1 Lezione del 02-10-25

Continuiamo la discussione dei processi:

1.0.1 Stato dei processi

In un **S/O monoprogrammato** il processo può trovarsi in uno di due stati:
grafico attivo - bloccato

- **Attivo**: il processo stato creato, è in esecuzione ed ha ancora istruzioni da eseguire.

  Creare un processo significa in primo luogo allocare le strutture dati che lo descrivono. In seguito, si deve allocare un po' di memoria al processo stesso per contenere i suoi dati (istruzioni, pila, ecc...) come visto in 4.6. Abbiamo però che allocare particolari descrittori al processo quando questo è l'unico in esecuzione sarebbe inutile: effettivamente un sistema monoprogrammato può essere tale solo se il processo in esecuzione è in qualche modo il S/O.

  A questo punto il processo è l'unico in esecuzione sulla CPU, e resterà tale fino alla fine del suo *lifetime*. Dovrà però *bloccarsi* se vuole accedere a risorse sistema: fara ciò usando una *chiamata a sistema*;

- **Bloccato**: il processo o qualche altro attore ha causato un qualche evento che ha determinato il passaggio di controllo al S/O (richiesta risorse di I/O, di risorse logiche, interruzioni esterne, ecc...).

  A questo punto sarà nuovamente un evento esterno (interruzione esterna, azione dell'utente) a riportare in esecuzione un processo, creandone uno nuovo se lo scorso aveva terminato, o rimettendo il corrente in esecuzione se si era bloccao su una operazione di I/O o simile.

  Notiamo che la memoria del processo potrebbe cambiare anche questo è bloccato: ad esempio se si hanno dispositivi che operano in DMA.

La transizione fra attivo e bloccato è detta *sospensione*, mentre fra bloccato e attivo è è detta *riattivazione*.

Possiamo dire che gli stati *attivo* e *bloccato* sono in qualche modo in corrispondenza con le fasi descritte in 2.1.4 di **CPU-Burst** e **I/O-Burst**.

Se il numero di CPU è minore del numero dei processi, cioè in un sistema **monoprocessore**, ci dotiamo di più stati:
grafico esecuzione - pronto - bloccato

- **Pronto**: in questo caso il processo è in attesa del tempo del processore. Abbiamo che nella maggior parte delle implementazioni questo stato è rappresentato da una *coda* di descrittori di processo, la cosiddetta **coda pronti** (utile per implementare politiche prioritarie $\leftrightarrow$ code prioritarie);

- **Esecuzione**: il processo è in esecuzione sulla CPU;

- **Bloccato**: il processo è in attesa, come nell'esempio precedente.

  In questo caso, come nel precedente, la memoria processo potrebbe essere modificata da dispositivi in DMA. Potrebbero poi essere modificati, in particolari situazioni, i descrittori di processo: magari a causa di operazioni di altri processi col S/O (pipe fra processi, ecc...).

La transizione da pronto e esecuzione è detta *assegnazione CPU*, mentre la contraria è detta *revoca CPU*, o in inglese **preemption**. Notiamo che quest'ultima transizione non è prevista da tutti i sistemi.

L'operazione di assegnazione CPU è eseguita da un componente del S/O detto **scheduler**. Lo scheduler viene eseguito quando il processo in esecuzione cambia stato, quindi in caso di *revoca CPU* o *sospensione*: sostanzialmente ogni volta che si richiede un nuovo processo da mettere in esecuzione. Questo comprende anche la situazione in cui il processo corrente *termina*.

Abbiamo poi che la transizioni allo stato bloccato sono le stesse dell'esempio precedente, con l'unica differenza che la *riattivazione* del processo non lo mette in esecuzione ma nello stato *pronto*.

C'è inoltre un'altro stato, oltre a quello di *terminazione* prima accennato, di cui dobbiamo tener conto: quello in cui il processo viene *creato*, cioè la transizione di un nuovo processo allo stato di *pronto* (detta *attivazione*):

grafico creato - esecuzione - pronto - bloccato - terminato

In questo caso possiamo anticipare che in S/O che adottano politiche *prioritarie* allo scheduling potrebbe essere necessario eseguire lo scheduler (per mettere in esecuzione un nuovo processo di priorità più alta).

1.0.2 Descrittori di processo

Abbiamo introdotto come la gestione dei processi bisogna di apposite strutture dati dette *descrittori* di processo (in inglese **PCB**, *Process Control Block*).

Questa dovrà associare ad ogni processo:

- Nome del processo: questo è il solito **PID** (*Process IDentifier*), ed identifica univocamente ogni processo. Richiedere che i PID siano univici è una *politica* che li trasforma in una *risorsa*: l'S/O dovrà impegnarsi a gestire i PID in modo che non avvengano collisioni;
- Stato del processo: codifica uno degli stati definiti prima;
- Modalità di servizio: questo riguarda la priorità (se implementata) o il tipo di scheduling che si usa per gestire il processo;
- Informazioni sulla gestione della memoria: conterrà puntatori alla memoria dedicata al processo;
- Contesto del processo: l'immagine dei registri all'ultima sospensione del processo;
- Utilizzo delle risorse: conterrà puntatori alle risorse logiche e fisiche a cui ha accesso il processo;
- Idefntificazione del processo successivo: questo serve semplicemente ad implementare, come abbiamo accennato, le code prioritarie di processi (come ad esempio la *coda pronti*).

Una volta definito il descrittore di processo, potremo volerci fornire di:

- La coda dei processi pronti;
- Una o più code per i processi bloccati (solitamente una coda è associata a una risorsa su cui i processi si bloccano);
- Un registro di qualche tipo che contiene il processo attualmente in esecuzione.

1.0.3 Cambio di contesto

Il **cambio di contesto** è l'operazione attraverso cui l'uso del processore viene commutato da un processo all'altro. Questo consiste in:

- Salvataggio del contesto del processo in esecuzione nel suo descrittore (cioè salvataggio di *stato*);
- Inserimento del descrittore di processo corrente in coda *bloccati* o *pronti*.
- Selezione di un nuovo processo da mettere in esecuzione e caricamento nel registro del descrittore del processo corrente (*short term schedulilng*);
- Caricamento del contesto del nuovo processo e cessione a questi del controllo.

Notiamo che per realizzare il cambio di contesto al S/O (che si occupa poi di portare avanti queste operazioni) abbiamo bisogno di funzionalità implementate in hardware: il "processo" S/O è semplicemente il processo in esecuzione al tempo del cambio, che è forzato a passare al contesto sistema nell'istante in cui si mette ad eseguire codice sistema.

Questo significa che per realizzare un S/O si necessita di un processore che implementi il cambio di contesto (x86 dal 286, ecc...).

Possiamo iniziare ad approfondire il modo in cui indicizziamo il processo. Dal punto di vista concettuale, vorremo usare una tripla:

$$S =< \text{PID}, \text{UID}, \text{GID} >$$

composta da **PID** (*Process IDentifier*, già visto), e **UID** e **GID** (*User IDentifier* e *Group IDentifier*), che rappresentano il proprietario del processo.

In caso di cambi di contesto al contesto sistema, quello che faremo è semplicemente cambiare UID e GID in modo da eseguire lo stesso processo, ma con privilegi diversi.

1.0.4 Creazione e terminazione di processi

Vediamo le operazioni che si possono svolgere sui processi.

Avremo che vorremo supportare *gerarchie* di procssi, dove un processo (padre) può richiedere la creazione di un nuovo processo (figlio). Questo significherà che ogni processo sarà figlio di un altro processa, e potra a sua volta essere padre di altri processi. Chiaramente, le informazioni relative alle relazioni parentali saranno mantenute da S/O nei descrittori di processi.

Se un processo termina, di base:

- Il padre può rilevare il suo stato di terminazione;
- Tutti i suoi figli terminano.

1.0.5 Processi concorrenti

Specifichiamo come più processi vengono eseguiti su una stessa macchina. Abbiamo che questi possono alternarsi (*interleaving*), tipico delle macchine a singolo processore, o eseguire effettivamente l'uno contemporaneamente all'altro. Questo è tipico delle macchine a più processori.

Nel caso di più processi in esecuzione contemporaneamente (diciamo processi *concorrenti*) vengono a verificarsi alcune problematiche:

- Processi **indipendenti**: se due processi $P1$ e $P2$ sono indipendenti, cioè non influenzano l'uno l'esecuzione dell'altra, dobbiamo assicurarci che questo resti vero, cioè dobbiamo risolvere il *problema della riproducibilità*. Visto che le risorse sistema sono condivise, dobbiamo infatti assicurarci che non ci siano effetti collaterali sensibili da un processo o dall'altro.
- Processi **interagenti**: se due processi $P1$ e $P2$ devono interagire fra di loro, possono farlo in maniera *esplicita* (per **cooperazione**), scambiandosi messaggi, o *implicita* (per **competizione**), magari competendo per la stessa risorsa in mutua esclusione.