

Processi

Monday, 20 March 2023

12:03

- Programma = Entità passiva
- Processo = Entità attiva astratta caricata in memoria per eseguire un programma
- Un programma può eseguire più processo
- Noi vogliamo massimizzare l'utilizzo della cpu = Mantenere la CPU il più impegnata possibile

E qui sono nate 2 tecniche:

- Multiprogrammazione
 - Tanti processi in memoria
 - Quando la cpu non è impegnata, prende un processo
 - Quando un processo è bloccato, la cpu viene riassegnata
 - Swapping se troppi processi in memoria
 - Quantità processi eseguiti contemporaneamente = grado di multiprogrammazione
- Multitasking
 - Tipo multiprogrammazione ma la cpu viene sottratta
 - Tutti i programmi progrediscono in maniera continua
 - Non permette ai programmi batch di monopolizzare CPU
- Operazione sui processi:

- Creazione

Ogni padre prende cura dei propri figli, si crea un albero di processi

Ognuno ha un proprio PID

E per la creazione, i permessi che il padre ha verso il figlio varia dalle varie politiche

Es:

- Tutte le risorse condivise
- Sottoinsieme di risorse
- Nessuna

Es:

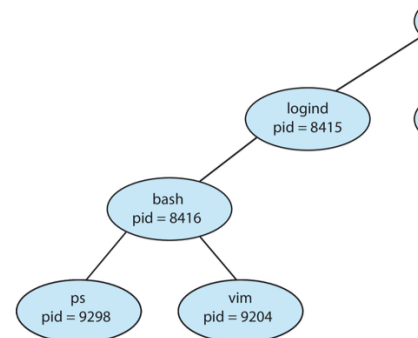
- Figlio duplicato padre
- Figlio diverso dal padre

- Terminazione:

- Padre è sospeso finché i figli non terminano
- Padre può terminare senza che i figli

I comandi che abbiamo sono:

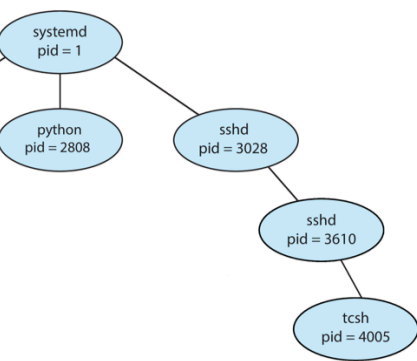
- Fork -> Crea figlio come il padre, ritorna PID
- Exec -> Sostituisce il programma di un processo con un altro
- Wait -> Padre attende figlio
- Exit -> Termina il processo in maniera graceful



ssibile

mazione

che.



- Exit -> Termina il processo in maniera graduale
- Abort -> Terminazione forzata

Ed a seconda delle politiche di sopra, potrebbe avvenire una terminazione a cascata

Implementazione di un processo

- Prima di implementare, dobbiamo comprendere la struttura

La struttura di un processo è fatta in:

- Registri
 - Memoria centrale/immagine
- Processi distinti hanno immagini distinte

E questa immagine è formata da:

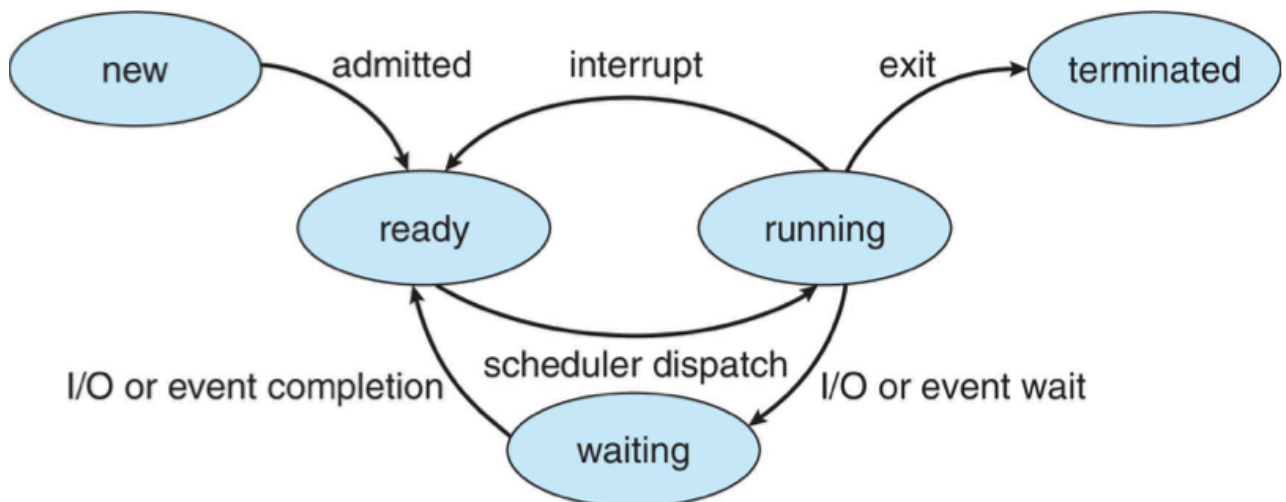
- Stack
Chiamate contenenti parametri, variabili locali, indirizzo ritorno
- Heap
Memoria allocata dinamicamente
- Data
Variabili globali
- Text
Codice di programma

- Stato del processo

- Risorse aperte

Le risorse potrebbero essere condivise

- Stati di un processo

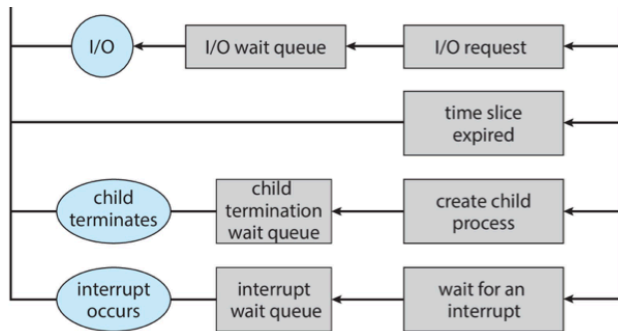


Per il passaggio da ready e running, la scelta viene fatta dallo scheduling dei processi. Esso ha 2 code:

- Ready queue, chi è pronto
- Wait queue, quelli in attesa per IO

Durante la loro vita i processi, con PCB, migrano da una coda all'altra dello stato del pr





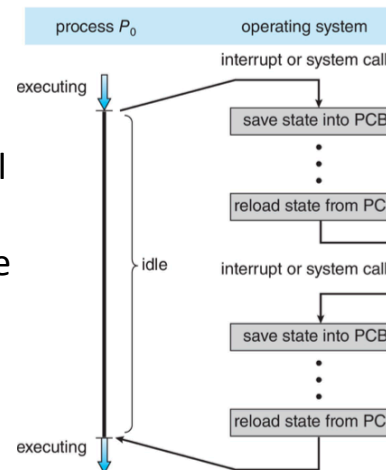
- PCB/TCB

Process control block = Task control block

E' il blocco di informazione che contiene tutto ciò che il kernel deve avere per far avvenire il content switch. -> Onoroso

\-> Viene chiamato overhead, aka lavoro non utile

+ complesso sistema operativo, + poverhead



Da notare che, ogni processi tra di loro possono essere:

- Indipendenti
- Cooperare

Si dice che un processo vuole cooperare quando il suo comportamento viene/influenza

Per permettere questo si mettono a disposizione

Le primitive IPC: comunicazione inter-processo

E questo mette a disposizione:

- Memoria condivisa

Praticamente, fra i 2 processi si decide una parte di memoria in comune

Però questo dà tanti problemi di sincronizzazione

- Message passing

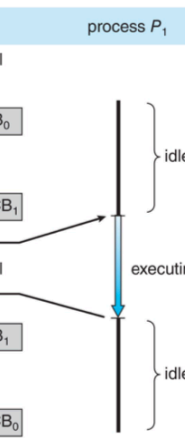
Qui noi abbiamo una lista dove inviamo dei messaggi che l'altro legge

Quindi, non c'è bisogno di una sincronizzazione -> Sincronizzazione implicita

Si utilizzano le primitive:

- Send
- Receive

Reti distribuite <-



to da altri