I processi

Cos'è un processo?

Un processo è una attività, controllata da un programma, che si svolge su un processore.

- Il **programma** è una entità *statica* che descrive la sequenza di istruzioni che devono essere svolte.
- Il **processo** è una entità *dinamica*, che rappresenta la particolare esecuzione di un programma, quindi evolve nel tempo.
- Il **processore (o CPU):** è la principale risorsa del sistema di elaborazione che consente l'evoluzione di un processo.

Per esempio la ricetta di cucina può essere considerato un programma che diventa un processo quando, assegnate tutte le risorse (utensili e ingredienti), sarà effettivamente eseguita dal processore (il cuoco). In realtà si possono generare più processi: l'approvvigionamento degli utensili e degli ingredienti (eventuale acquisto), preparazione della ricetta scomposta in diverse fasi, presentazione finale della pietanza.

In linea di massima un processo può essere definito come un programma in esecuzione composto dalla coppia (E,S)

- E: **codice eseguibile**
- S: **stato** del processo

Un programma può dare origine a diversi processi. Un certo processo è associato ad un unico programma.

Affinché un processo avanzi è necessaria l'assegnazione delle risorse a lui necessarie, fra cui la **CPU**.
Se la CPU è una sola, solo uno fra i processi esistenti è nello stato di esecuzione. Gli altri sono in qualche altro stato, ma non sono in esecuzione.
Normalmente (nei sistemi multiprogrammati) sono presenti sulla macchina decine di processi (osserva il task manager di Windows).
Un processo nasce e occupa le risorse del sistema. In particolare occupa la RAM, che è il punto di partenza per potere andare in esecuzione. Infatti la CPU è in grado si eseguire codice solo se quest'ultimo è in RAM.
I processi del kernel sono sempre in RAM, essendo "pronti" per essere eseguiti.

La componente principale dell'elaboratore, e anche la più costosa, è la CPU (Central Processing Unit) che è un esecutore sequenziale di operazioni. Tutti i programmi hanno bisogno della CPU per essere eseguiti, ma questa è unica e quindi viene "contesa" dai vari processi in esecuzione. Riveste allora particolare importanza quella parte del sistema operativo chiamato **gestore dei processi** che si occupa dell'assegnazione della CPU ai singoli processi che ne richiedono l'uso. Quindi la gestione dei processi è affidata a quei moduli del sistema operativo che formano il nucleo e che sono sempre residenti in memoria centrale. Il nucleo, chiamato anche **Kernel** (nocciolo), è la parte del sistema operativo più vicina alla macchina ed è strettamente dipendente dall'hardware. Le funzioni fondamentali del nucleo sono:

- Gestire il ciclo di vita dei processi: avvio e terminazione dei processi
- assegnare la CPU ai diversi processi: selezionare quale tra i processi in stato di "pronto" deve essere mandato in esecuzione
- sincronizzare i processi: gestire la cooperazione e la concorrenza tra i processi nell'accesso alle risorse
- sincronizzare i processi con l'ambiente esterno

Il nucleo comprende tutte le **routine di risposta alle interruzioni** e le **procedure che assegnano la risorsa CPU** ai diversi processi. Le norme che regolano queste assegnazioni vengono chiamate **politiche di scheduling.** Fanno quindi parte del nucleo:

- lo **schedulatore dei lavori;**
- lo **schedulatore dei processi;**
- il **controllore del traffico;**
- il **gestore delle interruzioni**;
- le **procedure di sincronizzazione e comunicazione** tra più processi necessarie per lo scambio di dati e informazioni.

Ogni elaboratore ha una propria configurazione hardware, così come ogni sistema operativo ha differenti caratteristiche. La configurazione (hardware e software) che si trova più di frequente è quella di un sistema di elaborazione **multiprogrammato** e **interattivo**, dotato di **un solo processore centrale**, di uno o più processori dedicati (canali di I/O) e delle periferiche standard (dischi, stampanti….)

In **memoria centrale** (RAM) in un dato istante possiamo trovare un certo numero di processi uno dei quali *sta evolvendo*, altri sono *pronti* per essere serviti, altri devono *attendere* la terminazione di operazioni di I/O. Altri ancora sono momentaneamente *parcheggiati* su memoria di massa. L'esecuzione di un programma genera almeno un processo, costituito dalla sequenza delle operazioni del programma. Un processo che deve essere eseguito può trovarsi, in ogni istante, in uno "**stato**", cioè in una condizione particolare determinata dalle esigenze del processo stesso e dallo stato globale del sistema (cioè le altre componenti hardware e software).

> Ogni **stato corrisponde a una situazione** in cui si trova il programma da eseguire (è in esecuzione, è in attesa della stampante, è in attesa di un dato da inserire da tastiera ecc.)
>
> Il **passaggio da uno stato all'altro** è deciso dal **sistema operativo** sulla base delle **politiche di schedulazione**. A ogni passaggio corrisponde una transazione che è effettuata tramite l'esecuzione di uno specifico programma del sistema operativo (controllore del traffico).

Descriviamo il ciclo di avanzamento di un programma seguendone tutta la sua storia, dal momento in cui il lavoro è proposto al sistema a quello in cui termina la sua esecuzione.
Come esempio di processo che deve essere svolto si può considerare quello di effettuare un viaggio in aereo dove *la situazione in cui si trova il passeggero* viene considerato il suo ***stato*** ed è determinato non solo dalle esigenze del passeggero ma anche dalle condizioni esterne: ad esempio *si trova all'aeroporto* in attesa dell'imbarco in aereo, che può essere pronto o deve ancora arrivare ed è in ritardo, oppure il passeggero *è in volo*, oppure ancora *si trova in un aeroporto di transito* per uno scalo o, finalmente, *è arrivato* a destinazione.

Processo utente e supervisore

Un **processo utente** è un processo generato da un programma scritto dall'utente.

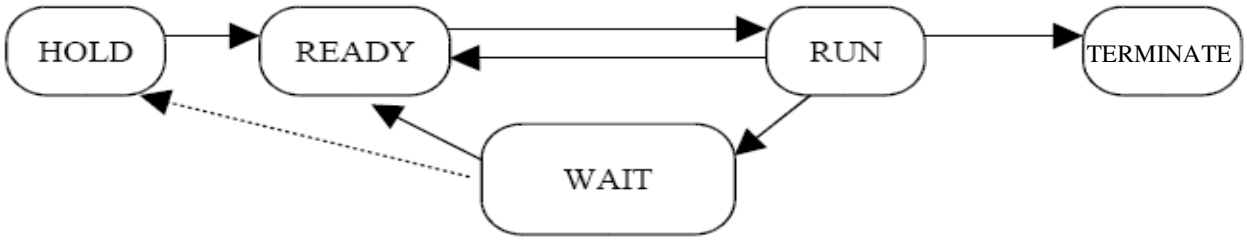
Un **processo supervisore** è un processo generato da un programma del Sistema Operativo.

Tra i processi in esecuzione sulla CPU ci sono quindi i processi utente, ma anche i **processi supervisore** del SO e questi ultimi hanno una **priorità** e una **importanza maggiore** dei processi utente
Ad esempio quando si chiede attraverso un programma scritto in C++ di acquisire un dato da tastiera con l'istruzione *cin …* oppure si visualizza un messaggio sul monitor con *cout …*, in realtà il nostro programma chiede l'intervento al sistema operativo per poter colloquiare opportunamente con le due periferiche menzionate, tastiera e monitor, e il processore entra in stato supervisore per trattare le richieste del nostro programma perchè comincia ad eseguire le routine preposte appunto alla gestione della tastiera e del monitor. Al termine dell'operazione richiesta potrà riprendere l'esecuzione del nostro programma (processo utente).

Gli stati di un processo

Gli **stati possibili** nei quali si può trovare un processo sono:

- **HOLD (parcheggio)**: il programma (chiamato job) è stato proposto al sistema e **attende di essere caricato in memoria centrale**. Esso è *in attesa di esecuzione* e *si trova in memoria di massa* .
- **READY (pronto)**: il programma è diventato **processo** e *si trova in memoria centrale*, pronto per essere eseguito. Esso **attende che gli venga assegnata la CPU**.
- **RUN (esecuzione)**: il **processo è in esecuzione** (in ogni istante un solo processo si trova in questo stato) perché gli è stata assegnata la CPU.
- **WAIT (attesa)**: il processo è in attesa (ad esempio deve attendere la fine di un'operazione di I/O).
- **TERMINATE (terminazione)**: il processo è terminato e può essere tolto dalla memoria centrale.



Stati di un processo

A ogni **stato** è associata una **lista** che contiene l'elenco di **tutti i processi** che si trovano in quel particolare stato in quel momento.

Transizione HOLD–READY

Avviene quando il lavoro viene caricato in memoria centrale. All'inizio, il Job scheduler, che ha ricevuto una richiesta di esecuzione di un programma, ha creato il **lavoro** (job) corrispondente al programma e lo ha messo in stato di Hold. Al momento opportuno lo **schedulatore dei lavori** sceglie tra i lavori in stato di Hold quello da **caricare nella memoria centrale**, se c'è un'area sufficientemente grande per contenerlo, passandolo così nello stato di Ready. La scelta del processo da portare in stato di "pronto" viene fatta in base all'ordine di arrivo dei processi oppure in base alla priorità dei processi o altro (politiche di schedulazione).

Transizione READY–RUN

Avviene quando al processo viene assegnato il **processore**; la scelta del processo da portare in stato di Run è fatta dallo **schedulatore dei processi** in base alla politica di schedulazione adottata dal gestore del processore.

Transizione RUN–READY

Questa transizione avviene quando il processo termina il time slice assegnatogli (time sharing), oppure quando arriva una interruzione esterna.

Transizione RUN–WAIT

Avviene quando il processo chiede un sevizio del S.O. ad esempio un'operazione di I/O (visualizzazione su monitor, richiesta di inserimento dati da tastiera, richiesta di una stampa ….)

Transizione WAIT–READY

Avviene quando un'interruzione esterna segnala il completamento dell'operazione per cui il processo è in attesa.

Transizione RUN–TERMINATE

Quando l'esecuzione del processo termina.

Transizione WAIT–HOLD

Non esiste sempre. E' la "schedulazione di medio livello", nella quale il S.O. decide di recuperare dello spazio di memoria "swappando" un processo su disco. E' una operazione lenta perché impone il salvataggio su disco dello stato del processo, e viene utilizzata più raramente possibile.

LO SCHEDULATORE DEI LAVORI ***(O SCHEDULATORE A LUNGO TERMINE)***

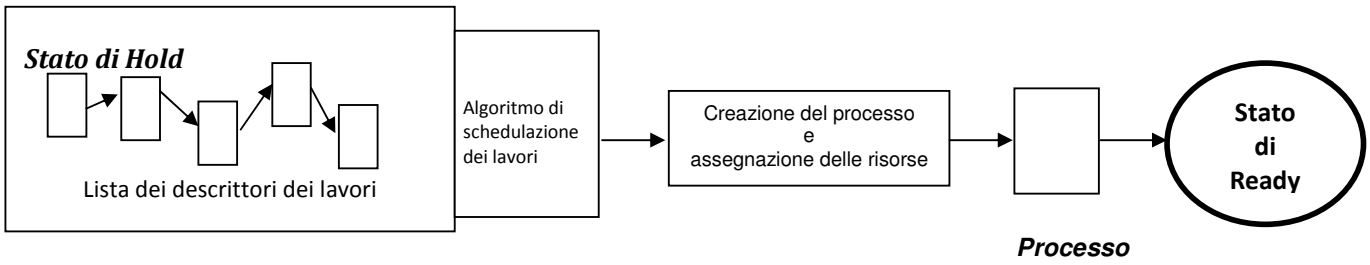
Determina in ogni istante il numero di lavori che risiedono in memoria centrale e quindi condiziona e controlla il grado di multiprogrammazione del sistema.

- Tiene aggiornate apposite *tabelle* che contengono le informazioni dello stato di ciascun lavoro (job)
- Sceglie ogni volta quale lavoro deve acquisire le risorse per avanzare, selezionandolo da una lista dei lavori in stato di *Hold*
- Quando un job viene scelto genera un processo
- Quando un processo termina sovrintende e coordina il recupero delle risorse assegnate a quel processo.

Per fare questo consulta i **descrittori dei lavori.**

Ogni **descrittore** contiene le seguenti informazioni:

- il *nome* del lavoro
- la *data e ora* di accettazione
- la *priorità*
- il *tempo* previsto di CPU
- *l'elenco delle risorse da assegnare staticamente* (dedicate per l'intero tempo di elaborazione)
- *l'elenco delle risorse da assegnare dinamicamente* (assegnate al momento dell'uso e rilasciate subito dopo)
- l'indicatore dello *stato di avanzamento* del lavoro (step). Dato che un lavoro può generare più processi viene indicato ogni processo già concluso e quello eventualmente in esecuzione.
- il puntatore al lavoro successivo nella lista



Politiche di schedulazione dei lavori

First In First Out: I lavori vengono scelti nell'ordine in cui sono giunti. In questo caso può succedere che i tempi di risposta siano imprevedibili e spesso c'è una scarsa ottimizzazione delle risorse.

Priorità statica: in questo caso viene assegnata ad ogni lavoro che arriva una priorità calcolata in base ad alcuni parametri, ad esempio al tempo presunto di utilizzo di alcune risorse. C'è però il rischio che alcuni lavori, quelli con priorità basse, restino troppo in attesa se arrivano continuamente lavori con priorità più alta.

Priorità dinamica: in questo caso viene assegnata ad ogni lavoro che arriva una priorità calcolata in base ad alcuni parametri, come nel caso precedente, ma con il crescere del tempo di attesa viene incrementata la priorità; si tiene quindi conto dell'anzianità del lavoro per evitare che rimanga troppo tempo nella lista.

SCHEDULATORE DEI PROCESSI *(O SCHEDULATORE A BREVE TERMINE)*

Determina in ogni istante il processo che deve essere portato da stato di pronto in stato di esecuzione, assegnandogli quindi la risorsa processore.

Per fare questo consulta i **descrittori dei processi (PCB - Process Control Block).**

Ogni **descrittore** contiene le seguenti informazioni:

- il *nome* del processo
- *l'indirizzo del descrittore del processo che l'ha generato* (es.: per un processo di I/O viene posto in questo campo l'indirizzo del processo che l'ha richiamato)
- *la lista degli indirizzi dei descrittori dei processi richiamati*
- *lo stato di avanzamento*
- *l'immagine nel processore* (contenuto dei registri di lavoro, del registro dei flags, del program counter,…)
- *l'immagine nella memoria* (le informazioni sulle aree di memoria assegnate al processo)
- *le informazioni relative alle altre risorse* assegnate al processo
- *il tempo* di CPU già usato, i limiti di tempo, …
- il puntatore al processo successivo nella lista

Tutti i descrittori dei processi che si trovano nello stesso stato di avanzamento sono collegati tra loro in modo da formare *liste bidirezionali multiple*.

Politiche di schedulazione dei processi

Esistono diversi algoritmi di schedulazione dei processi, ma sicuramente una buona politica deve considerare le seguenti necessità del sistema di elaborazione:

- *Rendere massimo il tempo di utilizzo della CPU*: il processore deve essere mantenuto attivo il più possibile, evitando che in un certo istante la lista dei processi in stato di pronto sia vuota;
- *Servire il maggior numero di processi nell'unità di tempo* rendendo quindi massimo il *throughput* di sistema e cioè il numero di processi completati nell'unità di tempo;
- *Valutare i tempi di esecuzione dei processi.* Viene misurato il tempo che un processo rimane attivo dal momento della sua generazione fino alla sua terminazione, tenendo conto sia dei tempi passati in attesa che di quelli impiegati nell'uso delle diverse risorse;
- *Rendere il più possibile limitato il tempo di attesa nello stato di pronto;*
- *Minimizzare il tempo di risposta.* In un sistema interattivo il tempo di risposta dipende sia dall'utente che fornisce i dati, sia da ciò che deve essere eseguito, sia dalla velocità di risposta dei dispositivi di output. Allora è sicuramente più significativo valutare il tempo che intercorre tra una singola richiesta e la prima risposta prodotta;
- *In un sistema multiutente, soddisfare il maggior numero di utenti;*
- *Minimizzare l'overhead di sistema (sovraccarico di sistema),* considerando che l'assegnazione della CPU da un processo ad un altro prevede sempre l'esecuzione della procedura di salvataggio dello stato del vecchio processo e il caricamento dello stato del nuovo processo (*context switch*).

L'algoritmo di rotazione tra i processi in esecuzione deve essere il più equo ed efficiente possibile.
C'è da tenere presente anche che per un corretto funzionamento del sistema è opportuno avere sia processi che usano molto la CPU, sia processi che usano molto le periferiche (I/O). In questo modo si evitano tempi di risposta inaccettabili per i programmi interattivi in un caso e inattività del processore nell'altro.

Anche lo schedulatore dei processi usa precisi criteri per operare la scelta di quale processo far avanzare.
Tratteremo successivamente alcune di queste politiche.

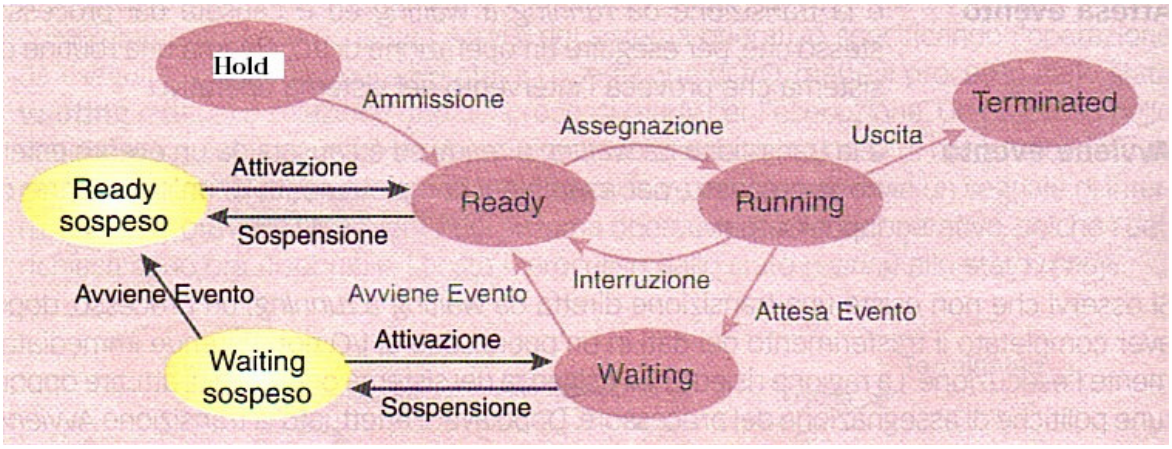
SCHEDULATORE DEI PROCESSI *A MEDIO TERMINE*

Esiste talvolta anche lo SCHEDULATORE A MEDIO TERMINE che gestisce lo **swapping** (trasferimento del contenuto di un'area di memoria centrale in un'area della memoria di massa detta appunto area di **swap**).

Spesso le memoria centrale non è sufficientemente estesa per contenere tutti i programmi concorrenti, allora il S.O. decide di recuperare della memoria "swappando" un processo su disco; lo schedulatore porta, quindi, il processo in stato di *Waiting-sospeso* o *Ready-sospeso* e questo succede a turno per i processi "fermi" troppo a lungo in *stato di attesa* oppure per i processi "fermi" troppo a lungo in *stato di ready* che occupano inutilmente la memoria RAM. Questo garantisce che tutti i processi avanzino, ma rallenta le operazioni in quanto richiede il salvataggio e poi il ripristino dello stato dei processi su una memoria molto più lenta della RAM, il disco.

Alcuni dei motivi per i quali può verificarsi l'esigenza di effettuare uno **swap out** di uno o più programmi, rimuovendoli dalla memoria centrale e portandoli temporaneamente su disco sono:

- L'arrivo di un processo con alta priorità che richiede molto spazio in memoria centrale in un momento in cui tale spazio non è disponibile perché occupato da altri processi che si trovano in stato di Ready; allora il S.O. porta alcuni di questi processi su disco ponendoli in stato Ready-sospeso e carica nella memoria Ram liberata il processo ad alta priorità che in questo modo ha lo spazio di memoria che gli necessita;
- Molti processi sono in stato di Wait in attesa di un evento di I/O e il processore è sottoutilizzato e quindi il S.O. decide di portare in memoria centrale altri processi che richiedono la CPU spostandone alcuni su disco e ponendoli da stato di Wait a stato di Waiting-sospeso.



L'insieme dei processi in stato di Ready, Running e Waiting costituisce l'insieme dei **processi attivi** e definisce anche il **grado di multiprogrammazione** del sistema esaminato.

I processi invece che si trovano in stato Waiting-sospeso e Ready-sospeso sono considerati **processi dormienti.**

L'operazione di *sospensione* di un processo (trasferendolo da memoria centrale a disco rigido) viene chiamata **swap out**.

L'operazione inversa di *attivazione* di un processo (trasferendolo da disco rigido a memoria centrale) viene chiamata **swap in**.

IL CONTROLLORE DEL TRAFFICO

Ha il compito di controllare l'avanzamento dei processi:

- Tiene traccia dello stato dei processi attivi
- Compila il descrittore di processo ogni volta che cambia qualcosa.
- Gestisce le transizioni da uno stato all'altro: sposta i descrittori da una lista all'altra, carica l'immagine del processo quando lo pone in esecuzione; salva l'immagine quando sottrae la CPU.
- Coordina sincronizzazione e cooperazione fra i processi, ovvero coordina gli eventi in modo che accadano nel giusto ordine.

Riepilogando il controllore del traffico:

- Si accorge quando un processore e' libero.
- Chiama lo schedulatore e si fa comunicare qual è il processo da schedulare.
- Aggiorna lo stato di avanzamento di quel processo e inizializza tutti i registri del processore con i dati prelevati dal descrittore del processo.
- Se il processo termina, gli toglie le risorse, elimina il descrittore, lo comunica allo schedulatore dei lavori affinché proponga un nuovo processo (il prossimo che ne ha diritto).

Inoltre:

- Se finisce il time slice, salva lo stato del processo nel descrittore, lo sposta il processo in stato di ready e aggiorna il descrittore coi nuovi dati che potranno servire allo schedulatore.
- Poi invoca lo schedulatore affinché scelga per un nuovo processo da passare in stato di Run.
- Se arriva un segnale di interruzione, salva l'immagine del processore, sposta il processo in stato di ready, e mette in esecuzione il processo associato al programma di risposta all'interruzione.
- Se il processo chiede I/O, salva l'immagine del processore, sposta il processo in stato di wait, crea il descrittore del processo di I/O e lo pone fra quelli in stato di pronto. Poi chiama lo schedulatore per scegliere un nuovo processo a cui assegnare la CPU. Il processo di I/O seguirà le stesse sorti, solo che il processore NON sarà necessariamente la CPU, ma potrà essere un canale di I/O.

Passaggi di stato e le interruzioni

Affinché un processo possa transitare da uno stato all'altro occorre che si verifichino determinati eventi. Tali eventi devono essere comunicati all'unità di controllo della CPU tramite apposite linee hardware, utilizzando il **meccanismo delle interruzioni**.

Un'**interruzione** (o **interrupt**) è una segnalazione che arriva alla CPU per comunicarle qualcosa e può essere di natura **software o hardware.**

Si parla di **interruzioni software** quando è lo stesso processo in esecuzione che invia un'apposita istruzione nella quale specifica il tipo di comportamento che richiede alla CPU.
Si parla di **interruzioni hardware** quando la segnalazione di richiesta di interruzione arriva da parte di una **periferica** di input/output tramite un **segnale elettrico**. Tale segnale viene inviato su una linea dedicata a questo tipo di interruzioni, e giunge direttamente su un piedino del microprocessore.
Un'interruzione hardware è quindi, una segnalazione di natura tipicamente **asincrona** (cioè la CPU non è a conoscenza del momento in cui l'interruzione si verificherà); di conseguenza, tale segnalazione, proveniente dall'esterno del processore, non è sincronizzata con le attività che la CPU sta svolgendo in quell'istante.
Esempi di interruzioni asincrone sono: la fine di un processo di stampa, la pressione di un tasto sulla tastiera, un'anomalia aritmetica del processo in esecuzione (ad esempio, un overflow, un underflow o una divisione per zero).
Un altro tipo di interruzione è quella **generata dal clock del sistema** che causa lo **scadere del *quanto* di tempo** del processo in esecuzione. Se, ad esempio, un quanto è pari a 100 cicli di clock, allo scadere di ogni 100 cicli viene generata un'interruzione hardware di "quanto scaduto". Queste interruzioni sono, però, di natura **sincrona,** poiché sono eventi interni al processore, previsti e sincronizzati con le attività del processore stesso.

Gestione delle interruzioni

Quando la CPU rileva la presenza di un'interruzione (hardware o software) deve:

- **completare** l'istruzione in corso del processo **P1** in esecuzione;
- **salvare** il contesto in cui stava operando **P1** (**cambio di contesto o context switch**). In particolare deve salvare in un'apposita area tutte le informazioni necessarie a descrivere la situazione raggiunta nell'istante immediatamente prima che si verificasse l'interruzione: il contenuto del *Program Counter* e di tutti i *registri* del processore che poi servono per poter riprendere l'esecuzione del processo interrotto (queste informazioni vengono salvate in un'area apposita nel Process Control Block o PCB del processo);
- **passaggio del modo di operare del processore** da ***modalità utente*** a ***modalità supervisore*** (privilegiata) per poter eseguire tutte le istruzioni e poter accedere a tutte le aree di memoria senza restrizioni;
- **modificare** lo stato di **P1** ad esempio da running a ready, aggiornando tutte le informazioni del PCB tra le quali il tempo usato del processore, e inserendo il processo nella lista dei processi in stato di ready;
- **eseguire** (in modalità privilegiata) una particolare routine per la gestione di quel particolare tipo di interruzione e per far questo viene forzato il contenuto del *Program Counter* inserendovi l'indirizzo di partenza della routine che gestisce l'interruzione invocata;
- **selezionare,** da parte del S.O. (schedulatore dei processi), il prossimo processo **P2** al quale deve essere assegnata la CPU per l'esecuzione o per la ripresa dell'esecuzione dall'istruzione successiva a quella in cui era stato interrotto;
- **aggiornare** il PCB del processo **P2** selezionato con modifica del suo stato da ready a running ed eliminare il processo dalla lista dei processi in stato di ready;
- **ripristinare** il contesto salvato del processo **P2** selezionato recuperando i valori dei registri e del Program Counter prima di riprendere l'esecuzione effettiva del processo;
- **passaggio del modo di operare del processore** da ***modalità supervisore*** a ***modalità utente****;*
- **Riprendere** l'esecuzione di **P2.**

Context switch

Il passaggio dell'esecuzione da un processo ad un altro *richiede quindi tempo* per il salvataggio ed il caricamento dei registri e delle mappe di memoria, aggiornamento di tabelle e liste ecc.
Questa operazione, chiamata **context switch,** avviene in diverse occasioni per motivi diversi; uno dei possibili eventi che comportano un context switch è l'esaurimento del ***quanto di tempo*** che è stato assegnato ad un processo in esecuzione. Possiamo allora fare delle considerazioni sulla **durata** del **quanto di tempo.**

La durata del **quanto di tempo** (time slice):

- **non deve essere troppo breve** per evitare: **molti context switch** tra i processi con la conseguente **riduzione dell'efficienza della CPU (overhead)**
- **non deve essere troppo lunga** per evitare: **tempi di risposta lunghi** per i processi interattivi con tempo di esecuzione breve che verrebbero altrimenti penalizzati dai processi che richiedono un tempo di esecuzione maggiore.

VALUTAZIONE DELLE PRESTAZIONI DI UN S.O.

Il processore viene usato in parte per *eseguire programmi utente* e in parte per *eseguire attività di gestione del S.O.*; in alcuni momenti può restare inutilizzato perché non ci sono operazioni da eseguire.

Naturalmente le prestazioni dei SO si valutano a parità di Hardware.
Alcuni parametri che possono servire per valutare l'**efficienza complessiva di un sistema** sono i seguenti.

Percentuale di attività

Misura il tempo in cui si mantiene "attiva" la CPU in quanto esegue i programmi utente e quelli di gestione del S.O.

$$PercentualeDiAttività = (TempoUtente + TempoDiSistema)*100 / TempoTotale$$

Ovviamente, più è elevata, meglio è.

Throughput

Valuta quanto lavoro è stato eseguito nell'unità di tempo e può essere misurato come rapporto tra il numero dei processi eseguiti e il tempo in cui il sistema è funzionante o tra il tempo in cui il processore esegue i programmi utente e il tempo in cui il sistema è funzionante.

$$Throughput = NumeroProcessiEseguiti / TempoTotale$$ processi al secondo

$$Throughput = TempoUtente / TempoTotale$$

Overhead

Misura quanto tempo di CPU si "perde" per attività non strettamente legate all'esecuzione dei programmi utente. Il tempo di sistema è il tempo in cui il processore è impegnato ad eseguire programmi di gestione del S.O.:

$$Overhead = TempoDiSistema / TempoTotale$$

Valuta il tempo che NON interessa l'utente. Questo valore deve essere basso.

La **percentuale di attività** si può ottenere come somma di ***Throughput + Overhead*** e calcolando la percentuale.