Sistemi Operativi

GESTIONE

DEI

PROCESSI

Processi

- Concetto di Processo
- Scheduling di Processi
- Operazioni su Processi
- Processi Cooperanti
- Concetto di Thread
- Modelli Multithread
- I thread in diversi S.O.

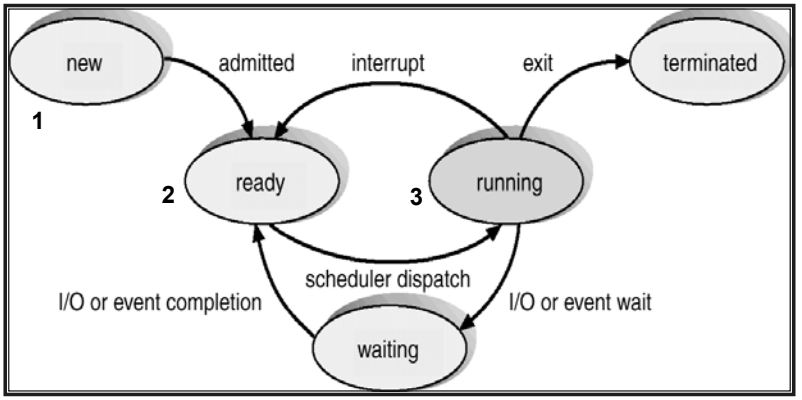
Concetto di Processo

- L'esecuzione di programmi ha diversi nomi in diversi contesti:
  - Sistemi Batch – **job**
  - Sistemi Time-sharing– **processo** o **task**
- I termini ***job*** e ***processo*** si usano spesso come sinonimi.
- **Processo** : un programma in esecuzione; l'esecuzione di un singolo processo avviene in maniera sequenziale.
- Un processo include:
  - sezione testo (codice),
  - il program counter,
  - lo stack,
  - la sezione dati.

Stato del Processo

- Durante la sua esecuzione un processo cambia il proprio ***stato*** che può essere:
  - **new**: Il processo viene creato.
  - **running**: Il processo (le sue istruzioni) è in esecuzione.
  - **waiting**: Il processo è in attesa di un dato evento.
  - **ready**: Il processo è pronto per essere eseguito.
  - **terminated**: Il processo ha completato la sua esecuzione.

Diagramma di stato di un Processo



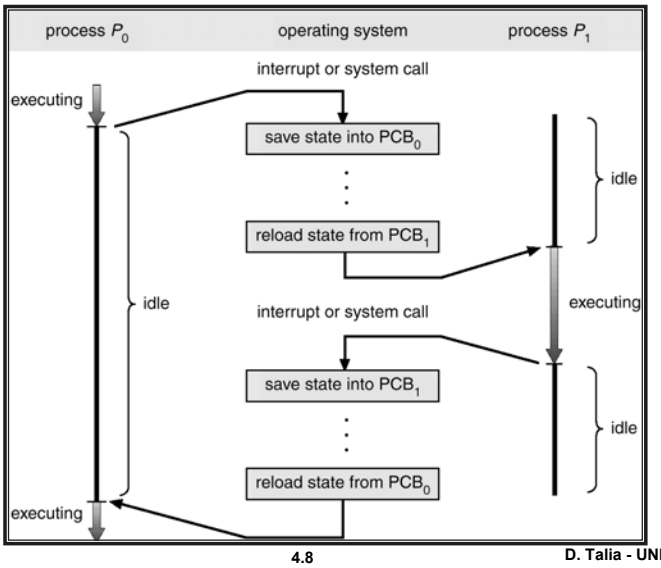
Process Control Block (PCB)

- Il PCB contiene l'informazione associata ad ogni processo:
  - stato del processo
  - program counter
  - registri della CPU
  - info sullo scheduling della CPU
  - informazioni di memory-management
  - informazioni di accounting
  - stato dell'I/O.

Process Control Block (PCB)

puntatore
Stato del processo
Program counter
Registri
limiti di memoria
Lista dei file aperti
. . .

Commutazione della CPU da Processo a Processo



Code di Scheduling

- **Coda dei processi** – l'insieme di tutti i processi nel sistema.
- **Ready queue** – l'insieme dei processi in memoria centrale pronti per essere eseguiti.
- **Coda del dispositivo** – l'insieme dei processi in attesa di usare un dispositivo. (Più code)
- I processi passano da una coda all'altra mentre cambiano stato.

Ready Queue e code dei dispositivi di I/O

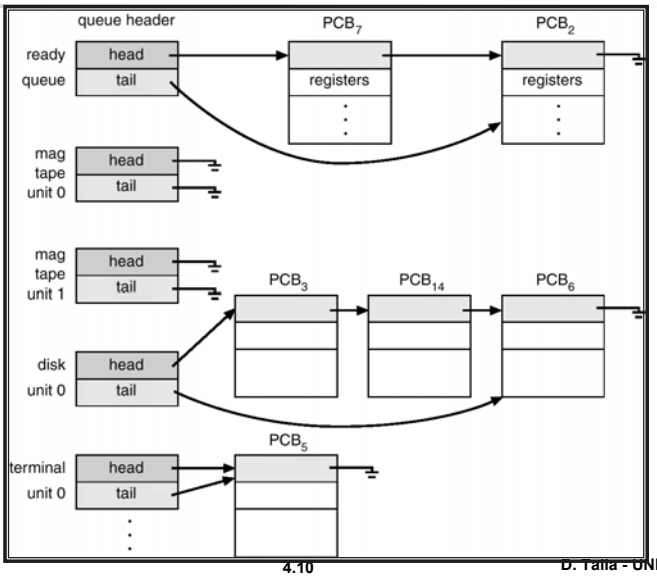
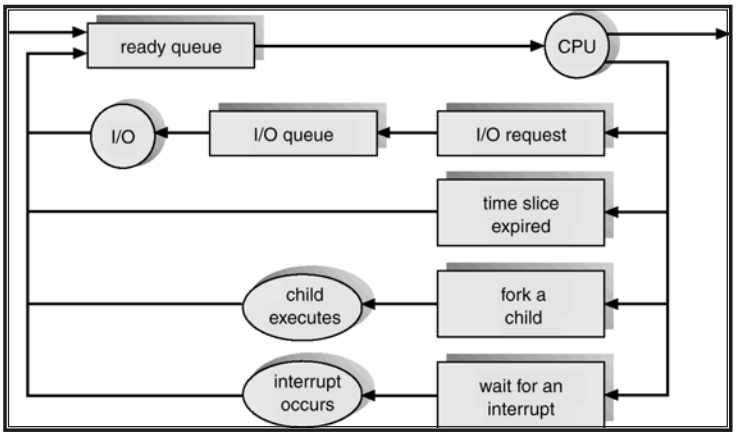


Diagramma di accodamento



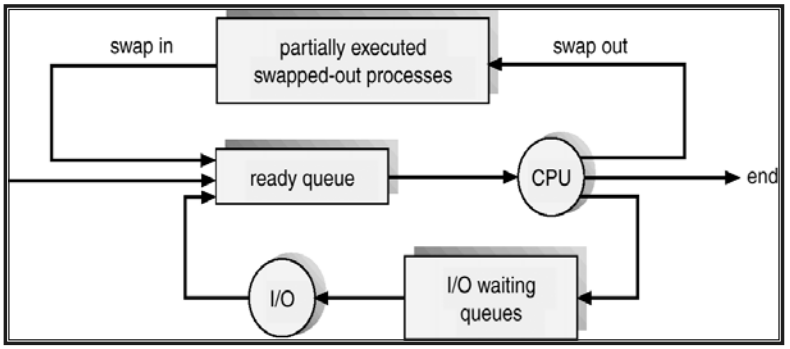
Scheduler

- In un sistema possono esistere più scheduler (es. sistemi batch):
- **Scheduler a lungo termine** (or **job scheduler**) – seleziona i processi da inserire nella *ready queue* (la coda dei processi pronti).
- **Scheduler a breve termine** (or **CPU scheduler**) – seleziona tra i processi pronti quelli che devono essere eseguiti.

Scheduler a medio termine

- In alcuni sistemi time-sharing esiste uno **scheduler a medio termine** che gestisce i processi pronti in memoria centrale. (**swapper**)
- In alcuni casi rimuove i processi dalla memoria (**swap-out**) per riportarli in memoria (**swap-in**) quando sarà possibile.
- Questo migliora l'utilizzo della memoria in caso di una alta richiesta di esecuzione di processi.

Scheduler a medio termine



Schedulers

- Lo scheduler a breve termine è invocato molto frequentemente (millisecondi) ⇒ (deve essere veloce).
- Lo scheduler a lungo termine è invocato non molto spesso (secondi, minutei) ⇒ (può essere lento).
- Lo scheduler a lungo termine controlla il grado di multi-programmazione (n° dei processi in memoria).
- I processi possono essere classificati come:
  - *processi* I/O-*bound* – basso uso della CPU e elevato uso dell'I/O.
  - *processi CPU-bound* – elevato uso della CPU e basso uso dell'I/O.

Context Switch

- **Context switch**: operazione di passaggio da un processo all'altro da parte della CPU.
- Il tempo impiegato per il context-switch time è un costo: il sistema non effettua lavoro utile per nessun processo utente.
- Il tempo di context switch dipende dal supporto offerto dall'hardware.

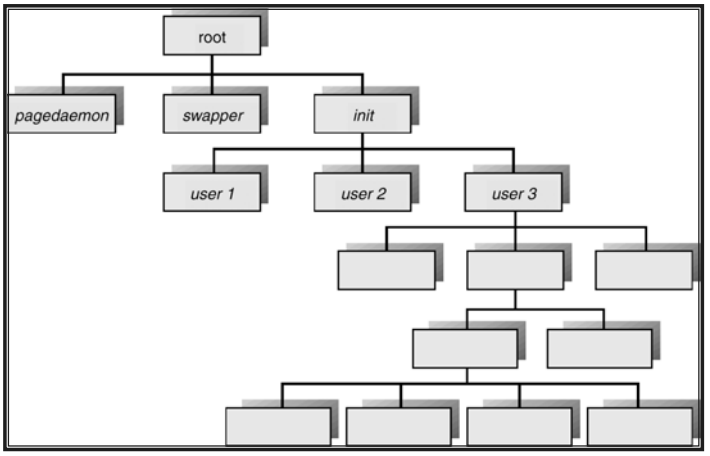
Operazioni sui Processi: Creazione

- Un processo qualsiasi può creare altri processi come suoi figli i quali possono creare altri processi, e cosi via.
- Il sistema operativo crea i processi utente come processi figli.
- Possibile condivisione di risorse:
  - processi padri e figli condividono le risorse.
  - Un processo figlio condivide una parte delle risorse del padre.
  - Processi padri e figli non condividono risorse.
- Approcci di esecuzione:
  - Processi padri e figli eseguono concorrentemente.
  - Il padre rimane in attesa della terminazione dei figli.

Creazione

- Spazio di indirizzi:
  - Il processo figlio viene duplicato dal processo padre.
  - I processo figlio ha un proprio codice.
- Esempio: UNIX
  - **fork** : system call che crea un nuovo processo copiando lo spazio del padre.
  - **exec :** system call per sostituire allo spazio degli indirizzi un nuovo programma.
- Esempio: Windows NT
  - **due modelli:** con e senza duplicazione.

Albero dei processi in UNIX



Terminazione di un processo

- Un processo esegue l'ultima istruzione e chiede al sistema operativo di terminare (**exit**).
  - risultati dal figlio al padre (tramite **wait**).
  - Le risorse del processo sono dealloccate dal sistema operativo.
- Un processo può eseguire la terminazione dei propri figli (tramite **abort**) perché:
  - Il processo figlio non è più utile.
  - il figlio ha usato risorse in eccesso.
  - Il processo padre termina.
    - Molti sistemi non permettono ai figli di eseguire quando il processo padre termina.
    - Terminazione a cascata.

Processi Indipendenti o Cooperanti

- I *processi indipendenti* non interagiscono con altri processi durante la loro esecuzione.
- *I processi cooperanti* influenzano o possono essere influenzati da altri processi. Il comportamento dipende anche dall'ambiente esterno.
- Vantaggi della cooperazione:
  - Condivisione dell'informazione
  - Velocità di esecuzione
  - Modularità
  - Convenienza.

Processi Cooperanti

- *I processi cooperanti* possono interagire tramite:
  - Scambio esplicito di dati,
  - Sincronizzazione su un particolare evento.
  - Condivisione dell'informazione.
- I sistemi operativi offrono meccanismi per realizzare queste diverse forme di cooperazione.
- Ad esempio:
  - send e receive
  - semafori
  - monitor
  - chiamata di procedura remota
- Alcuni linguaggi offrono anche meccanismi di cooperazione (es: Java).

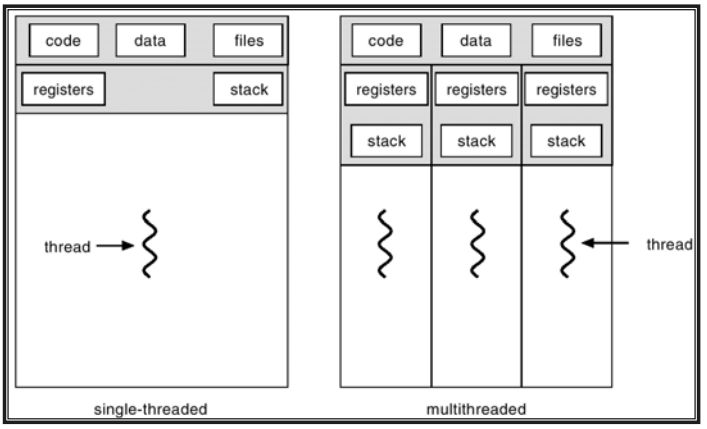
Thread

- Un **thread**, detto anche processo leggero, è una unità di esecuzione che consiste di un program counter, lo stack e un insieme di registi.
- Un thread condivide con altri thread la sezione codice, la sezione dati, e le risorse che servono per la loro esecuzione.
- Un insieme di thread associati prendono il nome di **task**.
- Un **processo** equivale ad un task con un unico thread.
- I thread rendono più efficiente l'esecuzione di attività che condividono lo stesso codice.

Thread

- Il context switch tra thread è molto più veloce.
- Un sistema operativo composto da thread è più efficiente.
- I thread non sono tra loro indipendenti perché condividono codice e dati.
- E' necessario che le operazioni non generino conflitti tra i diversi thread di un task.

Task con thread singoli e multipli



Benefici

- Velocità di risposta
- Condivisione di risorse
- Economia
- Uso di architettura parallele (multiprocessore)

Thread utente

- Generalmente esistono thread di utente (user threads) e thread di sistema (kernel threads)
- Nei thread di utente la gestione è fatta tramite una libreria d thread
- I thread utente sono implementati sopra il kernel.
- Esempi
  - POSIX *Pthreads*
  - Mach *C-threads*
  - Solaris *threads*

Kernel Threads

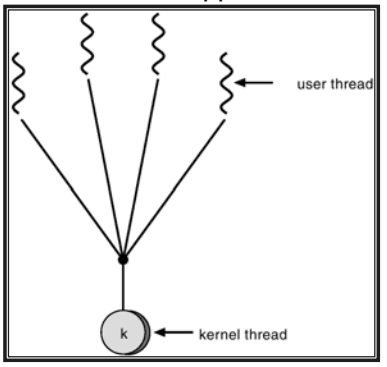
- I thread di sistema sono implementati e gestiti dal kernel.
- La gestione dei thread del kernel è più flessibile.
- Esempi
  - Windows 95/98/NT/2000
  - Solaris
  - Tru64 UNIX
  - Linux

Modelli di Multithreading

- Alcuni S.O. implementano sia thread di sistema che thread di utente.
- Questo genere differenti modelli di gestione dei thread:
  - Molti-ad-Uno
  - Uno-ad-Uno
  - Molti-a-Molti

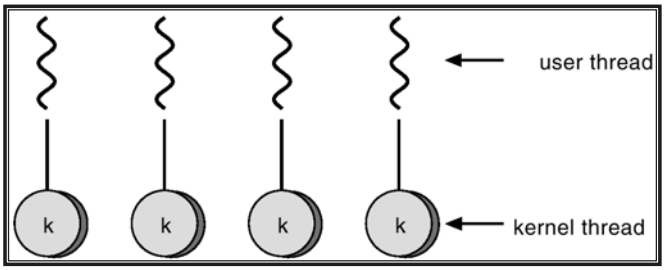
Modello Molti-ad-Uno

- Più user thread sono mappati su un singolo kernel thread.
- Usato nei sistemi che non supportano kernel threads.



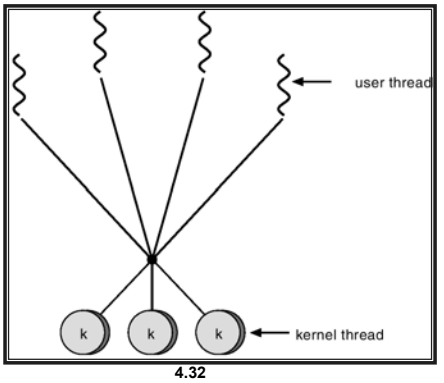
Modello Uno-ad-Uno

- Ogni user thread è associato ad un kernel thread.
- Esempi:
  - Windows 95/98/NT/2000
  - OS/2



Many-to-Many Model

- Molti user thread possono essere associati a diversi kernel threads.
- Permette al sistema operativo di creare un numero sufficiente kernel thread.
- Esempi: Solaris 2 e Windows NT/2000 con l *ThreadFiber* package

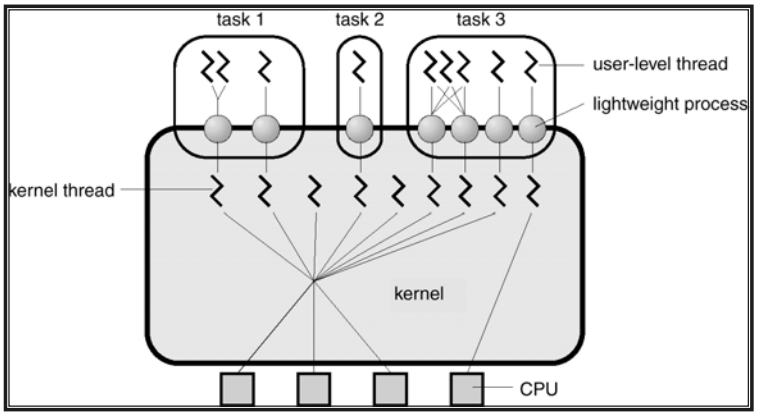


Pthreads

- Pthread è un modello basato sull'API standard POSIX (IEEE 1003.1c)per la creazione e la sincronizzazione di thread.
- Le API specificano il comportamento della libreria dei thread, ma non sono una sua implementazione.
- Esempi di primitive:
  pthread_create(), pthread_join(), pthread_exit()
- Usato in diverse versioni di UNIX.

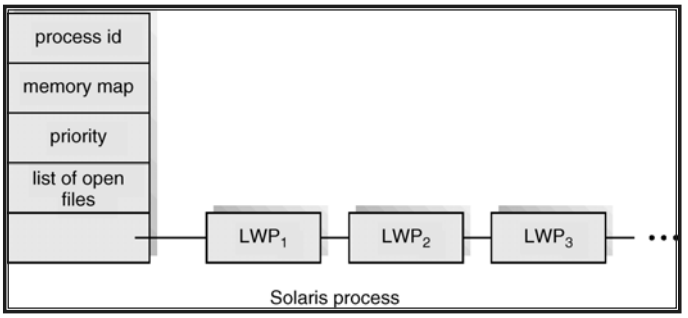
Solaris 2 Threads

Oltre ai thread utente e ai thread di sistema in Solaris esiste un livello intermedio (processi lightweigth) che sono associati a thread utente.



Processo Solaris

Oltre alle normali informazioni un processo Solaris contiene le informazioni sui processi lightweight associati.



Thread di Windows 2000

- Ogni thread utente è associato ad un thread del kernel (modello uno-ad-uno).
- Ogni thread contiene
  - un identificatore del thread
  - un insieme di registri
  - uno stack utente e uno stack kernel
  - un'area di memoria privata del thread
- Con le fibre Windows fornisce anche un modello molti-a-molti.

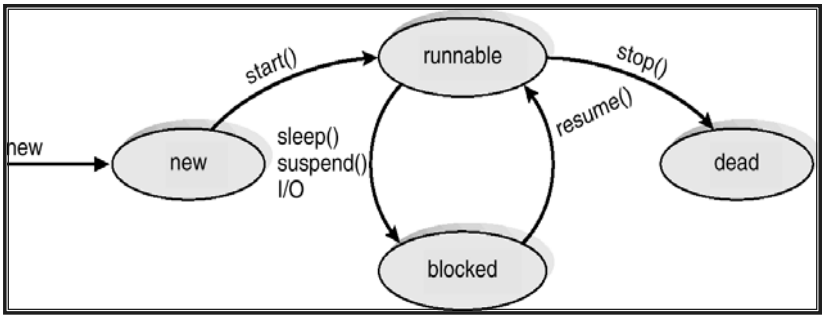
Thread di Linux

- Linux usa il termine *task* per indicare sia *processi* sia *thread*.
- Tuttavia per la creazione di un thread definisce la system call **clone()**.
- **Clone()** permette ad un task figlio di condividere lo spazio di indirizzi del task genitore.
- Tramite un insieme di flag è possibile specificare il livello di condivisione tra i task padre e figlio.

Thread Java

- Java offre la possibilità di usare i thread che possono essere implementati :
  - estendendo la classe Thread,
  - Implementando l'interfaccia Runnable.
- Il metodo **start()** è usato per creare un nuovo thread.
- I thread in Java sono eseguiti e gestiti dalla JVM.

Digramma di stato di un Thread Java



Domande

- Descrivere le variazioni di stato di un processo che effettua molte operazioni di I/O.
- Elencare le differenza principali tra i processi e i thread.
- Discutere le differenze tra i diversi modelli di gestione dei thread.
- Quali sono le differenze principali tra thread utente e therad kernel ?
- Quale è la differenza tra una fork() e una clone() ?