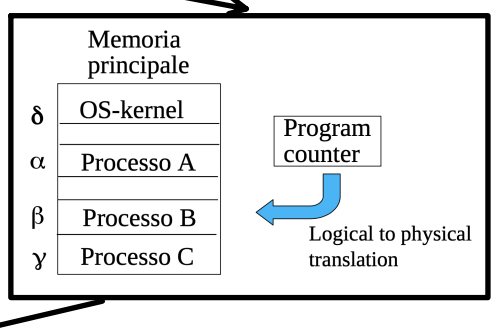
Introduzione

martedì 28 marzo 2023 12:25

INTRODUZIONE

Sappiamo che un processo in un sistema operativo Time-Sharing, quelli che comunemente utilizziamo, è di fatto un'applicazione attiva.

Se noi consideriamo una memoria di lavoro, tipicamente in un S.O time-sharing abbiamo che una zona è riservata al kernel del S.O, il software cuore del sistema che quando noi mandiamo in start-up la macchina viene ad essere caricato all'interno della memoria, e poi riserviamo spazio per un processo A,B,C e così via.



Se guardiamo individualmente questi contenitori di memoria, sappiamo che c'è un program counter che sta lavorando individualmente all'interno di un contenitore, quello che noi sappiamo è che attualmente la CPU sta lavorando all'interno di un processo, ad esempio il processoB.

Quindi sta all'interno di quella zona beta eseguendo e pescando un'istruzione una dopo l'altra.

β+0
β+1
β+2
β+3

Questa è una sequenza di istruzioni macchina che caratterizza l'esecuzione del processo B, ma lo stesso vale per gli altri processi.

L'esecuzione di ogni processo puo' essere caratterizzata tramite una sequenza di istruzioni denominata ***traccia*** **(Istruzioni macchina)**

Un sistema operativo Time-Sharing garantisce una esecuzione ***interleaved*** delle tracce dei singoli processi

Consideriamo una funzione: Questa funzione contiene l'istruzione macchina A,B,C,D ed E!

A
B
C
D
E

Supponiamo di dare all'interno dell'applicazione, il controllo a questa funzione.
Noi non sappiamo cosa fanno queste istruzioni. Potremmo avere che l'istruzione D è un salto condizionato verso l'istruzione B.

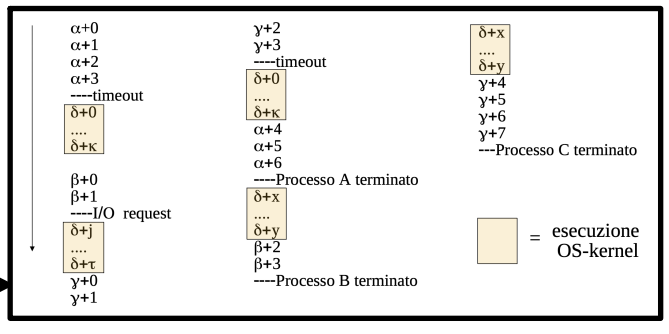
Qual'è la traccia che noi possiamo generare, eseguendo all'interno di una specifica applicazione questa funzione?
Qual'è la porzione della traccia che viene fuori quando l'applicazione esegue esattamente questa funzione F()?

Sicuramente entriamo ed eseguiamo l'istruzione macchina A. Poi B,C,D (Se la condizione del salto non è verificata eseguiamo E). E usciamo dalla funzione. A,B,C,D,E è ciò che la funzione F() ha incluso all'interno della traccia d'esecuzione di questa applicazione. Avremmo potuto avere anche A,B,C,D,B,C,D,E. Questa è un'altra *traccia*.

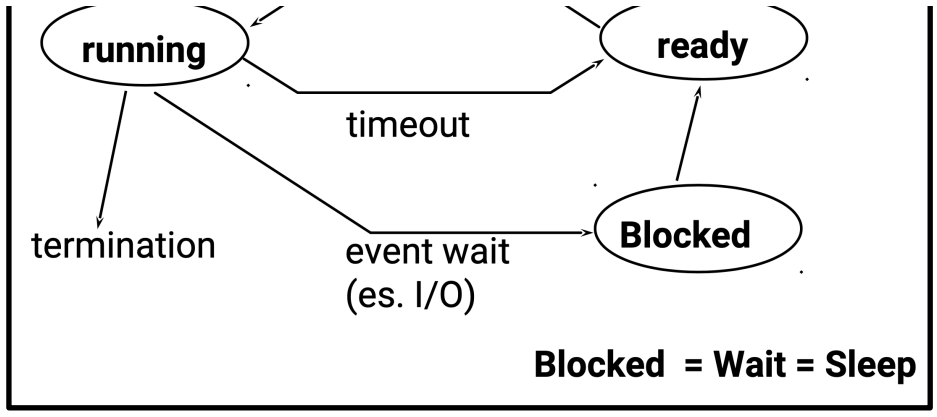
La traccia effettiva di un'applicazione è la sequenza delle istruzioni macchina che vengono mandate in fetch.

Un esempio di esecuzione interleaved

L'esecuzione è mischiata in CPU: Inizialmente la CPU è assegnata al processo A, e quindi la CPU esegue le istruzioni del processo A.
Poi c'è un timeout e arriva un interrupt di un timer che ci porta il controllo da un'altra parte: cambiamo lo stato e prende il controllo un'altra parte del software.



Stati fondamentali dei processi



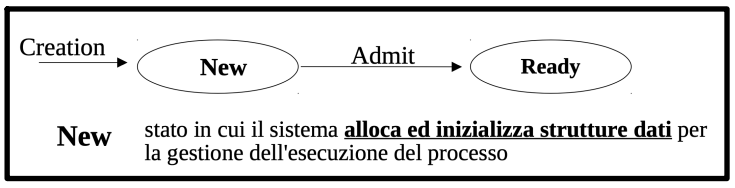
In un sistema potremmo avere più di un'applicazione caricata in memoria e quindi più di una traccia d'esecuzione, cioè sequenza di istruzioni macchina user space dell'applicazione, che potrà includere trap al kernel (la parte kernel non fa parte di questa traccia, solamente le chiamate).

Il sistema operativo dovrà quindi gestire l'Interleave delle varie tracce d'esecuzione user space, secondo la tecnica del time-sharing.

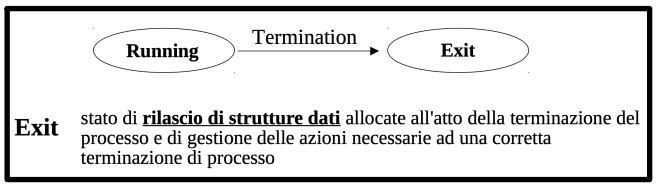
Un modello basico può includere due stati: un processo può essere RUNNING o NOT RUNNING, cioè la sua traccia è in esecuzione o no, si passa da uno stato all'altro tramite transazioni di stato, cioè quando il kernel assegna il controllo al processo oppure lo mette in pausa per dare il controllo ad un altro.

Lo stato READY, cioè in questo stato un processo è candidato a girare, e NOT RUNNING viene chiamato WAIT, quest'ultimo stato rappresenta un'attesa di eventi (per esempio interazione con dispositivi di I/O).

WAIT, quest'ultimo stato rappresenta un'attesa di eventi (per esempio interazione con dispositivi di I/O).



Un'applicazione non nasce RUNNING, perciò bisogna inserire uno stato NEW, il processo alla sua nascita è NEW, tramite un evento di admit viene poi portato a READY, cioè dopo che si sono fatte delle operazioni di setup.



Ovviamente c'è poi il duale, cioè lo stato EXIT per consentire il rilascio delle risorse da parte del kernel per conto del processo. Una volta in EXIT un'applicazione non può più essere ri-schedulata.

Il livello di multiprogrammazione

Il processore e' tipicamente molto piu' veloce dei sistemi di I/O

↳ esiste la possibilita' che la maggior parte dei processi residenti in memoria siano simultaneamente nello stato **Blocked** in attesa di completamento di una richiesta di I/O

↳ Rischio di sottoutilizzo del processore

Infatti spesso in questi casi di processi BLOCKED si prende il loro address space e lo si porta sul disco e lo si toglie dalla RAM, per far eseguire processi che sono ready.

Noi possiamo spostare applicazioni, ossia i loro address space, dalla memoria di lavoro alla memoria disco e riportarli in memoria di lavoro quando serve: ossia quando essi diventano immediatamente READY possiamo riassegnare la CPU, che ovviamente lavora in RAM, e questa operazione si chiama SWAPPING.

SWAP OUT

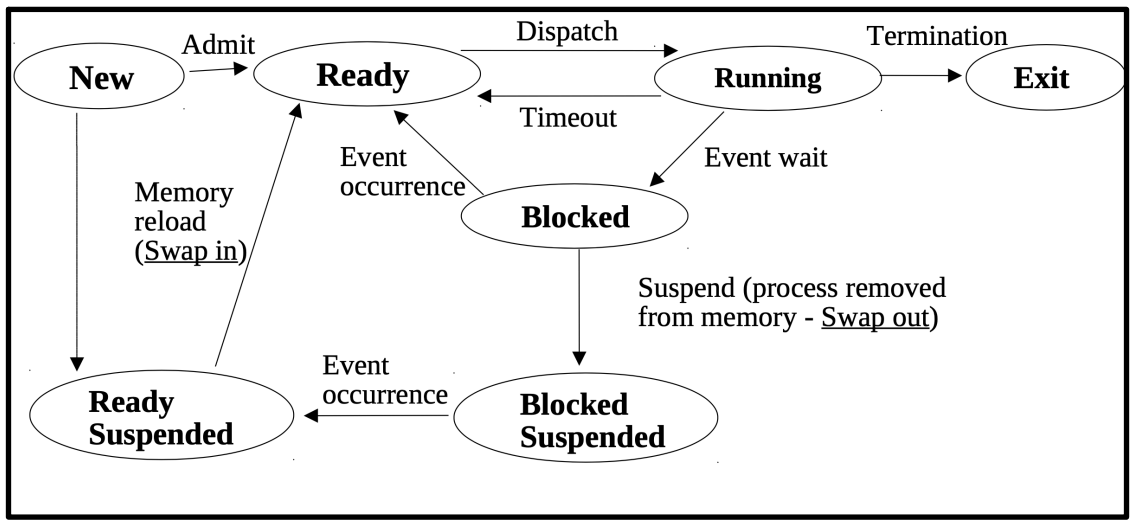
Escludere un'applicazione dalla RAM e la carichiamo su disco.

SWAP IN

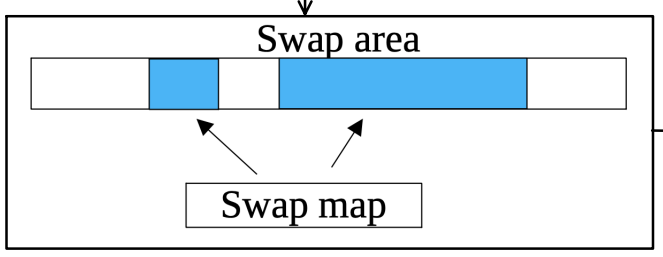
Carichiamo un'applicazione (anche per la prima volta) all'interno della memoria di lavoro e gli diamo il controllo.

Lo Swap Out tipicamente viene applicato a processi che sono BLOCKED=WAIT, ovvero che sono entrati in stato di attesa che accada qualche evento. Finché non accade questi non possono usare la CPU e non ha senso tenerli nella RAM. Così evitiamo il SOTTOUTILIZZO DEL PROCESSORE.
In questo caso l'applicazione diventa "SOSPESA".
Un'applicazione BLOCKED può essere portata fuori dalla memoria di lavoro con un'operazione di SWAP OUT, E COSÌ DIVENTA SOSPESA.

SCHEMA FINALE



La zona della memoria secondaria dove noi andiamo a caricare gli address space quando li eliminiamo dalla memoria principale viene denominata "SWAP AREA".



Esiste una zona della memoria secondaria che viene ad essere utilizzata come swap area, ovviamente al livello del sistema operativo, tra le informazioni per gestire questa swap area, abbiamo una SWAP MAP che ci va ad indicare le zone occupate rispetto alle zone libere di questa swap area.

Il sistema preleva anche address space dalla swap area per portarli sulla main memory e viceversa.