuzione di questo Processo.
```

Per determinare il Processo più adatto ad avanzare è sufficiente ordinare i Processi Runnable per priorità. L'assegnazione della priorità ai Processi non è funzione del Dispatcher, ma dello scheduler ad alto livello.

Normalmente le priorità sono calcolate da quest'ultimo secondo parametri quali la quantità delle risorse richieste, il tempo trascorso dall'ultima volta che il Processo è andato in esecuzione e la sua importanza relativa.

Per quanto riguarda il Dispatcher è quindi ovvio che le priorità dei Processi sono state definite in precedenza al suo intervento.

Nel nostro Sistema Operativo, i Descrittori di Processo in stato Runnable saranno posti in una coda ordinata in priorità decrescente, cosicché il processo più adatto ad essere eseguito sarà in testa. Questa coda, illustrata in fig. 5.9, sarà indicata come **Coda dei Processi**.

Il ruolo del Dispatcher è di mandare in esecuzione il primo Processo nella Coda dei Processi che non sia già in esecuzione su qualche altro processore. Esso potrà o meno, essere lo stesso Processo che era in esecuzione prima dell'attivazione del Dispatcher.

E' necessario notare che l'introduzione della Coda dei Processi comporta una duplice azione da parte della procedura di servizio dell'interruzione ogni volta che un Processo è reso Runnable.

Per primo essa deve alterare lo stato nel Descrittore e per secondo essa deve collegare il Descrittore nella Coda dei processi alla posizione indicata dalla sua priorità.

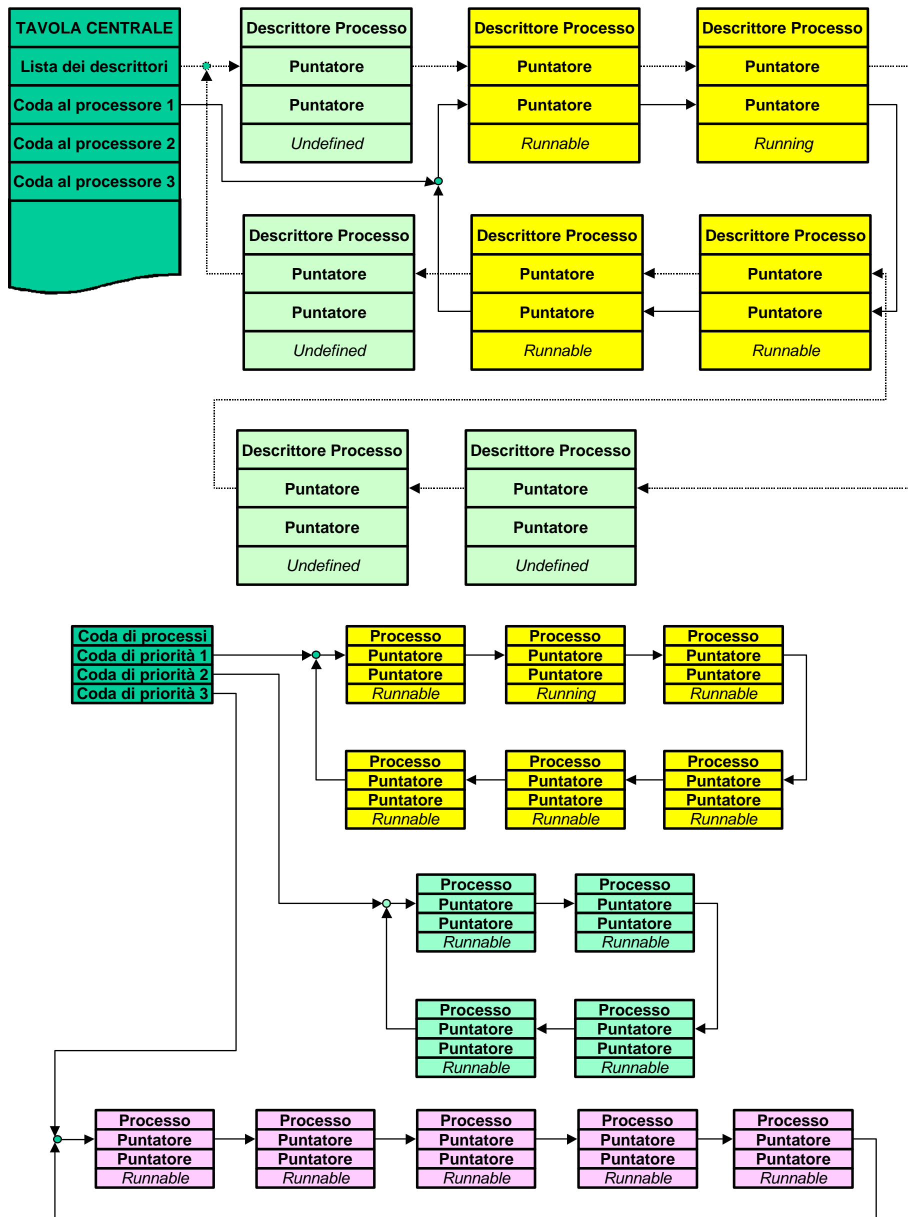
Nel prossimo paragrafo si mostrerà come questa operazione possa essere effettuata convenientemente eseguendo l'operazione di signal su un semaforo opportuno sul quale il Processo interessato ha eseguito una operazione di wait.

E' naturalmente possibile che, in un particolare momento, la Coda dei Processi contenga un numero di Processi minore dei processori disponibili, questo può verificarsi quando diversi Processi sono in attesa del completamento di operazioni di I/O. Questa situazione è probabilmente il risultato di una cattiva programmazione dello

Scheduler ad alto livello ed implica che non c'è lavoro per qualche processore.

Piuttosto che lasciare un processore inattivo, è conveniente introdurre un ulteriore Processo, chiamato **Processo Nullo** che ha la più bassa priorità ed è sempre Runnable. Il Processo Nullo invece di effettuare inutili cicli di attesa può compiere qualche utile funzione come ad esempio eseguire programmi di test del processore. La sua posizione alla fine della Coda dei Processi è mostrata nella figura seguente:

Fig. 5.9. Struttura dei Processi e Coda dei Processi



Il funzionamento del Dispatcher, con queste ulteriori precisazioni, può quindi essere riassunto nei seguenti punti:

Fig. 5.10. Pseudocodice del Dispatcher con coda di processi

Dispatcher

```
Il processo corrente è ancora il primo processo in coda di attesa?
Se è così:
Riprendi l'esecuzione.
Altrimenti:
Salva l'Immagine del Processo corrente nel suo Descrittore;
Preleva l'Immagine del primo Processo dalla Coda di attesa;
Riprendi l'esecuzione di questo Processo.
```

Nella impostazione del Nucleo si è scelta una struttura molto semplice per i Processi Runnable. Molti sistemi operativi dividono i processi Runnable in diverse classi, distinguendoli in base a criteri quali:

- risorse richieste,
- massimo tempo di attesa tollerato,
- massimo tempo di esecuzione assegnato, etc.

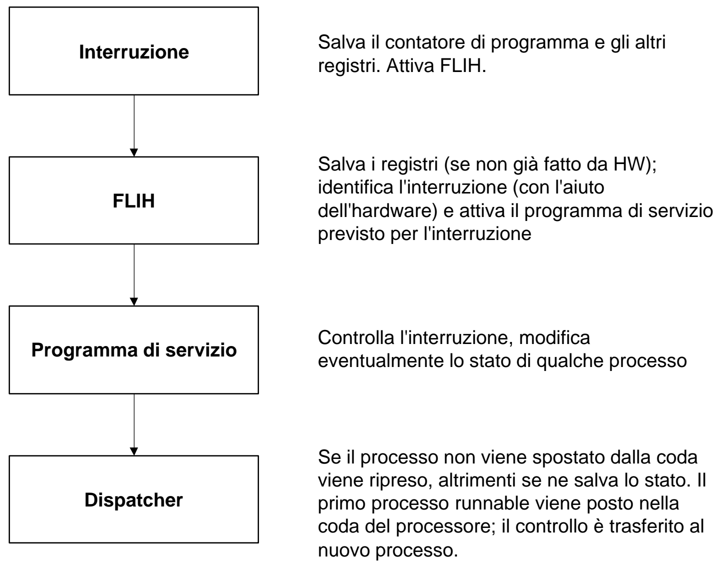
In alcuni Sistemi per esempio esistono tre code di attesa, caratterizzate rispettivamente da un tempo di esecuzione di 2, 0.2 e 0.02 secondi.

Ogni coda è servita su base FIFO, la coda di più alta priorità è quella con più breve tempo di esecuzione. Un Processo è inizialmente inserito nella coda di 0.02 sec; se esso avanza per tutto il tempo assegnato è relegato nella coda di 0.2 sec, e successivamente, se esso avanza per il tempo consentito è relegato nella coda di 2 sec. L'obiettivo è quello di assicurare che i Processi associati con terminali interattivi, che tipicamente richiedono poco impegno della unità centrale, ricevano un servizio rapido, mentre i Processi che richiedono un tempo elevato di unità centrale ricevano un servizio di maggior durata ma meno frequente.

L'esame approfondito di questi problemi è rimandato alla trattazione dello Scheduler ad alto livello.

In conclusione le relazioni intercorrenti tra il FLIH ed il Dispatcher possono essere riassunte nel diagramma seguente:

Fig. 5.11. Relazione tra FLIH e Dispatcher



5.6 Realizzazione delle primitive Wait e Signal

L'ultima parte del nucleo realizza i meccanismi elementari di comunicazione tra i Processi.

Fig. 5.12. Realizzazione delle primitive semaforiche

Nella realizzazione proposta si utilizzeranno le operazioni primitive di ***wait*** e ***signal*** che operano su semafori. Oltre a tale scelta, generalmente accettata, esistono altri meccanismi di comunicazione, forse meno facili da implementare, ma meno inclini ad essere mal adoperati.

Le primitive wait e signal sono incluse nel Nucleo poiché:

- devono essere utilizzabili da tutti i processi e perciò devono essere realizzate a basso livello;
- l'operazione di wait, eseguita da un Processo può provocare il suo blocco e quindi causare l'attivazione del Dispatcher per la riallocazione del processore. L'operazione di wait deve quindi avere accesso al Dispatcher.

Un modo conveniente per una procedura di interruzione per risvegliare (rendere

Runnable) un Processo è di eseguire il signal sul semaforo sul quale il Processo ha eseguito wait. Perciò l'operazione di signal deve essere accessibile alle procedure di interruzione.

Le operazioni da realizzare, con riferimento al semaforo s sono:

Fig. 5.13. Wait e Signal

```
wait(s)
Se val(s>)0
Decrementa val(s)
signal(s)
Incrementa val(s)
```

La realizzazione delle primitive nel Nucleo avverrà concordemente alle seguenti direttive:

Fig. 5.14. Wait e Signal su semaforo con coda

```
wait(s):
Se val(s)>0
Decrementa val(s).
Altrimenti
Aggiungi il processo alla coda del semaforo e rendilo unrunnable.
```

```
signal(s)
Se la coda è vuota
Incrementa val(s).
Altrimenti:
Togli un processo dalla coda del semaforo e rendilo runnable.
```

L'operazione di wait implica che i processi siano bloccati quando un semaforo ha valore zero e liberati in seguito di una operazione signal. Un modo naturale di realizzare ciò è di associare a ogni semaforo una coda (coda del semaforo). Quando un processo compie un'operazione di wait su un semaforo a valore nullo esso è inserito nella coda del semaforo e reso Unrunnable.

A. ATTESA

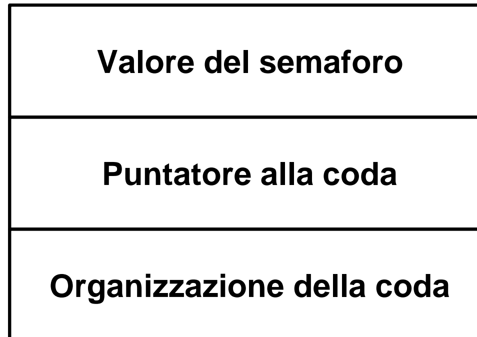
L'operazione di wait implica che i processi siano bloccati quando un semaforo ha valore zero e liberati in seguito di una operazione signal. Un modo naturale di realizzare ciò è di associare a ogni semaforo una coda (coda del semaforo). Quando un processo compie un'operazione di wait su un semaforo a valore nullo esso è inserito nella coda del semaforo e reso unrunnable.

Viceversa, quando è effettuata un'operazione di signal su un semaforo, un processo può essere tolto dalla coda del semaforo (a meno che essa non sia vuota) e reso nuo-

vamente Runnable.

Il semaforo deve perciò essere realizzato con una struttura a due componenti: un valore intero ed un puntatore (queue pointer) eventualmente nullo. A questo punto la implementazione può essere così descritta:

Fig. 5.15. Struttura di un semaforo



Si noti che, se un processo sta per essere rilasciato, non è necessario incrementare il semaforo dentro la primitiva signal, infatti il processo liberato dovrebbe immediatamente decrementare il semaforo nel completare la sua operazione di wait.

B. ACCODAMENTO

Non si è detto ancora quale processo debba essere rimosso dalla coda del semaforo dopo una operazione di signal, né si è stabilito se un processo aggiunto alla coda su una operazione di wait senza successo debba essere inserito in testa, alla fine o nel mezzo di essa. In altre parole non si è ancora specificato l'organizzazione della coda.

Per la maggior parte dei semafori una organizzazione adeguata è quella FIFO poiché essa assicura che tutti i processi bloccati saranno liberati. In alcuni casi può essere preferibile ordinare la coda con un altro criterio, probabilmente secondo la priorità usata nella coda di attesa del processore. Questa ultima organizzazione assicura che i processi ad alta priorità non attendano per lunghi periodi di tempo sulla coda di un semaforo.

Il punto importante da sottolineare è che semafori differenti possono richiedere differenti organizzazioni; nella realizzazione del semaforo si può includere una informazione aggiuntiva per indicare la organizzazione della coda. Questa informazione può essere ad esempio una descrizione codificata dell'organizzazione della coda o, in casi più complessi, può essere un puntatore al segmento di programma che realizza le operazioni di accodamento e disaccodamento. La struttura del semaforo nella realizzazione appena descritta è mostrata in fig. 5.15.

C. ALLOCAZIONE DEL PROCESSORE

Sia la wait che la signal possono alterare lo stato di un Processo, la prima rendendolo Unrunnable, la seconda facendo l'opposto.

Deve quindi essere prevista una chiamata al Dispatcher perché decida quale Processo debba successivamente essere mandato in esecuzione. Nei casi in cui non viene modificato lo stato del Processo (cioè una wait su un semaforo con valore positivo o una signal su un semaforo con coda vuota) il Dispatcher riattiverà il Processo corrente poiché esso sarà ancora il primo processo Runnable nella coda di attesa del processore.

D. INDIVISIBILITÀ

In accordo con la loro definizione, sia wait che signal devono essere operazioni indivisibili nel senso che solo un Processo alla volta può eseguirle, ne deriva che esse devono essere implementate come procedure che iniziano con un meccanismo di ***lock*** e terminano con una operazione di ***unlock***.

Su una configurazione monoprocessore la operazione di lock può essere implementata facilmente disabilitando le interruzioni. Ciò assicura che un Processo non può perdere il controllo del processore mentre sta eseguendo wait e signal: non c'è modo infatti di interromperlo. L'operazione di unlock è realizzata semplicemente riabilitando le interruzioni.

Su di un elaboratore con più CPU questa realizzazione è inadeguata poiché è possibile per due Processi in esecuzione su processori diversi eseguire simultaneamente wait e signal. In questo caso occorrono altri meccanismi per realizzare la lock e unlock come ad esempio l'impiego dell'istruzione di ***test and set***. Questa istruzione verifica e modifica il contenuto di una locazione di memoria in una sola operazione. Durante l'esecuzione dell'istruzione sono inibiti i tentativi di altri Processori di accedere alla stessa locazione.

L'idea di base è di usare una particolare locazione come indicatore di accesso alle procedure di wait e signal. Se l'indicatore è TRUE l'entrata è permessa altrimenti no. L'operazione di lock consiste nell'eseguire una istruzione di test and set sull'indicatore, con ciò se ne determina il valore e nello stesso tempo lo si pone FALSE.

Se il valore dell'indicatore è TRUE il Processo avanza, altrimenti esso cicla sull'istruzione test e set finché il Processo attivo su una delle procedure di wait e signal ripristinerà il valore dell'indicatore al valore TRUE.

Una alternativa al test e set, adottata per la prima volta sugli elaboratori Burroughs 6000, è una istruzione di scambio dei contenuti di due locazioni di memoria. L'operazione di lock parte scambiando i valori di un flag e di una locazione che è stata preventivamente posta a zero.

Il valore di questa seconda locazione è esaminato per vedere se è consentito l'accesso, mentre ogni altro Processo che tenti di guadagnare l'entrata troverà lo zero lasciato dallo scambio. Un Processo che, dopo lo scambio, trova il flag con il valore zero ripete semplicemente l'operazione di lock, finché il flag è reso diverso da zero dal Processo che esegue una operazione di unlock.

Si consideri ad esempio un insieme di più processi costituiti da un produttore e più consumatori con il seguente comportamento:

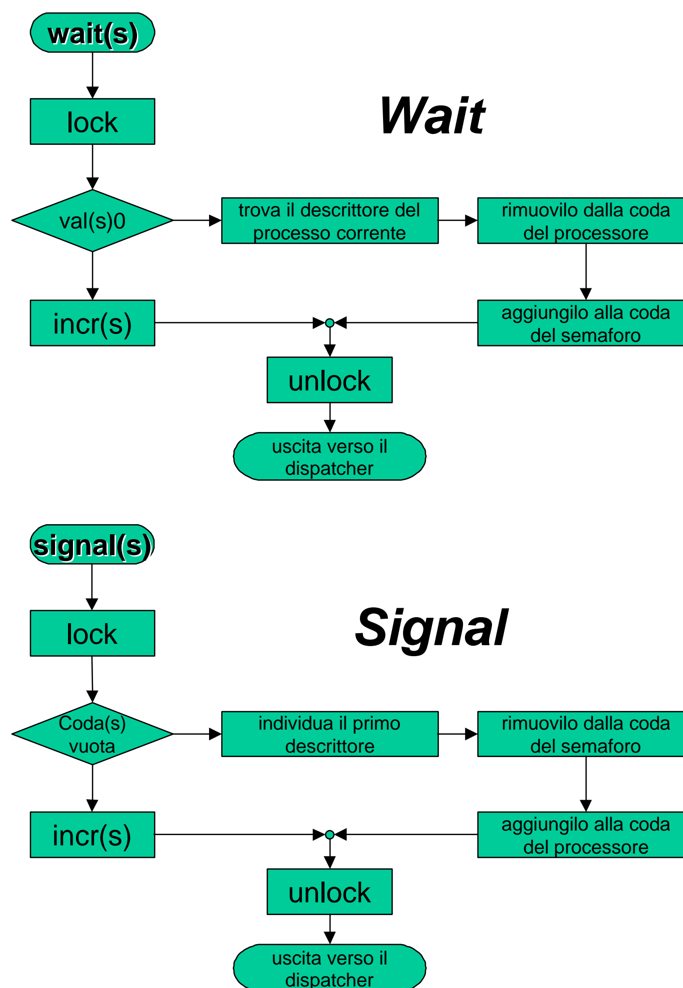
Fig. 5.16. Codice processi Produttore e Consumatore

Produttore	Consumatore
wait(s_1) produci il dato signal(s_2)	wait(s_2) consuma il dato signal(s_1)
Condizioni iniziali: val(s_1)=1, val(s_2)=0	

supponendo che i consumatori effettuino due tentativi di prelievo prima che il produttore inizi la sua attività, si avranno due richieste in coda al semaforo s2; il produttore con `signal(s2)` non rende `val(s2)=1`, bensì soddisfa una delle richieste dei processi in coda al semaforo s2, dopo di che il `signal(s1)` del consumatore incrementa il valore di s1 in modo che il produttore possa generare un altro dato.

Si conclude che wait e signal possono essere realizzate con il codice di figura:

Fig. 5.17. Realizzazione di wait e signal



Le due procedure saranno chiamate da extracodici (SVC) che entrano nel repertorio delle istruzioni dei processi.

Con ciò si può ritenere completato il Nucleo del Sistema Operativo: realizzato dalla FLIH, dal Dispatcher e dalle procedure wait e signal, tutte eseguite in modo Supervisore.

E' da notare che entrambi questi meccanismi impiegano una forma di **busy waiting**, cioè un processo che non può passare l'operazione di lock blocca il suo processore in un ciclo continuo tentando ripetutamente l'operazione finché il flag è soddisfatto.

Fig. 5.18. Paragone tra primitive hardware e software

	Wait & Signal	***Lock & Unlock***
Scopo	Sincronizzazione generale dei processi	Mutua esclusione dei processi dalle procedure *wait* e *signal*
Implementazione	Software	Hardware
Meccanismo di ritardo	Attesa su coda	Busy waiting e disabilitazione delle interruzioni
Tempo tipico di ritardo	Diversi millisecondi	Diversi microsecondi

Per quanto semplici le procedure di wait e signal il tempo impiegato nella busy waiting dovrebbe essere estremamente limitato.

Le operazioni di lock e unlock non possono essere usate come sostitutive di wait e signal. L'attesa in busy waiting o l'inibizione delle interruzioni, accettabili per l'operazione di lock, non sarebbero accettabili nella scala dei tempi richiesta da una operazione di wait. La tabella di fig. 5.18 mostra i due livelli concettualmente differenti cui afferiscono i due tipi di operazioni.

Pagina lasciata intenzionalmente vuota