21-03-2023

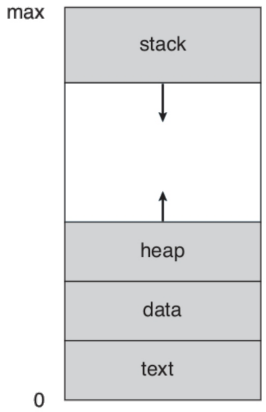
Processi, Thread, IPC e Scheduling

- Affinchè i processi comunichino fra loro devono **condividere le risorse** o **parti di memoria** ed è gestito dal GESTORE DI PROCESSI. Inoltre esso si occupa dello scheduling dei programmi (ordine di esecuzione).

Processo

Un processo è **una porzione di programma** che va in esecuzione (quindi non è l'intero programma). E' una sua **astrazione**.

> *Si riesce a dare una rappresentazione come se si avessero più CPU.*



Le informazioni contenute nel PCB variano a seconda delle implementazioni, ma in generale sono presenti:

Program counter
Area per il salvataggio dei registri general purpose, di indirizzamento
Area salvataggio registro di stato
Area di salvataggio per i flag
Stato corrente di avanzamento del processo (Pronto, In Esecuzione, Bloccato)
Identificatore unico del process
Un puntatore al processo padre
Puntatore ai processi figli se esistenti
Livello di priorità
Informazioni per il memory management(Gestione della memoria) (in particolare memoria virtuale) del processo
Identificatore della CPU su cui è in esecuzione
Informazioni per lo scheduling (gestione) del processo, come il tempo di run (esecuzione) o wait (attesa) accumulato o tempo stimato di esecuzione rimanente
Informazioni di accounting di un processo
Segnali che pendono
Informazioni sullo stato di I/O del processo
Registro nel quale è presente un puntatore alla page table

> Un programma è un'entità **PASSIVA** mentre il processo è un'entità **ATTIVA**

- Quando un processo viene sostituito da un'altro (interrotto) si devono conservare i **dati essenziali affinchè possa ripartire** da dove si è fermato.
- Ad ogni processo viene associato uno **spazio degli indirizzi** proprio per questo motivo.
  Un processo **contiene** anche:
- codice eseguibile
- dati del programma
- stack
- copia dei **registri** della CPU (*per la **ripresa** del processo*)
  - Ci interessa sapere cosa il processo ha fatto prima che esso viene fermato, ovvero le informazioni contenute in una parte di memoria (registri)
- **file aperti**, che possono essere diversi. Ogni file aperto ha il **puntatore a tale file** e ognuno di essi deve essere memorizzato per sapere da dove ripartire. Questo salvataggio avviene mediante la **chiamata** detta **READ**
- **allarmi** pendenti
- processi imparentati

Vi è la **TABELLA DEI PROCESSI** che contiene tutte le info del processo che sono diverse dai contenuti elencati sopra. Ogni voce in tabella prende il nome di **Process Control Block**.

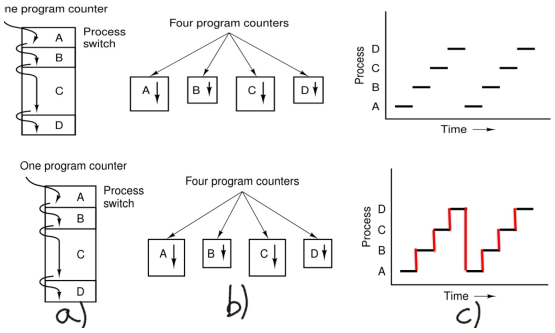
- **Ogni processo** ha una **propria** tabella dei processi

In definitiva, un processo è formato dal suo **spazio degli indirizzi** e dalla sua **tabella dei processi.**

Modello dei processi

Con un'architettura multiprocessore, si possono eseguire più processi in maniera pseudo-parallela perchè se dovessi verificare un'istante di tempo, al suo interno avrò un solo processo e non 5 processi.

Lo **pseudo-parallelismo** avviene perchè in un istante di tempo si riescono a eseguire più processi quasi allo stesso tempo. E' come se si avesse la rappresentazione di **CPU virtuali** e ognuna si dedica a uno specifico processo



> Più volte, un determinato tempo contiene più di un processo

- Ogni processo ha **un SOLO** *Program Counter* (PC)

Dalla figura soprastante:

- figura a: se esegue un programma alla volta, si esegue prima il programma A, poi B ecc...
- figura b: rappresentazione di come se fosse eseguito con CPU virtuali in un certo intervallo di tempo. Si rappresentano i 4 programmi come se fossero eseguiti insiemi. Ogni CPU virtuale esegue un processo distinto
- figura c: se si prende un'istante temporale (linea rossa) si riesce a identificare solo un processo. In quell'istante di tempo si ha lo switch fra processi

> *E' come se ogni processo, con la propria CPU virtuale, avesse un proprio registro PC ma* ***FISICAMENTE*** *è sempre uno*

Processi distinti

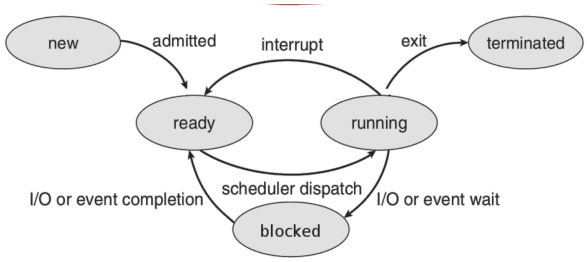
- Un programma può essere eseguito in diverse parti cioè eseguito da **diversi processi** ognuno che si occupa di uno specifico compito.
- Due o più processi dello **stesso programma** non vuol dire che sono uguali. **Un processo è sempre DIVERSO da un'altro processo** anche se essi fanno parte dello stesso programma)
- I processi possono condividere delle informazioni ma comunque rimangono diversi fra loro

Osservazioni importanti

Visto che la CPU passa da un processo a un'altro molto velocemente, l'esecuzione non può essere mai uniforme e uguale ad una parte precedente di processo. Essi **NON devono essere programmati in base alle loro tempistiche** perchè esse dipendono dal consto di quel **momento di esecuzione**.

Stato di un processo

> Un solo processo occupa l'unità di elaborazione (in un istante di tempo).
> Tutti gli altri processi sono in attesa (**come in una coda**) della terminazione di altri processi in modo da occupare l'unità di elaborazione



- **Un processo può fermare la sua esecuzione** anche prima della fine del suo tempo di lavoro (errori, interruzioni, processi con priorità più alta)
- Tutti i processi che **non possono continuare nella loro esecuzione**, piuttosto che essere in coda che competono per l'unità di elaborazione, vengono messi nella coda dei processi bloccati che richiedono delle risorse non ancora disponibili.
  - Quando la loro richiesta disponibile per essere eseguita, allora il processo passa allo stato Ready venendo tolto dalla coda dei processi **BLOCCATI**

Un processo può trovarsi in diversi **STATI**:

1. **In esecuzione (*RUNNING*)**: Il processo occupa l'unità di elaborazione. E' sempre **UNICO**
2. **In attesa (*READY*)**: Qui ci sono tutti i processi in attesa diversi dal processo in running.
3. **Bloccati (*BLOCKED*):** E' una coda diversa dalla coda dei processi in stato di *READY* e contiene tutti quei processi che non possono andare avanti con la propria esecuzione perchè non si trovano nelle condizioni di essere eseguito (*risorse non ancora disponibili*). Un processo viene bloccato se vengono fatte richieste e tali richieste non possono essere soddisfatti in un determinato istante.
4. **Nuovo processo (*NEW*)**: un processo che viene appena creato e
5. **Terminato (*TERMINATED*)**: un processo ha finito tutti i suoi compiti

Creazione di un processo

Un processo può essere creato:

- **in fase iniziale** quando si avvia il sistema
- **da altri processi**
- **dall'utente** mediante l'operazione di apertura di un file o altro attraverso le chiamate di sistema.

Si distinguono:

- processi **ATTIIVI**: interagiscono con l'utente
- processi **PASSIVI**: sono background e non sono associati ad utenti ma hanno specifiche attività da svolgere (detti anche **PROCESSI DEMONI**)
  - Principalmente sono programmi di sistema che, una volta avviati, **rimangono dormienti per la gran parte della loro esistenza**. Ogni tanto si svegliano, eseguono attività e tornano nello stato PASSIVO. (per esempio "*controllo delle email*" oppure "*Stampa*")

Un nuovo processo può creare altri processi tramite **chiamate di sistema**. Essa indica al SO di creare un nuovo programma che deve essere poi eseguito

> In **UNIX** la chiamata di sistema è detta `fork`. **Inizialmente** i due processi hanno la **stessa immagine** (identici) e successivamente il processo creato dalla fork, attraverso la chiamata `exec` cambia la sua immagine e diventa autonomo e i **due processi si separano** e ognuno di essi hanno il **PROPRIO spazio degli indirizzi** (*modifiche dello spazio degli indirizzi del padre non sono visibili nello spazio degli indirizzi del figlio. Vale anche il viceversa*)

> Mentre in **Windows** la chiamata di sistema è `CreateProcess`

Terminazione di un processo

Un processo può essere terminato per vari motivi:

- **USCITA NORMALE** perchè termina normalmente la sua esistenza, terminano la loro esecuzuione
  - Viene eseguita una chiamata di sistema `exit` (UNIX) oppure `ExitProcess` (Windows)
- **USCITA SU ERRORE** causato dal processo stesso
- **ERRORE CRITICO**: per esempio compilare un file non esistente `g++ ...`
- **TERMINATO DA UN ALTRO PROCESSO**: con il comando `kill` (UNIX) o `TerminateProcess` (Windows)

Gerarchia dei processi

Il processo che crea un processo e chiama `fork` è detto processo **PADRE**. Il processo creatocon la `fork` è detto processo **FIGLIO**.
A sua volta quest'ultimo può creare tanti altri processi.

> Un **figlio** ha **un solo padre** ma un **padre** può avere **più figli**.

- In UNIX un processo (con i figli) formano un **gruppo di processi**. Quando un utente **invia un segnale**, esso viene inviato a tutti i membri di questo gruppo. Ogni membro può decidere se **considerare** il segnale oppure **ignorarlo**.
  - Esiste un processo speciale (init) che è la RADICE di tutto da dove nascono tutti i processi figli e il suo primo compito (in fase di avvio) è leggere il numero di terminali presenti nel sistema e inizia a eseguire le chiamate di sistema un numero di volte pari al numero di terminali.
  - I processi (che sono associati a un utente) eseguono 5 (per esempio) fork distinte. Questi processi creati attendono il login dell'utente.
- In **WINDOWS** non c'è un vero e propria gerarchia di processi ma **i processi sono tutti allo stesso livello**.
  - Quando un processo ne crea un altro, il processo **genitore avrà un token** attraverso il quale si riesce a **controllare il processo** appena generato

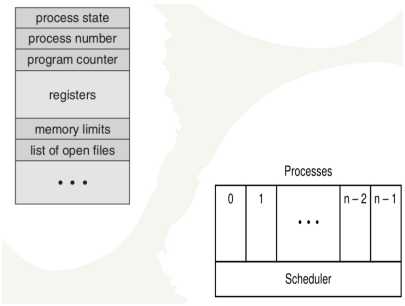
Tabella dei processi

Ad ogni processo viene assegnato uno specifico spazio degli indirizzi e una specifica tabella dei processi. All'interno di quest'ultima c'è:

- Process Control Block (PCB) che indica ogni informazione importante per il corretto funzionamento (stato del processo, PC, numero processo, puntatore stack, registri, informazione sulla CPU e scheduling ecc)
- Le informazioni importanti sullo stato del processo
- ***L'avvio di processi, la loro terminazione, gli interrupt e il tempo di ogni processo*** vengono gestiti dallo **SCHEDULER**.

    - Lo **scheduling** (è il **livello più basso del sistema operativo**) è **l'ordine** stabilito per l'esecuzione dei processi. I processi **BATCH** sono considerati di egual importanza. I processi **INTERATTIVI**, invece, possono avere priorità maggiore/minore rispetto ad altri
    - L'algoritmo che determina l'ordine di esecuzione è detto **ALGORITMO DI SCHEDULING**.
- **La priorità**, man mano, **viene diminuita nel tempo** così che i processi con priorità minima venga eseguito anche (prima o poi)

Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		



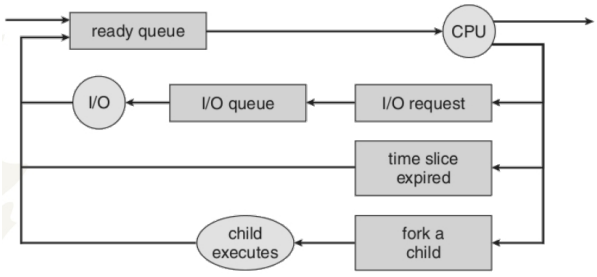
Gli aspetti negativi della tabella dei processi sono i possibili errori, quindi serve il **GESTIRE GLI INTERRUPT**. Bisogna sapere se c'è stato un errore e il relativo **motivo**.
In caso di exit(n), n può avere un valore che serve per capire se l'operazione è andata a buon fine oppure no. Questo n può servire anche per un altro processo e il relativo avvio.

Il **VETTORE DI INTERRUPT** contiene l'indirizzo alla procedura di servizio dell'interrupt. In caso di interrupt:

- **si salva nello stack del PC e del PSW le informazioni memorizzate** allo stato attuale che si devono mantenere
- **caricamento del vettore di interrupt l'indirizzo della procedura associata**
- **salvataggio registri** e impostazione di un nuovo stack
- **esecuzione** procedura di servizio dell'**interrupt**
- **interrogazione sullo scheduler** per sapere con quale processo proseguire
- **ripristino** dal PCB (*Process Control Block*) dello stato di tale processo (registri, mappa memoria)
- **ripresa** nel processo corrente

Diagramma di accodamento dei processi

Quando si mandano in esecuzione i vari processi si ha:



- Al momento della creazione di un processo, esso viene inserito nella **ready queue** (processi per competono per ottenere l'unità di elaborazione)
- Un processo, poi, va in esecuzione in **CPU**.
  Un processo termina perchè
- finisce il tempo di esecuzione: viene reinserito nella *ready queue*
- fa una richiesta di I/O allora non può andare ulteriormente avanti perchè, probabilmente, la risorsta non è disponibile. In questo caso il processo va nella I/O queue.
- genera un processo figlio e ciò viene creato per un motivo specifico. Dopo la fork di un nuovo processo si deve decidere se deve essere eseguito prima il figlio o il padre. Nel disegno, il processo figlio viene eseguito.