# CAPITOLO 3: I PROCESSI

## 3.1 CONCETTO DI PROCESSO

Un **processo** è un’entità attiva, ossia è un programma in esecuzione, talvolta noto anche come **sezione di testo**: comprendente l’attività corrente rappresentata dal valore del contatore di programma e dal contenuto dei registri della CPU; normalmente comprende anche la pila(stack), contenente a sua volta i dati temporanei come i parametri di un metodo, gli indirizzi di rientro e le variabili locali, e una sezione di dati contenenti le variabili globali. Un processo può includere uno heap, ossia della memoria dinamicamente allocata durante l’esecuzione del processo.

Un **programma** è un’entità passiva, è costituito da un file binario residente nel disco che contiene il codice; un programma diventa processo quando viene caricato in memoria ed attende di essere eseguito dalla CPU. Un processo si può trovare in vari stati:

* **Nuovo:** il processo viene creato
* **Attesa:** il processo attende che si verifichi qualche evento (come un’operazione di I/O o la ricezione di un segnale).
* **Esecuzione:** Un’unità di elaborazione esegue le istruzioni del relativo programma.
* **Pronto:** il processo è pronto per essere assegnato alla CPU.
* **Terminato:** il processo ha terminato la sua esecuzione, e può terminare in modo **anormale** (interrotto per qualche errore di esecuzione) o in modo **normale**.

Per tener traccia delle informazioni di un processo, esso viene rappresentato dal blocco di controllo di un processo (PCB). È un blocco che contiene molte informazioni sul processo quali il suo **stato**, **informazioni sullo scheduling della CPU**, i **registri della CPU**, il **program counter, informazioni sulla gestione della memoria**, **informazioni sulle risorse utilizzate dal processo** **e informazioni sullo stato di I/O**.

## 3.2 SCHEDULING DEI PROCESSI

L’obiettivo della **multiprogrammazione** consiste nel disporre dell'esecuzione contemporanea di alcuni processi in modo da massimizzare l’utilizzo della CPU, mentre l’obiettivo della **partizione del tempo** è di commutare l'utilizzo della CPU tra vari processi così frequentemente che gli utenti possano interagire con ciascun programma mentre è in esecuzione.

Per raggiungere questi obiettivi, lo scheduler dei processi seleziona dalla coda dei processi un processo il quale viene posto in esecuzione nella CPU. In un sistema mono-processore non vi sarà mai più di un processo in esecuzione; gli altri processi dovranno attendere finché la CPU sia nuovamente disponibile.

**3.2.1 Code di scheduling**

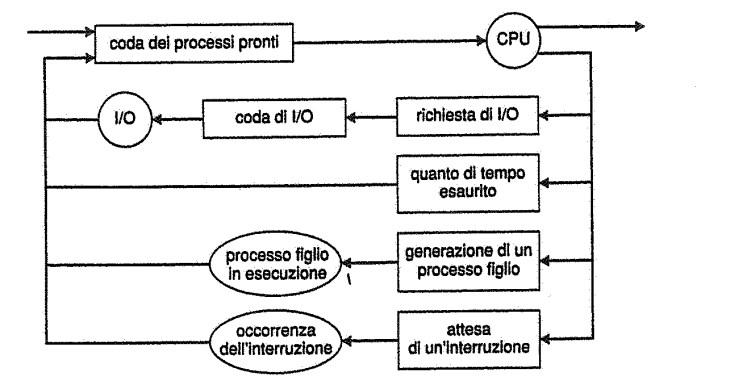
Ogni processo è inserito nella coda dei processi, composta da tutti i processi del sistema. I processi presenti in memoria centrale che sono pronti e nell’attesa di essere eseguiti, vengono collocati nella **coda dei processi pronti** (ready queue), questa coda si memorizza come una lista concatenata: un’intestazione della coda dei processi pronti contiene i puntatori al primo e all’ultimo PCB dell’elenco, e ciascun PCB è esteso con un campo puntatore che indica il prossimo processo contenuto nella coda dei processi pronti. Quando viene assegnato ad un processo la CPU, questo viene tolto dalla coda dei processi pronti ed eseguito finché non si verifica un’interruzione, una richiesta di I/O oppure esso termini la sua esecuzione. Poiché nel sistema esistono molti processi, il disco può essere occupato con una richiesta di I/O di un altro processo. L'elenco dei processi che attendono la disponibilità di un dispositivo di I/O si chiama coda del dispositivo.

Un nuovo processo inizialmente si colloca nella coda dei processi pronti, dove attende finché non è selezionato per essere eseguito (Dispatched). Una volta che gli viene assegnata la CPU ed è nella fase di esecuzione, si può verificare uno dei seguenti eventi:

-il processo richiede una operazione di IO e quindi viene inserito nella coda dei processi del dispositivo;

-il processo può creare un nuovo processo ed attende la terminazione;

-il processo può essere rimosso forzatamente dalla CPU a causa di un’interruzione ed essere inserito nella coda dei processi pronti.



## Scheduler

Nel corso della sua esistenza, un processo si trova in varie code di scheduling. Il SO attraverso lo scheduler seleziona i processi dalle suddette code. Spesso, nei sistemi a lotti, accade che più processi vengono sottoposti in esecuzione di quanti se ne possono eseguire. Quindi questi processi vengono trasferiti in memoria secondaria, dove si tengono fino al momento della loro esecuzione(spooling).

Lo scheduler a lungo termine provvede a scegliere questi lavori e a caricarli in memoria centrale dove attendono affinché siano eseguiti.

Lo scheduler a breve termine (o Scheduler della CPU) invece fa la selezione tra i processi pronti per l'esecuzione e assegna la CPU a uno di loro.

Questi due scheduler si differenziano maggiormente per la frequenza in cui queste operazioni vengono effettuate, lo scheduler a breve temine seleziona processi molto velocemente, nell’ordine dei millisecondi mentre per lo scheduler a lungo termine possono passare anche diversi minuti prima che selezioni un nuovo processo. È importante che lo scheduler a lungo termine faccia un’accurata selezione dei processi che possono essere processo con prevalenza di IO (IO bounds) e processo con prevalenza di esecuzione (CPU bounds). Inoltre, lo scheduler a lungo termine controlla il grado di multiprogrammazione del sistema, cioè il numero di processi presenti in memoria, se è stabile, la velocità media di creazione dei processi dev’essere uguale alla velocità media con cui i processi abbandonano il sistema; quindi lo scheduler a lungo termine si può richiamare solo quando un processo abbandona il sistema.

Gli altri sistemi operativi, come quelli a partizione del tempo, privi di scheduler a lungo termine, si limita semplicemente a caricare in memoria tutti i processi presenti nel sistema che vengono gestiti dallo scheduler a breve termine. La stabilità del sistema dipende dai limiti fisici degli stessi, quando ci si accorge che il rendimento della macchina scende a livelli inaccettabili si possono semplicemente chiudere delle attività.

Nei sistemi a partizione di tempo è stato introdotto un livello di scheduling intermedio. L’idea alla base di un tale scheduler è che a volte può essere vantaggioso eliminare processi dalla memoria (e dalla contesa attiva per la CPU), riducendo il grado di multiprogrammazione del sistema. In seguito, il processo può essere reintrodotto in memoria, in modo che la sua esecuzione riprenda da dove era stata interrotta. Questo schema si chiama **avvicendamento dei processi in memoria (SWAPPING)**.

## 

## Cambio di contesto

A volte sono le interruzioni ad indurre il SO a sospendere un processo ed eseguire routine del kernel. In presenza di una interruzione, il sistema deve salvare il **contesto** del processo corrente, per poterlo poi ripristinare quando il processo stesso potrà ritornare in esecuzione. Il contesto e rappresentato all’interno del PCB del processo, in termini generali si esegue un **salvataggio dello stato corrente della cpu**, sia essa che sia in modalità utente o in modalità di sistema; in seguito, si attuerà un corrispondente **ripristino dello stato** per poter poi riprendere l’elaborazione dal punto in cui era stata interrotta.

Il passaggio della CPU a un nuovo processo implica la registrazione dello stato del processo vecchio e il caricamento dello stato precedentemente registrato del nuovo processo.

Questa procedura e nota col nome di **cambio di contesto** *(context switch*), in questa procedura il sistema salva nel suo **PCB** il contesto del processo uscente, e carica il contesto del processo subentrante, salvato in precedenza, il **cambio di contesto** comporta un calo delle prestazioni, in genere di qualche millisecondo.

## 3.OPERAZIONE SUI PROCESSI

## 

## Creazione di un processo

## Durante la propria esecuzione, un processo può creare nuovi processi tramite la chiamata a sistema *create\_process*. Il processo creante si chiama processo genitore, mentre il nuovo processo si chiama processo figlio che allora volta può creare altri figli, formando un albero dei processi. La maggior parte dei sistemi operativi identifica un processo per mezzo di un numero univoco chiamato PID (process identifier) quando un processo Crea un nuovo processo, quest'ultimo può essere in grado di ottenere le proprie risorse direttamente dal sistema operativo, oppure vincolato a un sottoinsieme delle risorse del processo genitore. Il processo genitore può avere necessità di spartire le proprie risorse tra i suoi processi figli, oppure può essere in grado di condividerne alcune, come memoria o file. Limitando le risorse di un processo figlio a un sottoinsieme di risorse del processo genitore, si può evitare che un processo sovraccarichi il sistema creando troppi sottoprocessi. Quando si crea un processo, oltre alle risorse fisiche e logiche, il processo figlio ottiene i dati di inizializzazione che il processo genitore gli può passare.

## Quando un processo ne creo uno nuovo, per quel che riguarda l'esecuzione ci sono due possibilità:

## 1-il processo genitore continua l'esecuzione in modo concorrente con i processi figli;

## 2-il processo genitore attende che alcuni o tutti i tuoi figli processi terminino.

## Ci sono due possibilità anche per quel che riguarda lo spazio di indirizzi del nuovo processo:

## 1-il processo figlio è un duplicato del processo genitore;

## 2-nel processo figlio si carica un nuovo programma.

## Terminazione di un processo

un processo termina quando finisce l'esecuzione della ultima estrazione e inoltra la richiesta al sistema operativo di essere cancellato usando la chiamata di sistema exit (); a questo punto, il processo figlio può riportare alcuni dati al processo genitore che li riceve attraverso la chiamata di sistema wait (). Tutte le risorse del processo sono poi liberate dal sistema operativo. La terminazione di un processo si può verificare anche in altre casi. Un processo può causare la terminazione di un altro per mezzo di un'opportuna chiamata di sistema. Generalmente solo il genitore del processo che si vuole terminare può invocare una chiamata di sistema di questo tipo, occorre notare che un genitore deve conoscere l'identità dei propri figli, perciò quando un processo ne crea uno nuovo, l'identità del nuovo processo viene passata al processo genitore.

Un processo genitore può porre termine all'esecuzione di uno dei suoi processi figli per diversi motivi:

* Il processo figlio ha ecceduto nell'uso di alcune tra le risorse che gli sono state assegnate.
* Il compito assegnato al processo figlio non è più richiesto.
* Il processo genitore termina e il sistema operativo non consente ad un processo figlio di continuare l'esecuzione in tale circostanza.

In un sistema Linux se un processo genitore termina, tutti i suoi processi figli sono affidati al processo init (), che assume il ruolo di nuovo genitore, cui i processi figli possono riportare i risultati delle proprie attività.

## 4.COMUNICAZIONE TRA PROCESSI

I processi eseguiti concorrentemente nel SO possono essere indipendenti o cooperanti.

-Un processo è **indipendente** se non può influire su altri processi, né tanto meno può subirne l'influsso;

-Un processo è **cooperante** se influenza o può essere influenzato da altri processi.

Un ambiente che consente la cooperazione tra processi può essere utile per diverse ragioni:

* **Condivisione dell’informazione:** poiché più utenti possono essere interessati alla stessa informazione
* **Accelerazione del calcolo**: Alcune attività d’elaborazione sono realizzabili più rapidamente se si suddividono in sotto attività eseguibili in parallelo.
* **Modularità**
* **Convenienza:** Anche un solo utente può avere la necessità di compiere più attività contemporaneamente.

Per lo scambio di dati e di informazioni tra processi cooperanti necessitano delle tecniche di comunicazione tra processi detti IPC (**Interprocess communication**). Tali tecniche prevedono diversi metodi per lo scambio di informazioni: attraverso **memoria condivisa** oppure attraverso **lo scambio di messaggi**. Lo scambio di messaggi è utile quando si devono trasmettere piccole quantità di informazioni, altrimenti è meglio usare la memoria condivisa.

## Sistemi a memoria condivisa

## La comunicazione tra processi basata sulla condivisione della memoria richiede che i processi comunicanti allochino una zona di memoria condivisa, di solito residente nello spazio degli indirizzi del processo che alloca: gli altri processi che desiderano usarla per comunicare dovranno annettersi al loro spazio degli indirizzi. Il tipo è la collocazione dei dati sono determinati dai processi, e rimangono al di fuori del controllo del sistema operativo. I processi hanno anche la responsabilità di non scrivere nella stessa locazione simultaneamente. Per illustrare il concetto di cooperazione tra processi, si consideri il problema del produttore/consumatore. Un processo produttore produce informazioni che sono consumate da un processo consumatore. Una possibile soluzione del problema del produttore/consumatore si basa sulla memoria condivisa. L'esecuzione concorrente dei due processi richiede la presenza di un buffer che possa essere riempito dal produttore e svuotato dal consumatore. I due processi devono essere sincronizzati in modo tale che il consumatore non tenti di consumare un'unità non ancora prodotta.

## Sono utilizzabili due tipi di buffer:

## Illimitato: non pone limiti alla dimensione del buffer, il consumatore può trovarsi ad attendere nuovi oggetti, ma il produttore può sempre produrne.

## Quello con buffer limitato: il problema del produttore e del consumatore con buffer limitato presuppone l'esistenza di una dimensione fissa del buffer in questione, in questo caso il consumatore deve attendere che il buffer sia vuoto e viceversa il produttore deve attendere che il buffer sia pieno.

## Sistemi a scambio di messaggi

Un altro modo per la comunicazione tra processi è quello dello scambio di informazioni attraverso lo scambio dei messaggi. Lo scambio di messaggi è un meccanismo che permette a due o più processi di comunicare e di sincronizzarsi senza condividere lo spazio degli indirizzi. Un meccanismo per lo scambio di messaggi deve prevedere almeno due operazioni: send () e receive (), i messaggi possono avere lunghezza fissa o variabile, nel primo caso implementazione a livello del sistema è elementare ma programmare applicazione diviene complicato mentre nel secondo caso, l'implementazione del meccanismo è più complesso e la programmazione utente risulta semplificata.

Se i processi **P** e **Q** vogliono comunicare, devono inviare e ricevere messaggi tramite un canale di comunicazione, ci sono diversi metodi per realizzare a livello logico un canale di comunicazione e le chiamate di sistema send () e receive ():

* Comunicazione diretta o indiretta;
* Comunicazione sincrona o asincrona;
* Gestione automatica o esplicita del buffer;

**Nominazione**

Per comunicare, i processi devono disporre della possibilità di far riferimento ad altri processi; a tale scopo è possibile servirsi di una comunicazione **diretta** o **indiretta**. con la comunicazione diretta, ogni processo che intenda comunicare deve nominare esplicitamente il ricevente o il trasmittente della comunicazione. In questo schema le funzioni primitive send () e receive () si definiscono come segue:

* **Send (P, messaggio):** invia messaggio al processo P;
* **Receive (Q, messaggio):** riceve, in messaggio, un messaggio dal processo Q.

All'interno di questo schema, un canale di comunicazione alle seguenti caratteristiche:

* tra ogni coppia di processi che intendono comunicare si stabilisce automaticamente un canale; i processi devono conoscere solo la loro reciproca identità;
* Un canale è associato esattamente a due processi.

Questo schema ha una simmetria nell’indirizzamento, cioè il trasmittente e il ricevente devono nominarsi a vicenda. Una variante di questo schema si avvale dell'asimmetria dell’indirizzamento: soltanto il trasmittente nominai ricevente.

In questo schema le primitive Send () e receive () si definiscono come segue:

* **Send (P, messaggio):** invia il messaggio al processo P;
* **Receive (id, messaggio):** riceve, in messaggio, un messaggio da qualsiasi processo; nella variabile id si riporta il nome del processo con cui è avvenuta la comunicazione.

Entrambi gli schemi, simmetrico e asimmetrico, hanno lo svantaggio di una limitata modularità delle risultanti definizioni dei processi.

Con la **comunicazione indiretta,** i messaggi s’inviano a delle porte(mail-box) che li ricevono, Una porta si può vedere in modo astratto come un oggetto in cui i processi possono introdurre e prelevare i messaggi, identificata in modo univoco.

Due processi possono comunicare solo se condividono una porta. Le primitive send () e receive () si definiscono come segue:

* **send (A, mex):** invia il messaggio alla porta A.
* **receive (A, mex):** riceve, in messaggio, un messaggio alla porta A.

In questo schema un canale di comunicazione ha le seguenti caratteristiche:

* tra una coppia di processi si stabilisce un canale solo se entrambi i processi della coppia condividono una stessa porta;
* Un canale può essere associato a più di 2 processi;
* Tra ogni coppia di processi comunicanti possono esserci più canali diversi, ciascuno corrispondente ad una porta.

Una porta può appartenere al processo o al sistema. se appartiene a un processo, occorre distinguere ulteriormente tra il proprietario, che può soltanto ricevere messaggi tramite la porta, e l'utente, che possono inviare i messaggi alla porta. Quando un messaggio che possiede una porta termina, questa scompare. Una porta posseduta dal sistema operativo è indipendente e non è legata ad alcun processo particolare.

Il sistema operativo offre un meccanismo che permette ad un processo le seguenti operazioni:

* Creare una nuova porta;
* Inviare e ricevere messaggi tramite la porta;
* Rimuovere una porta.

**SINCRONIZZAZIONE**

La comunicazione tra processi avviene attraverso chiamate delle primitive send () e receive (). Ci sono diverse possibilità nella definizione di ciascuna primitiva. Lo scambio di messaggi può essere:

**-Invio sincrono:** il processo che invia il messaggio si blocca nell’attesa che il ricevente riceva il messaggio (può essere una porta o un processo);

**-Invio asincrono:** il processo invia il messaggio e continua la sua esecuzione;

**-Ricezione sincrona:** il ricevente si blocca nell'attesa di ricevere un messaggio;

**-Ricezione asincrona**: il ricevente riceve un messaggio valido o un valore nullo;

**CODE DI MESSAGGI**

## Se la comunicazione è diretta o indiretta, i messaggi scambiati tra processi comunicanti risiedono in code temporanee. Fondamentalmente esistono 3 modi per realizzare queste code:

**-Capacità 0:** la coda ha lunghezza max = 0. In questo caso il trasmittente deve fermarsi finché il ricevente non riceva il messaggio, visto che il canale non può avere messaggi d’attesa al suo interno.

**-Capacità limitata:** la coda ha lunghezza finita n, se la coda non è piena quando si invia un nuovo messaggio, quest'ultimo è posto in fondo alla coda. Il trasmittente può proseguire la propria esecuzione senza essere costretto ad attendere. Il canale ha tuttavia capacità limitata, se è pieno, il trasmittente deve fermarsi nell'attesa che ci sia spazio disponibile nella coda

**-Capacità illimitata:** la coda ha lunghezza potenzialmente infinita, quindi al suo interno può attendere un numero indefinito di messaggi. Il trasmittente non si ferma mai.

Il caso con capacità 0 è detto sistema a scambio di messaggi NO BUFFERING (senza memorizzazione transitoria); gli altri 2 sistemi AUTOMATIC BUFFERING (memorizzazione transitoria automatica).

## 5.COMUNICAZIONE CLIENT SERVER (facoltativo)

Una socket è definita come l’estremità di un canale di comunicazione. Una coppia di processi che comunica attraverso una rete usa una coppia di socket, una per ogni processo e ogni socket è identificata attraverso un indirizzo IP concatenato a un numero di porta. Il cliente comunica con il server creando una socket e collegandosi per suo tramite alla porta su cui il server è in ascolto. Stabilita la connessione, il client può leggere dalla socket tramite le ordinarie istruzioni di IO. I messaggi vengono suddivisi in pacchetti ordinati correttamente ed inviati sulla socket. Si indirizzano ad un demone il quale provvederà a comporre il messaggio originale.