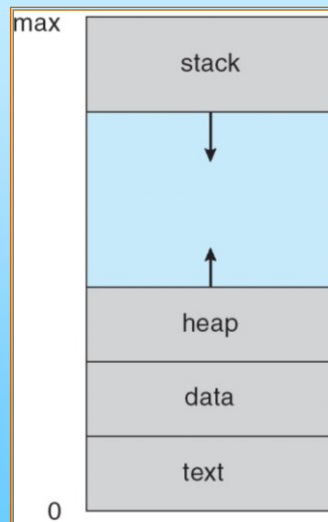


Gestione dei Processi

Processo

- Un sistema operativo esegue una varietà di programmi:
 - Sistemi batch – lavori o jobs
 - Sistemi in time-sharing – programmi utente o tasks
- I termini *job* e *process* sono usati quasi intercambiabilmente
- **Processo** – un programma in esecuzione; l'esecuzione del programma deve procedere in modo sequenziale
- Un processo include:
 - program counter
 - stack
 - sezione dati

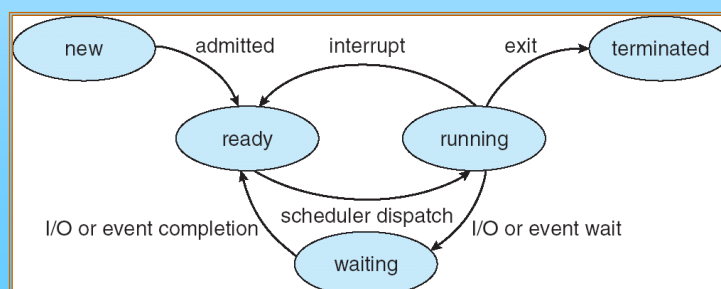
Un processo in memoria



Sistemi Operativi A.A 2016/2017

Stato di un processo

- Quando un processo è in esecuzione cambia il proprio *stato*
 - **new**: il processo è stato creato
 - **ready**: il processo è in attesa di essere assegnato alla CPU
 - **running**: istruzioni del processo sono in esecuzione
 - **waiting**: il processo è in attesa di un evento
 - **terminated**: il processo ha finito l'esecuzione

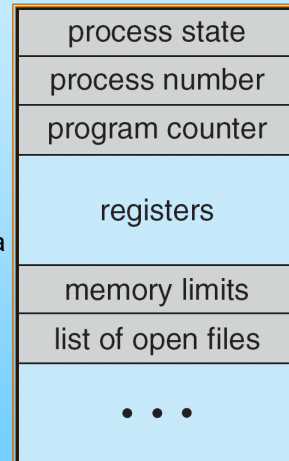


Sistemi Operativi A.A 2016/2017

Process Control Block (PCB)

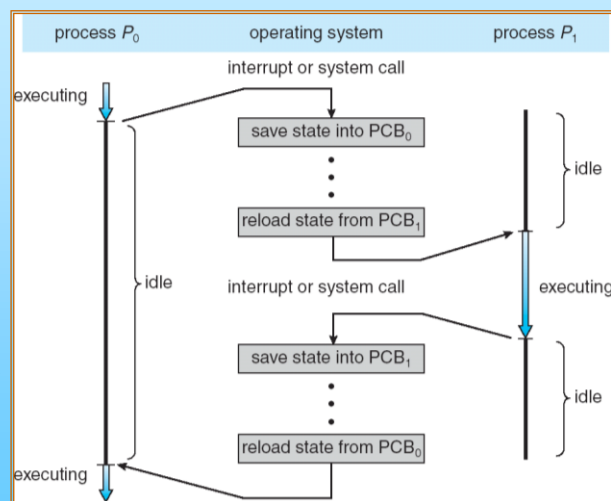
Informazioni associate al processo

- Stato del processo
- Program counter
- Contenuto registri CPU
- Informazioni per lo scheduling della CPU
- Informazioni per la gestione della memoria
- Informazioni di accounting
- Informazioni sullo stato dell'I/O



Sistemi Operativi A.A 2016/2017

Passaggio CPU da Processo a Processo



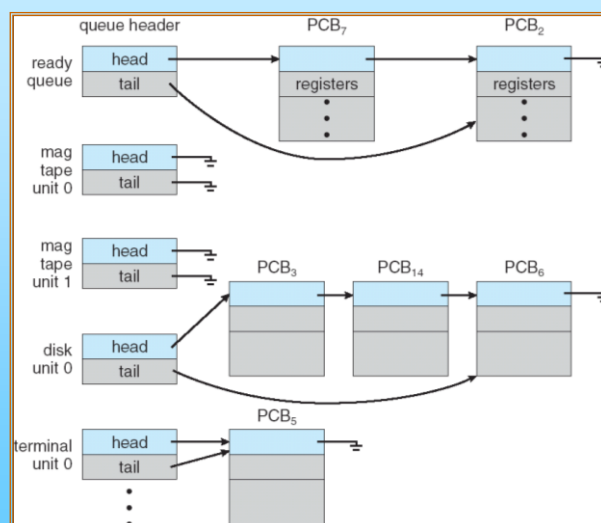
Sistemi Operativi A.A 2016/2017

Code per lo scheduling dei processi

- **Job queue** – insieme di tutti i processi del sistema
- **Ready queue** – insieme di tutti i processi che risiedono in memoria pronti e in attesa di essere eseguiti
- **Device queues** – insieme di processi in attesa di un I/O su device
- Processi migrano tra le varie code

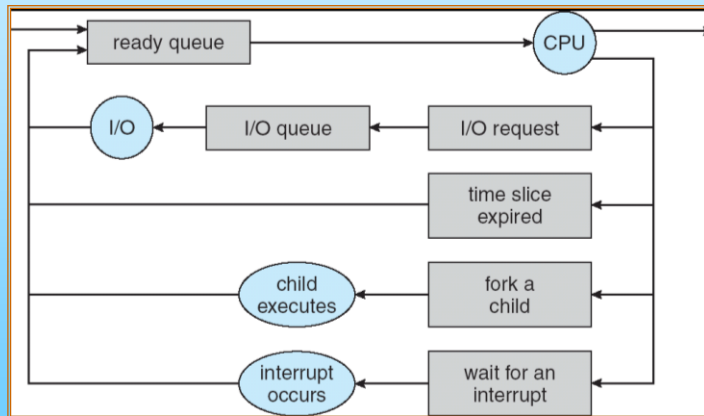
Sistemi Operativi A.A. 2016/2017

Ready Queue e varie I/O Device Queues



Sistemi Operativi A.A. 2016/2017

Rappresentazione dello scheduling dei processi



Sistemi Operativi A.A 2016/2017

Schedulers

- **Scheduler a lungo termine** (o job scheduler) – in un sistema batch seleziona quali processi tra quelli in attesa devono essere portati nella ready queue
- **Scheduler a breve termine** (o CPU scheduler) – seleziona quale processo deve essere eseguito e allocato sulla CPU

Sistemi Operativi A.A 2016/2017

Scheduling a medio termine

Serve ad eliminare dalla memoria processi parzialmente eseguiti riducendo il grado di multiprogrammazione

```
graph LR; subgraph Box [ ]; direction LR; PE[partially executed swapped-out processes]; RQ[ready queue]; CPU((CPU)); IOWQ[I/O waiting queues]; IO((I/O)); CPU --> end((end)); CPU --> IOWQ; IOWQ --> IO; IO --> RQ; RQ --> CPU; PE -- "swap in" --> RQ; RQ -- "swap out" --> PE; end
```

The diagram illustrates the flow of processes in a medium-term scheduling system. It is enclosed in a rectangular box. At the top, a gray box labeled "partially executed swapped-out processes" is connected to a "ready queue" (gray box) below it. An arrow labeled "swap in" points from the top box to the ready queue, and an arrow labeled "swap out" points from the ready queue back to the top box. The "ready queue" has three arrows pointing into it from the left and one arrow pointing to a "CPU" (blue circle) on the right. From the "CPU", an arrow points to the right labeled "end", and another arrow points down to a gray box labeled "I/O waiting queues". From this box, an arrow points left to a blue circle labeled "I/O". From the "I/O" circle, an arrow points up to the "ready queue".

Sistemi Operativi A.A 2016/2017

Schedulers (Continua)

- Scheduler a breve termine è invocato molto frequentemente (millisecondi) \Rightarrow (deve essere veloce)
- Scheduler a lungo termine è invocato di rado (secondi, minuti) \Rightarrow (può essere lento)
- Lo scheduler di lungo termine controlla il *grado di multiprogrammazione*
- I processi possono essere:
 - **A prevalenza di I/O (I/O-bound)** – passa più tempo a fare I/O che computazioni
 - **A prevalenza di CPU (CPU-bound)** – spende più tempo a fare computazioni

Sistemi Operativi A.A 2016/2017

Cambio di contesto

- Quando la CPU viene assegnata ad un altro processo, il sistema deve salvare lo stato del processo e caricare lo stato del nuovo processo (context-switch)
- Il cambio di contesto comporta un calo delle prestazioni; il sistema non fa nessun lavoro utile alla computazione mentre effettua il cambio di contesto
- Il tempo impiegato dipende dal supporto hardware

Sistemi Operativi A.A. 2016/2017

Creazione dei processi

- *Processo genitore* (parent) crea *processi figli* (children), che a loro volta creano altri processi, formando un albero di processi (process tree)
- Condivisione delle risorse (es. file aperti), si hanno 3 possibilità:
 - Processo genitore e processi figli condividono tutte le risorse
 - I processi figli condividono con i genitori alcune risorse
 - Processi genitori e figli non condividono nessuna risorsa
- Esecuzione, si hanno 2 possibilità:
 - Processo genitore e figli sono eseguiti in concorrenza
 - Processo genitore aspetta fino alla terminazione dei processi figli

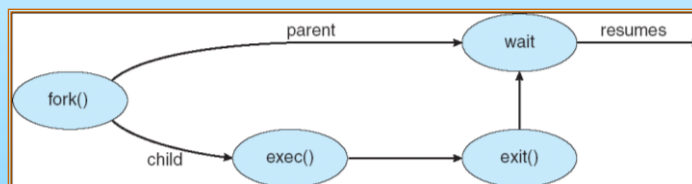
Sistemi Operativi A.A. 2016/2017

Creazione dei processi (Continua)

- Spazio indirizzi (memoria), due possibilità:
 - Processo figlio ha un duplicato del processo padre
 - Processo figlio creato con un programma caricato dentro
- UNIX:
 - La chiamata di sistema **fork** crea un nuovo processo duplicato del padre, l'unica cosa che cambia tra padre e figlio è il valore ritornato da fork: padre ritorna id del processo figlio nel figlio ritorna 0
 - La chiamata di sistema **exec** viene usata dopo una **fork** per sostituire lo spazio di memoria con quello di un nuovo programma

Sistemi Operativi A.A 2016/2017

Creazione dei processi UNIX



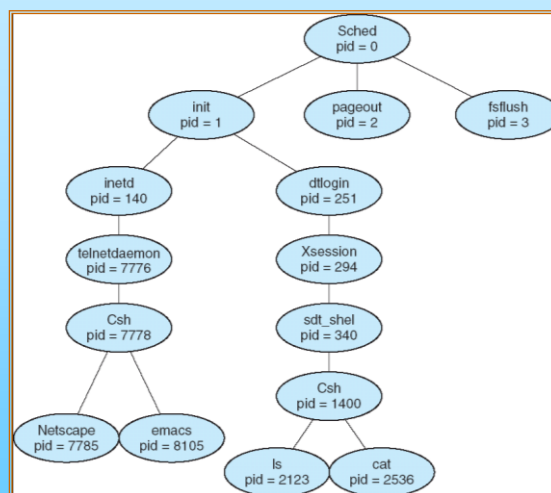
Sistemi Operativi A.A 2016/2017

Programma C che crea un processo separato

```
#include <stdio.h>
#include <unistd.h>
#include <wait.h>
int main()
{
    pid_t pid; /* process id */
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait (NULL);
        printf ("Child Complete");
        exit(0);
    }
}
```

Sistemi Operativi A.A 2016/2017

Un albero di processi su Solaris



Sistemi Operativi A.A 2016/2017

Terminazione dei processi

- Un processo esegue la sua ultima istruzione e chiede al sistema di terminarsi (**exit**)
 - Fornisce il valore risultato al processo padre che lo riceve dalla `wait`
 - Le risorse del processo sono rilasciate dal sistema operativo
- Un processo genitore può terminare l'esecuzione dei processi figli (**abort**)
 - Se il compito assegnato al figlio non è più richiesto
 - Se il processo padre termina:
 - ▶ Alcuni sistemi operativi non permettono ai figli di continuare se il processo padre termina
 - Tutti i figli sono terminati – *terminazione a cascata*
 - ▶ Altri sistemi operativi assegnano i figli al processo iniziale
- *Processo Zombie*: un processo figlio terminato ma che ha ancora pid e PCB per poter dare risultato al processo padre tramite `wait()`

Sistemi Operativi A.A. 2016/2017

Unix/Linux

- Altre funzioni disponibili:
 - **getpid()** – ritorna l'identificatore del processo attualmente in esecuzione
 - **getppid()** – ritorna l'identificatore del processo padre
 - **execv(char* path, char*[] args)**
 - **execve(char* path, char*[] args, char*[] env);**

Sistemi Operativi A.A. 2016/2017

Gestione processi in Java

- Il metodo `exec` della classe **RunTime** permette di eseguire un processo. Esempio (su Windows):

```
Process p = Runtime.getRuntime().exec("notepad.exe");  
System.out.println("attendo che finisca...");  
p.waitFor(); // attende che il processo generato finisca  
System.out.println("finito!");
```

Sistemi Operativi A.A 2016/2017

Gestione processi in Java

- Tramite l'oggetto `Process` si può accedere agli stream di input/output del processo in esecuzione
 - `p.getInputStream()` //stream per accedere all'**output** del processo
 - `p.getOutputStream()` //stream per fornire **input** al processo
 - `p.getErrorStream()` //stream per accedere all'`stderr` del processo
- Inoltre si può terminare il processo tramite metodo `destroy`
 - `p.destroy()`

Sistemi Operativi A.A 2016/2017

Esempio

```
class TestProcess1 {  
    static public void main(String[] args) {  
        try {  
            Process p = Runtime.getRuntime().exec("notepad.exe");  
            System.out.println("attendo 5 sec...");  
            Thread.sleep(5*1000); //aspetta 5 secondi  
            p.destroy();  
        } catch(IOException e) {  
            e.printStackTrace();  
        } catch(InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Sistemi Operativi A.A 2016/2017

Esempio

```
class TestProcess2 {  
    static public void main(String[] args) {  
        try {  
            Process p = Runtime.getRuntime().exec("cmd /c dir");  
            InputStream is = p.getInputStream(); //stream per prendere output comando dir  
            BufferedReader br = new BufferedReader(new InputStreamReader(is,"Cp850"));  
            String line;  
            while((line = br.readLine()) != null)  
                System.out.println("> "+line);  
            p.waitFor();  
        } catch (IOException | InterruptedException ex) { ex.printStackTrace(); }  
    }  
}
```

Esegue comando **dir**
da riga di comando (cmd.exe)

Stampa le righe prodotte
in uscita dal comando **dir**

Codifica caratteri
accentati usata da
windows

Sistemi Operativi A.A 2016/2017

Processi Cooperativi

- **Processo indipendente** un processo non viene influenzato e non influenza l'esecuzione di altri processi
- **Processo cooperativo** un processo che può influenzare e può essere influenzato da altri processi
- Vantaggi di processi cooperativi
 - Condivisione di informazioni
 - Velocizzazione delle elaborazioni
 - Modularità
 - Convenienza

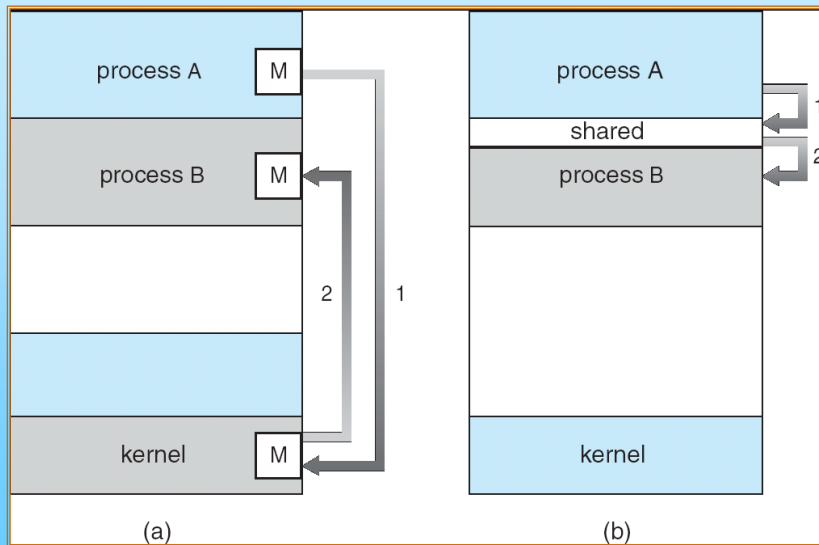
Sistemi Operativi A.A 2016/2017

InterProcess Communication (IPC)

- Meccanismi usati dai processi per comunicare e sincronizzare le loro azioni
- Due modelli principali per lo scambio di informazioni tra processi:
 - **Memoria condivisa** – viene creata una zona di memoria condivisa tra due o più processi
 - **Scambio di messaggi**
- I moderni sistemi operativi spesso forniscono entrambe le modalità.
 - Memoria condivisa, più efficiente, usata per scambiare grandi quantità di dati
 - Scambio di messaggi, meno efficienti, utili per sincronizzare processi

Sistemi Operativi A.A 2016/2017

Modelli di comunicazione



Sistemi Operativi A.A. 2016/2017

InterProcess Communication (IPC)

- IPC fornisce due operazioni di base:
 - **send(messaggio)** – messaggio a grandezza fissa o variabile
 - **receive(messaggio)**
- Se i processi P e Q vogliono comunicare, devono:
 - Stabilire un *canale di comunicazione* tra loro
 - Scambiare messaggi tramite send/receive
- Implementazione del canale di comunicazione:
 - Livello fisico (es. memoria condivisa, hardware bus)
 - Livello logico (es. proprietà logiche)

Sistemi Operativi A.A. 2016/2017

Comunicazione diretta

- I processi devono nominarsi esplicitamente:
 - **send** (P , $message$) – invia messaggio al processo P
 - **receive**(Q , $message$) – riceve un messaggio dal processo Q
- Proprietà logiche del canale di comunicazione
 - Canale stabilito automaticamente
 - Un collegamento viene stabilito tra due processi comunicanti
 - Tra due coppie di processi esiste esattamente un collegamento
 - Il collegamento può essere unidirezionale, ma di solito è bidirezionale

Sistemi Operativi A.A. 2016/2017

Comunicazione indiretta

- Messaggi sono diretti e ricevuti da una porta o mailbox
 - Una porta ha un id univoco
 - I processi possono comunicare solo se condividono una porta
- Proprietà del collegamento
 - Collegamento stabilito solo se i processi condividono una porta in comune
 - Un collegamento può essere associato a più processi
 - Ogni coppia di processi può condividere molti collegamenti
 - Collegamento può essere unidirezionale o bidirezionale

Sistemi Operativi A.A. 2016/2017

Comunicazione indiretta

■ Operazioni

- Creare una nuova porta
- Inviare e ricevere messaggi tramite una porta
- Distruggere una porta

■ Le primitive definite sono del tipo:

send($A, message$) – invia un messaggio alla porta A

receive($A, message$) – riceve un messaggio dalla porta A

Sistemi Operativi A.A. 2016/2017

Comunicazione indiretta

■ Condivisione di una porta

- P_1 , P_2 , e P_3 condividono la porta A
- P_1 , invia; P_2 and P_3 ricevono
- Chi prende il messaggio?

■ Possibili soluzioni

- Permettere che una porta possa essere associata al più a due processi
- Permettere che solo un processo alla volta possa eseguire la receive su una porta
- Permettere al sistema di decidere arbitrariamente a quale processo inviare il dato, al processo inviante viene notificato quale processo ha ricevuto il messaggio.

Sistemi Operativi A.A. 2016/2017

Sincronizzazione

- L'invio/ricezione di messaggi può essere bloccante o non bloccante
- **Bloccante** è considerato **sincrono**
 - **Invio bloccante:** il processo viene bloccato fino a che il messaggio non viene ricevuto
 - **Ricezione bloccante:** il processo viene bloccato fino a che un messaggio è disponibile
- **Non bloccante** è considerato **asincrono**
 - **Invio non bloccante:** il processo invia il messaggio e continua
 - **Ricezione non bloccante:** il processo riceve un messaggio valido o un messaggio nullo

Sistemi Operativi A.A. 2016/2017

Code di messaggi

- Coda di messaggi associata al collegamento:
 1. **Capacità zero** – 0 messaggi
Processo inviante deve aspettare il ricevente (rendezvous)
 2. **Capacità limitata** – lunghezza finita di n messaggi
Processo inviante deve aspettare se la coda è piena
 3. **Capacità illimitata** – lunghezza infinita
Processo inviante non aspetta mai

Sistemi Operativi A.A. 2016/2017