Gestione dei processi

Argomenti

- Concetto di processo
- Scheduling dei processi
- Operazioni sui processi
  - cooperazione fra processi
- Thread

Concetto di processo

- Un sistema operativo esegue programmi di varia natura:
  - Sistemi batch: *job*
  - Sistemi time–sharing: *programmi utente* o *task*
  - Spesso i termini *job* e *processo* sono usati come sinonimi
- Informalmente, il termine processo è utilizzato per indicare un programma in esecuzione
- Più processi possono essere associati allo stesso programma: ciascuno rappresenta l'esecuzione dello stesso codice
  - con dati di ingresso diversi, richieste da utenti diversi, …
- È l'unità di esecuzione all'interno del sistema
  - esecuzione sequenziale nel caso ci sia una sola CPU
  - un S.O. multiprogrammato consente l'esecuzione concorrente di più processi, commutando l'assegnazione della CPU tra di essi

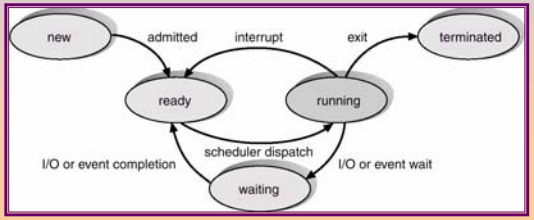
Stati di un processo

- Mentre viene eseguito un processo cambia **stato**
- Due stati principali:

- Se il numero di CPU è minore del numero dei processi (es. sistema monoprocessore), lo stato attivo si suddivide in due stati:
  - pronto (in attesa di una CPU)
  - in esecuzione (utilizza una CPU)

Diagramma di stato di un processo

- **New** (nuovo): Il processo viene creato.
- **Running** (in esecuzione): Le istruzioni vengono eseguite.
- **Waiting** (in attesa): Il processo è in attesa di un evento.
- **Ready** (pronto): Il processo è in attesa di essere assegnato ad un processore.
- **Terminated** (terminato): Il processo ha terminato la propria esecuzione.

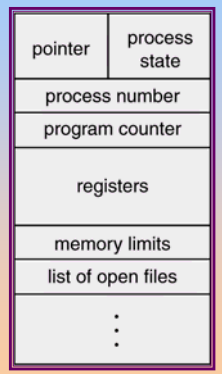


Rappresentazione di un processo

- Un processo è caratterizzato da:
  - Dati e codice nella memoria centrale
  - Program Counter
  - Contenuto degli altri registri della CPU
  - Risorse di I/O e file assegnati
- Le informazioni su un processo sono mantenute in un'apposita struttura dati:

Descrittore del processo (Process Control Block)

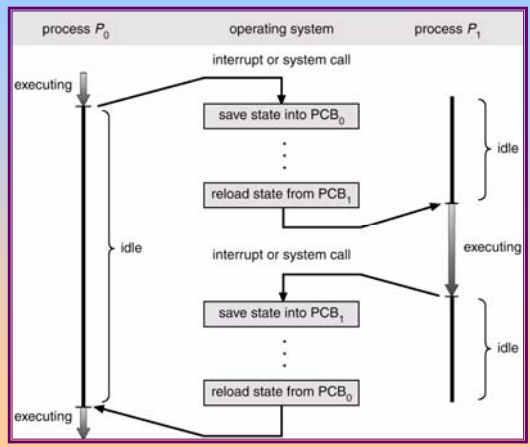
Process Control Block (PCB)



Informazione associata a ciascun processo:

- Identificatore del processo
- Stato del processo
- Program counter
- Registri della CPU
  - accumulatori, indice, stack pointer,…
- Informazioni per lo scheduling della CPU
  - priorità, …
- Informazioni sulla gestione della memoria
  - registri base e limite, tabella pagine/segmenti, …
- Informazioni sullo stato di I/O
  - lista dispositivi/file aperti
- Informazioni di accounting delle risorse
  - account, tempo di uso CPU, …

Commutazione della CPU fra processi



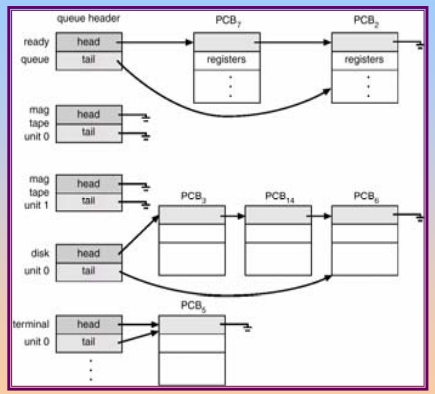
Context switch (cambio di contesto)

- Quando la CPU passa da un processo all'altro, il sistema deve salvare lo stato del vecchio processo e caricare lo stato precedentemente salvato per il nuovo processo
  - Salvataggio del contesto del processo in esecuzione nel suo descrittore (salvataggio stato)
  - Inserimento del descrittore nella coda dei processi bloccati o pronti
  - Selezione di un altro processo dalla coda dei processi pronti e caricamento del suo id nel registro processo in esecuzione (short term scheduling)
  - Caricamento del contesto del nuovo processo nei registri del processore (ripristino stato)
- Il tempo di *context–switch* è un sovraccarico (*overhead*)
  - il sistema non lavora utilmente mentre cambia contesto
- Il tempo di context–switch dipende dal supporto hardware
  - velocità di accesso alla memoria, numero di registri da copiare,…

Code per lo scheduling di processi

- **Coda dei processi**: insieme di tutti i processi presenti nel sistema
- **Ready queue** (Coda dei processi pronti): insieme di tutti i processi in attesa di esecuzione che risiedono in memoria centrale.
- **Code dei dispositivi**: Insieme di processi in attesa per un dispositivo di I/O.
- Consentono di regolare l'assegnazione delle risorse ai processi
  - Dispositivi di I/O, CPU, memoria

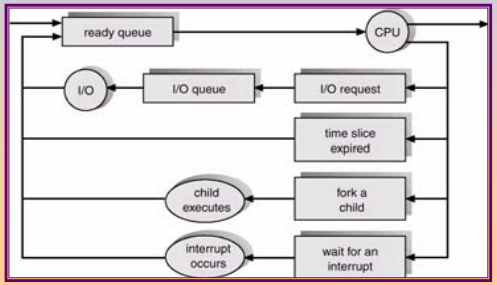
Ready queue e code ai dispositivi di I/O



I processi si "spostano" fra le varie code, in base a politiche di scheduling, implementate da componenti del SO dette scheduler

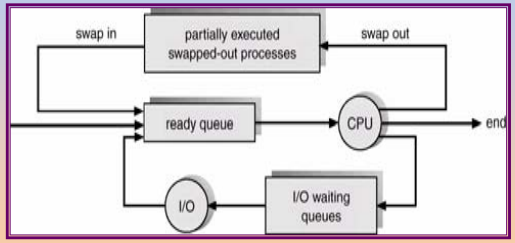
Scheduling dei processi

- Scheduler a **lungo termine** (o scheduler dei job, per sistemi batch):
  - seleziona quali processi devono essere portati dalla memoria di massa alla ready queue (in memoria centrale).
- Scheduler a **breve termine** (o scheduler della CPU) :
  - seleziona quale dei pocessi pronti deve essere eseguito successivamente, e gli assegna la CPU.



Scheduler a medio termine

- Scheduler a **medio termine** (*swapper*):
  - rimuove processi dalla memoria (e dalla contesa per la CPU)



Tipi di scheduler

- Scheduler a breve termine
  - è chiamato molto spesso (≈100 millisecondi)
  - ⇒ deve essere veloce
- Scheduler a lungo termine
  - Utilizzato per processi non interattivi -- tipico dei sistemi batch
  - viene chiamato raramente (secondi, minuti)
    - A regime è chiamato quando un processo termina
  - ⇒ può essere lento
  - controlla il *grado di multiprogrammazione*
  - Obiettivo principale: ottimizza l'uso del sistema (in particolare, CPU e I/O)
    - selezionando un giusto mix di processi I/O bound e CPU bound
- Scheduler a medio termine
  - Permette di assegnare la memoria centrale ad un sottoinsieme dei processi presenti nel sistema
  - Le operazioni di swap sono molto costose

Operazioni sui processi

- Meccanismi per la gestione dei processi (e thread di utente):
  - Creazione
  - Terminazione
  - Interazione tra processi (sincronizzazione e comunicazione)

➔ **Chiamate di sistema**

Creazione di processi

- Il processo padre crea processi figli che, a loro volta, creano altri processi, formando un albero di processi.
- Diverse politiche di gestione dell'interazione padre-figlio
  - Esecuzione:
    - Il padre e i figli vengono eseguiti concorrentemente.
    - Il padre attende la terminazione dei processi figli.
  - Condivisione di risorse:
    - Il padre e il figlio condividono tutte le risorse.
    - I figli condividono un sottoinsieme delle risorse del padre.
    - Il padre e il figlio non condividono risorse.
  - Spazio degli indirizzi
    - Il processo figlio è un duplicato del processo padre.
    - Nel processo figlio è stato caricato un diverso programma.
- Esempio (**UNIX**):
  - la system call **fork** crea un nuovo processo,
  - la **execlp** viene impiegata dopo una **fork** per sostituire lo spazio di memoria del processo originale con un nuovo programma.

Terminazione di processi

- Il processo (e.g.,dopo l'ultima istruzione) chiede al SO di essere terminato mediante una specifica chiamata di sistema (**exit** in UNIX) che compie le seguenti operazioni:
  - Può restituire dati (output) al processo padre (**wait**).
  - Le risorse del processo vengono deallocate dal SO.
- Il padre può terminare l'esecuzione dei processi figli (**abort**) se…
  - Il figlio ha ecceduto nell'uso di alcune risorse.
  - Il compito assegnato al figlio non è più richiesto.
  - Il padre termina.
    - Il SO non consente ad un processo figlio di continuare l'esecuzione se il padre è terminato.
    - Questo fenomeno è detto *terminazione a cascata*

Interazione fra processi

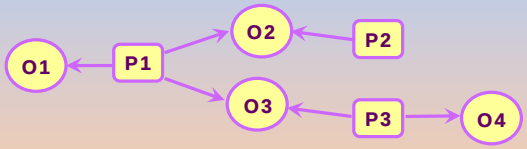
- Processi concorrenti
  - due processi P1 e P2 sono concorrenti se la loro esecuzione si sovrappone nel tempo
- Processi indipendenti:
  - l'esecuzione di P1 non è influenzata da P2, e viceversa
  - Proprietà della riproducibilità
- Processi interagenti:
  - l'esecuzione di P1 è influenzata da P2, e viceversa
  - L'effetto dell'interazione dipende dalla velocità relativa dei processi.
  - Comportamento non riproducibile

Tipi di interazione

- Due tipi di interazione
  - competizione: per l'uso di risorse comuni che non possono essere utilizzate contemporaneamente (mutua esclusione)
  - cooperazione: nell'eseguire un'attività comune mediante scambio di informazioni (comunicazione)
- Motivazioni per la concorrenza:
  - Condivisione di informazioni fra diversi utenti
  - Accelerazione del calcolo (in sistemi multiprocessore)
  - Modularità
  - Convenienza

Modello di cooperazione ad ambiente globale

Il sistema è visto come un insieme di processi e oggetti (risorse)

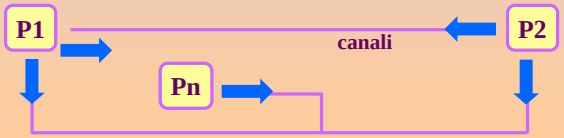


O1, O4 risorse private ▸ ***competizione***

O2, O3 risorse comuni ▸ ***cooperazione***

Modello a scambio di messaggi

- Il sistema è visto come un insieme di processi ciascuno operante in un ambiente locale non accessibile direttamente agli altri processi
  - Ogni interazione tra processi (comunicazione, sincronizzazione) avviene tramite scambi di messaggi
- IPC: Inter-process communication
- Se due processi P e Q vogliono comunicare, devono:
  1. stabilire fra loro un canale di comunicazione;
  2. scambiare messaggi mediante due tipi di operazioni:
     - send(messaggio)
     - receive(messaggio).



Sincronizzazione

- Sia in caso di competizione che di cooperazione è necessario imporre dei vincoli di sincronizzazione per garantire la corretta esecuzione dei processi
- In caso di competizione:
  - un solo processo alla volta può avere accesso alla risorsa comune (sincronizzazione indiretta o implicita)
- In caso di cooperazione:
  - le operazioni con cui i processi cooperano devono seguire una sequenza prefissata (sincronizzazione diretta o esplicita)

Esempio di cooperazione: il paradigma produttore-consumantore

- È un paradigma classico per processi cooperanti.
- Il processo produttore produce informazioni che vengono consumate da un processo consumatore.
- Esempi:
  - Un programma di stampa produce caratteri che verranno consumati dal driver della stampante
  - Il compilatore produce un codice che viene consumato da un assemblatore
- Nel caso di esecuzione concorrente dei due processi è necessario un meccanismo di sincronizzazione
- Possibili soluzioni
  - Uso di un canale di comunicazione e di scambio di messaggi
  - Uso di memoria condivisa (buffer)
    - Buffer limitato: si assume che la dimensione del buffer sia fissata
    - Buffer illimitato: non si ponggono limiti pratici alla dimensione

Soluzione con memoria condivisa e buffer limitato

- Dati condivisi

```
#define BUFFER_SIZE 10
Typedef struct {
   . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- La soluzione consente l'utilizzo di soli BUFFER_SIZE–1 elementi ….

Soluzione con buffer limitato

```
item nextProduced;

while (1) {
        while (((in + 1) % BUFFER_SIZE) == out)
                ; /* do nothing */
        buffer[in] = nextProduced;
        in = (in + 1) % BUFFER_SIZE;
}
```

Processo Produttore

```
item nextConsumed;

while (1) {
        while (in == out)
                ; /* do nothing */
        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
}
```

Processo Consumatore

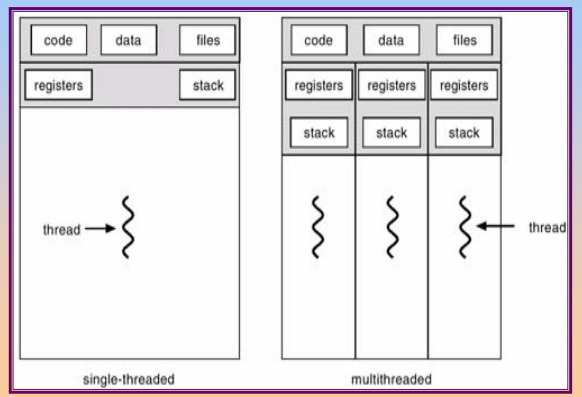
Thread

- Un processo è al tempo stesso:
  - Un elemento cui viene assegnata la CPU
  - Un elemento che possiede risorse (memoria, I/O, …)
- Separazione dei due aspetti:
  - thread: elemento cui viene assegnata la CPU
  - task: elemento che possiede le risorse
- Un ***thread*** (o *lightweight process*, **LWP**) rappresenta un flusso di esecuzione all'interno di un task
  - Multithreading: molteplicità di flussi di esecuzione all'interno di un task
  - Un processo tradizionale, o *heavyweight,* corrisponde ad un task con un solo thread.

Thread

- Informazioni associate ad un thread
  - Program counter
  - Insieme dei registri
  - Spazio dello stack
- Informazioni associate ad un *Task*
  - Segmento di codice
  - Segmento dati
  - Risorse del sistema

Task e thread



Vantaggi del multithreading

- In un task multithread, la cooperazione di più thread permette un minor tempo di esecuzione
  - mentre un thread è blocccato in attesa, un secondo thread nello stesso task può essere in esecuzione.
  - In architetture parallele thread diversi possone essere eseguiti in contemporanea
- Riduzione dei tempi di context-switch rispetto all'uso di task concorrenti
- Esempi di applicazione
  - Esecuzione concorrente di differenti compiti (es,. Browser web)
  - Gestione concorrente di richieste provenienti da utenti diversi (es.. server Web)

Realizzazione dei thread

- A livello utente:
  - La gestione dei thread è affidata alle applicazioni ed il kernel non è conscio della loro presenza
    - Uso di una libreria (thread package)
  - Il S.O. gestisce solo i task
- A livello kernel (es. NT, Linux, OS2):
  - Il S.O. gestisce direttamente i thread
  - Possibilità di utilizzare le potenzialità di un sistema multiprocessore
- Approcci ibridi (es.: Solaris 2).
  - implementano thread sia a livello kernel che a livello utente

Thread a livello utente (ULT)

- **ULT** = *User Level Thread*
- Vantaggi:
  - **Generalità**: gli ULT possono essere eseguiti su qualunque SO senza cambiare il kernel sottostante.
    - La libreria dei thread è un insieme di utilità a livello di applicazione.
  - **Efficienza**: il cambio di contesto fra thread non richiede privilegi in modalità kernel (risparmia il sovraccarico del doppio cambiamento di modalità).
  - **Flessibilità**: lo scheduling può essere diverso per applicazioni diverse.
- Svantaggi:
  - In caso di system call bloccanti, quando un thread esegue una chiamata di sistema, viene bloccato tutto il processo
  - Un'applicazione multithread non può sfruttare il multiprocessing: in un dato istante un solo thread per processo è in esecuzione

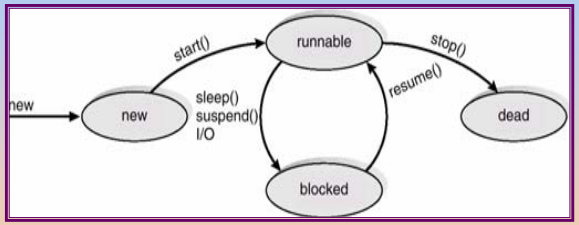
Thread a livello kernel (KLT)

- **KLT** = *Kernel Level Thread*
- Vantaggio:
  - Il kernel effettua lo scheduling a livello di thread:
    Se un thread di un processo è bloccato il kernel può schedulare un altro thread dello stesso processo
- Svantaggio :
  - Il trasferimento del controllo fra thread dello stesso processo richiede il passaggio in modalità kernel: l'aumento di prestazioni è molto meno rilevante rispetto all'approccio ULT.

Thread Java

- La gestione dei thread Java avviene a livello dello stesso linguaggio di programmazione
  - supportata dalla macchina virtuale Java (JVM)
- Tutti i programmi Java comprendono almeno un thread
  - Anche un programma costituito solo dal metodo main viene eseguito come un singolo thread.
  - Java fornisce strumenti che consentono di creare e manipolare thread aggiuntivi nel programma
- Esistono due modi per implementare thread in Java:
  - Definire una sottoclasse della classe Thread
  - Definire una classe che implementa l'interfaccia Runnable. Questa modalità è più flessibile, in quanto consente di definire un thread che è sottoclasse di una classe diversa dalla classe Thread.

Diagramma di stato di un Thread Java

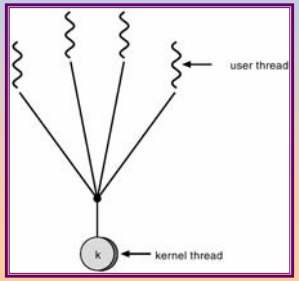


Modelli di Multithreading

- 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.

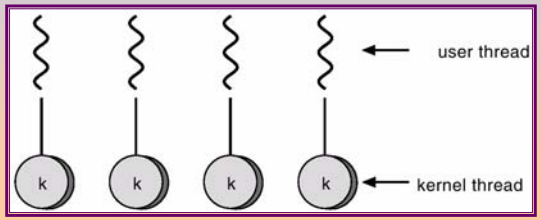
Modello Molti-ad-Uno

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



Modello Uno-a-Uno

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



Modello Molti-a-Molti

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