

Sistemi Operativi

Docente: Ugo Erra
ugoerr+so@dia.unisa.it



3° LEZIONE PROCESSI

*CORSO DI LAUREA TRIENNALE IN INFORMATICA
UNIVERSITA' DEGLI STUDI DELLA BASILICATA*



Sommario della lezione



- **Concetto di processo**
 - Stato di un processo
- **Scheduling dei processi**
 - Cambio di contesto
- **Operazioni sui processi**
- **Comunicazioni tra processi**
 - Memoria condivisa
 - Scambio di messaggi

Definizione di processo

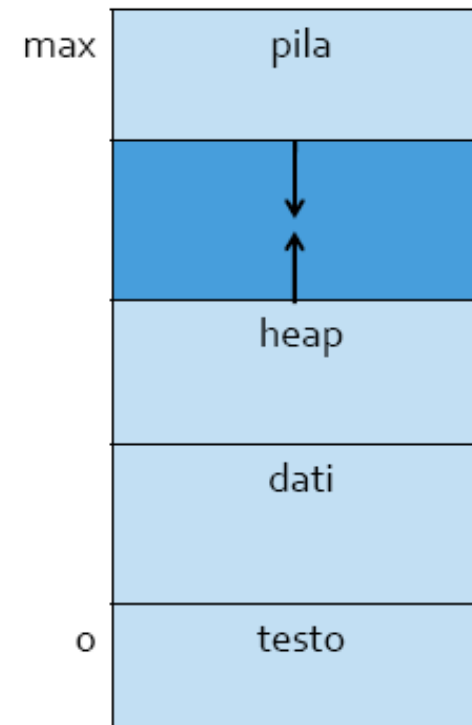


- Un Sistema Operativo può eseguire diversi tipi di programmi:
 - Un sistema a lotti (*batch*) esegue lavori (*job*)
 - Un sistema a partizione di tempo esegue **programmi utenti** o **task**
- Normalmente si utilizzano i termini *job* o *processo* in maniera intercambiabile
- Un processo è un programma in esecuzione
 - L'esecuzione avviene sempre in modo sequenziale
- Un processo include
 - Un contatore di programma
 - Una pila (*stack*)
 - Sezione di dati

Un processo in memoria



- Un processo è composto da:
 - Una *pila* in cui i dati vengono gestite in modalità LIFO (Last In First Out) per tenere variabili globali, indirizzi di ritorno
 - Un *heap* ovvero una memoria allocata dinamicamente
 - Una sezione *dati* per contenere le variabili globali
 - Il *testo* che rappresenta il codice

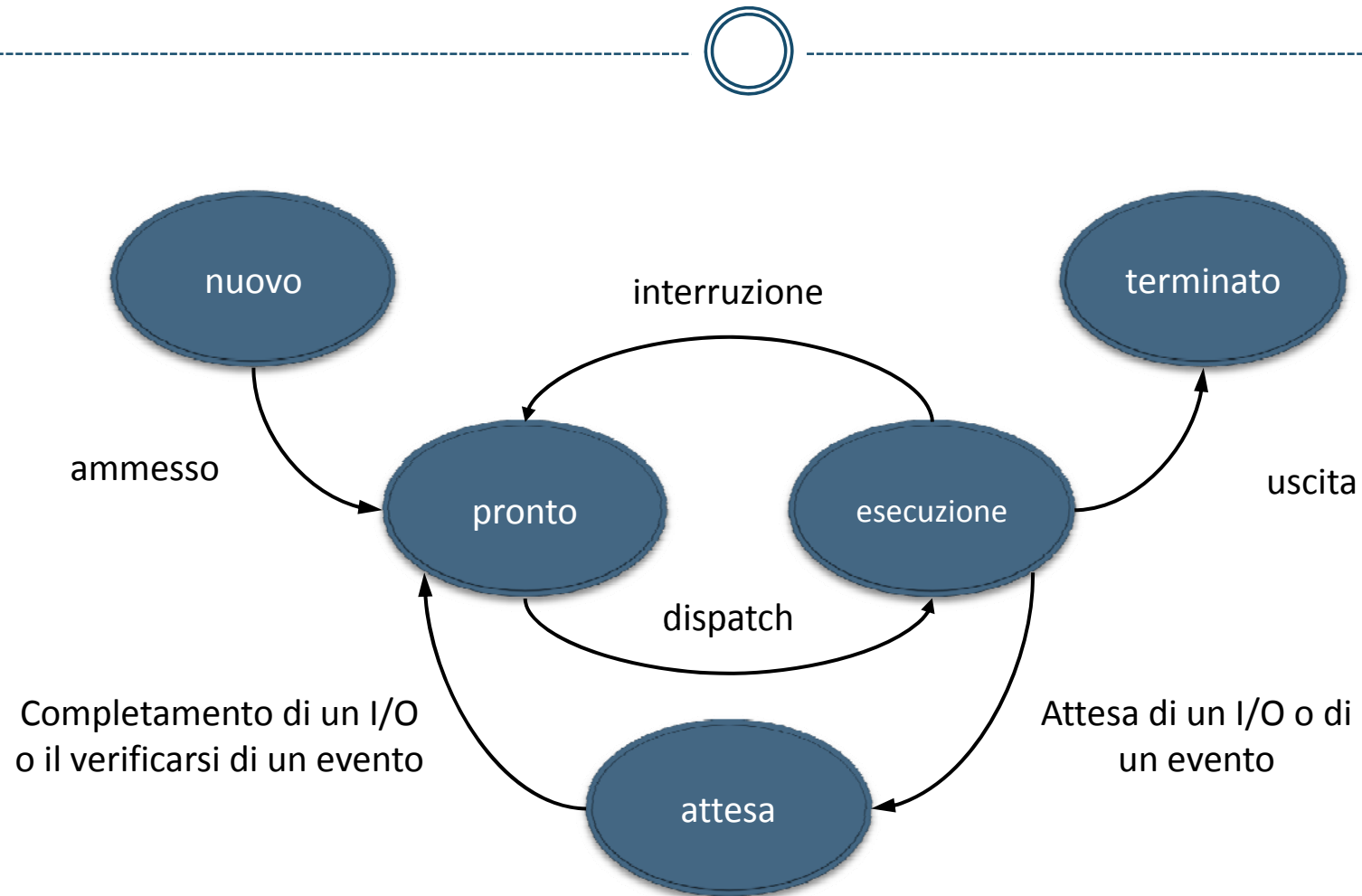


Stato di un processo



- Un processo durante la sua esecuzione può trovarsi in
- diversi stati:
 - **Nuovo:** Il processo è creato
 - **Esecuzione:** Le istruzioni di un processo sono eseguite
 - **Attesa:** Il processo attende che si verifichi qualche evento
 - **Pronto:** Il processo attende che sia assegnato ad un processore
 - **Terminato:** Il processo ha terminato la sua esecuzione

Diagramma di transizione degli stati di un processo

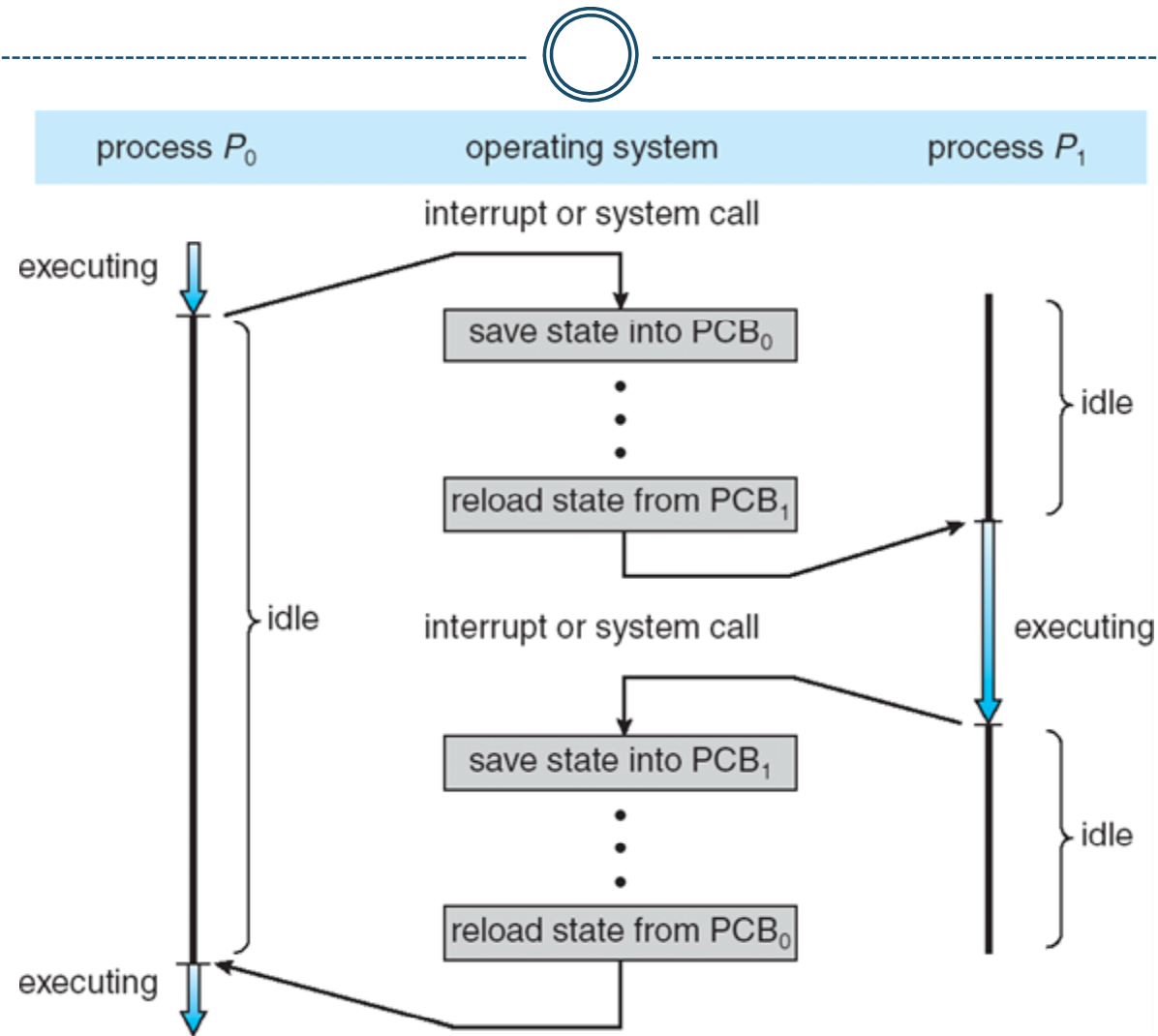


Blocco di controllo dei processi



- Ad ogni processo è associato il **blocco di controllo di un processo** (*process control block, PCB*) che mantiene le seguenti informazioni:
 - Stato del processo
 - Contatore di programma
 - Registri di CPU
 - Informazioni sullo scheduling della CPU
 - Informazioni sulla gestione della memoria
 - Informazioni di contabilizzazione delle risorse
 - ✦ Tempo di utilizzo della CPU, numeri di processi, limiti di tempo
 - Informazioni sullo stato dell'I/O
 - ✦ File aperti, dispositivi I/O assegnati

Commutazione della CPU tra processi



Code di scheduling



- In un sistema multiprogrammato più processi possono essere eseguiti contemporaneamente per massimizzare l'utilizzo della CPU
- Lo scheduler dei processi ha il compito di selezionare il processo da mandare in esecuzione
- L'insieme di tutti i processi del sistema è gestito attraverso una **coda di processi** (*job queue*)
- L'insieme di tutti i processi che si trovano in memoria e pronti per essere mandati in esecuzione si trovano nella **coda dei processi pronti** (*ready queue*)
- L'insieme dei processi in attesa dell'I/O da un dispositivo sono inseriti nella **coda del dispositivo** (*device queue*)
- Durante la sua vita un processo migra nelle diverse code

Code dei processi (CPU, I/O)

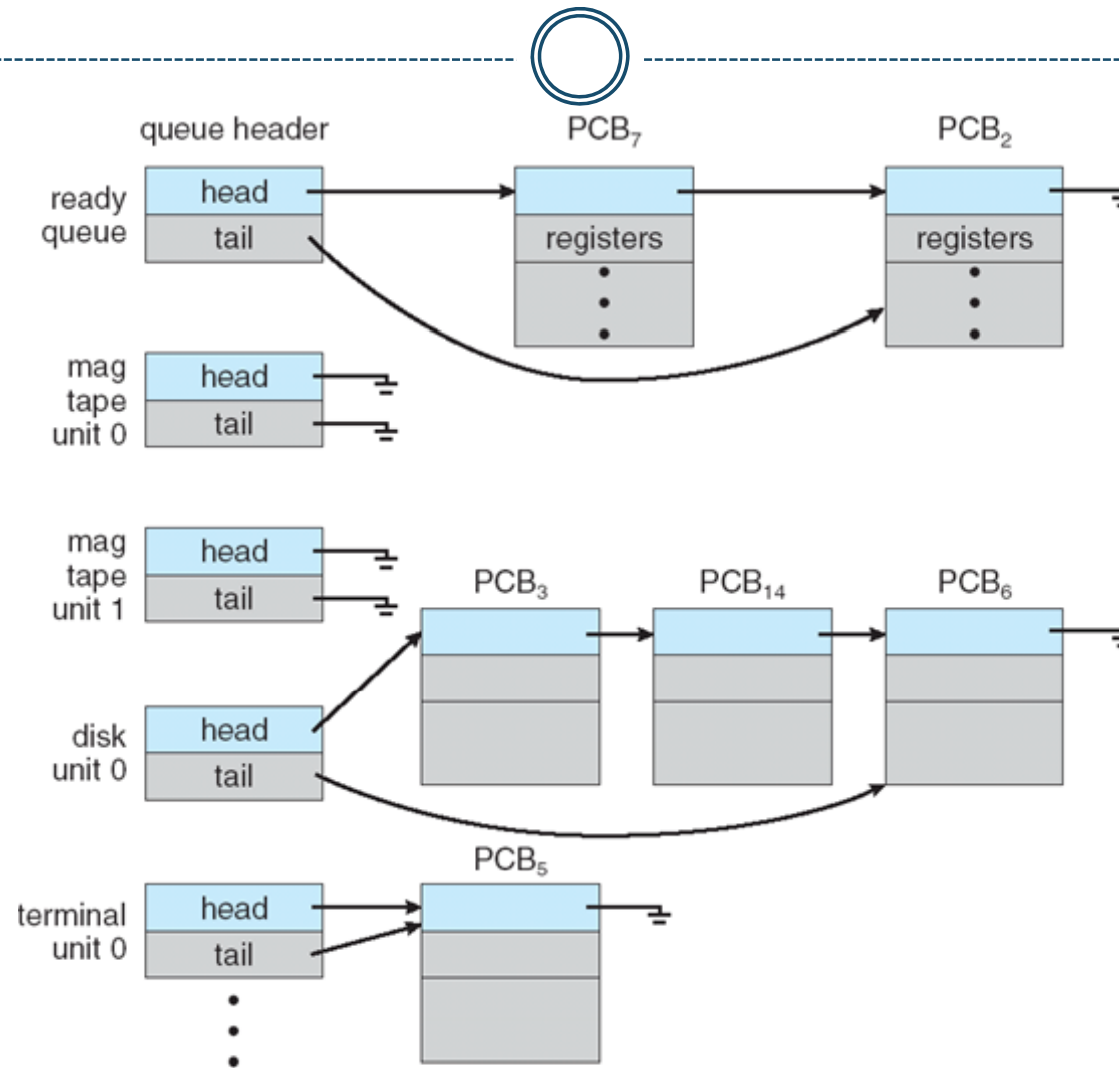
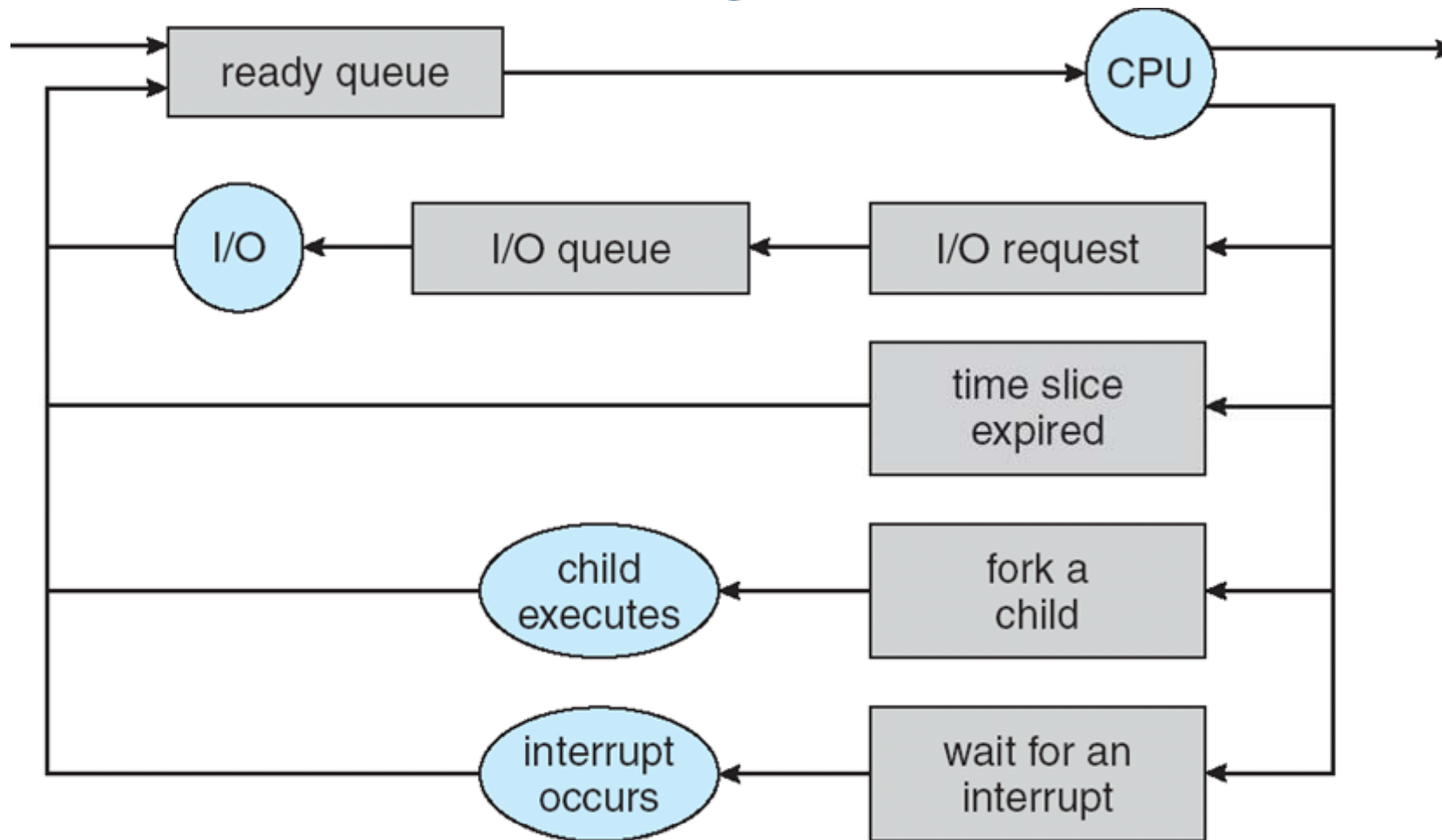


Diagramma di accodamento



Scheduler - 1



- Gli scheduler si dividono generalmente in due tipi:
 - **Scheduler a lungo termine** (*job scheduler*)
 - ✦ Seleziona quale processo deve essere prelevato ed inserito nella coda dei processi pronti
 - **Scheduler della CPU** (*CPU scheduler*)
 - ✦ Seleziona quale è il prossimo processo da mandare in esecuzione sulla CPU
- I sistemi operativi come Windows o Linux non hanno uno scheduler di lungo termine, normalmente la scelta di quali processi tenere in memoria è affidata alla sensibilità dell'utente

Scheduler - 2

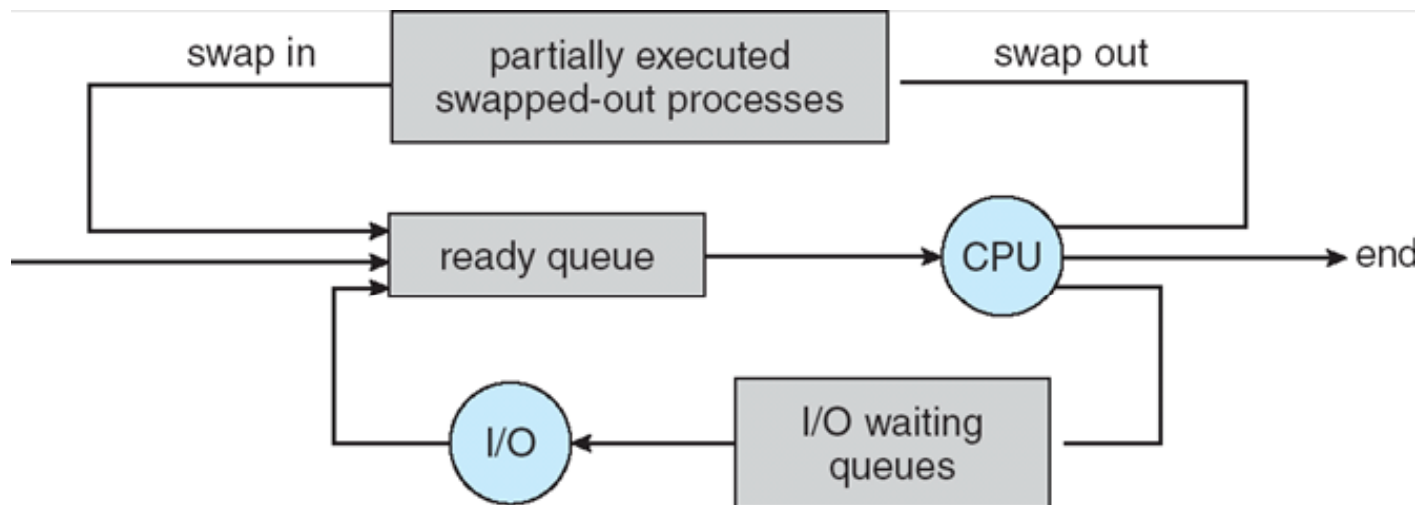


- Lo scheduler della CPU è invocato molto più frequentemente, ad esempio una volta ogni 100 millisecondi
 - Deve essere molto veloce altrimenti si sprecono cicli di CPU
- Lo scheduler a lungo termine è invocato meno frequentemente, ad esempio ogni secondo o minuto
 - Controlla il **grado di multiprogrammazione** ovvero il numero di processi presenti in memoria
- I processi possono essere descritti come:
 - **Processi I/O bound**
 - ✦ Il tempo è speso in prevalenza in attesa di I/O da un dispositivo
 - **Processi CPU bound**
 - ✦ Il tempo è speso in prevalenza in computazione e quindi sulla CPU

Scheduler a medio termine



- Uno **scheduler a medio termine** è impiegato per eliminare momentaneamente processi dalla memoria per ridurre il grado di multiprogrammazione
- Il meccanismo è chiamato avvicendamento dei processi in memoria (*swapping*)
 - Il processo viene rimosso dalla memoria per poi essere ricaricato in memoria



Cambio di contesto



- Il **cambio di contesto** (*context switch*) è il momento in cui la CPU passa all'esecuzione di un altro processo
- Il sistema deve salvare lo stato del processo corrente e caricare lo stato del prossimo processo
- Il cambio di contesto deve avvenire velocemente perché è una pura fase di overhead ovvero il sistema non fa nulla di utile in quel momento
- L'hardware gioca un ruolo importante per determinare il tempo necessario

Operazioni sui processi



- Nei sistemi concorrenti si devono prevedere dei meccanismi per
 - La creazione di un processo
 - La terminazione di un processo

Creazione di un processo

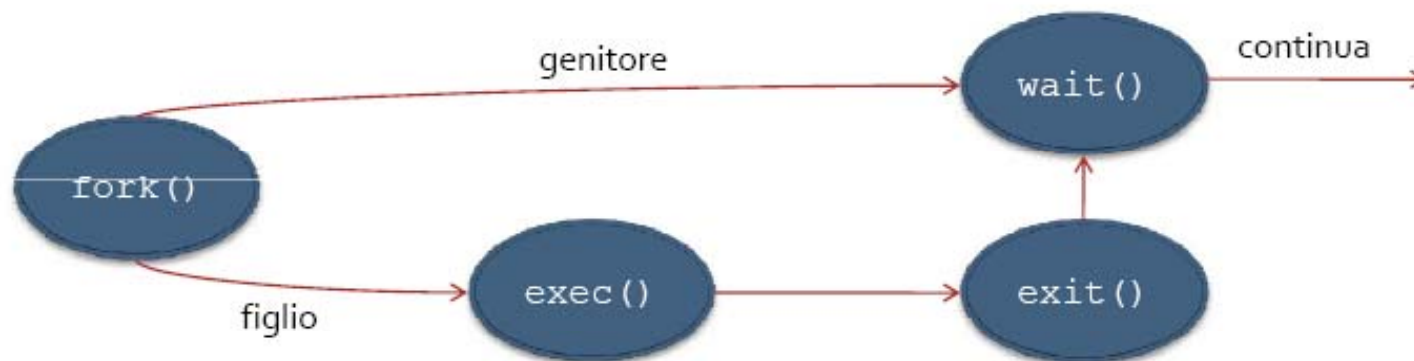


- La creazione dei processi inizia da un processo padre che crea dei processi figli che a loro volta creano altri processi formando un albero di processi
- Il processo padre ed il processo figlio possono condividere le risorse nei seguenti modi:
 - Padre e figlio condividono tutte le risorse
 - Il figlio condivide solo un sottoinsieme delle risorse
 - Padre e figlio non condividono risorse
- L'esecuzione del processo padre e del processo figlio può procedere nei seguenti modi:
 - Padre e figlio sono eseguiti in maniera concorrente
 - Il padre attende la fine del figlio
- Lo spazio di indirizzamento può essere gestito nel seguente modo:
 - Il figlio è un duplicato esatto del padre
 - Il figlio carica un programma nel suo spazio di memoria

Generazione di un processo sotto UNIX



- In un sistema UNIX ogni processo è identificato attraverso un intero univoco (pid)
- Un nuovo processo è creato invocando la chiamata di sistema `fork()`
 - Il processo figlio è una copia identica del padre
- Entrambi i processi continuano l'esecuzione ma la `fork()` riporterà zero al processo padre ed un nuovo pid per il figlio
 - Il figlio può utilizzare la chiamata di sistema `exec()` per caricare un programma
 - Il padre attende la fine del figlio

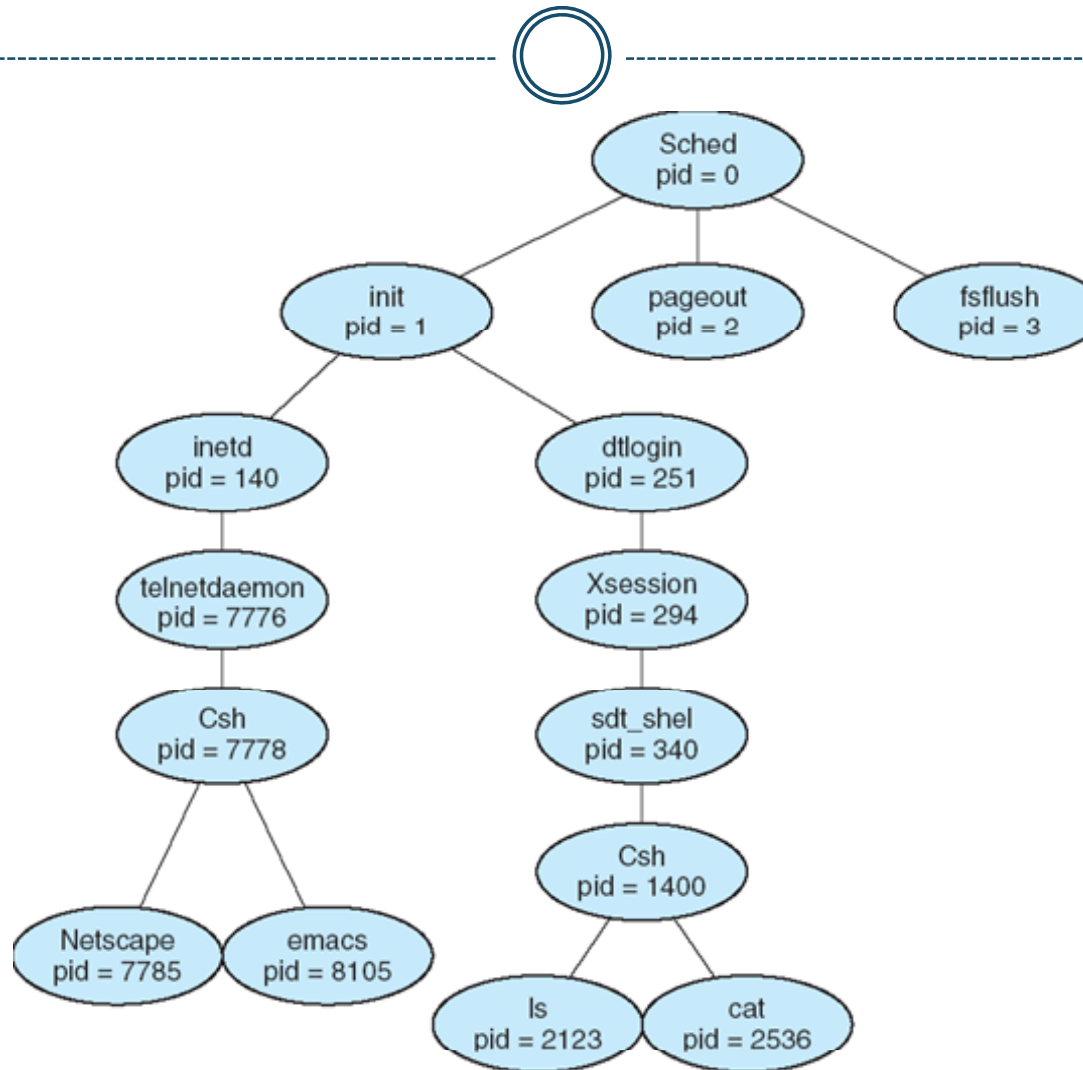


Creazione di un processo in C in UNIX



```
int main()
{
    pid_t pid;
    /* 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);
    }
}
```

Albero di creazione dei processi Solaris



Terminazione di un processo



- Un processo termina quando termina l'esecuzione della sua ultima istruzione ed il processo invoca la chiamata `exit()`
 - Il processo figlio può riportare alcuni dati di output al padre
 - Le risorse del processo sono deallocate dal sistema operativo
- Un processo padre può terminare l'esecuzione di processo figlio nei seguenti modi:
 - Il figlio ha ecceduto nell'uso delle risorse allocate
 - Il compito assegnato al figlio non è più richiesto
 - Il processo padre termina e di conseguenza anche il figlio viene terminato
 - ✦ In alcuni sistemi operativi questa operazione di terminazione a cascata in cui tutti i figli terminano è chiamata **terminazione a cascata**

Comunicazione tra processi



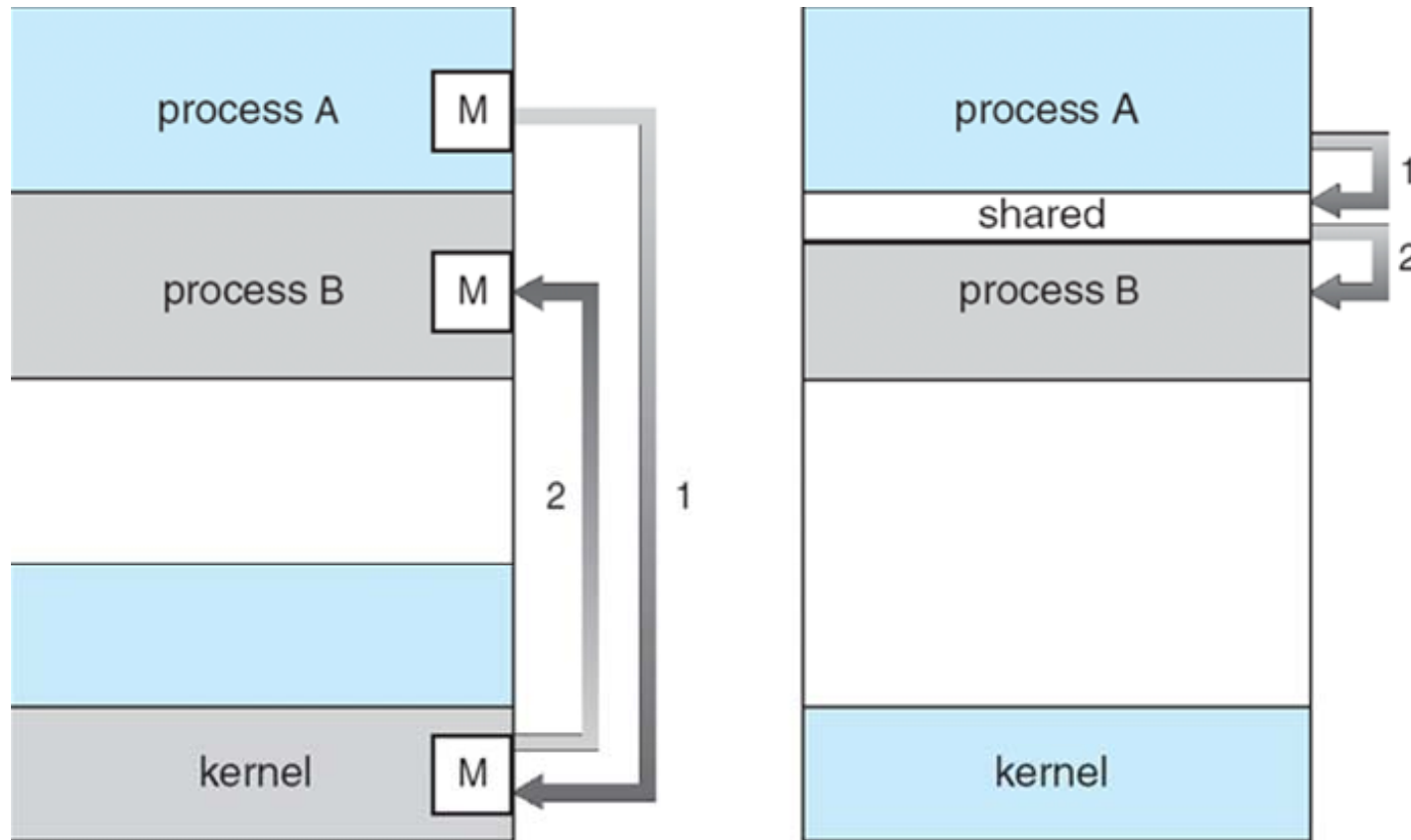
- I processi di un Sistema Operativo possono essere
 - **Indipendenti**
 - ✦ Ogni processo non può influire ne subire l'influenza di un altro processo
 - **Cooperanti**
 - ✦ Un processo può influire o essere influenzato dall'esecuzione di un altro processo
- I vantaggi della comunicazione tra processi sono:
 - Condivisione delle informazioni
 - Accelerazione del calcolo
 - Modularità
 - Convenienza

Modelli per la comunicazione tra processi



- Affinché i processi possano cooperare necessitano di un meccanismo per la **comunicazione tra processi** (*interprocess communication*)
 - I modelli fondamentali sono:
 - ✦ Memoria condivisa
 - ✦ Scambio di messaggi

Modelli di comunicazione



Memoria condivisa



- La comunicazione tra processi mediante memoria condivisa si basa sull'allocazione di un blocco di memoria condivisa che i processi utilizzano
 - Normalmente è impedito ad un processo di usare la memoria allocata da un altro processo. Il Sistema Operativo si deve incaricare di farla utilizzare a chi la richiede
- I processi necessitano di un meccanismo che permetta loro di accedere alla memoria condivisa tramite scritture e letture
- Il problema è che due o più processi potrebbero scrivere simultaneamente nella stessa locazione

Problema del Produttore/Consumatore



- Nella comunicazione tra processi si utilizza il paradigma del produttore/consumatore:
 - Un processo **produttore produce informazioni**
 - Un processo **consumatore consuma informazioni**
- Il produttore ed il consumatore utilizzano una memoria condivisa mediante un buffer
 - **Buffer illimitato che non pone limiti alla dimensione del buffer**
 - **Buffer limitato presuppone una dimensione fissa del buffer**

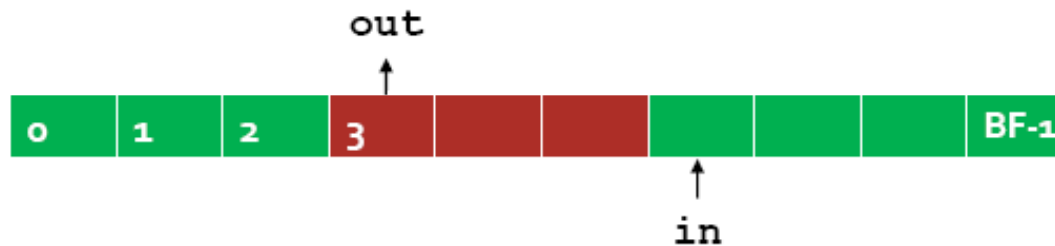
Esempio di Produttore/Consumatore



- Supponiamo di avere una array di elementi condivisi

```
#define BUFFER_SIZE 10
typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```



Processo Consumatore



- Sotto quali condizioni il buffer è vuoto?
 - Se $in == out$

```
while (true) {  
    while (in == out);  
    // do nothing -- nothing to consume  
  
    // remove an item from the buffer  
    item = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    return item;  
}
```



Processo produttore



- Sotto quali condizioni il buffer è pieno?
 - Se $((in + 1) \% DIM_BUFFER) == out$

```
while (true) {  
    /* Produce an item */  
    while ((in + 1) % BUFFER_SIZE) == out);  
    /* do nothing -- no free buffers */  
    buffer[in] = item;  
    in = (in + 1) % BUFFER_SIZE;  
}
```



Sistema a scambio di messaggi



- Lo scambio di messaggi permette a diversi processi di comunicare e di sincronizzarsi senza utilizzare delle variabili condivise
- Il meccanismo di scambio dei messaggi prevede due primitive di comunicazione:
 - *send(message)*
 - *receive(message)*
- Se due processi P e Q vogliono comunicare
 - Stabiliscono un canale di comunicazione tra di loro
 - Inviano e ricevono messaggi tra loro tramite send/receive
- L'implementazione logica di un **canale di comunicazione può essere fatto**
 - Fisicamente (memoria condivisa, hardware bus)
 - Logica

Alcune domande



- In che modo è stabilito un collegamento?
- Un collegamento può essere associato a più di due processi?
- Quante connessioni ci possono essere tra ogni coppia di processi comunicanti?
- Quale è la capacità di una connessione?
- La dimensione di un messaggio è fissata o dinamica?
- Un collegamento è uni-direzionale o bi-direzionale?

Comunicazione diretta



- Nella comunicazione diretta i processi devono conoscere la reciproca identità
- Le primitive sono definite nel seguente modo:
 - *send(P, message)* – invia un messaggio al processo *P*
 - *receive(Q, message)* – riceve un messaggio dal processo *Q*
- Le caratteristiche della comunicazione diretta sono:
 - I collegamenti sono stabiliti automaticamente
 - Un canale è associato esattamente a due processi
 - Esiste un canale tra ogni coppia di processi
 - Il link deve essere uni-direzione ma solitamente è bidirezionale

Comunicazione indiretta - 1



- Nella **comunicazione indiretta i messaggi sono inviati e ricevuti** attraverso delle porte
 - Ogni porta ha un unico id
 - I processi possono comunicare solo se condividono la porta
- Le **caratteristiche della comunicazione indiretta sono:**
 - Tra una coppia di processi si stabilisce una connessione se e solo se condividono una porta
 - Un canale può essere associato a più di una porta
 - Tra ogni coppia di processi possono esserci più canali e quindi più porte
 - I collegamenti possono essere uni-direzionali oppure bidirezionali

Comunicazione indiretta - 2



- Le operazioni previste nella comunicazione indiretta sono:
 - Crea una nuova porta
 - Invia e ricevi un messaggio attraverso una porta
 - Cancella una porta
- Le primitive sono definite nel seguente modo:
 - ***send(A, message)*** – *invia un messaggio alla porta A*
 - ***receive(A, message)*** – *ricevi un messaggio dalla porta A*

Comunicazione indiretta - 3



- Un porta può essere condivisa tra più processi:
 - P1, P2 e P3 condividono la porta A
 - P1 invia un messaggio
 - P2 e P3 leggono il messaggio dalla porta A
 - Chi dei due riceverà il messaggio?
- Le possibili soluzioni sono:
 - Si permette un solo canale tra due processi
 - Si permette a un solo processo alla volta di eseguire una operazioni di ricezione
 - Il Sistema Operativo si incarica di scegliere un solo processo a cui consegnare il messaggio. La scelta sarà comunicata al mittente

Sincronizzazione



- La comunicazione tra processi può essere
 - **Sincrono** (*bloccante*)
 - **Asincrono** (*non-bloccante*)
- Nella comunicazione sincrona
 - **Invio sincrono** - blocca il mittente finché il destinatario non ha ricevuto il messaggio
 - **Ricezione sincrona** - blocca il destinatario nell'attesa della ricezione del messaggio
- Nella comunicazione asincrona
 - **Invio asincrono** - il processo non è bloccato dall'invio del messaggio
 - **Ricezione asincrona** - il destinatario legge un messaggio oppure un valore nullo

Code di messaggi



- I messaggi scambiati tra processi risiedono all'interno di un buffer di memoria
- Il buffer è normalmente una coda di messaggi
- La coda dei messaggi può essere implementata in diversi modi:
 - **1. Capacità zero** – Il mittente deve attendere che il destinatario prenda il messaggio (*rendezvous*)
 - **2. Capacità limitata** – La coda ha lunghezza n . Il mittente attende se la coda è piena
 - **3. Capacità illimitata** - La coda ha lunghezza infinita. Il mittente non attende mai