Il nucleo e la gestione dei processi

1) Programmi, processi e risorse

Fino a questo momento, la nostra idea di processo ha coinciso con quella di "programma in esecuzione".
In realtà tra i due c'è una netta differenza.

Il **programma** rappresenta la descrizione del procedimento logico *(algoritmo)* che deve essere eseguito per risolvere un determinato problema. Tale procedimento è descritto attraverso un idoneo formalismo *(linguaggio di programmazione)* che ne consente l'esecuzione da parte di un specifico computer.

Quando il programma è in esecuzione produce una *sequenza di eventi,* ossia esegue, una dopo l'altra, una serie di operazioni tra quelle che il computer stesso sa eseguire.

Un processo è un programma in esecuzione. Ogni volta che si lancia un'istanza di programma, si avvia un processo.

Il programma, quindi, essendo una descrizione di un algoritmo, è **un'entità statica,** mentre il processo, essendo un'istanza di tale programma, è **un'entità astratta e dinamica** e identifica l'attività del computer relativa all'esecuzione del programma. L'esecuzione di un programma (cioè un processo) può prevedere di seguire un determinato flusso logico, cioè solo alcune istruzioni di quel programma (si pensi all'esecuzione di un'istruzione di selezione o di ciclo) ma un'altra esecuzione dello stesso programma potrebbe seguirne un altro.
Ciò dipende tra l'altro dai dati forniti in input al programma, dal valore contenuto nelle variabili e da altri parametri.

Il programma in esecuzione è solo una componente del processo. Un processo, infatti, è caratterizzato da più elementi:

- **il programma (il codice)** che deve essere eseguito;
- **l'esecutore che lo esegue**, il computer;
- **i dati su cui opera il programma** (dati iniziali, intermedi e finali);
- **l'ambiente (di esecuzione)** nel quale opera il computer per eseguire il programma e le
  strutture dei dati di servizio che il sistema operativo utilizza per l'esecuzione (un program
  counter, uno stack e un'area di memoria riservata).

I processi, durante la loro evoluzione, utilizzano le risorse del computer. Ogni risorsa, in base al numero di processi che la possono utilizzare contemporaneamente, può essere

classificata come risorsa a molteplicità:

- **unaria**, quando la risorsa può essere utilizzata da un solo processo alla volta. Una stampante è un esempio di risorsa unaria in quanto un solo processo può utilizzarla sino alla fine della stampa, dopodiché essa viene assegnata al primo processo in coda d'attesa;
- **finita**, quando una risorsa può essere utilizzata contemporaneamente da N processi. In questo caso si parla anche di risorsa a molteplicità N. Un esempio di risorsa di questo tipo è un canale condiviso;
- **infinita**. La molteplicità si dice infinita quando la risorsa può essere utilizzata da un numero qualsiasi di processi. Ne è un esempio un file in sola lettura.

Ogni risorsa può essere assegnata a un processo in modo:

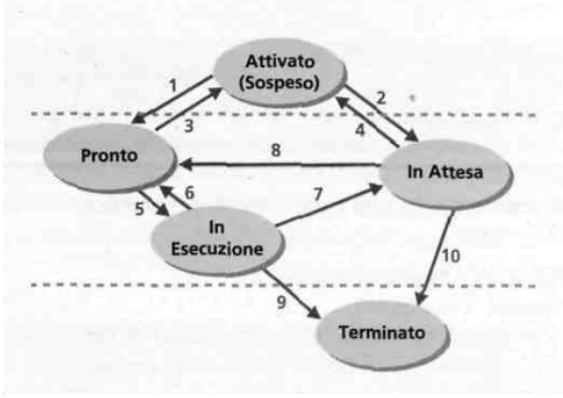
- **statico**. La risorsa è assegnata per tutta la durata del processo. Di solito tale assegnazione avviene quando il sistema operativo conosce quali risorse assegnare a un processo al momento della sua nascita. Tali risorse, generalmente, vengono rilasciate al termine del processo;
- **dinamico**. Si ha quando un processo richiede e rilascia una risorsa durante l'esecuzione.

Un sistema operativo ha, quindi, un certo numero di risorse hardware e software da gestire e un certo numero di processi che vengono continuamente creati e mandati in esecuzione su risorse assegnate staticamente o dinamicamente durante la loro esecuzione.

2) Lo stato di un processo

Un processo in esecuzione è soggetto a transizioni di stato definite in parte dalla corrente attività del processo stesso e in parte da eventi esterni asincroni con la sua esecuzione.

Ogni processo all'interno di un sistema operativo è descritto mediante una struttura dati, nota come descrittore di processo o PCB (Process Control Block), che raccoglie tutte le informazioni necessarie al sistema per identificare il processo, individuare il suo stato interno e gestirne le risorse. La tabella dei descrittori di processo contiene l'elenco dei descrittori di tutti i processi presenti in quell'istante nel sistema.
Fotografando il sistema in un determinato istante, è possibile individuare i differenti stati in cui ogni processo si viene a trovare.



Il descrittore di un processo appena creato viene inserito nella tabella dei descrittori dei processi. Lo stato del processo è **Attivato** e il processo è pronto per essere preso in considerazione dal nucleo. Un processo può ritornare in questo stato quando è **Sospeso**, quando, cioè, non può temporaneamente competere per l'uso della CPU o di altre risorse.

Dallo stato Attivato, il processo può transitare:

- allo stato **Pronto** quando il nucleo recupera tutte lei risorse di cui il processo necessita, tramite la CPU o quando viene riattivato dopo una sospensione (1);
- allo *stato* **In Attesa** quando il nucleo non riesce a recuperare tutte le risorse di cui il processo necessita o quando viene riattivato dopo una sospensione che lo aveva trovato in questo stato (2).

Nello stato Pronto il processo attende che gli sia assegnata la CPU e il suo descrittore viene inserito in una coda di processi detta **ready list** o **lista dei processi pronti**.

Da questo stato il processo può transitare:

- allo stato **Sospeso** quando, trovandosi nella ready iist, viene momentaneamente allontanato dalla competizione per !a CPU, per migliorare l'efficienza (3);
- allo stato **In Esecuzione** quando gli viene assegnata la CPU, secondo la politica di schedulazione dei processi in ready list (5).

Nello stato In Esecuzione sono eseguite le azioni elementari di cui è composto il processo. Da questo stato il processo può transitare:

- allo stato **Pronto** quando deve prelasciare la CPU. Questo può avvenire perché è scaduto il tempo a disposizione (time slice) oppure, ad esempio, per dar precedenza a un processo con priorità maggiore (6);
- allo stato **in Attesa** quando ha effettuato una richiesta di utilizzo di una risorsa momentaneamente occupata ed è in attesa di un messaggio da parte di un altro processo o genericamente del verificarsi di un particolare evento (7).
- allo stato **Terminato** quando la sua esecuzione è andata a buon fine (9).

Nello stato **In Attesa** il processo è in attesa di riprendere l'esecuzione o di essere reinserito nella ready list. Da questo stato il processo può transitare:

- allo stato **Pronto** quando la risorsa che ha richiesto è disponibile oppure quando l'evento che stava aspettando si è verificato (8);
- il processo passa da *In Attesa* a **Sospeso** quando viene momentaneamente allontani dalla competizione per le risorse (4);
- allo stato **Terminato** quando l'attesa supera il periodo massimo stabilito. In questo caso si suppone che la risorsa richiesta non sia più disponibile o l'evento non possa mai verificarsi (10).

Nello stato Terminato vengono rilasciate tutte le risorse impegnate e il descrittore rimosso dalla tabella dei descrittori e distrutto. Vengono inoltre terminati anche tutti i processi figli che eventualmente sono stati generati.

La terminazione può avvenire in modalità normale o anomala (quando, ad esempio. processo ha effettuato una richiesta di accesso a una risorsa non conforme ai suoi diritti (all'accesso a quella risorsa si è verificato un errore).

3) Processi e processore

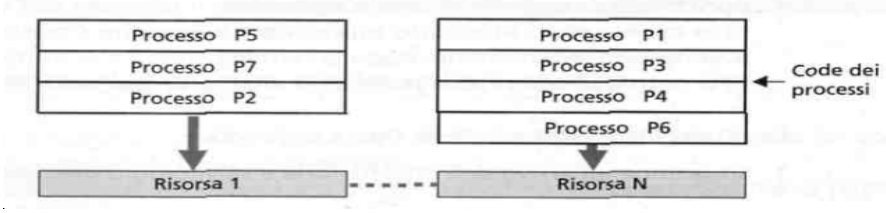
I compiti del nucleo, genericamente indicati con **gestione dei processi** e, di riflesso, **gestione del processore** (in quanto risorsa privilegiata assegnata ai processi) sono:

- **creare** un processo;
- **gestire** la transizione dallo stato Pronto (**schedulatore a basso livello**) allo stato *In Esecuzione*. Questa attività consiste nella gestione della ready list e prende il nome di schedulazione a basso livello.
- **terminare** un processo;
- fornire primitive, al livello superiore, per la **comunicazione** tra processi;
- fornire primitive, al livello superiore, per la **sincronizzazione** dei processi e per il loro passaggio da uno stato all'altro.

La politica (algoritmo) con il quale viene effettuata la scelta è detta politica di scheduling ed ha il compito di selezionare il processo da mandare in esecuzione, mentre il dispatcher, o schedulatore a basso livello, assegna il processo alla CPU.
Lo scheduling deve:

- *garantire* l'assegnazione della risorsa a tutti i processi che ne facciano richiesta;
- *minimizzare* i tempi medi di attesa dei processi in coda;
- tener conto della *priorità* dei processi.

L'ultimo punto si riferisce al fatto che l'ordine in cui alcuni processi in coda vengono serviti dipende dalla "priorità" assegnata a ciascuno di essi. Tale priorità non tiene conto solo dell'ordine di arrivo, ma dipende anche dalla finalità di massimizzazione del servizio e, in generale, dall'obiettivo di ottimizzare il lavoro complessivo del computer.



4) Passaggi di stato e interruzioni

Affinchè 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 nel registro istruzioni della CPU 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 con funzioni di CPU.
Un'interruzione 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.

5)Gestione delle interruzioni

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

1) **completare** l'istruzione in corso del processo in esecuzione;
2) **salvare** il contesto in cui stava operando (cambio di contesto). In particolare deve salvare in un'apposita area il **contesto minimo** composto da:
   - i suoi *registri* interni;
   - il *Program Counter;*
   - il registro di Stato (Status *Register).*

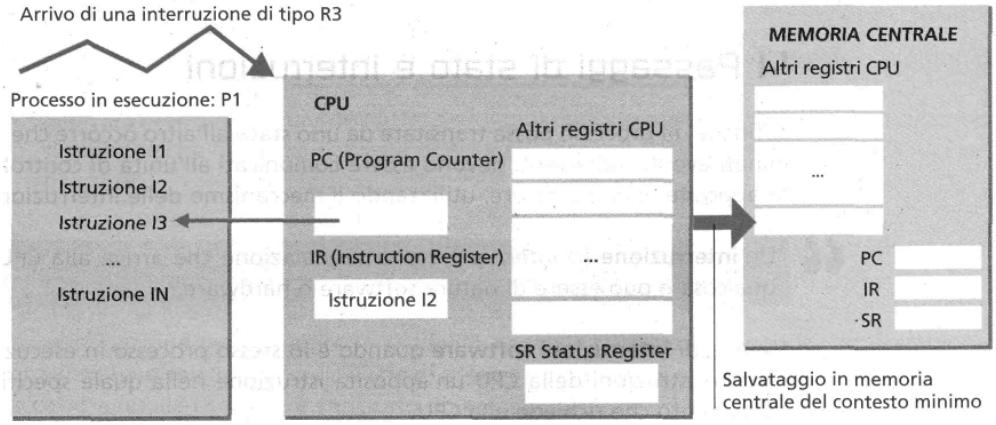
Devono essere salvate, quindi, tutte le informazioni necessarie a descrivere la situazione

raggiunta nell'istante immediatamente prima che si verificasse l'interruzione;

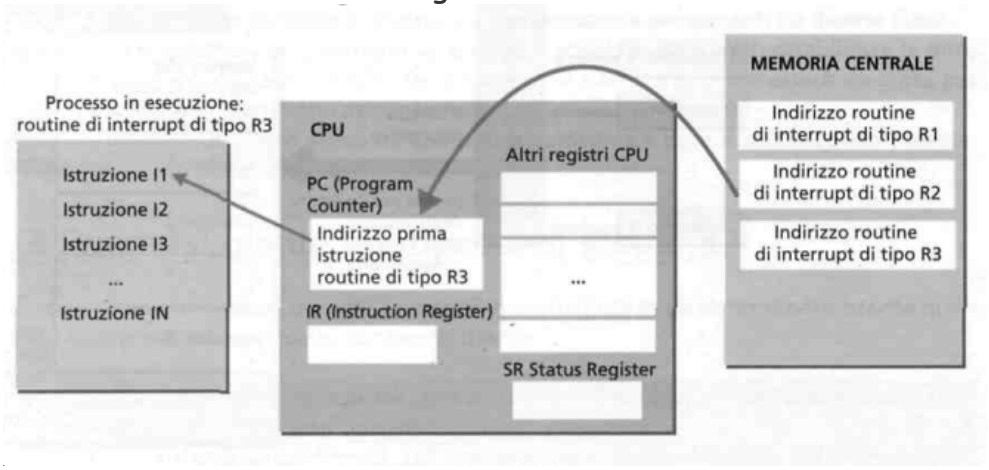
3) **eseguire** una particolare routine per la gestione di quel tipo di interruzione, detta **routine di interrupt;**
4) **ripristinare** il contesto salvato e **riprendere** il processo dall'istruzione successiva a quello in cui era stato interrotto (ripristinare il *Program Counter).* Non è detto che questo avvenga immediatamente dopo la terminazione della routine di interrupt, in quanto la CPU potrebbe essere assegnata a un altro processo della ready list.

La situazione è illustrata nelle figure seguenti.

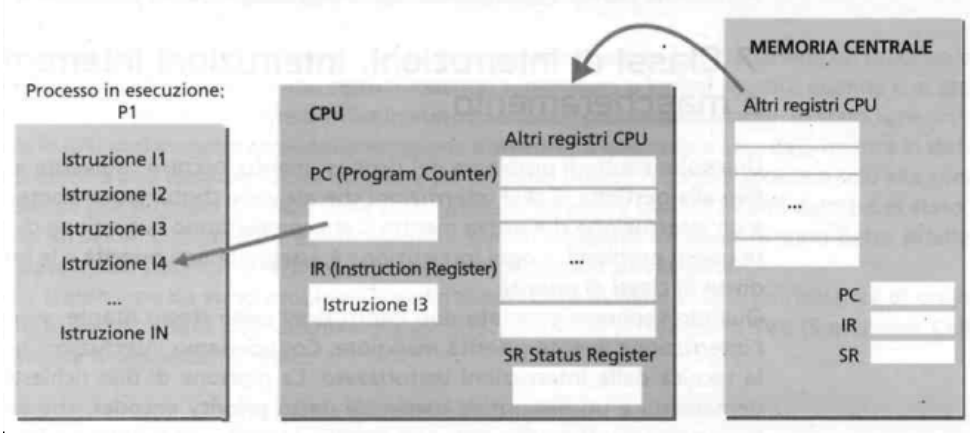
Situazione all'arrivo dell'interruzione e salvataggio del contesto minimo:



Esecuzione della routine di gestione dell'interruzione:



Ripristino del contesto minimo e ripresa del precedente processo in esecuzione.



6) Riconoscimento delle interruzioni

La CPU si accorge che si è verificata un'interruzione hardware attraverso un'apposita linea coliegata a un suo piedino, la **linea INT**.
Essendo un'unica linea collegata a un unico piedino, nasce il problema di poter distinguere il dispositivo dal quale l'interruzione è stata generata. Esistono due tecniche per il riconoscimento dell'interruzione:

- una tecnica software, detta **polling;**
- una tecnica hardware, detta delle **interruzioni vettorizzate**.

La tecnica del **polling** richiede che sia la CPU a individuare il dispositivo che ha generato l'interrupt, interrogando tutti i dispositivi, fino ad avere una risposta affermativa dal dispositivo interessato. Il tempo richiesto per la scansione però può essere considerevole e spesso si preferisce la seconda tecnica.
La tecnica **delle interruzioni vettorizzate** richiede, invece, che siano gli stessi dispositivi a farsi riconoscere, emettendo un **codice identificativo** sul bus indirizzi.

7) Classi di interruzioni, interruzioni interrompibili e mascheramento

Una volta risolto il problema del riconoscimento, occorre risolvere quello relativo alla gestione di due interruzioni che arrivano contemporaneamente e quello relativo a un'inter-ruzione che arriva mentre si sta completando la gestione di un'altra. Per risolvere questi problemi, a ogni interruzione è assegnata una priorità e le interruzioni sono suddivise in classi di priorità.
Quando vengono generate due interruzioni nello stesso istante, viene servita per prima l'interruzione avente priorità maggiore.
Mentre si sta servendo un'interruzione, un altro componente hardware ha il compito d mascherare le richieste di altre interruzioni, che potrebbero arrivare da un altro dispositivo con la stessa priorità (cioè della stessa classe), non facendo arrivare il segnale INT al piedino della CPU.
Il componente che contiene la logica del mascheramento emette gli opportuni segnali per

abilitare o disabilitare le interruzioni provenienti da diverse classi. Mentre si sta servendo un'interruzione, quindi, è possibile abilitare o disabilitare le altre interruzioni in arrivo. Quando un'interruzione deve necessariamente essere eseguita per intero, si parla di **interruzione non interrompibile**.
Quando un'interruzione in arrivo ha una priorità altissima e deve essere eseguita necessariamente, si dice che è **non mascherabile**.

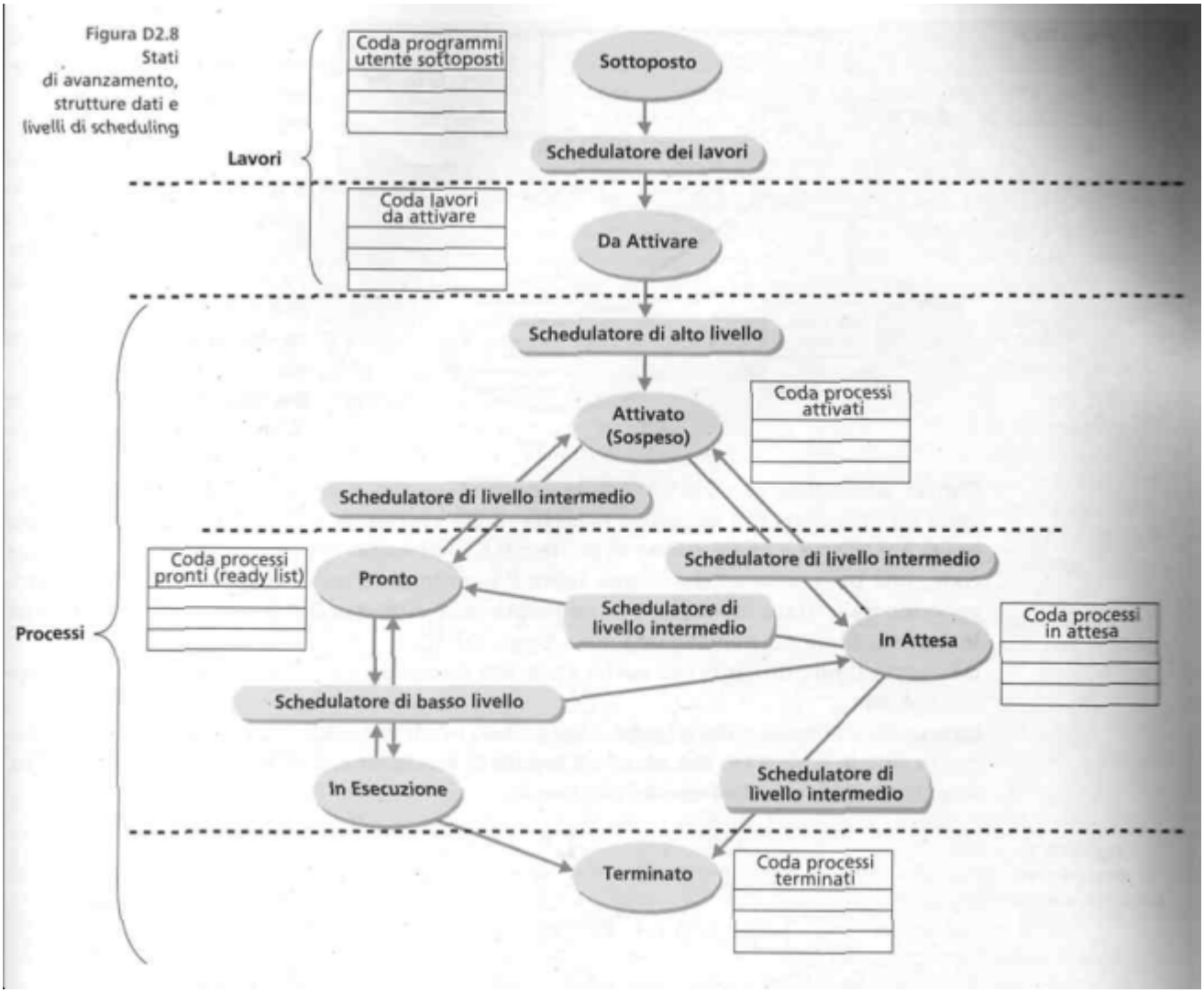
8) Scheduling dei lavori e scheduling dei processi

Quando sono presenti più processi contemporaneamente, spesso accade che essi siano in competizione per l'uso delle risorse del sistema. È quindi indispensabile che ci sia una politica di assegnazione delle risorse necessarie ai processi: lo **scheduling**.

Una richiesta dell'utente non diventa subito un processo, ma transita prima in uno stato detto stato **submit** o **sottoposto** o **dei lavori sottoposti**, nel quale transitano le richieste degli utenti sotto forma di lavori o **job**, cioè programmi correlati di informazioni (come il nome del proprietario, i diritti di accesso sulle risorse ecc.).
Dallo stato Sottoposto, uno schedulatore, detto **schedulatore dei lavori**, fa transitare un lavoro nello stato *Da Attivare,* dopo che i controlli sui diritti di accesso alle risorse richieste sono andati a buon fine, assegnandogli una priorità e una lista di risorse necessaria per l'esecuzione. Ora il lavoro è pronto a essere trasformato in processo e si parlerà di **scheduling dei processi**.
Occorre, però, distinguere tre diversi livelli di scheduling dei processi:

- **scheduling di basso livello** o **dispatcher** (già visto parlando della gestione della ready list, ovvero della politica con la quale si sceglie il processo, tra quelli nello stato *Pronto,* cui assegnare il processore (e di conseguenza far transitare allo stato *In Esecuzione*);
- **scheduling di alto livello**, detto genericamente **scheduler**. Consiste nella scelta del programma di cui avviare il processo. La scelta viene effettuata in base alla richiesta *(statica)* di risorse e alla priorità assegnatagli;
- **scheduling di livello intermedio**. È la politica di scelta di un processo, tra quelli che sono già presenti, in stato di *Attivato* o di *In Attesa,* in base all'evoluzione dei processi, alle loro richieste dinamiche (cioè a tempo di esecuzione) di risorse, in particolare della memoria e alla velocità con cui avanzano.

Riconsideriamo, ora, gli stati di avanzamento di un processo e, tenendo conto che ogni processo gestore possiede strutture dati di tipo FIFQ, tali da consentire la memorizzazione dei descrittori dei processi in una coda, possiamo riassumere, nella figura D2.8, gli stati di avanzamento, le strutture dati e i livelli di scheduling.



Dalla figura notiamo che:

- lo schedulatore dei lavori preleva i programmi utente presenti nella coda "programmi utenti sottoposti" e li inserisce nella coda "lavori da attivare", assegnando loro una priorità (statica) e una lista di risorse richieste;
- lo schedulatore di alto livello preleva da questa coda i processi da attivare e li inserisce nella coda "processi attivati";
- lo schedulatore di livello intermedio, in base alla disponibilità delle risorse richieste, fa transitare il descrittore nella coda dei processi pronti oppure in quella dei processi in attesa;
- lo schedulatore di basso livello si occupa della transizione dallo stato *Pronto* a quello *In Esecuzione,* facendo transitare il descrittore di processo dalla ready list alla coda dei processi in esecuzione.

Abbiamo visto che un processo può perdere l'assegnazione della CPU per diversi motivi. Lo scheduling può essere effettuato in **base** a due politiche fondamentali:

- **scheduling senza diritto di prelazione** (*non preemptive scheduling*): quando il processo può venire sospeso solo in quanto richiede esplicitamente una

funzione del sistema operativo;

- **scheduling con diritto di prelazione** (*preemptlve scbedullng*): quando il processo può essere obbligato a lasciare lo stato di esecuzione, ad esempio a causa della terminazione del quanto di tempo.

9) Lo Scheduling a basso livello

La più semplice tra le politiche di scheduling a basso livello è quella dettata **dall'algoritmo FCFS** (*First Come First Served*) che assegna il processore ai processi nell'ordine in cui si trovano nella coda dello stato di *Pronto* (ossia al processo pronto in attesa da più tempo) e lo mantengono sino alla terminazione o alla sospensione. La coda dei processi pronti è gestita secondo la tecnica FIFO (*First In First Out*) e l'assegnazione del processore non viene mai revocata se non per motivi di protezione dei processi difettosi.
La politica di gestione FCFS è particolarmente inadatta alla gestione di processi server e a quelli che richiedono molti trasferimenti e su periferiche diverse poiché queste operazioni di I/O finirebbero per provocare attese molto lunghe, inattività del processore e scarsa interattività con l'utente.
Per questo motivo la politica FCFS può essere considerata una politica di scheduling senza diritto di prelazione valida solo per i sistemi batch.
Una politica di scheduling con diritto di prelazione progettata appositamente per i sistemi in time sharing è la **Round Robin**.
Quando scade il quanto di tempo (time – slice), il processo viene fatto transitare nello stato di *Pronto,* in attesa che venga nuovamente riconsiderato ossia viene riposto in coda. In questo modo a tutti i processi pronti viene ciclicamente assegnata fa CPU, realizzando un parallelismo apparente. La soluzione è semplice da implementare in quanto utilizza la sola coda dei processi pronti, ma ha lo svantaggio di non implementare la priorità tra i processi.