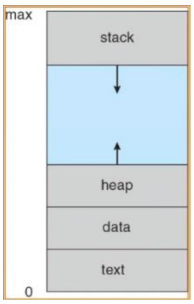
5.0 STATI DI UN PROCESSO

Supponiamo di avere un programma in C e di effettuarne una compilazione. Quando generiamo il nostro file a.out e lo mandiamo in esecuzione, stiamo creando un'**istanza del programma**, ossia un **processo** che deve eseguire delle istruzioni scritte all'interno di un programma. Un processo rappresenta un'esecuzione dell'eseguibile. Di un medesimo programma possono esserci più processi associati.

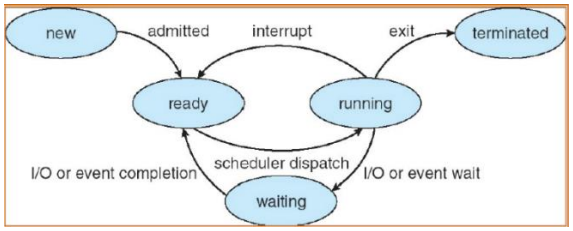


Dal punto di vista del SO, la prima cosa che succede, quando a.out viene mandato in esecuzione, è che deve essere trovato un posto all'interno della memoria principale. Una volta trovato questo pezzo in memoria principale all'interno di questo spazio sono presenti le seguenti informazioni: il codice eseguito, i vari dati del programma e poi viene lasciato uno spazio completamente libero, il quale verrà utilizzato come spazio di servizio, per esempio nel momento in cui avviene un'allocazione dinamica della memoria, questo spazio allocato dinamicamente è presente in questo spazio libero. Ci sarà anche uno stack all'interno dello spazio del processo.

Supponiamo che nel nostro codice sono presenti delle chiamate a funzione, quando chiamiamo una funzione, tutte le informazioni contenute nel programma principale vengono conservate all'interno dello stack. Dopo che la funzione chiamata è stata eseguita, tutte le informazioni del programma principale vengono recuperate dallo stack e l'esecuzione può procedere normalmente.

Una volta eseguito il primo step (trovare spazio nella memoria principale), si deve caratterizzare il processo. Un processo è caratterizzato da: lo spazio allocato, il program counter (ci dice a che punto siamo dell'esecuzione del codice), contenuto registri CPU (in un certo istante, quali contenuti dei registri della CPU sono usati), sezione testo (codice), sezione dati, heap, stack.

Tutte queste informazioni ci servono per sapere ad un certo istante, un processo in che stato si trova.



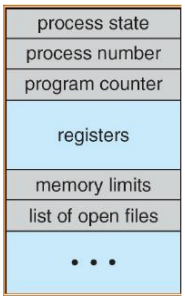
Un processo durante la sua esecuzione può trovarsi in uno di 5 stati:

- **New**: il processo è stato creato (appena do il comando a.out);
- **Running**: le sue istruzioni vengono eseguite. Questo è il momento in cui gli viene assegnata la CPU;
- **Waiting**: il processo è in attesa di qualche evento (ad esempio fine di un'operazione di I/O);
- **Ready**: il processo è in attesa di essere assegnato ad un processore;
- **Terminated**: il processo ha terminato l'esecuzione.

Appena si crea un processo, siamo nello stato **new**, in questo stato si assegna lo spazio di memoria. Una volta "apparecchiato", il processo entra nello stato **ready**, in questo stato è pronto per l'esecuzione e si aspetta che venga assegnata la CPU. Dallo stato **ready** si esce e si va nello stato **running** in cui viene assegnata la CPU e si comincia con l'esecuzione del programma. Dallo stato **running** si può uscire per vari motivi: quando c'è *un'operazione di I/O* il processo viene fatto uscire dalla CPU, si passa allo stato **waiting** in cui si aspetta la computazione. Appena l'operazione finisce si torna allo stato **ready** in cui si indica che il processo è pronto e che rimane in attesa della CPU. Un altro modo per uscire dallo stato **running** è quando arriva un *interrupt*. Un esempio di interrupt si ha quando il tempo di un processo all'interno della CPU è limitato, superato un certo tempo scatta un interrupt che fa passare il processo allo stato **ready**. Mediante un interrupt, si torna allo stato **ready**, in attesa che venga riassegnata la CPU. Dallo stato **running** si passa allo stato **terminated** mediante una *exit* nel momento in cui ho terminato il codice da eseguire.

Quando un processo perde la CPU, in qualche modo si deve effettuare una fotografia per sapere a che punto si trova il processo, per poi poter ripartire dal punto in cui ci si è fermato in precedenza. Queste informazioni sono presenti all'interno del **Process Control Block**.

Il **Process Control Block** rappresenta una serie di informazioni sullo stato del processo, su quella che è la situazione attuale, e sono:



- **Stato del processo**: a quale degli stati si trova un processo;
- **Numero del processo**: ad ogni processo viene assegnato un identificatore numerico che lo identifica univocamente. Questo viene chiamato il **PID** di un processo;
- **Program counter**: a che punto dell'esecuzione del codice sono arrivato. Utile nel momento in cui si riprende la CPU si sa da dove riprendere l'esecuzione;
- **Registri;**
- **Memory limits**: da dove comincia e dove finisce il pezzo di memoria assegnato ad un processo;
- **Lista dei file aperti dal processo;**
- **Informazioni di contabilizzazione** (quanto tempo ho usato la CPU), **informazioni sullo stato dell'I/O**, etc…

Supponiamo di avere un processo P_0 in esecuzione (stato **running**). Per un qualche motivo arriva un interrupt, dunque il processo perde la CPU a favore di un altro processo.

Quando il processo esce dalla CPU deve essere aggiornato il suo **Process Control Block**.

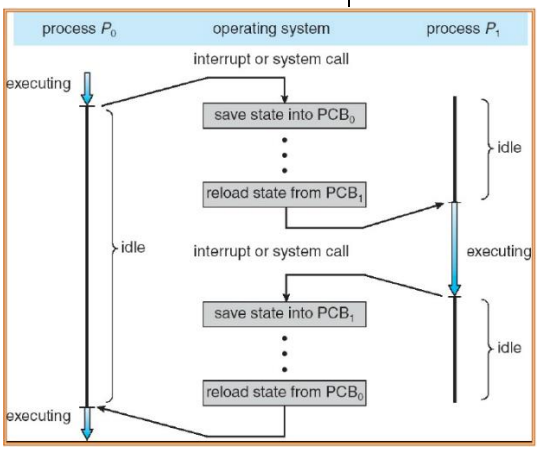
Esce dalla CPU, vengono salvate le sue informazioni nel PCB_0.

Il SO deve scegliere tra tutti i processi **ready**, quello che deve prendere la CPU. Supponiamo che tra tutti i processi **ready**, viene scelto P_1.

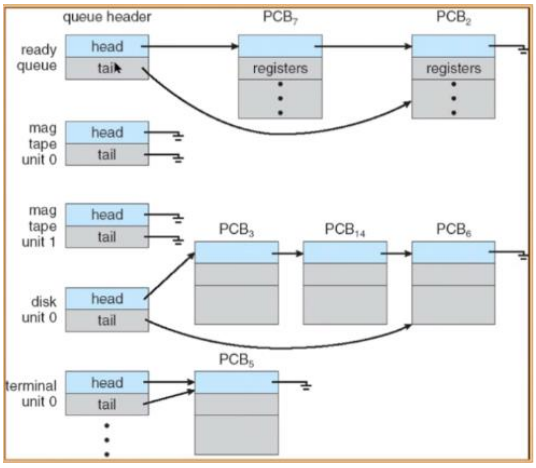
Dal PCB di P_1 devono essere prese tutte le informazioni per poter ripartire da dove era stato fermato in precedenza.

Questo lasso di tempo viene chiamato **cambio di contesto** (rappresenta un **tempo perso**).

Una volta fatto ciò il processo P_1 viene eseguito fino a quando per un qualche motivo c'è un interrupt e gli viene tolta la CPU e si ripete quanto detto in precedenza.

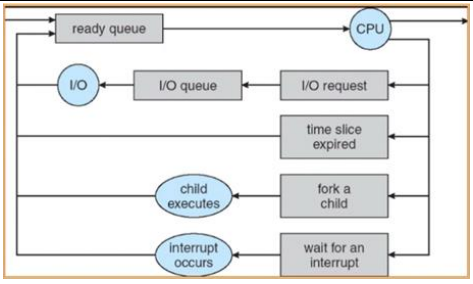


Un concetto importante sono le **Code di scheduling per i processi**, cioè, quando un processo esce dalla CPU, può capitare che non è l'unico processo che magari aspetterà nuovamente la CPU.
I processi che sono nello stato **ready** e sono in attesa vengono messi all'interno di una **Ready queue**. È presente anche una **coda dei dispositivi**, ossia l'insieme dei processi in attesa di qualche dispositivo I/O.
Più in generale abbiamo una **job queue**, ossia l'insieme di tutti i processi nel sistema. I processi, durante la loro vita, migrano tra varie code.
Questo è un esempio di ready queue in cui ogni processo è in attesa ed è conosciuto mediante il suo PCB →



5.1 CICLO DI VITA DI UN PROCESSO & SCHEDULATORI

Un processo, dopo lo stato **new**, arriva nella **ready queue** e sta lì fino a che non gli viene assegnata la CPU. Dalla CPU può uscire quando il codice del programma è terminato, oppure perché c'è una richiesta di I/O, e se c'è una richiesta di I/O allora si deve entrare all'interno della coda dei dispositivi e aspetta che possa essere eseguita una certa operazione.
Completata questa operazione si ritorna nella **ready queue**.
Dalla CPU si può anche uscire perché è finito il tempo assegnato a priori all'interno della CPU.
Non si passa per altre code ma si ritorna subito alla **ready queue**.
Si può uscire dalla CPU perché è stata fatta una **fork** per creare un figlio.
L'ultimo caso per cui si esce dalla CPU è quando c'è un **interrupt**.

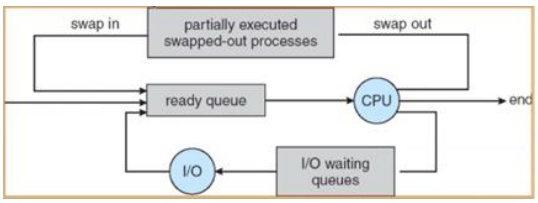


Nella gestione dei processi sono importanti gli **schedulatori**, che sono degli strumenti che servono a selezionare i processi dalle varie code, quando si è per esempio nella **I/O queue**, ci sono vari criteri per stabilire quali delle richieste servire per prima. Lo schedulatore sceglie tra i processi che devono entrare nell'I/O quale eseguire. La scelta avviene secondo vari algoritmi. Ci sono vari schedulatori:

- **Schedulatore a lungo termine (job scheduler)**: seleziona i processi dal disco che devono essere caricati in memoria centrale (**ready queue**). Questo avviene perché non tutti i processi possono stare contemporaneamente in memoria principale altrimenti si consuma molto spazio.
- **Schedulatore a breve termine (CPU scheduler)**: seleziona il prossimo processo che la CPU dovrebbe eseguire.

Sistemi come Unix e Windows, sono privi dello schedulatore a lungo termine e si limitano a caricare in memoria tutti i nuovi processi ed a gestirli con lo scheduler a breve termine. È l'utente che, se vede che le prestazioni della macchina diminuiscono, decide di chiudere alcune applicazioni.

Altro tipo di schedulatore è quello a **medio termine** che viene usato quando non esiste quello a lungo termine.
Una volta che si rende conto che la situazione si sta intasando, prende il processo che ha la CPU e lo mette sul disco.
Una volta che si libera dello spazio in memoria principale, lo riporta sulla memoria.



I processi possono essere descritti come:

- **Processi I/O bound**: consumano più tempo facendo I/O che computazione, contengono molti e brevi CPU burst.
- **Processi CPU-bound**: consumano più tempo facendo computazione; contengono pochi e lunghi CPU burst.

È compito dello **scheduler a lungo termine** quello di scegliere una giusta combinazione tra i processi di I/O bound e quelli CPU-bound che sono presenti in memoria centrale.