Processi

Concetto di processo

Un sistema operativo esegue differenti tipi di programmi:

- un sistema a lotti (batch system) esegue lavori (job)
- un sistema time-shared esegue programmi utente (task)

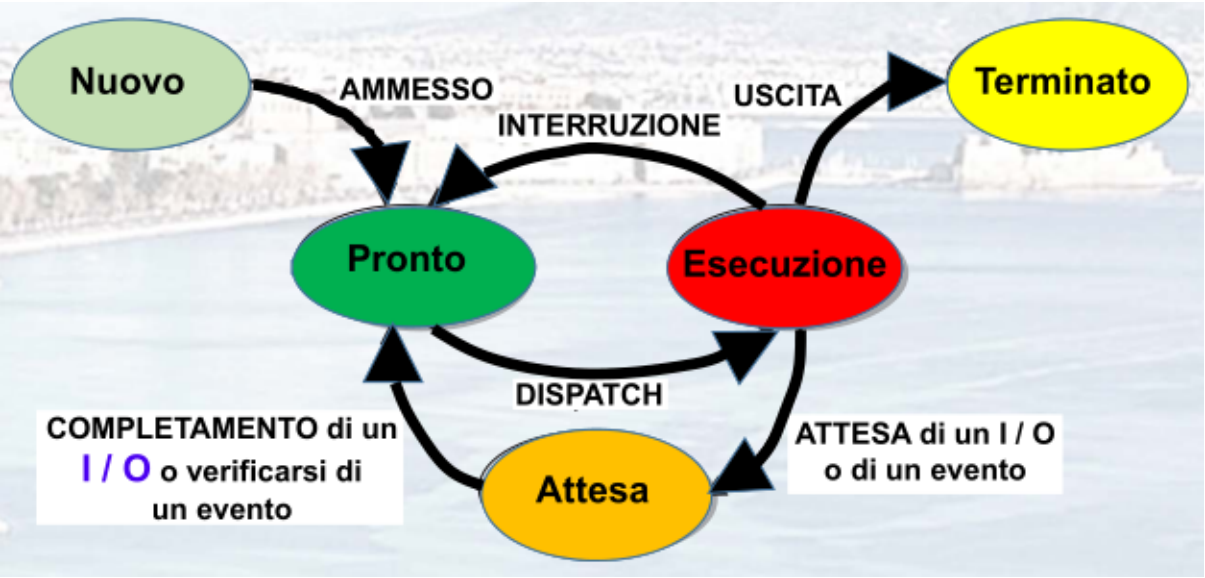
Processo → un programma in esecuzione
L'esecuzione di un processo deve avvenire in modo sequenziale

Un processo comprende:

- Contatore di programma
- Pila (stack)
- sezione di dati

Stato del processo

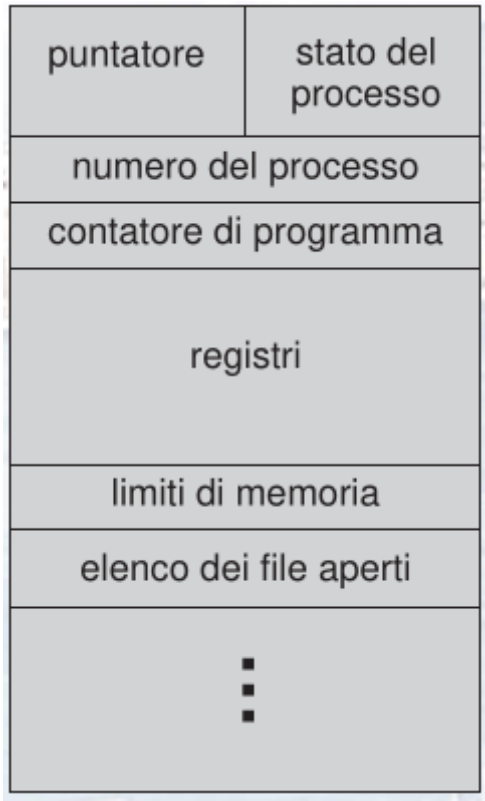
- Nuovo → si crea il processo
- Esecuzione → le istruzioni vengono eseguite
- Attesa → il processo attende che si verifichi qualche evento
- Pronto → il processo attendo di essere assegnato a un'unità di elaborazione
- Terminato → il processo ha terminato l'esecuzione



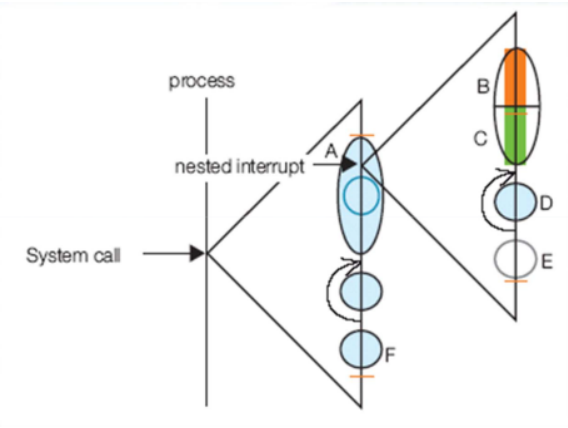
Blocco di controllo di un processo (PCB)

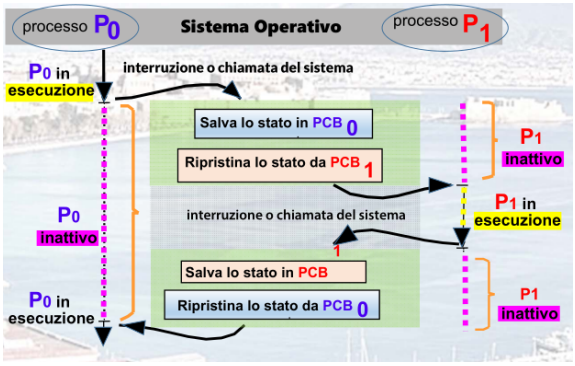
Informazioni connesse a un processo specifico:

- Stato del processo
- Contatore di programma
- Registri di CPU
- Informazioni sullo scheduling di CPU
- Informazioni sulla gestione della memoria
- Informazioni di contabilizzazione delle risorse
- Informazioni sullo stato dell'I/O



Commutazione della CPU tra i processi





Code di scheduling dei processi

- Coda di processi (job queue) → insieme di tutti i processi del sistema
- Coda di processi pronti (ready queue) → processi presenti nella memoria centrale pronti per essere eseguiti
- Coda del dispositivo (device queue) → elenco dei processi che attendono la disponibilità di un particolare dispositivo di I/O

Scheduler

Scheduler a lungo termine (job scheduler)

Seleziona i processi che devono essere caricati nella coda dei processi pronti e viene invicato meno frequentemente. Può anche essere lento e controlla il grado di multiprogrammazione.

Scheduler di CPU (short term scheduler o CPU scheduler)

Fa la selezione tra i lavori pronti per essere eseguiti e assegna la CPU a uno di essi. Viene invocato frequentemente, di conseguenza deve essere più veloce

Cambio di contesto

Avviene quando la CPU passa a un nuovo processo, con conseguente registrazione dello stato del processo vecchio e il caricamento dello stato precedentemente registrato del nuovo processo

Il tempo necessario per il cambio di contesto è puramente sovraccarico: il sistema non compie alcun lavoro utile durante la commutazione

Creazione di un processo

Un processo genitore può creare numerosi processi figli che possono a loro volta creare altri processi figli, generando un albero di processi.

Gestione delle risorse ed esecuzione

Genitore e figlio possono condividere tutte o parte delle risorse, o possono anche non condividerne alcuna. Allo stesso tempo possono essere eseguiti in modo concorrente o sequenziale.

Spazio di indirizzi

Il processo figlio è un duplicato del processo genitore e nel processo figlio viene caricato un programma.

Terminazione di un processo

Un processo termina quando finisce l'esecuzione della sua ultima istruzione, e chiede al sistema operativo di essere cancellato (chiamata del sistema exit)

Un processo genitore può anche porre termine all'esecuzione di uno dei suoi processi figli (abort) perché:

- Il processo figlio ha ecceduto nell'uso di alcune risorse
- Il compito assegnato al processo figlio non è più richiessto
- Il processo genitore termina
    - Un processo non può esistere senza un proprio genitore
    - Si esegue una terminazione a cascata

Processi cooperanti

Un processo si dice indipendente se non può influire su altri processi nel sistema o subirne l'influsso

Un processo è cooperante se influenza o può essere influenzato da altri processi in esecuzione nel sistema. Le informazioni vengono quindi condivise garantendo accelerazione del calcolo, modularità e convenienza

---

Problema del produttore e del consumatore

Un processo produttore produce informazioni che vengono poi usate da un processo consumatore.

Nel caso in cui si disponga di memoria limitata si presuppone l'esistenza di una dimensione fissata del vettore

Memoria limitata - soluzione con memoria condivisa

Supponiamo di usare una zona di memoria condivisa, e usiamo un vettore circolare di grandezza DIM_VETTORE che fa uso di 2 indici per l'accesso.

Con l'indice inserisci puntiamo alla prima casella disponibile del vettore

Con l'indice preleva puntiamo alla prima posizione occupata del vettore

Posso quindi ricavare:

- se (inserisci == preleva) → il vettore è vuoto
- se (((inserisci+1) % DIM_VETTORE) == preleva) → il vettore è pieno

```
// ESEMPIO DI CODICE -------- //
#define DIM_VETTORE 10
Typedef struct {
  ...
} elemento;
elemento vettore[DIM_VETTORE];
int inserisci = 0;
int preleva = 0;
// ------------------------- //
item appena_Prodotto;
while (1) {
  while (((inserisci + 1) % DIM_VETTORE) == preleva); /* non fa niente */
  vettore[inserisci] = appena_Prodotto;
  inserisci = (inserisci + 1) % DIM_VETTORE;
}
// ------------------------- //
while (1) {
  while (inserisci == preleva); /* non fa niente */
  da_Consumare = vettore[preleva];
  preleva = (preleva + 1) % DIM_VETTORE;
}
// ------------------------- //
```

Comunicazione tra processi (IPC)

Attraverso le funzioni di IPC i processi possono comunicare tra loro e sincronizzare le loro azioni.

Un sistema IPC fornisce le operazioni di send e recieve.

Se due processi vogliono comunicare devono stabilire un canale di comunicazione e scambiare messaggi tramite i comandi send/recieve

Un canale di comunicazione può essere visto in modo fisico (memoria condivisa, bus reti) e in modo logico (proprietà logiche)

Comunicazione diretta

I processi devono nominarsi in modo esplicito nei send e recieve.

I canali vengono stabiliti automaticamente, ogni canale e associato a una coppia di processi e il canale può essere sia unidirezionale che bidirezionale.

Comunicazione indiretta

I messaggi vengono inviati a delle porte (mailbox) che li ricevono. Ciascuna porta è identificata in modo univoco e due processi possono comunicare solo se condividono una porta.

Un canale di comunicazione (unidirezionale o bidirezionale) si stabilisce solo se entrambi i processi condividono una stessa porta, può essere associato a più di due processi e ciascuna coppia può condividere due o più canali.

Le operazioni principale consentono di creare una nuova porta, inviare e ricevere messaggi e rimuovere una porta.

Si può creare un problema nel momento in cui 3 canali (P1,P2,P3) condividono una stessa porta A.
Se P1 invia un messaggio ad A e P2 e P3 eseguono una recieve, chi riceve il messaggio?
Per risolvere il porblema si può:

- consentire che un canale sia associato al massimo a due processi
- consentire a un solo processo alla volta di eseguire la recieve
- consentire al sistema di decidere quale processo riceverà il messaggio

---

Sincronizzazione

Lo scambio di messaggi può essere sincrono (bloccante) oppure asincrono (non bloccante)

Anche le primitive send e recieve possono essere sincrone oppure asincrone

---

Code di messaggi (buffering)

Se la comunicazione è diretta o indiretta, i messaggi scambiati risiedono in code temporanee.

Esistono 3 modi per realizzare queste code:

- Capacità zero → 0 messaggi → il trasmittente deve fermarsi finché il ricevente prende in consegna il messaggio
- Capacità limitata → coda di lunghezza n → il trasmittente deve attendere se il canale è pieno
- Capacità illimitata → coda potenzialmente infinita → Il trasmittente non si ferma mai

---

Comunicazione nei sistemi client-server

Socket

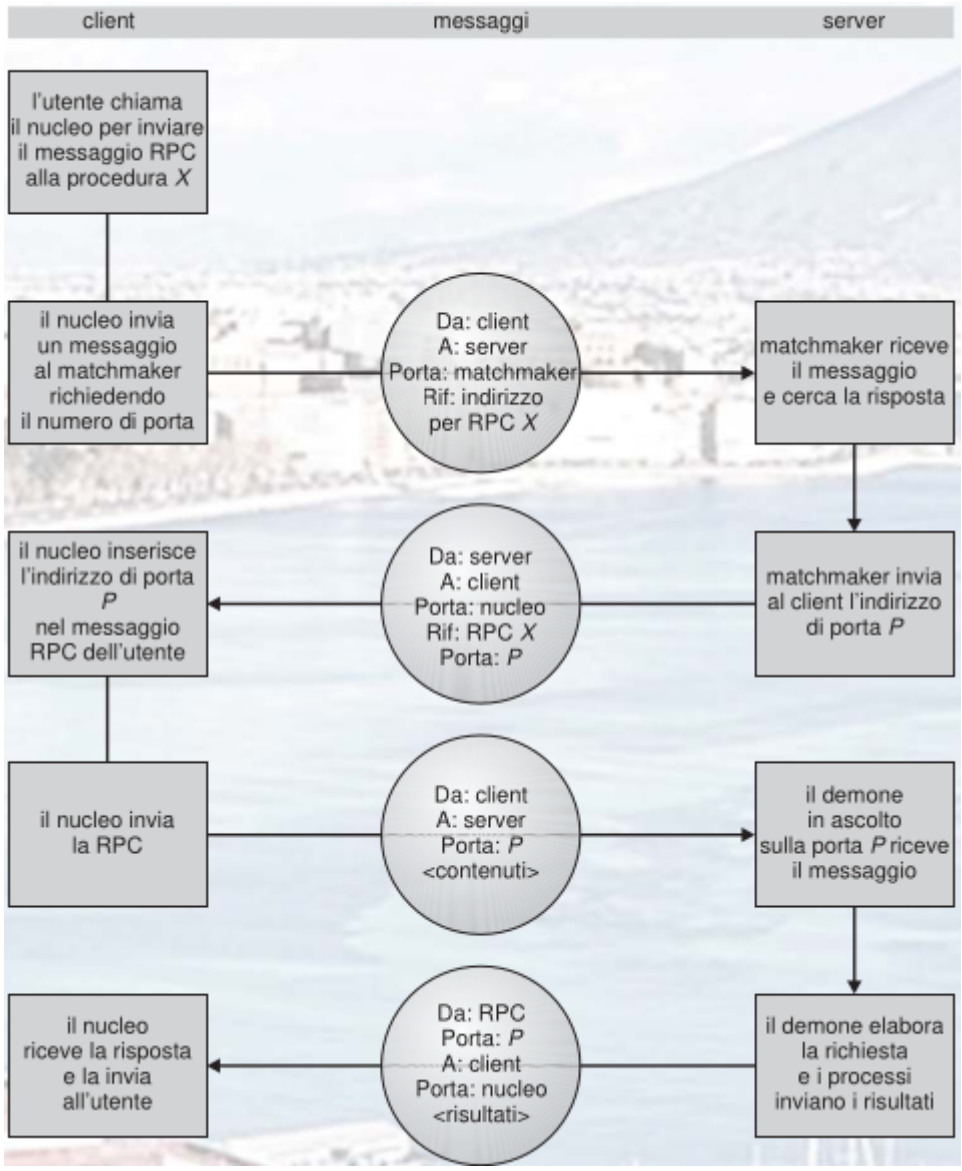
> Una socket è definita come l'estremità di un canale di comunicazione

Ogni socket è identificata da un indirizzo IP collegato a un numero di porta. La comunicazione avviene attraverso una coppia di socket

Chiamate di procedure remote (RPC)

La RPC è stata progettata per astrarre il meccanismo della chiamata di procedura affinché di possa usare tra sistemi collegati tramite una rete

Stub → segmento di codice: proxy legato lato client per la procedura in corso sul server. Il segmento di codice individua la porta del server e struttura i parametri (marshalling).
Un analogo segmento di codice di riferimento nel server riceve il messaggio e invoca la procedura nel server.



Chiamate di metodi remoti (RMI)

L'invocazione di metodi remoti è una funzione di Java simile alla RPC, e consente a un thread di invocare un metodo di un oggetto remoto

← Capitolo 3: Strutture dei S.O.

Capitolo 5: Thread →