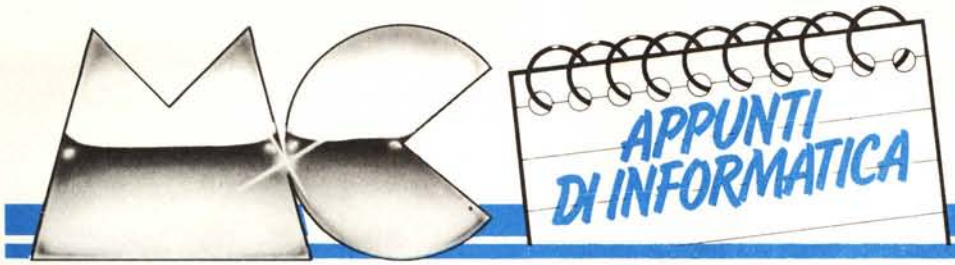

di Andrea de Prisco

Processi, Multiprogrammazione e Time-Sharing

Con la multiprogrammazione di un calcolatore è possibile mandare in esecuzione più programmi contemporaneamente, sia che questo abbia effettivamente a disposizione molti processori che lavorano in parallelo, sia che il processore disponibile sia uno solo e che effettivamente riesca «a farsi in quattro», o più all'occorrenza, per soddisfare simultaneamente più utenze.

Definiamo i processi

Un processo è un'entità di elaborazione attiva capace di provocare il verificarsi di eventi. Questa definizione potrebbe anche essere letta come: «Un processo è un programma in esecuzione» anche se, come vedremo, non sempre tale asserto è verificato (oltreché da usare con le dovute cautele). Vogliamo comunque soffermarci maggiormente sulla differenza abissale esistente tra un programma nudo e crudo e un processo. Il primo infatti è una cosa effettivamente toccabile con mano: una manciata di istruzioni (possibilmente appoggiate su di un pezzo di carta) per descrivere cosa il processore dovrà fare. Il processo, di fatto, è «la cosa» descritta dal programma.

Si usa dire che i programmi all'interno di un calcolatore «girano»...

Nulla di più falso: possiamo assicurarvi che in memoria centrale stanno ben saldi alle celle riservate loro. Quando sulla tastiera del nostro personal digitiamo RUN, si dice: «facciamo partire il programma»... per dove?

Il programma non parte: semmai è il processo da esso descritto che «parte»... ma sempre in senso lato. Allo stesso modo è inutile cercare di fermare un programma: abbiamo già detto che finché è in memoria non si sposta nemmeno di un millimetro. Tuttalpiù si muoverà qualcosa quando lo carichiamo da disco: effettivamente in questo caso il programma «parte» dal disco e si «ferma» in memoria.

Perché mi dice ciò?... Qualcuno obbietterà!

Innanzitutto per mettere un po' d'ordine (e ordinatezza) a tutto l'articolo. Nell'introduzione in testa si parla di programmi mandati in esecuzione: forse che qualcuno volesse fucilarli? No, è solo che iniziare di botto con la parola processo poteva sconvolgere qualcuno e nella filosofia di «Appunti» ciò non dovrebbe mai capitare... almeno si spera!

Visione convenzionale di calcolatore

Siamo abituati a vedere un calcolatore come un sistema composto da una memoria dove sono parcheggiati dati e programma, un processore capace di eseguire istruzioni e le dovute unità di ingresso/uscita atte all'interfacciamento col mondo esterno. Immesso «il programma» in memoria con un apposito tasto o comando siamo anche in grado di farlo eseguire ottenendone i risultati sul video... del televisore piccolo. Questo normalmente accade a chi ha in casa un qualsiasi personal/home computer piccolo o grande che sia. Magari i risultati della elaborazione sono una bella navicella che spara e si sposta sul video, pilotata da un ulteriore input-device chiamato volgarmente joystick.

Se però spostiamo la nostra attenzione sui grossi calcolatori, magari di qualche centro di calcolo, possiamo notare come l'andazzo della situazione sia notevolmente diverso. Infatti vedremo collegati ad un unico computer molti video e molte tastiere dalle quali altrettante persone dialogano con l'unità centrale come se questa fosse un oggetto privato di ognuno. Il tizio X potrebbe ad esempio scrivere programmi matematici per ottenere su terminale ad alta risoluzione grafici di funzioni; l'utente Y potrebbe gestire in Cobol archivi di dati e l'utente Z... giocare con la sua brava navicella, tanto per cambiare.

Prima di vedere il funzionamento di un sistema multiprogrammato occorre fare alcune considerazioni. Quando su di una macchina «girano» più processi contemporaneamente si dice che esiste del parallelismo, discendente dal fatto che più processi possono avanzare parallelamente: ciò dipende qualitativamente e quantitativamente dal numero di processori di cui si di-

spone. Possono essercene più d'uno in modo da avere un parallelismo già al livello dell'hardware della macchina in considerazione o un solo processore che grazie al sistema operativo riesce a simulare ugualmente un grado di parallelismo, ovviamente stavolta a livello software. Comunque anche per i calcolatori multiprocessor il parallelismo non è mai solo hardware in quanto normalmente i processi in esecuzione sono più del numero dei processori e ciò vuol dire che nell'ambito di ognuno di questi un ulteriore livello di parallelismo è implementato dal sistema operativo della macchina. Per fare un esempio se il computer che stiamo considerando dispone di 10 processori e noi riusciamo a far girare contemporaneamente 20 processi è auspicabile pensare che su ogni processor girano in parallelismo simulato due processi.

L'idea di base

Da questo momento in poi, concentreremo la nostra attenzione sul parallelismo, come abbiamo detto prima, simulato: cioè sul come sia possibile che un processore riesca a far avanzare contemporaneamente più processi. Tale particolarità, riservata solo a computer troppo «seri» per le tasche di un comune mortale, ha la sua ragion d'essere non tanto per accontentare più utenti di un centro di calcolo simultaneamente ma quanto per poter sfruttare meglio le potenzialità di calcolo dei computer. Infatti i calcolatori l'unica cosa che sanno fare è calcolare: se fanno questo nel minor tempo possibile è meglio per tutti, a partire da chi deve aspettare davanti al terminale i risultati di un'elaborazione, per finire alla bolletta ENEL che se lievita di meno sono contenti in molti, ecologisti compresi.

Detto questo, cerchiamo di scoprire quand'è che i computer perdono tempo. Mentre calcolano sicuramente no: sono al lavoro, nulla da ridire. È mentre usano le periferiche di ingresso/uscita che «rallentano». Infatti, per prelevare ad esempio dati da disco, occorre spostare testine, attendere rotazioni del disco, trasferire messaggi su linee: tutto ciò magari in un secondo. Il processore è costretto a fermarsi in attesa del dato richiesto, mentre potrebbe occuparsi d'altro. Pensate che in un secondo un processore può fare anche diversi milioni di operazioni... e invece sta fermo ad aspettare.

No, così proprio non va.

Meglio elaborare un altro programma, già presente in memoria, e se anche questo richiede accessi al disco niente paura, prendiamo un altro programma e lo elaboriamo, e così via. Posto che il primo processo ottiene il dato richiesto (in altre parole è intanto passato il secondo di cui sopra) quanto prima ripartirà in occasione della sospensione di qualche altro processo.

Stato di un processo

Formalizziamo meglio quanto appena detto indicando innanzitutto in quanti e in quali stati può trovarsi un processo. Un processo può essere in Esecuzione, in stato di Pronto o essere Sospeso. Dato che il processore può eseguire un solo processo per volta, dei processi che stanno all'interno di

Figura 1 - Descrittore di processo.

un calcolatore, solo uno può effettivamente essere in stato di Esecuzione: quello che, nell'istante che stiamo considerando, è processato (scusate il rigiro di parole) dal processore. Nel caso di architetture parallele se n sono i processori, saranno correntemente in Esecuzione n processi alla volta, come abbiamo già anticipato.

Un altro stato in cui un processo può trovarsi è detto di Pronto: riguarda tutti quei processi che appena il processore si libera, ossia sgancia il processo in Esecuzione, possono partire o ripartire immediatamente. Nel caso multiprocessor, appena si libera uno dei processori funzionanti.

Infine un processo può essere in stato di Sospeso se, come abbiamo detto prima, non può proseguire con l'elaborazione in attesa di un dato che deve ad esempio arrivare da disco.

Altre informazioni circa un processo non in Esecuzione riguardano la posizione di memoria dove il programma che lo descrive è parcheggiato, l'indirizzo della prima istruzione che dovrà essere eseguita «al risveglio», il contenuto dei registri adoperati del processo stesso durante l'elaborazione. Inoltre, per comodità, daremo anche un nome o un numero d'ordine ad ogni processo per poterlo identificare univocamente. Tutte queste informazioni sono contenute nei descrittori di processi, opportune strutture dati in memoria, e usate in scrittura quando un processo viene sospeso e in lettura quando si tratta di risvegliarlo per mandarlo in esecuzione.

Ambiente multiprogrammato

Vediamo ora cosa succede un po' più in dettaglio in un calcolatore multiprogrammato nel quale, per sfruttare meglio il processore (o i processori), si fanno avanzare sempre processi in modo da non lasciare mai inattiva la CPU.

Supponiamo di avere in memoria i vari programmi da elaborare: riserveremo per ognuno di essi uno spazio in memoria opportuno. Come abbiamo detto prima daremo un nome o un numero ad ognuno per poterlo identificare e setteremo per tutti i relativi descrittori di processo. Essendo all'inizio, tutti i processi sono in stato di Pronto e ovviamente per ognuno di questi la prima istruzione da eseguire, quando saranno attivati, sarà la prima istruzione dei corrispondenti programmi in memoria. Stiamo per dare il via, se volete pensate pure agli attimi prima della partenza di un Gran Premio.

Caso uniprocessor: si prende un descrittore di processo e prelevate da questo le informazioni che ci interessano (indirizzo in memoria del programma, istruzione prossima da eseguire, registri adoperati) si può iniziare ad elaborare il primo programma. Sempre per comodità immaginate di tenere tutti i descrittori di processo in stato di pronto in una ben precisa zo-

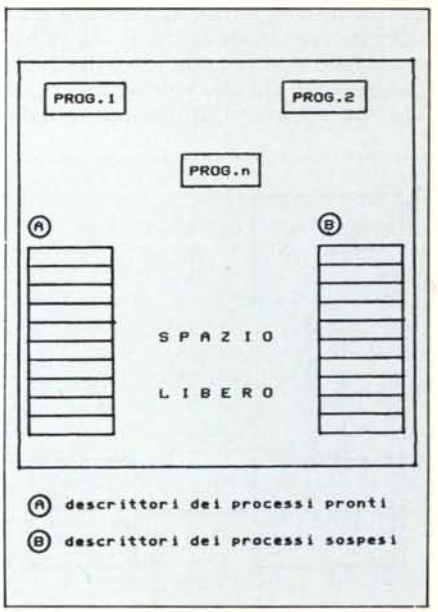

Figura 2 - Memoria di un calcolatore multiprogrammato.

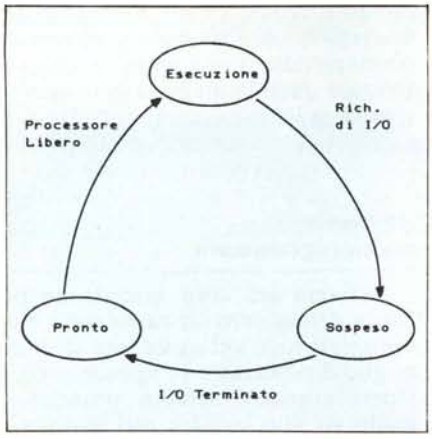

Figura 3 - Schema evolutivo di un processo in ambiente multiprogrammato.

na di memoria, riservata per tale scopo. Analogamente per i processi sospesi. Bene, a questo punto, durante l'elaborazione del primo programma, come è facile prevedere, supponiamo che il processo corrispondente richieda un accesso al disco: operazione, come ben sappiamo, lunghissima in confronto alla velocità della CPU. Sospendiamo il processo in corso, ponendolo in stato di Sospeso: questa operazione si effettua semplicemente inserendo nel descrittore di processo tutte le informazioni che ci serviranno al risveglio: registri e prossima istruzione da eseguire. Porremo il descrittore nella zona riservata ai processi sospesi. La CPU preleva un altro descrittore dai processi pronti e fa partire l'elaborazione di un altro programma.

Intanto l'unità a disco preleva il dato richiesto e lo manda all'unità centrale, accompagnato naturalmente da un vigoroso interrupt: il processore smette per un attimo di elaborare il programma in corso, aggiorna lo stato del processo sospeso che aveva richiesto il dato e lo pone in stato di pronto (ha ottenuto ciò che voleva, può ripartire non appena il processore lo risveglia) spostando nuovamente il corrispondente descrittore di processo nella zona «Processi Pronti». Continua così l'esecuzione del processo interrotto dall'interrupt.

Nel caso di calcolatori multiprocessor, occorre naturalmente apportare alcune banali variazioni all'algoritmo appena descritto. Ovviamente se sono i processori, al momento del via, i descrittori di processo saranno prelevati per mandare in esecuzione parallela i processi. Per quanto riguarda le sospensioni non vi sono differenze rispetto al caso uniprocessor, e quando si tratterrà di risvegliare un processo, questo sarà fatto dal processore che si è appena liberato.

Per finire, in figura 3 è mostrato lo schema evolutivo di un processo in un sistema multiprogrammato: i nidi di questo grafo rappresentano i tre stati di un processo e gli archi tra i nodi le transizioni di stato, etichettati con l'evento che l'ha provocato.

Time-sharing

Come abbiamo appena descritto, in un sistema multiprogrammato, si ha parallelismo simulato come conseguenza del fatto che vogliamo a tutti i costi non far restare mai inutilizzata la CPU di un calcolatore. Ovvero: quanto più i processi fanno uso dei lenti dispositivi di ingresso/uscita tanto più li vedremo avanzare parallelamente.

A questo punto è d'obbligo una domanda: se un processo deve solo compiere una grossa quantità di calcoli, senza fare operazioni di ingresso uscita fino al termine dell'elaborazione, possiamo ancora parlare di parallelismo?

Certamente no: infatti mentre il primo processo avanza (imperterrito) per ore e ore di elaborazione, tutti gli altri, sebbene in stato di pronto, non vedrebbero per ore un solo pezzo di CPU manco a pagarlo (si fa per dire) un occhio della testa. Questo fatto, anche se in termini di ottimizzazione dell'uso dei processori non è controproducente può in alcuni casi essere perlomeno indesiderabile. Spieghiamoci meglio: all'inizio dell'articolo abbiamo detto che finché un processore non sta con le mani in mano sono tutti più contenti. Quindi se calcola (dicevamo) nulla da ridire: stiamo sfruttando al massimo la CPU.

Il fatto è che sebbene i processi in attesa di esecuzione siano privi di animo e quindi intuibilmente non dovrebbero seccarsi della situazione angosciosa (pensate alla fila presso uno sportello), non possiamo dire altrettanto dei programmatori che hanno inserito i loro elaborati nel computer e sono costretti a aspettare a lungo prima di ottenerne i risultati.

Quindi, in generale, un sistema semplicemente multiprogrammato a cui attaccare una manciata di terminali per soddisfare più utenti certamente non basta. L'idea è quella di condividere l'uso della CPU da parte di ogni processo in base non solo al numero di accessi a dispositivi secondari, ma anche in base alla durata della singola elaborazione.

Ciò vuol dire che: «Caro processo, noi ti diamo la CPU per un certo periodo di tempo, se riesci a portare a termine l'esecuzione bene, altrimenti (da bravo) cedi il passo ai tuoi colleghi che hanno diritto quanto te a proseguire l'elaborazione... OK?».

Questo a parole. Nella pratica, implementare un tale meccanismo, detto appunto a divisione di tempo (time sharing) non è assolutamente difficile: basta solo aggiungere un dispositivo detto Timer che ad intervalli di tempo regolari provvede a mandare un interrupt al processore avvertendolo così che è scaduto un altro quanto di tempo.

In figura 5 è mostrato lo schema evolutivo di un processo in ambiente time-sharing: quanto detto per la memoria e i descrittori di processo resta invariato. Supponiamo di dover elaborare un certo numero di programmi. All'inizio saranno tutti in stato di pronto quindi, come prima, preleviamo un descrittore per far partire il primo processo: contemporaneamente regoliamo il timer al quanto di tempo prestabilito (generalmente dell'ordine di centesimi di secondo). Bene, il primo processo avanza. Se questo effettuerà una operazione di ingresso/uscita lo metteremo in stato di sospeso, se ciò non avviene si possono presentare due casi: o il processo termina, o scade il quanto di tempo e il timer manda l'interruzione al processore. In questo caso il processo è posto in stato

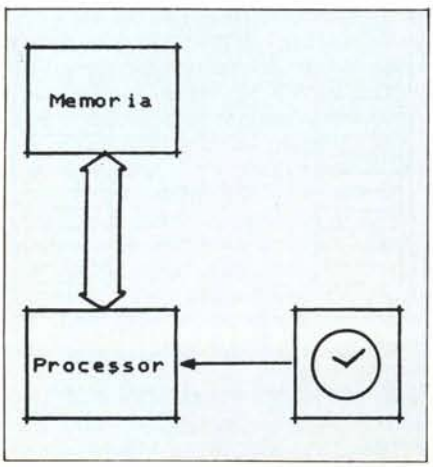

Figura 4 - In un sistema time-sharing è sempre presente un dispositivo «Timer».

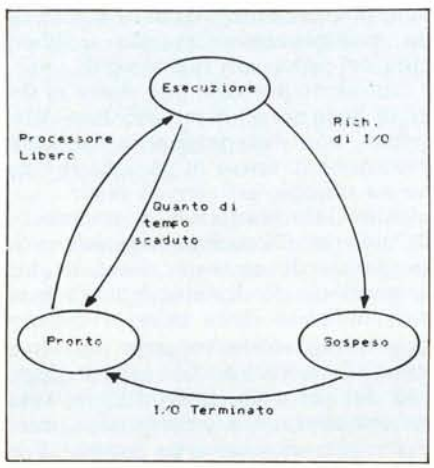

Figura 5 - Schema evolutivo di un processo in ambiente Time-Sharing.

di pronto e un nuovo processo potrà così disporre della CPU.

Per essere inoltre il più giusti possibile useremo il nostro spazio dei descrittori di processi pronti come una coda F-I-F-O (first-in-first-out, il primo ad entrare è anche il primo ad uscire) in modo che processi interrotti molto tempo fa andranno in esecuzione prima di quelli che «puzzano» ancora di CPU. Se invece preferiamo che alcuni processi avanzino più rapidamente di altri, possiamo dare ad ognuno un grado di priorità sfruttando tale informazione come segue.

Un primo modo consiste nell'usare la coda dei processi pronti non come una coda F-I-F-O ma come una coda a priorità: quando dovremo prelevare un nuovo descrittore, sceglieremo in base alla priorità dei processi corrispondenti: inseriremo cioè anche tale informazione nel descrittore. In tale modo processi meno importanti useranno meno di altri la CPU del sistema.

Il principale svantaggio di tale metodo è che il più delle volte processi meno importanti rischiano di non avanzare mai. Per questo si preferisce adottare altri metodi per gestire la priorità.

Un secondo metodo consiste nel variare il quanto di tempo disponibile a seconda della priorità del processo stesso. Si verificherà così che processi con priorità più alta disporranno del processore per intervalli più lunghi e di conseguenza verranno interrotti meno volte, nel corso dell'elaborazione, avanzando così più velocemente di altri. Ovviamente la coda dei processi pronti sarà nuovamente F-I-F-O in modo che processi con priorità più bassa non rischieranno l'attesa infinita: semplicemente evolveranno più lentamente dato che vedranno meno la CPU.

Concorrenza

L'ultimo problema circa i sistemi multiprogrammati (in genere, ovvero time-sharing e non) riguarda la gestione delle cosiddette risorse condivise: più processi evolventi in parallelo, concorrono per accedere a qualcosa strettamente sequenziale comune a più processi: un qualsiasi oggetto fisico o logico al quale può accedere solo un processo alla volta.

Ovvero: se il processo X sta usando un array condiviso con gli altri e il processo Y contemporaneamente cerca di fare lo stesso, sarà sospeso (allo stesso modo visto prima) in attesa di poter usare la risorsa. Quando X rilascerà la risorsa, provvederà a togliere Y dallo stato di sospeso per porlo in stato di pronto...

Questo solo come piccolo antipasto: analizzeremo per intero il problema sul prossimo numero. Arrivederci. MK

Architetture tipiche

Le diversità strutturali dell'architettura dei calcolatori riguardano essenzialmente come le varie componenti processore-memoria-periferiche sono fra loro collegate per formare il «sistema» di calcolo.

In figura A (P sta per processor, M per memoria, I/O per unità di ingresso/uscita) è mostrato un primo schema di collegamento delle varie unità tramite Bus singolo, sul quale in un verso o nell'altro viaggiano dati e segnali dovuti alle esecuzioni dei processi in corso. Essendo la più semplice è anche l'organizzazione con più basso livello di parallelismo hardware e per questo la più lenta. Infatti il bus di collegamento può essere usato per inviare dati solo da una unità alla volta, eventualmente costringendo le altre ad aspettare che la prima abbia finito e che quindi rilasci il bus. Ciò si paga parecchio quando ad attendere sono unità importanti come il processor che a causa di una operazione di scrittura in memoria da parte di un'unità di ingresso/uscita non può procedere nell'esecuzione dei processi. Per ovviare a questo inconveniente è sufficiente complicare un po' l'architettura, introducendo due bus distinti per il funzionamento delle unità ingresso/uscita. Tale architettura è mostrata in figura B. Il primo bus è usato dal processore per accedere alla memoria e per dialogare con le periferiche solo in termini di comandi da eseguire. Il bus DMA (Direct Memory Access) è invece usato dalle periferiche per i trasferimenti diretti in e dalla memoria in modo da non occupare il bus principale che così è sempre libero per il processor.

Un ulteriore incremento delle prestazioni si ha con lo schema di figura C in cui si utilizza un collegamento dedicato (quindi velocissimo) tra processore e memoria. In questo modo il processore può contemporaneamente dialogare con le periferiche e fare operazioni in memoria, fermo restando che il bus DMA, come prima, permette questo anche alle periferiche.

L'ultimo schema, mostrato in figura D riguarda l'architettura di un calcolatore multiprocessor. Le due strutture di interconnessione permettono a differenza del bus, un collegamento parallelo fra i due fronti: tutti i processori nello stesso istante possono accedere a locazioni di memoria così come possono comandare le periferiche di ingresso uscita. Anche per tale architettura è possibile aggiungere un bus DMA che come prima permette di migliorare le prestazioni.