DESCRIZIONE, CONTROLLO e SCHEDULAZIONE dei PROCESSI

In ambienti multi-programmati

PROCESSO Process

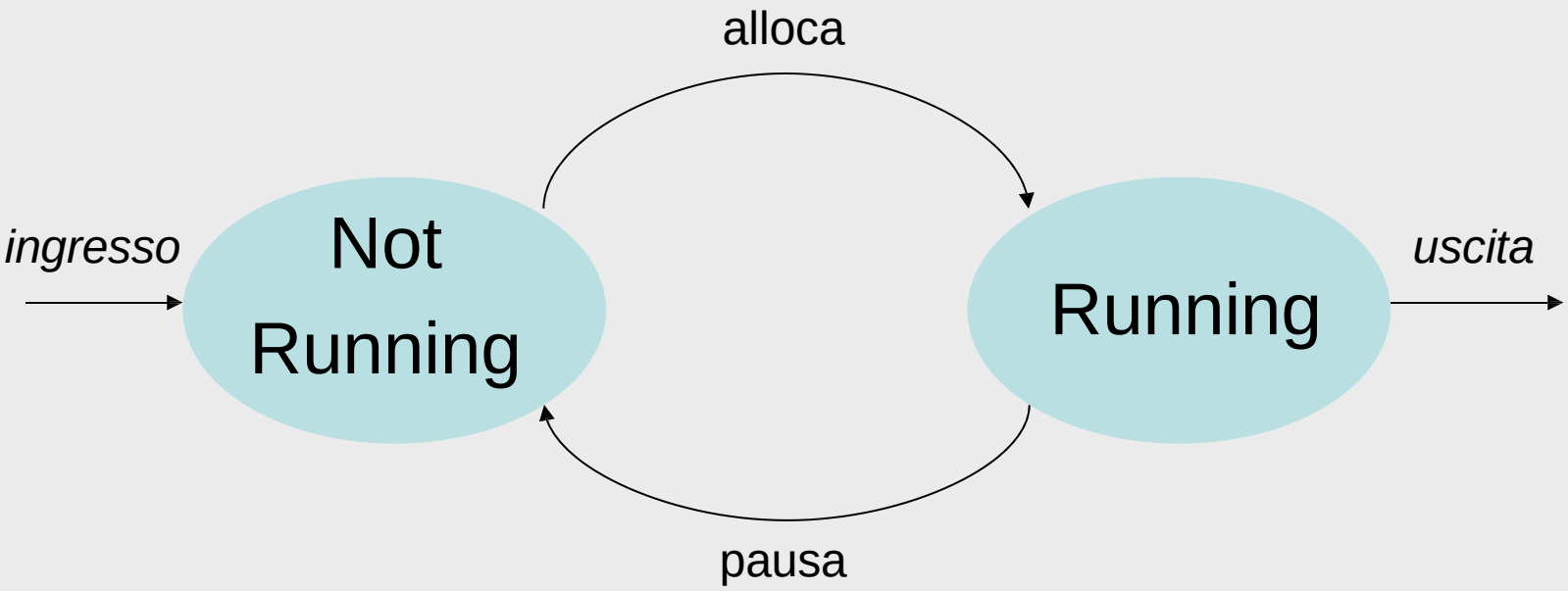
- Una entità che deve essere eseguita su una CPU
- Una attività caratterizzata dall'esecuzione di una sequenza di istruzioni, uno stato corrente e un set di istruzioni di sistema

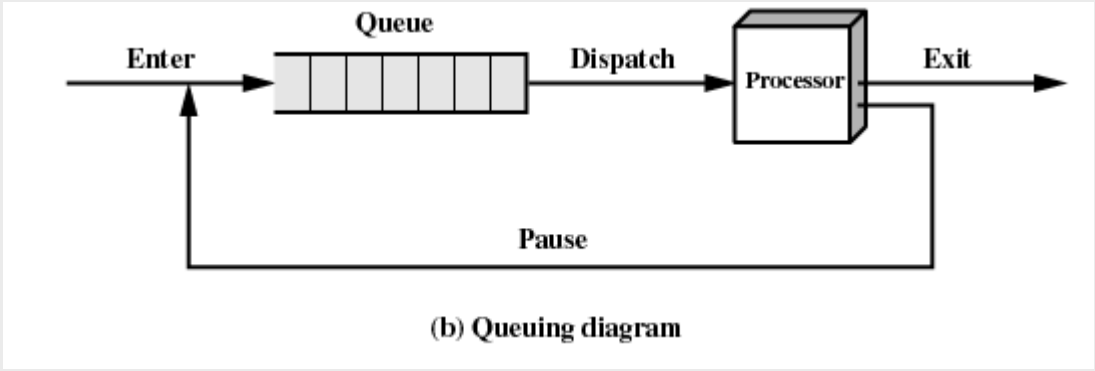
Componenti

- Programma
  - codice eseguibile
- Dati
  - variabili
  - spazio di lavoro
  - Buffer
- Contesto di esecuzione (info necessarie al SO per gestire il processo)
  - contenuto dei registri della CPU
  - Priorità, stato di attesa su un dispositivo di I/O

Modello a due stati

Compito principale di un SO è il controllo dell'esecuzione dei processi





(b) Queuing diagram

Descrizione dei Processi

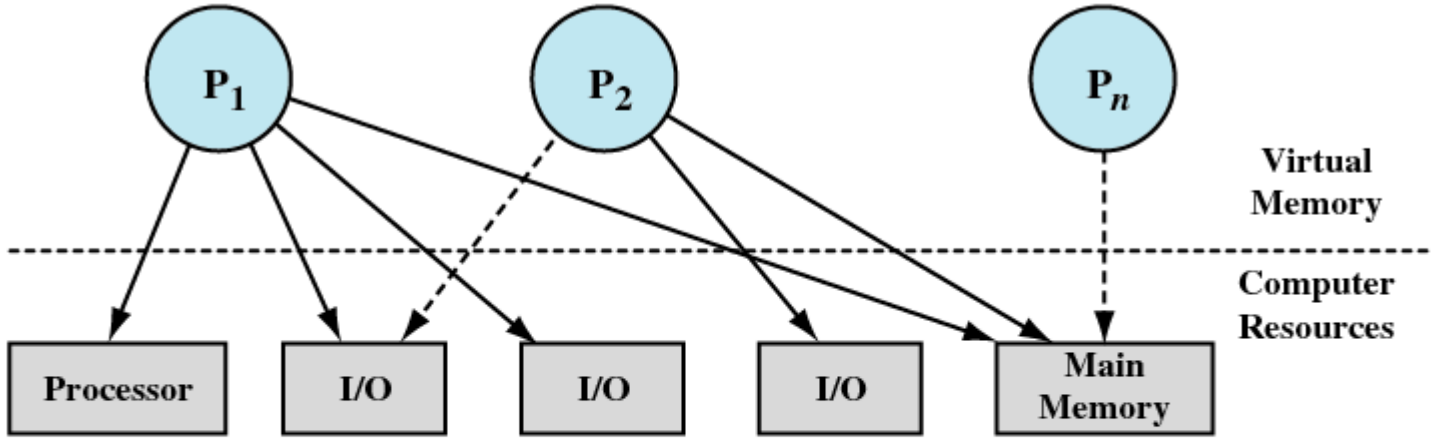


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

Il SO ha necessità di uno strumento per gestire i processi, che tenga traccia di tutte le info disponibili

Process Control Block PCB

Descrittore di processo attraverso il quale il SO può gestirlo

IDENTIFICATORE DEL PROCESSO

Process IDentification (PID) - Valore numerico
(PID del processo genitore)

INFO SULLO STATO DEL PROCESSORE

Registri Dati visibili all'utente (dipendono dall'arch. del calcolatore)

Registri di controllo e di stato

PC – indirizzo della prossima istruzione da eseguire
Registri di stato – includono i flag per l'abilitazione degli interrupt
Registri che contengono codici di condizione (segno, overflow, ecc.)

Puntatori allo stack

Usato per procedure e funzioni

Process Control Block

INFO DI CONTROLLO DEL PROCESSO

Schedulazione e info di stato

- stato del processo (running, ready, blocked, ecc.)
- priorità nelle code di scheduling
- info correlate alla schedulazione (tempo di attesa, tempo di esecuzione, ecc.)
- evento (del quale è in attesa)

Strutturazione dati

- puntatori ad altri processi (figli/padre o per l'implementazione di code)

Comunicazione tra Processi

- flag, segnali e messaggi per la comunicazione

Privilegi

- in relazione all'uso di memoria, dispositivi, ecc.

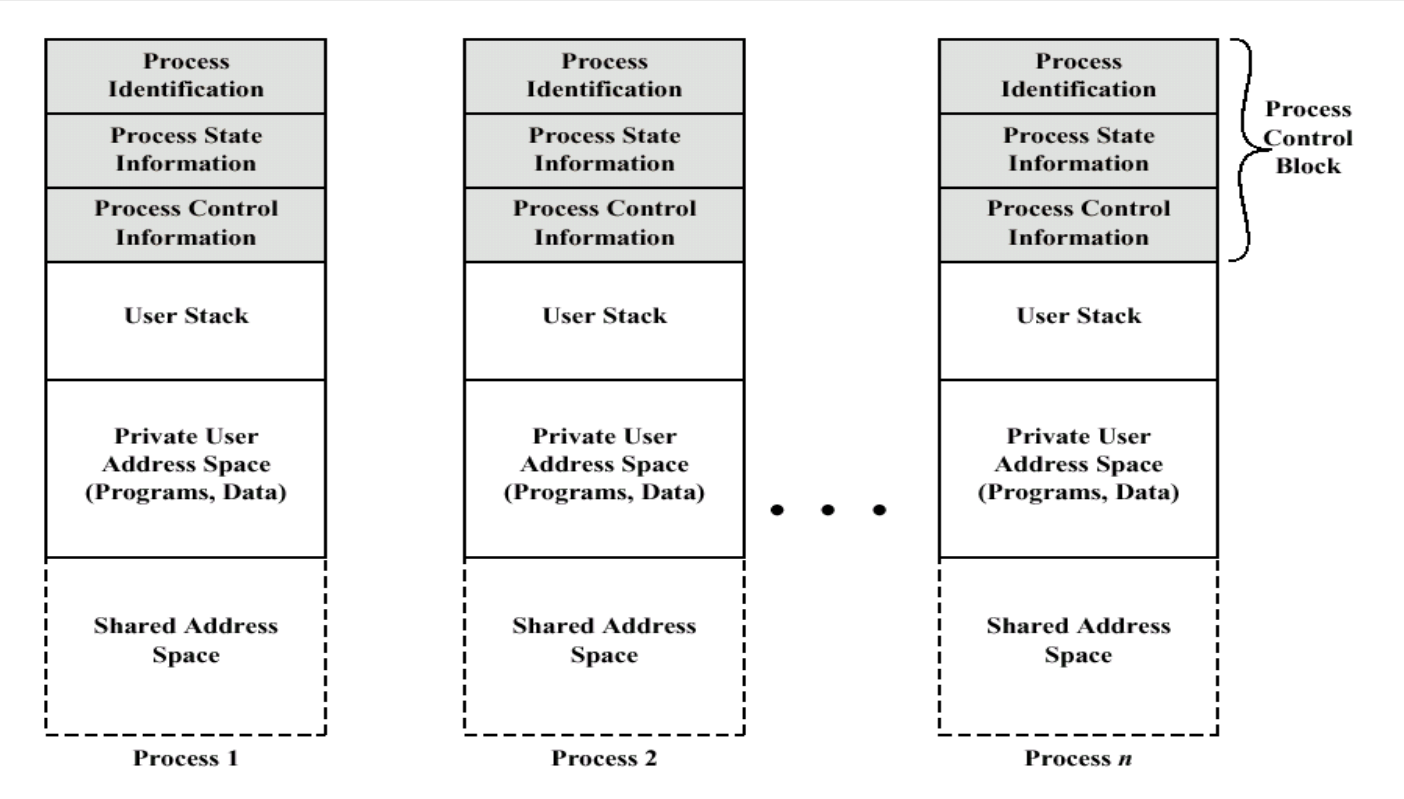
Gestione memoria

- limiti di memoria: insieme degli indirizzi accessibili (base, indice)

Contabilizzazione delle risorse

- risorse controllate dal processo (lista file aperti, lista dispositivi I/O) e loro storia

Immagine dei processi in memoria



Nell'esempio le immagini occupano locazioni contigue di memoria, in una implementazione reale ciò può non essere vero. Dipende dalla politica di gestione della memoria

Creazione e Terminazione dei Processi

Process

Eventi che portano alla **creazione** del processi

1. Richiesta da terminale (un utente accede al sistema)
2. Il SO genera un processo sulla base della richiesta di un processo utente
   Es: stampa – il processo generatore continua la sua esecuzione
3. Un processo utente genera un nuovo processo
   Processo padre - Processo figlio (generazioni)
   Es. sfruttare il parallelismo
   processo server che genera diverse istanze per gestire diverse richieste

Eventi che portano alla **terminazione** del processi

1. Terminazione normale (end)
2. Uscita dell'utente dalla applicazione
3. Superamento del tempo massimo
4. Memoria non disponibile
5. Violazione dei limiti di memoria
6. Fallimento di una operazione (aritmetica - I/O)
7. Terminazione del genitore
8. Richiesta del genitore

Modello a 5 stati Process

Nel modello a 2 stati, lo stato “not-running” include 2 possibilià:

- Not-running
  - ready to execute
- Blocked
  - In attesa di I/O

  Il Dispatcher non può semplicemente scegliere il processo da più tempo in attesa, poichè esso potrebbe essere in attesa di trasferimento I/O

Lo stato “not-running” viene suddiviso in due stati: ready e blocked

- New
- Ready
- Running
- Blocked (Waiting)
- Exit (Terminated)

Modello a 5 stati

Stati e Transizioni

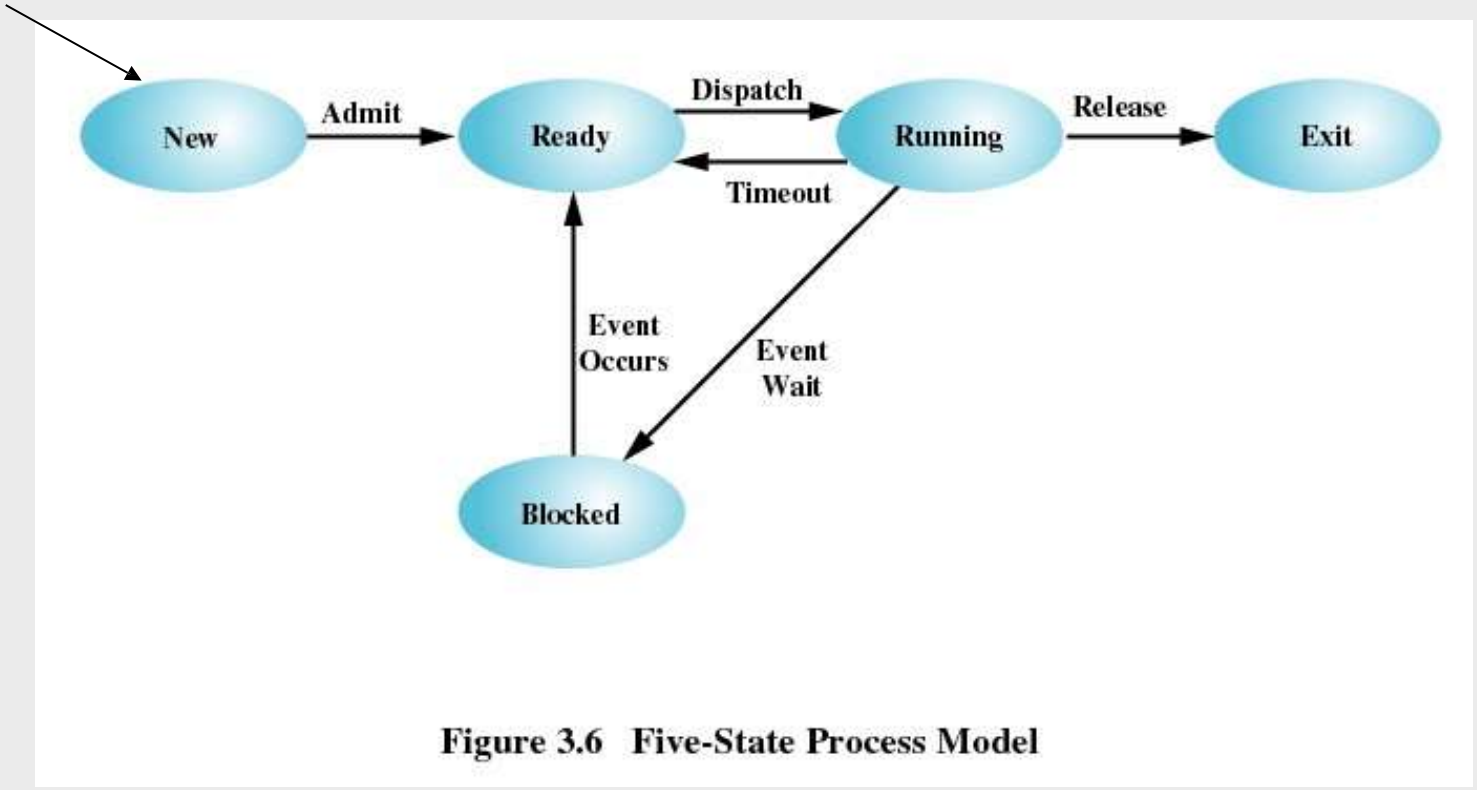


Figure 3.6 Five-State Process Model

***New**: SO associa al processi il PID, alloca e costruisce le tabelle per la gestione del processo. NB: il processo non'è caricato in memoria*

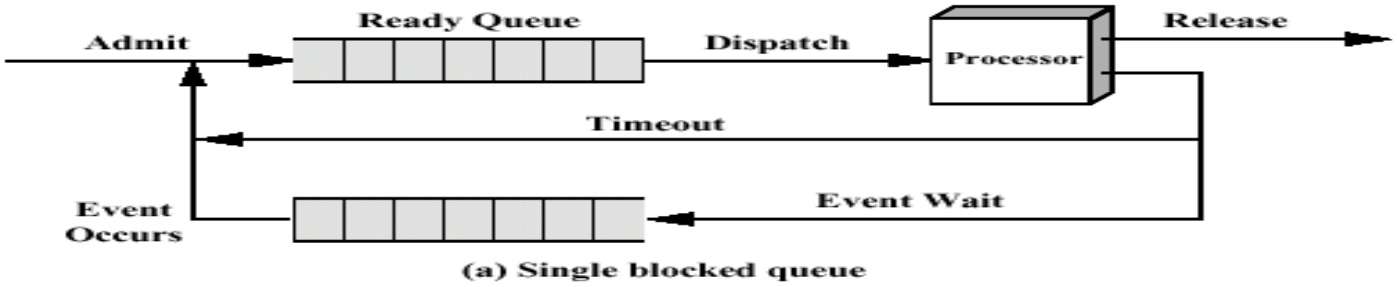
***Exit**: rilascio delle risorse. Il SO può mantenere alcune info (es. contabilità)*

Ulteriori Transizioni:

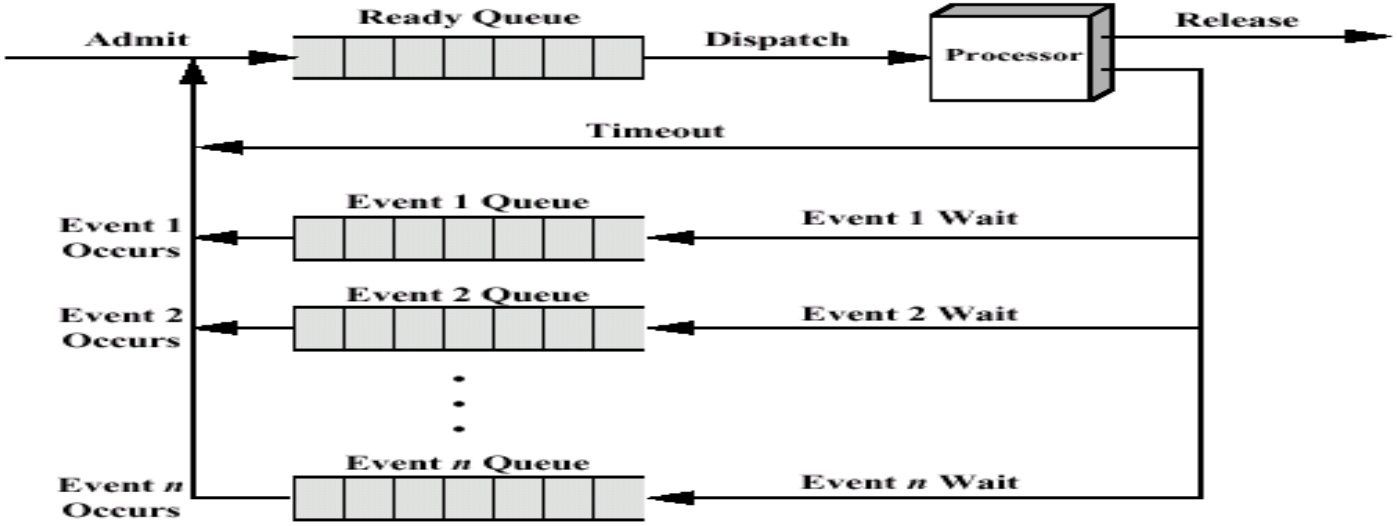
ready -> exit (genitore che termina figlio)

blocked -> exit (come sopra)

Strategie di accodamento



(a) Single blocked queue



(b) Multiple blocked queues

Context Switch

Riguarda il passaggio della CPU ad un nuovo processo

Cause:

- Clock interrupt
  - Il processo ha terminato il tempo a sua disposizione (torna in ready)
- I/O interrupt
  - Una operazione di I/O è terminata, il SO sposta il processo in attesa di tale evento da blocked a ready e decide se riprendere l'esecuzione del processo precedente
- Memory fault
  - L'indirizzo di memoria generato è sul disco (memoria virtuale): deve essere portato in RAM. Il SO carica il blocco, nel frattempo il processo che ha generato la richiesta è in blocked, al termine del trasferimento andrà in ready
- Trap
  - Errore di esecuzione (il processo potrebbe andare in exit)
- Supervisor call
  - Es.: file open, il processo utente va in blocked

Context Switch

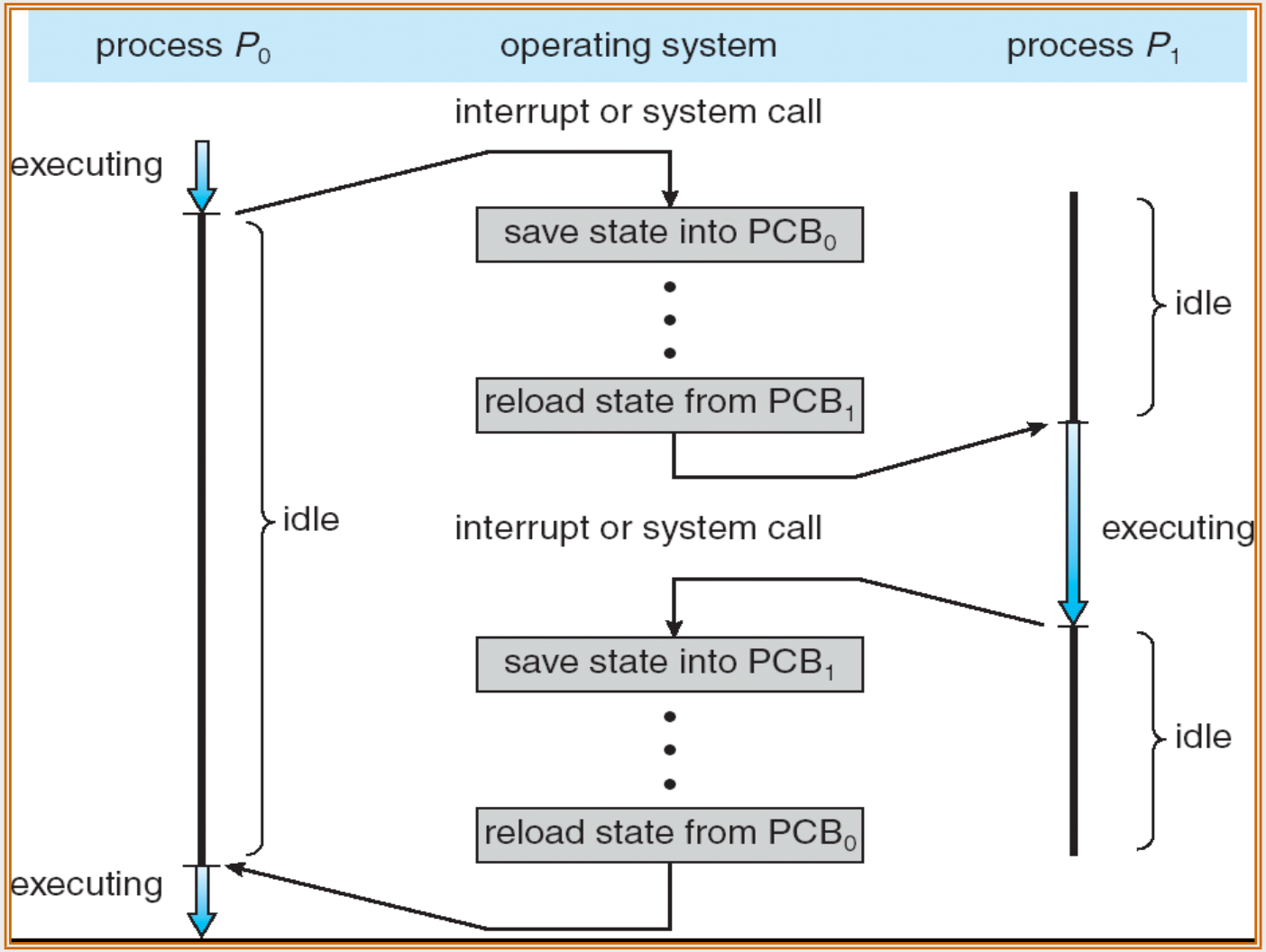
Operazioni svolte dal SO in modalità supervisor al momento del cambio di processo in stato di running:

- Salvataggio del contesto del processo che abbandona la CPU (valori dei registri della CPU: pc, psw, reg, ecc.)
- Cambio del valore di stato nel PCB (running -> [ready, blocked, exit])
- Spostamento del PCB in nuova coda (ready o blocked) o deallocare le sue risorse (exit)
- Aggiornamento delle strutture dati gestione memoria (area dello stack)

- selezione di nuovo processo per lo stato running (dispatcher)
- aggiornamento del suo stato nel PCB
- ripristino del contesto

- Context-switch time è *overhead*; il sistema non svolge nessun compito utile (all'utente)
- Il tempo dipende dalla complessità del SO e dall'hardware

Context Switch



Gestione dei Processi

Modi di esecuzione dei Processi:

- Modo Utente: esecuzione di processi utenti
- Modo Sistema o Kernel o Controllo: esecuzione di istruzioni che hanno come scopo:
  - Gestione dei Processi
    - Creazione e terminazione
    - Schedulazione
    - Cambio di contesto
    - Sincronizzazione
    - PCB
  - Gestione della Memoria
    - Allocazione
    - Trasferimento disco/RAM e vicersa
    - Gestione paginazione, segmentazione
  - Gestione I/O
    - Gestione buffer
    - Allocazione canali I/O
  - Supporto
    - Gestione Interruzioni
    - Contabilità

Creazione dei Processi

1) Assegnare al processo un PID unico;
   aggiungere una entry level alla tabella dei processi

2) Allocare lo spazio per il processo e per tutti gli elementi della sua immagine (PCB, User Stack, Area di memoria dati e istruzioni, aree condivise). Info valutate per difetto, indicazione dell'user, dal processo padre.

   Es. Address space
   - Child duplicate of parent
   - Child has a program loaded into it

   Possibilità di esecuzione:
   - *Processo padre e figlio sono concorrenti*
   - *Il padre attende la terminazione del figlio*

3) Inizializzazion del PCB
   stato del processore = 0
   PC = prossima istruzione
   puntatori allo stack…
   stato = ready (ready-suspended)

4) Inserimento nella coda ready

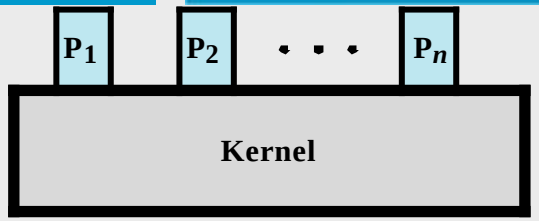
5) Estende le strutture al fine della fatturazione o delle statistiche.

Terminazione dei Processi

- Il processo termina poichè ha eseguito l'ultima istruzione, chiede al SO di cancellarlo
  (**exit**)
  - Un processo figlio può riportare dati al padre (in **wait**)
  - Le risorse del processo sono deallocate

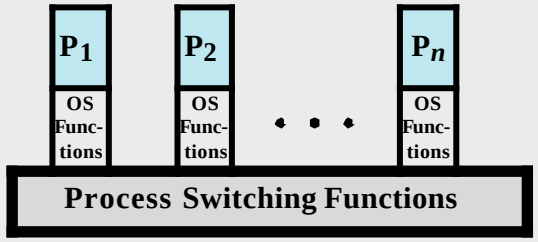
- Un processo padre può uccidere unprocesso figlio (**abort**)
  - Il figlio ha ecceduto l'uso di risorse
  - Il task del figlio non è più richiesto
  - Se un genitore termina, in base all'applicaziione alcuni SO non consentono che i figli rimangano in esecuzione: *cascading termination*
    - *Es: terminazione di firefox causa la terminazione di tutte le schede ad esso associate*

Modalità di esecuzione del SO



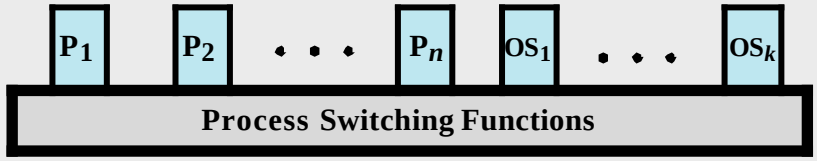
(a) Separate ke rnel

- Kernel separato dai processi (obsoleto)
  - Occupa una regione di memoria specifica
  - Ha il proprio stack
  - Eseguito come entità separata



(b) OS functions execute within user processes

- All'interno dei processi utente sono presenti anche programmi, dati e stack del kernel
  - Quando vi è una chiamata a SO, il Kernel cambia il modo di esecuzione (salvataggio contesto utente, modo utente -> kernel) NB: non c'è cambio di processo (non interviene lo schedulatore) poichè la funzione del kernel è nel processo utente



(c) OS functions execute as separate processes

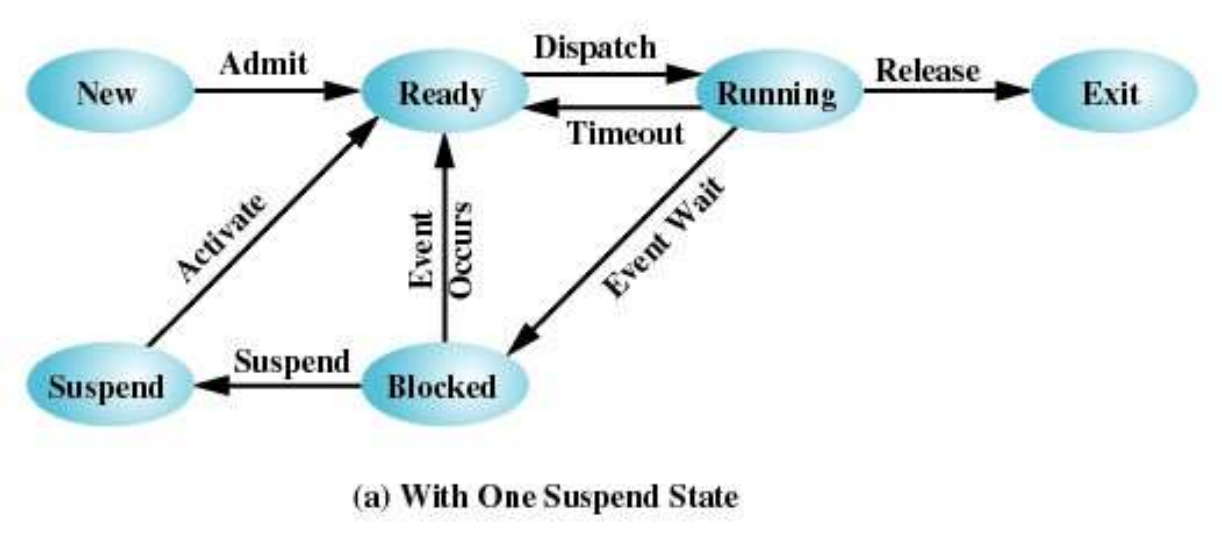
- SO basato su processi
  - Processi di sistema
  - Pro: interfacce pulite e semplici tra i moduli
  - Utile in sistemi multi processore

Evoluzione del modello a 5 stati

- Nonostante la memoria Virtuale, un programma per essere eseguito deve essere in RAM
- Con elevata probabilità tutti i processi in memoria restano in attesa di operazioni di I/O

- Processore inattivo (molto più veloce di I/O)
- **Soluzioni**
  - espandere la memoria
    - Costo
    - Poco efficiente (programmi sempre più grandi)
  - effettuare lo swapping: spostare un processo dalla RAM al disco
    - introduzione nuovo stato - stato suspend
    - NB: lo swapping è una ulteriore op. di I/O, ma in generale è la più rapida tra le op. di I/O

Evoluzione del modello a 5 stati
Modello a 6 stati



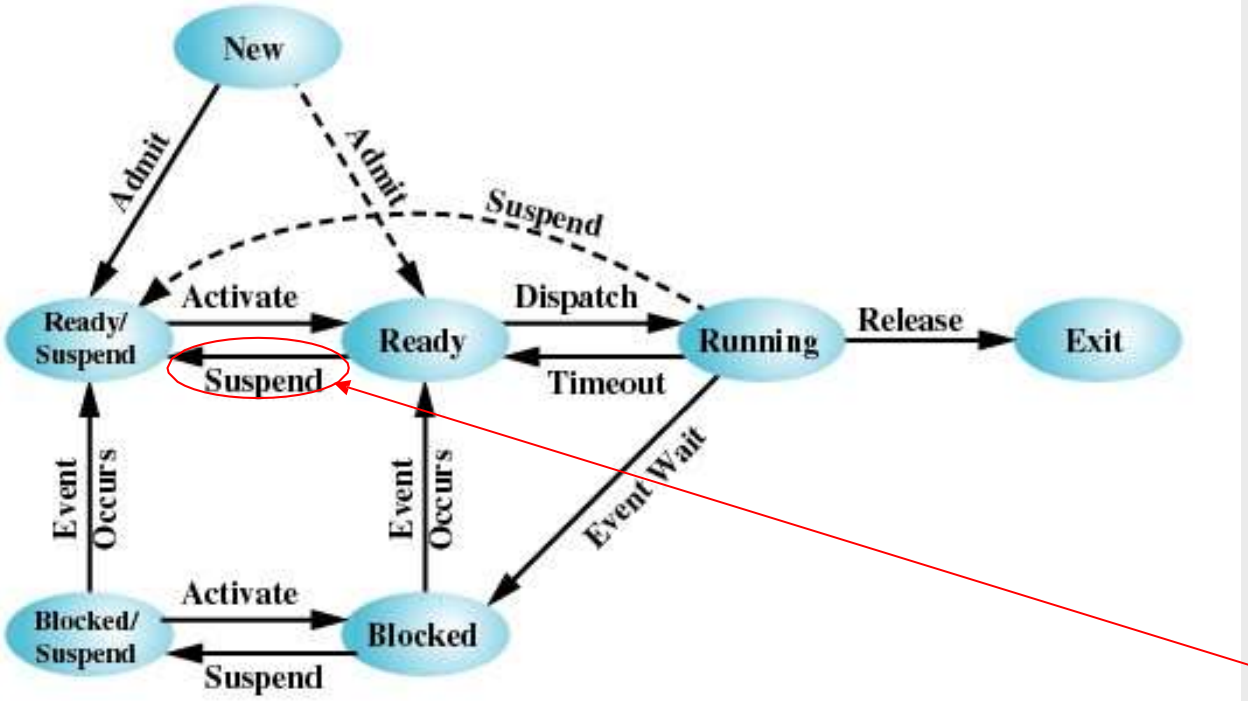
(a) With One Suspend State

Swap out: scaricamento del processo sul disco blocked -> suspended
Swap in..

Tra processi in new e in suspended quale scelgo da portare in Ready??
Ricordiamoci che il Suspended è in attesa di un evento..
Se l'evento si verifica quel processo potrebbe andare in ready

suspended → *Ready-suspended*
suspended → *Blocked-suspended*

Evoluzione del modello a 5 stati
Modello a 7 stati



(b) With Two Suspend States

Figure 3.9 Process State Transition Diagram with Suspend States

Osservazione
Memoria virtuale: un processo può trovarsi solo parzialmente Ram, quando si fa riferimento a un indirizzo su disco, questo viene caricato
⇒Inutilità del suspended

MA: immaginiamo il caso di molti processi tutti parzialmente presenti in RAM…

Necessità di maggiore memoria per allocare un processo più grande o a maggiore priorità

tutti -> exit

UNIX System 5 –SVR4

SO eseguito nell'ambiente di un processo che può avere due modalità: Utente, Kernel

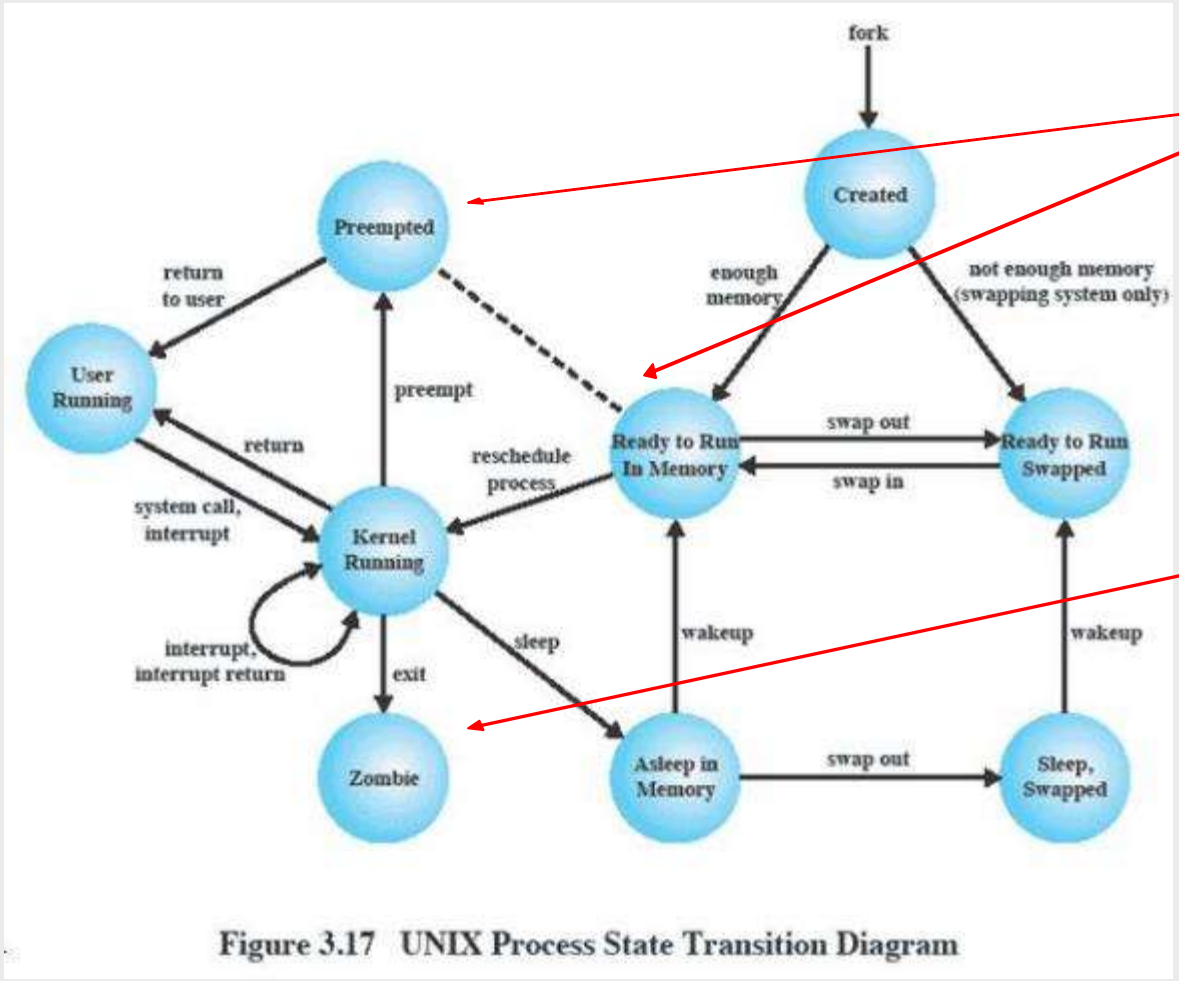


Figure 3.17 UNIX Process State Transition Diagram

Sono sostanzialmente lo stesso stato (coda unica), Nel passare da runK a runU, può giungere un processo a più alta priorità, in questo caso pre-rilascia il processo che va in preempted

Processo terminato del quale restano solo le info utilizzate dal padre

Scheduling

Con scheduling si intende un insieme di tecniche e di meccanismi interni del sistema operativo che amministrano l'ordine in cui il lavoro viene svolto

Obiettivo primario dello scheduling è l'ottimizzazione delle prestazioni del sistema.

Il sistema operativo può prevedere fino a 3 tipi di scheduler:

- *scheduler di lungo termine (SLT)*
- *scheduler di medio termine (SMT)*
- *scheduler di breve termine (SBT)*

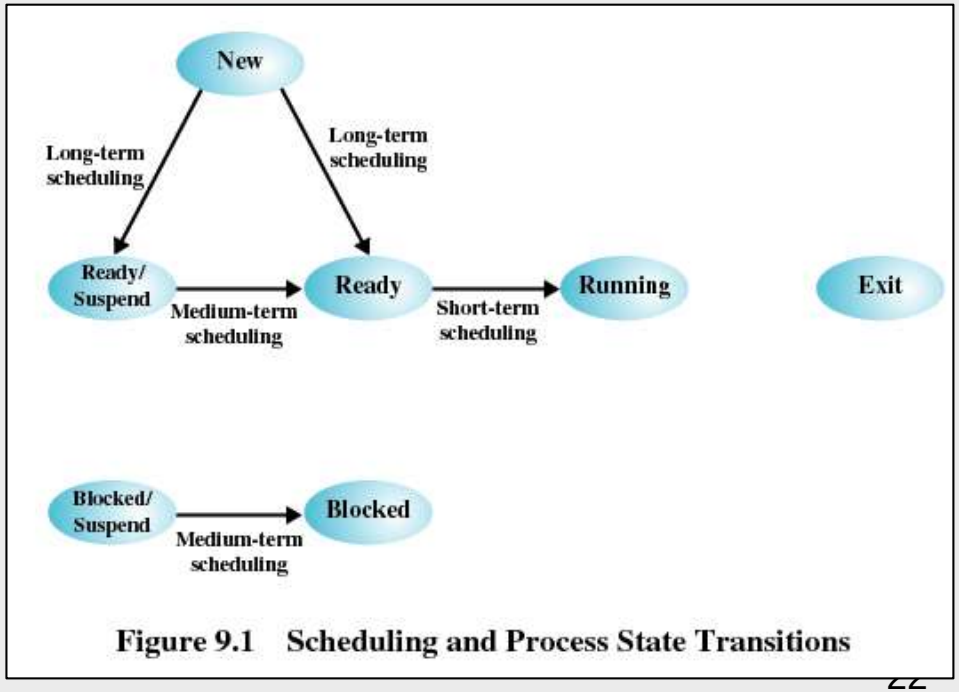


Figure 9.1 Scheduling and Process State Transitions

Scheduling

Scheduling a lungo termine:

- Determina quali programmi sono ammessi nel sistema per essere processati (new -> ready o ready suspended)
- Controlla il grado di multiprogrammazione (new -> ready)
  - Più processi = minor tempo percentuale di esecuzione per ognuno di essi
- Stime effettuate dal programmatore (o dal sistema) forniscono informazioni sulle risorse necessarie alla esecuzione, come le dimensioni della memoria, il tempo di esecuzione totale, ecc.
- Il lavoro dello scheduler di lungo termine si basa quindi sulla stima del comportamento globale dei job.

Le strategie principali:

1. fornire alla coda dei processi pronti (e quindi allo scheduler di breve termine) gruppi di processi che siano bilanciati tra loro nello sfruttamento della CPU e dell'I/O;
2. aumentare il numero di processi provenienti dalla coda batch, quando il carico della CPU diminuisce;
3. diminuire (fino anche a bloccare) i lavori provenienti dalla coda batch, quando il carico aumenta e/o i tempi di risposta del sistema diminuiscono.

La frequenza di chiamata dell'SLT è bassa e consente di implementare strategie anche complesse di selezione dei lavori e di dimensionamento del carico dei processi da inviare alla coda pronti (ready)

Scheduling

Scheduling a medio termine:

- ready suspended ->ready
- Blocked suspended ->blocked
- Parte della funzione di swapping
- Basato sulla necessità di gestire il livello di multi-programmazione

La presenza di molti processi sospesi in memoria, riduce la disponibilità per nuovi processi pronti. In questo caso lo ***scheduler di breve termine*** *è obbligato a scegliere tra i pochi processi pronti…*

- *utilizza le informazioni del* ***Descrittore di Processo*** *(PCB) per stabilire la richiesta di memoria del processo;*
- *tenta di allocare spazio in memoria centrale;*
- *riposiziona il processo in memoria nella coda dei pronti.*

Viene attivato:

→*quando si rende disponibile lo spazio in memoria;*

→ *quando l'arrivo di processi pronti scende al di sotto di una soglia specificata.*

Scheduling

Scheduling a breve termine (dispatcher):

- Ready -> Run
- Eseguito molto frequentemente
- Invocato quando si verifica un evento:
  - Clock interrupts
  - I/O interrupts
  - Operating system calls
  - Signals

La sua principale strategia è orientata alla massimizzazione delle prestazioni del sistema secondo un specifico insieme di obiettivi.

Scheduling della CPU (scheduling a breve termine)

Esecuzione di un processo:

1. ciclo di elaborazione (CPU)
2. attesa di completamento di I/O

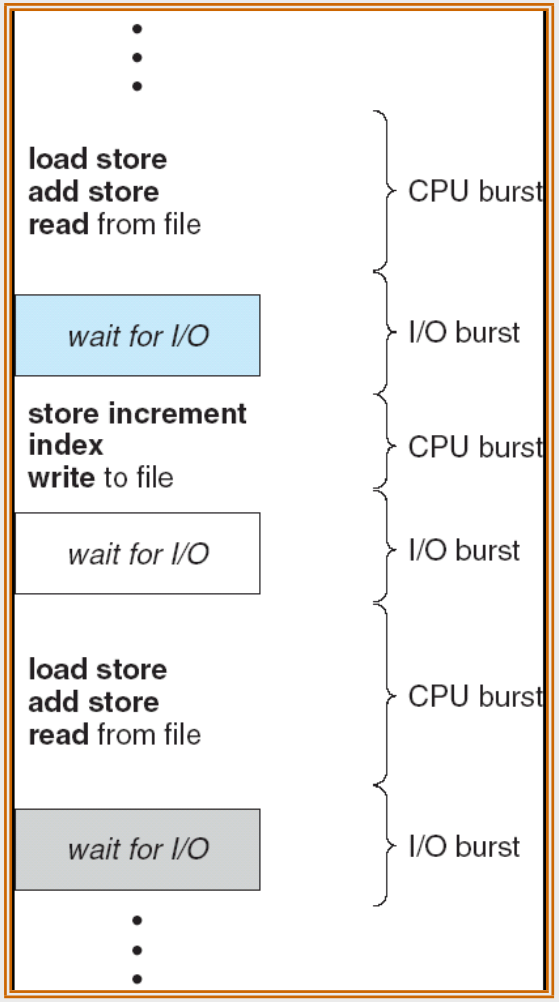
Lo scheduling della CPU riguarda la distribuzione delle sequenze di elaborazione della CPU

Processo I/O bound:

molte sequenze di operazioni di CPU di breve durata

Processo CPU bound:

poche sequenze di operazioni di CPU di lunga durata



Scheduling della CPU: valutazione Delle prestazioni

Approcci:

User-oriented

- Turnaround time: tempo intercorso tra la sottomissione di un processo e la sua terminazione.
- Tempo di risposta: per un processo interattivo è il tempo trascorso tra la sottomissione di richiesta e il ritorno del primo output. Il processo può continuare nel suo ciclo di esecuzione mentre produce output.
- Deadlines: quando la deadline di un processo è specificata, lo scheduling defe fare in modo di completare (o avviare) il processo entro la deadline

System-oriented

- Throughput: numero di processi terminati per unità di tempo
- Utilizzo del processore: *% del tempo in cui la CPU è impegnata*
  - *per eseguire programmi degli utenti*
  - *per eseguire moduli del sistema operativo*
- Evitare la starvation
- Utilizzare tutte le risorse (dispositivi di I/O, CPU, ecc.)

Criteri di ottimizzazione:

- Max utilizzazione della CPU
- Max throughput
- Min turnaround time
- Min tempo di risposta

Scheduling della CPU: valutazione Delle prestazioni

Tempo di ricircolo (Turnaround time)

Tempo trascorso l'avvio (immissione nel sistema) e la terminazione di un processo

$T_{loading}+T_{ready}+T_{CPU}+T_{I/O}$

Tempo di attesa

- misura il tempo che un processo trascorre in attesa delle risorse a causa di conflitti con altri processi.
- E' la penalità che si paga per condividere risorse ed è espressa come: tempo di ricircolo - tempo di esecuzione.
- Valuta in sostanza la sorgente di inefficienza

Tempo di risposta

- nei sistemi time-sharing:

misura il tempo che trascorre dal momento in cui viene introdotto l'ultimo carattere al terminale all'istante in cui viene restituita la prima risposta

- nei sistemi real-time:

misura il tempo che trascorre dal momento in cui viene segnalato un evento esterno all'istante in cui viene eseguita la prima istruzione della relativa routine di gestione

Priorità – Event Driven

A ogni processo viene assegnato un livello di **priorità**. Lo scheduler sceglie sempre il processo pronto con priorità maggiore

La priorità può essere assegnata dall'utente o dal sistema e può essere di tipo statico o dinamico
la priorità dinamica varia in funzione:

- del valore iniziale
- delle caratteristiche del processo
- della richiesta di risorse
- del comportamento durante l'esecuzione

*Modello **Event Driven** applicato nei sistemi dove il tempo di risposta, soprattutto ad eventi esterni, è critico.*

Caratteristiche:

- *il sistemista può influire sull'ordine in cui uno scheduler serve gli eventi esterni modificando le priorità assegnate ai processi*
- *le prestazioni sono dipendenti da una accurata pianificazione nell'assegnazione delle priorità*
- *non è in grado di garantire il completamento di un processo in un intervallo di tempo finito dalla sua creazione.*

Priorità – Event Driven

- Un numero di priorità (intero) è associato a ciascun processo
- La CPU è allocata al processo con la più alta priorità (di solito la più alta priorità corrisponde all'intero più piccolo)
  - Preemptive: requisizione della CPU se la priorità del nuovo processo in ready è maggiore
  - Non-preemptive
- Le priorità possono essere definite:
  - internamente al SO (utilizzando grandezze misurabili): uso di memoria, file aperti, rapporto tra picchi medi di di I/O e di CPU
  - Esternamente al SO rilevanza del processo, criticità
- Problemi: Starvation (processi a bassa priorità non vengono mai eseguiti)
- Soluzione: Aging, al passare del tempo in stato di ready la priorità del processo aumenta

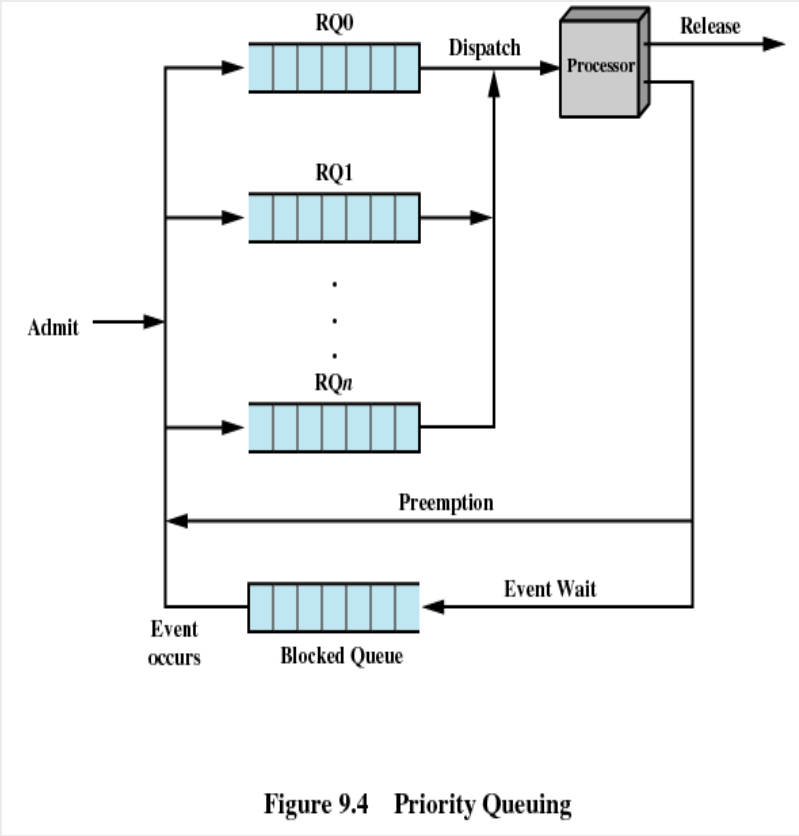


Figure 9.4 Priority Queuing

Scheduling della CPU

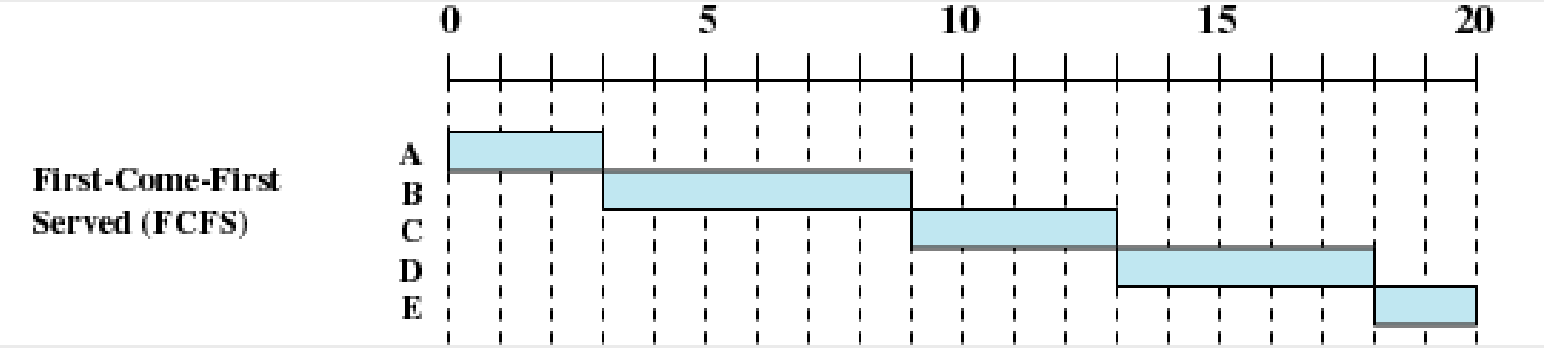
DECISION MODE:

- Non-Preemptive
  - Un processo nello stato di running abbandonerà tale stato solo se termina l'esecuzione o se si blocca per una operazione di I/O
- Preemptive
  - Il processo attualmente nello stato di running può essere interrotto e spostato nello stato di ready dal SO
  - PRO: Nessun processo può monopolizzare il processo
  - PROBLEMI: processi che condividono dati -> meccanismi di sincronizzazione

DISPATCHER:

- Modulo del SO che passa il controllo della CPU al processo selezionato dallo scheduler a breve termine attraverso:
  - switch del contesto
  - switch del l modo utente/supervisore
  - Salto alla locazione opportuna del programma utente (contenuta nel PCB) per farlo ripartire
- *Dispatch latency* – tempo che il dispatcher usa per fermare un processo e far partire un altro.

First Come First Served - FCFS



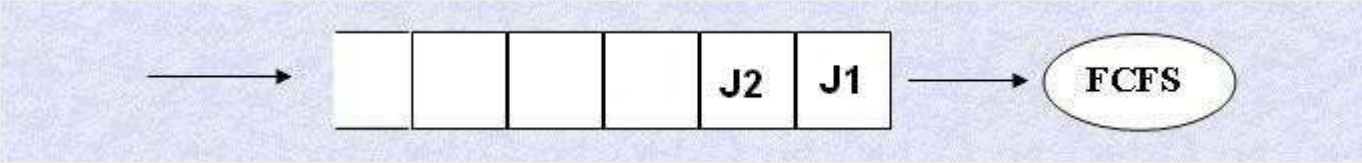
- Ogni processo entra nella coda di ready
- Quando un processo abbandona lo stato di running si seleziona il processo che da più tempo è nello stato di ready
- Un processo I/O bound o che richiede poco tempo di esecuzione potrebbe attendere molto tempo prima che gli venga assegnata la CPU
- Favorisce i processi CPU-bound
- Tempo medio di attesa in coda elevato
- Effetto convoglio: tutti I processi in coda attendono che un processo CPU-bound termini
- Senza prelazione:
  - *basso sfruttamento dei componenti*
  - *basso lavoro utile del sistema*

First Come First Served - FCFS

Le prestazioni dipendono dall'ordine di arrivo dei jobs

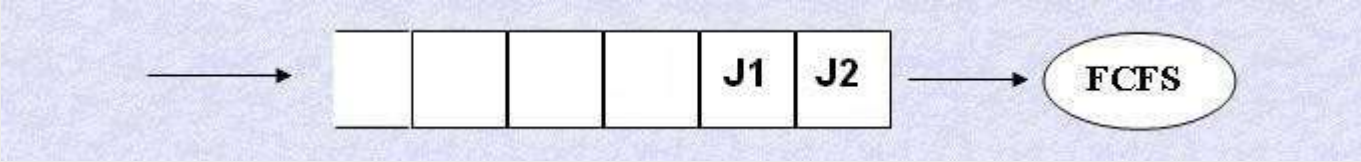
esempio : Siano J_1 e J_2 due job con tempi di esecuzione totali rispettivamente pari a t_1=20 e t_2=2 unità di tempo.

I CASO



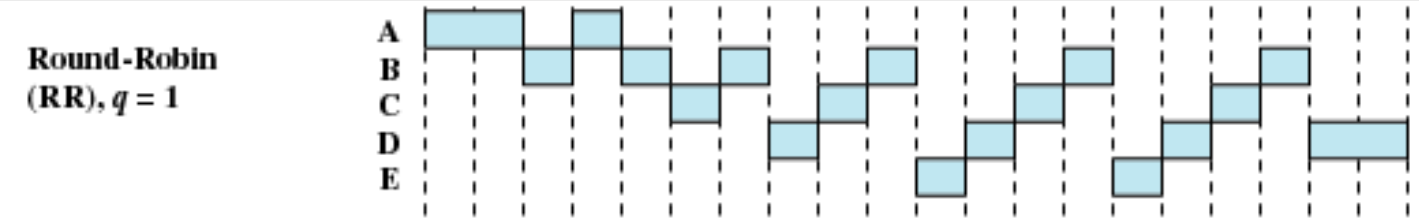
Tempo di riclico di J_1=20; Tempo di riclico di J_2=22; Tempo medio di riclico=21;
Tempo di attesa di J_1=0; Tempo di attesa J_2=20; Tempo medio di attesa=10;

II CASO



Tempo di riclico di J_2=2; Tempo di riclico di J_1=22; Tempo medio di riclico=12;
Tempo di attesa di J_2=0; Tempo di attesa J_1=2; Tempo medio di attesa=1;

Round Robin – time slice



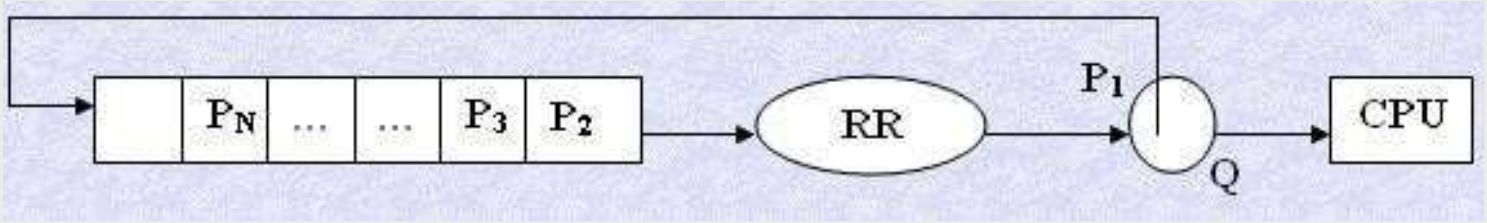
- Preemption basata sul clock (clock interrupt)
- Ogni processo utilizza il processore per un dato intervallo di tempo (time slice). Valori tipici: 10-100msec
- Al verificarsi dell'interrupt il processo in esecuzione viene portato nella coda di ready (gestita FIFO)
- Con *n* processi in ready e un time quantum *q*, ogni processo ottiene 1/*n* del tempo di CPU in frazioni di tempo al più pari a q. Tempo massimo di attesa in ready: (*n*-1)*q*.
- Prestazioni dipendenti dal quanto di tempo:
  - q grande $\Rightarrow$ FCFS
  - q piccolo $\Rightarrow$ incrementa il numero di context switch

Round Robin – time slice

La schedulazione round-robin fornisce una buona condivisione delle risorse del sistema:

- i processi più brevi possono completare l'operazione in un q (buon tempo di risposta)
- i processi più lunghi sono forzati a passare più volte per la coda dei processi pronti (tempo proporzionale alle loro richieste di risorse)
- per i processi interattivi lunghi, se l' esecuzione tra due fasi interattive riesce a completarsi in un q, il tempo di risposta è buono

La realizzazione di uno scheduler RR richiede il supporto di un Timer che invia un'interruzione alla scadenza di ogni q, forzando lo scheduler a sostituire il processo in esecuzione. Il timer viene riazzerato se un processo cede il controllo al Sistema Operativo prima della scadenza del suo q.



Highest Response Ratio Next

Siano:

w – tempo speso in coda di ready (attesa della disponibilità del processore)
s – tempo di servizio previsto

si definisce il Response Ratio come:

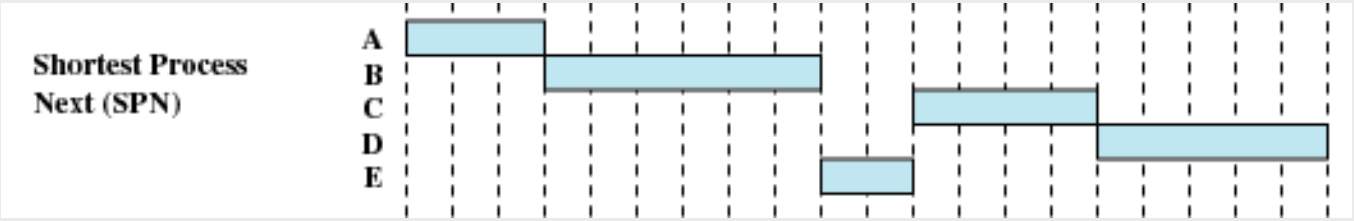
$$RR = \frac{w + s}{s}$$

Si sceglie il processo con il più alto valore di RR

Osservazioni:

- *quando un processo entra in coda per la prima volta, RR=1*
- *tiene considerazione l'età del processo*

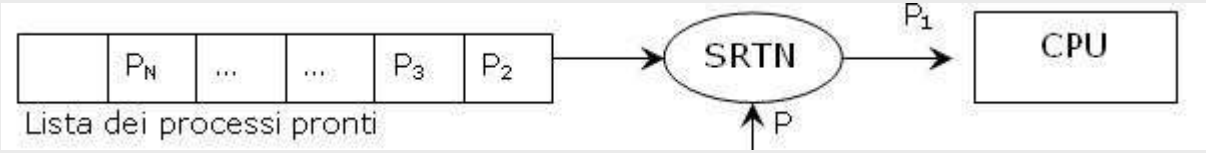
Shortest Process Next - SPN



- Il processo scelto dalla coda di ready è quello con il più breve tempo di esecuzione stimato: più breve sequenza di operazioni svolte dal processore
- Possibilità di starvation per processi fortemente CPU bound
- SPN è ottimale nel senso che formisce il tempo medio di attesa minimo, per un dato set di processi.
- Utilizzato nello scheduling a lungo termine
- Difficile stimare la durata della prossima sequenza di CPU (oltre che oneroso)
- Due possibili schemi:
  - nonpreemptive;
  - preemptive – se arriva nuovo processo con una sequenza di CPU minore del tempo necessario per la conclusione della sequenza di CPU del processo attualmente in esecuzione, si ha il prerilascio della CPU a favore del processo appena arrivato. Questo schema è anche noto come Shortest-Remaining-Time-First (SRTF)

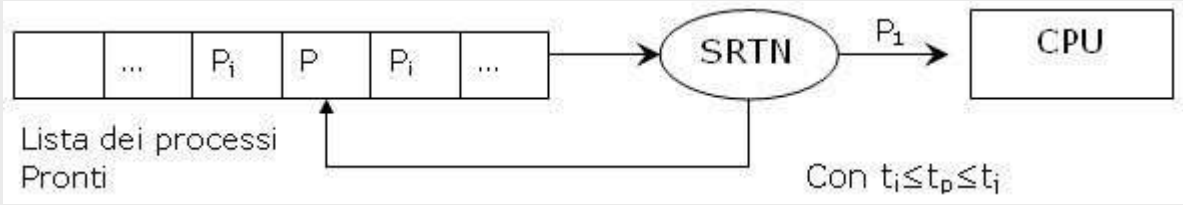
Shortest Remaining Time Next

SPN Con Prerilascio

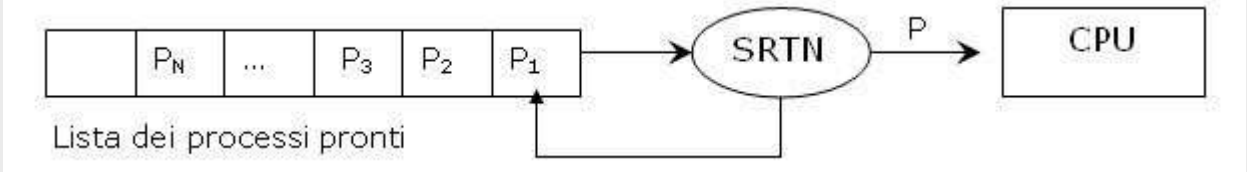


P_i i-esimo processo con i=1,…,N;
t_i tempo di esecuzione dell'i-esimo processo;
Per ogni i,j =1,...,N i≤j: $t_i \leq t_j$;
t_p tempo di esecuzione del processo P;
t_1^r tempo di esecuzione residuo del processo P_1;

I CASO : $t_{1_r} \leq t_p$



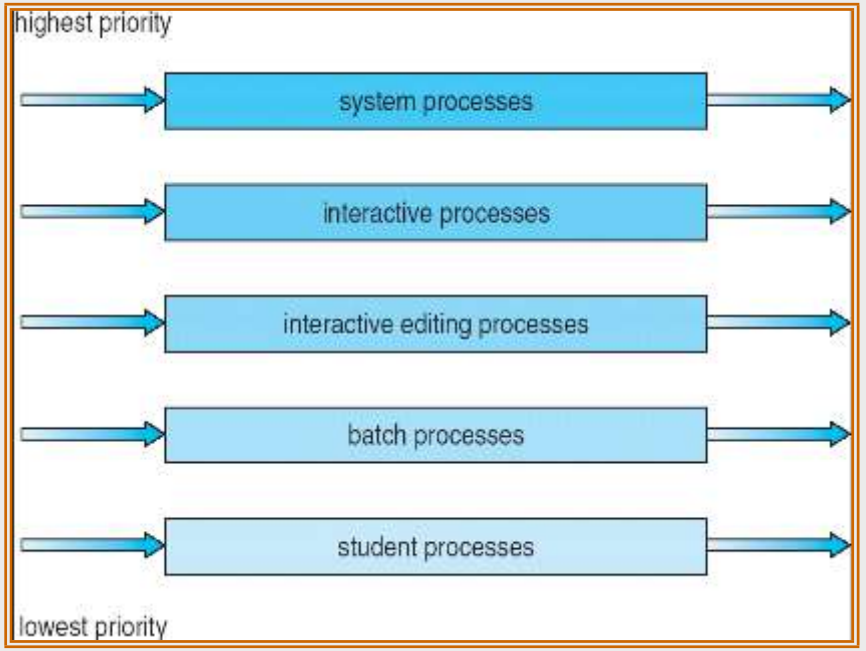
II CASO : $t_{1_r} > t_p$



Schedulazione a code multiple

- La coda di ready è suddivisa in sotto-code:
  - foreground (interactive)
  - background (batch)
- Ogni coda ha un proprio algoritmo di schedulazione

  Es.: foreground – RR, background – FCFS
- Lo Scheduling deve essere effettuato tra le code:
  - Scheduling per priorità fissa e con prelazione (i.e., serve all from foreground then from background). Possibilità di starvation.
  - Time slice – ad ogni coda è associato un certo ammontare di tempo di CPU; i.e., 80% to foreground in RR, 20% to background in FCFS

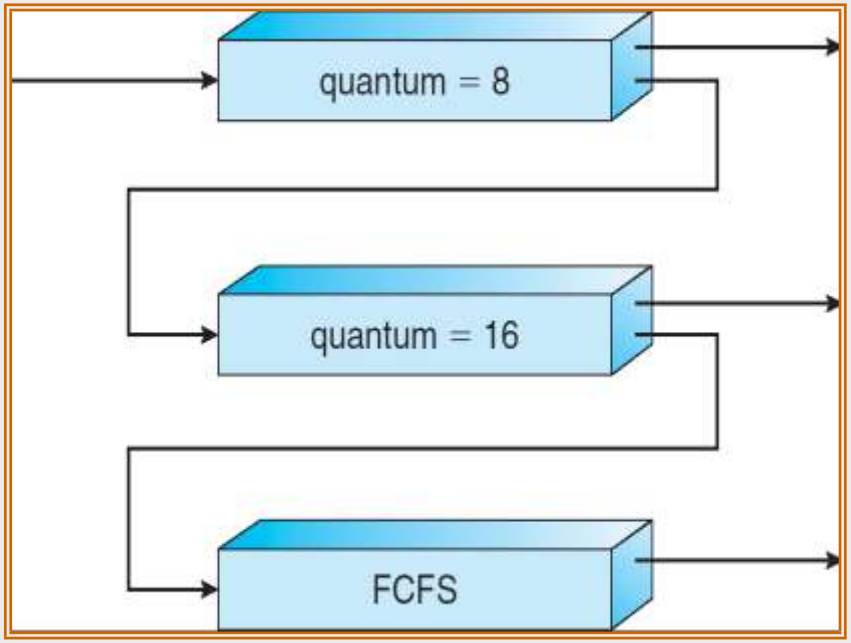


Schedulazione a code multiple con feedback

- Un processo può essere spostato da una coda all'altra (implementazione dell'aging)
- Code Multilevel-Feedback definite dai seguenti parametri:
  - Numero di code
  - Algoritmo di scheduling per ogni coda
  - Metodi usati per l'up-grading e il down-grading di ogni processo

Esempio:

- Tre code:
  - Q_0 – RR con time quantum 8 milliseconds
  - Q_1 – RR con ime quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - Un nuovo processo entra nella coda Q_0 (FCFS).
  - Quando ottiene la CPU, la impegna per 8 ms. Se non termina entro gli 8 ms è spostato in Q_1.
  - Il processo in Q_1 viene nuovamente servito con politica FCFS e riceve la CPU per ulteriori 16 ms.
  - Se ancora non termina viene spostato in Q_2.



Ricapitolando

•First come first served: Seleziona il processo che ha il tempo di attesa più lungo.

•Round robin: Usa il time-slicing per limitare tutti i processi in fase di esecuzione ad un piccolo periodo d'uso del processore e ruota su tutti i processi Ready.

•Shortest process next: Seleziona il processo con il minore tempo di uso del processore previsto e non prerilascia il processo.

•Shortest remaining time: Seleziona il processo con il minore tempo di uso del processore previsto; un processo puo essere prerilasciato quando un altro processo diventa Ready.

•Highest response ratio next: Basa la decisione di scheduling su una stima del tempo di turnaround normalizzato.

•Feedback: Crea un insieme di code per lo scheduling, ed alloca i processi in tali code a seconda della loro storia di esecuzione, o su altri criteri.

Windows

Dispatcher: porzione del kernel che si occupa dello scheduling
Un thread viene eseguito fino a:

1. Terminazione
2. Prelazione da parte di un thread a priorità più alta
3. Esaurimento del quanto di tempo
4. Chiamata bloccante

32 livelli di priorità divisi in due macro-classi:

variable: thread con priorità da 1 a 15
real time: …da 16 a 31
priorità 0: thread per la gestione della memoria

Una coda per ogni livello di priorità.
Se non esiste nessun thread in ready viene eseguito un thread di idle

La priorità di un thread dipende dalla priorità della classe a cui appartiene e dalla priorità relativa del thread nella classe

Valori di ciascuna classe

Priorità relativa del thread nella classe	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Windows

- Quando il time slice termina il thread riceve una priorità più bassa tornando in ready
- L'abbassamento della priorità non porta mai la priorità sotto il livello minimo per una specifica classe
- Se un thread va in blocked la sua priorità sarà aumentata in dipendenza del tipo di evento di attesa (thread che attende dati da tastiera riceverà priorità maggiore rispetto a quello che attende dati da HD)
- *La strategia mira a fornire tempi di risposta rapidi per thread interattivi (interfacce basate su mouse,..)*
- Tipi di processi interattivi:
  - in primo piano (*foreground*): correntemente selezionati sullo schermo
  - in secondo piano (*background*): non attualmente selezionati
  - Quando un processo passa da background a foreground la sua priorità viene aumentata di un fattore 3

Linux Scheduling

- Due algoritmi:
  - time-sharing con diritto di prelazione ad equità
  - real-time: le priorità assolute sono più importanti dell'equità
- Time-sharing
  - Partizionamento del tempo basato sui crediti: il processo con più crediti viene schedulato
  - Ad ogni interruzione dovuta allo scadere del tempo i crediti sono decrementati di una unità
  - Quando credit=0, il processo viene sospeso e si sceglie un altro processo
  - Periodicamente vengono riassegnati i crediti per tutti i processi (utente e di sistema) con la regola:
    - Crediti=crediti/2 + priorità
    - In questo modo i processi blocked acquisiscono grande priorità
- Real-time
  - Posix.1b compliant – due classi
    - FCFS (senza prelazione) and RR (con prelazione)
    - Viene sempre eseguito il processo con più alta priorità
  - Soft real time: nessuna garanzia sul completamento dei processi entro le loro deadlines (in linux non si può sospendere un processo kernel per dare spazio ad uno utente)

Schedulazione MULTIPROCESSORE

Classificazione dei sistemi multi-processore

- Processori debolmente accoppiati o distribuiti, o cluster
  - Ogni processore ha la propria memoria e i propri canali di I/O
- Processori specializzati per funzioni
  - Esempio: I/O processor
  - Controllati da un processore master a cui forniscono servizi
- Processori fortemente accoppiati
  - I processori condividona la memoria centrale
  - Sono sotto il controllo del SO

Granularità di sincronizzazione tra i processori

- Parallelismo indipendente,:
  - Non c'è sincronizzazione tra i processi (sistemi time-sharing)
  - Da un punto di vista di scheduling è indifferente avere uno o più processori
- Parallelismo a grana grossa e molto grossa:
  - Bassa sincronizzazione tra i processi
  - Può essere immaginato pari al caso di un processore multiprogrammato
- Parallelismo a grana media:
  - Una applicazione è un insieme di threads che interagisnono frequentemente
  - Lo scheduling può influenzare le prestazioni
- Parallelismo a grana fine:
  - Per applicazioni ad elevato parallelismo / Sperimentale…

Assegnazione dei processi ai processori

- Se i processori sono tutti uguali (SMP) possono essere trattati come un pool di risorse e i processi possono essere eseguiti su ogni processore

  - Assegnazione permanente (statica):
    - Un processo viene eseguito sempre su un specifico processore
    - Coda a breve termine dedicata per ogni processore
    - Poco overhead nella gestione dello scheduling
    - Un processore può rimanere inutilizzato a seguito di una coda vuota mentre un altro ha un carico arretrato
  - Assegnazione dinamica:
    - Coda globale
    - Non c'è svantaggio rispetto al caso precedente se i PCB sono in RAM condivisa

Assegnazione dei processi ai processori

- Architettura Master/slave
  - Kernel functions sempre eseguite su un particolare processore
  - Master responsabile per lo scheduling
  - Slave inviano richieste di servizio al master
  - Vantaggi:
    - Solo il master ha il controllo della memoria e dei dispositivi di I/O (gestione conflitti semplificata)
  - Svantaggi
    - Un fallimento del master determina il fallimento dell'intero sistema
    - Master può diventare il collo di bottiglia delle prestazioni

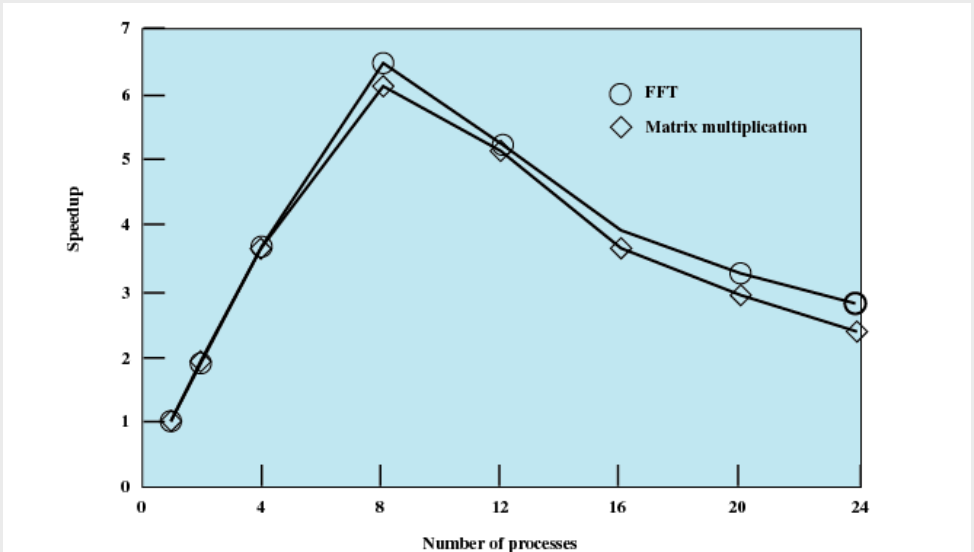
- Architettura Peer
  - OS eseguito su ogni processore
  - Ogni processore fa fronte al proprio scheduling
  - Difficoltà per l'OS:
    - Due processori non devono scegliere lo stesso processo

Scheduling dei threads

- Applicazione: set di threads che cooperano e sono eseguiti concorrentemente nello stesso spazio di indirizzamento
- Su monoprocessore i threads vengono usati per sovrapporre sequenze di I/O a sequenze di esecuzione
- Elevati guadagni di performance se threads di una stessa applicazione vengono eseguiti simultaneamente su più processori
- Approcci per l'assegnazione:
  - Load sharing – condivisione del carico
    - I processi non sono assegnati a specifici processori ed esiste una coda globale dei threads pronti
  - Gang scheduling – schedulazione a gruppi
    - Un set di threads correlati è schedulato per essere eseguito nello stesso istante su un set di processori, sulla base di un processo per processore
  - Assegnazione dedicata
    - I Threads sono assegnati ad uno specifico processore
    - Teoricamente ad un processo sono assegnati tanti processori quanti sono i suoi threads
  - Scheduling Dinamico
    - Il numero dei threads può essere modificato durante l'esecuzione

Assegnazione di processore dedicato – Scheduling Statico

- Forma estrema di gang scheduling: un gruppo di processori viene dedicata ad una applicazione fino al suo termine
- Quando una applicazione è schedulata ognuno dei suoi threads viene assegnato ad un processore fino al termine dell'esecuzione dell'applicazione
- SVANTAGGI:
  - Alcuni processori potrebbero rimanere "fermi"
  - No multiprogramming of processors
- VANTAGGI:
  - Caso di sistemi con molti processori (centinaia)…
  - Annullamento totale dei cambi di contesto per tutta l'esecuzione di un programma



Es.

-Due applicazioni in esecuzione contemporanea
-16 processori
-Ogni applicazione può essere suddivisa in più threads
-L'incremento di velocità diminuisce se il numero di threads diventa maggiore del numero di processori:
-Maggior numero di switching e schedulazione

Scheduling dinamico

- Il numero dei threads di un processo può essere modificato dinamicamente
- Il SO può modificare il carico per migliorare l'utilizzazione dei processori
- SO e applicazione collaborano nell'attività di scheduling:
  - SO responsabile dell'assegnazione dei processori ai processi
  - L'applicazione decide quali threads eseguire, quali sospendere, ecc.
- Politiche applicate dal SO:
  - Assegnare il processore ad un nuovo processo se tale processore è in idle
  - Assegnare un processore ad un nuovo processo requisendolo ad uno che ne detiene già più di uno
  - Se una richiesta non può essere soddisfatta, essa viene sospesa fino a quando un processore non diventa disponibile
  - Assegna il processore ad un processo che non ha correntemente alcun processore allocato

NB: il vantaggio derivante dallo scheduling dinamico può essere annullato dall'overhead della sua gestione