

La gestione della CPU

Prof. **Marco Camurri**

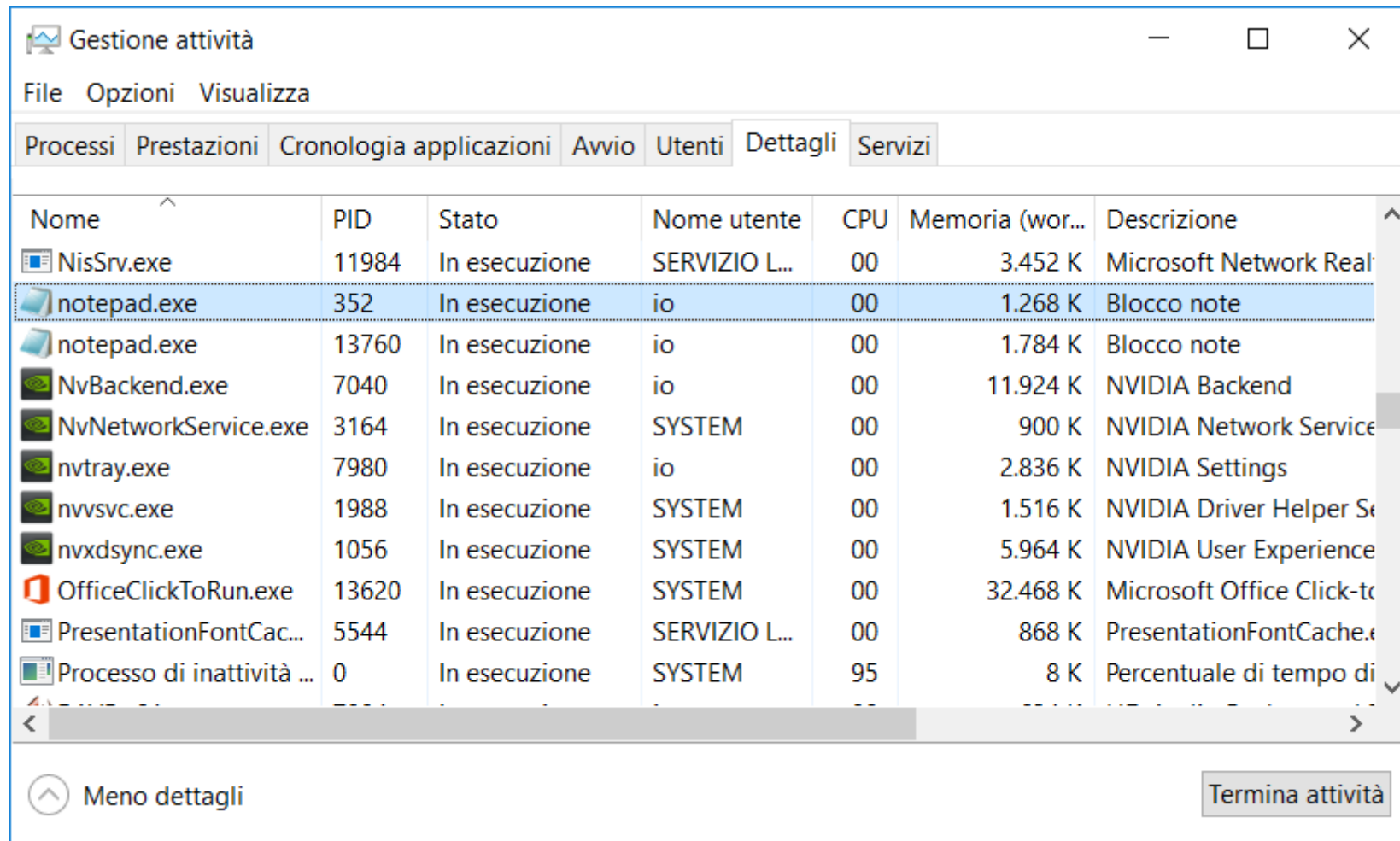
Argomenti

- Programmi e processi
- Multitasking
- Time-slice
- Il dispatcher e l'attività di context-switch
- Descrittori di processo o PCB
- Stati di un processo: ready, running e wait
- Processi CPU-bound e I/O-bound
- Scheduling della CPU
 - Algoritmo Round-Robin
 - Algoritmi con priorità (cenni)
- Thread

Differenza fra programma e processo

processo = programma in esecuzione

Se avvio due volte "blocco note" ottengo **due processi**, che corrispondono a due diverse istanze in esecuzione dello **stesso programma**



The screenshot shows the Windows Task Manager window titled "Gestione attività". The "Processi" tab is selected, displaying a list of running processes. The table has columns for Name, PID, Status, User, CPU, Memory, and Description. Two instances of "notepad.exe" are visible, both running under the user "io".

Nome	PID	Stato	Nome utente	CPU	Memoria (wor...	Descrizione
NisSrv.exe	11984	In esecuzione	SERVIZIO L...	00	3.452 K	Microsoft Network Real...
notepad.exe	352	In esecuzione	io	00	1.268 K	Blocco note
notepad.exe	13760	In esecuzione	io	00	1.784 K	Blocco note
NvBackend.exe	7040	In esecuzione	io	00	11.924 K	NVIDIA Backend
NvNetworkService.exe	3164	In esecuzione	SYSTEM	00	900 K	NVIDIA Network Service
nvtray.exe	7980	In esecuzione	io	00	2.836 K	NVIDIA Settings
nvsvc.exe	1988	In esecuzione	SYSTEM	00	1.516 K	NVIDIA Driver Helper Se
nvxdsync.exe	1056	In esecuzione	SYSTEM	00	5.964 K	NVIDIA User Experience
OfficeClickToRun.exe	13620	In esecuzione	SYSTEM	00	32.468 K	Microsoft Office Click-to
PresentationFontCac...	5544	In esecuzione	SERVIZIO L...	00	868 K	PresentationFontCache.e
Processo di inattività ...	0	In esecuzione	SYSTEM	95	8 K	Percentuale di tempo di

Cos'è un processo?

- Il processo è un'astrazione creata dal S.O per rappresentare il concetto di "programma in esecuzione"
- Per ogni programma avviato il S.O deve mantenere una serie di informazioni come l'utente che ne ha richiesto l'avvio, la posizione in memoria del processo, ecc.
- Ogni processo occupa delle risorse del sistema (CPU, memoria) per cui esiste un limite al numero di processi che il S.O è in grado di creare

Multi-tasking

- Il concetto di processo è ciò che rende possibile il multitasking

Un S.O si dice **multi-tasking** (task=attività) se è in grado di eseguire più programmi "contemporaneamente"

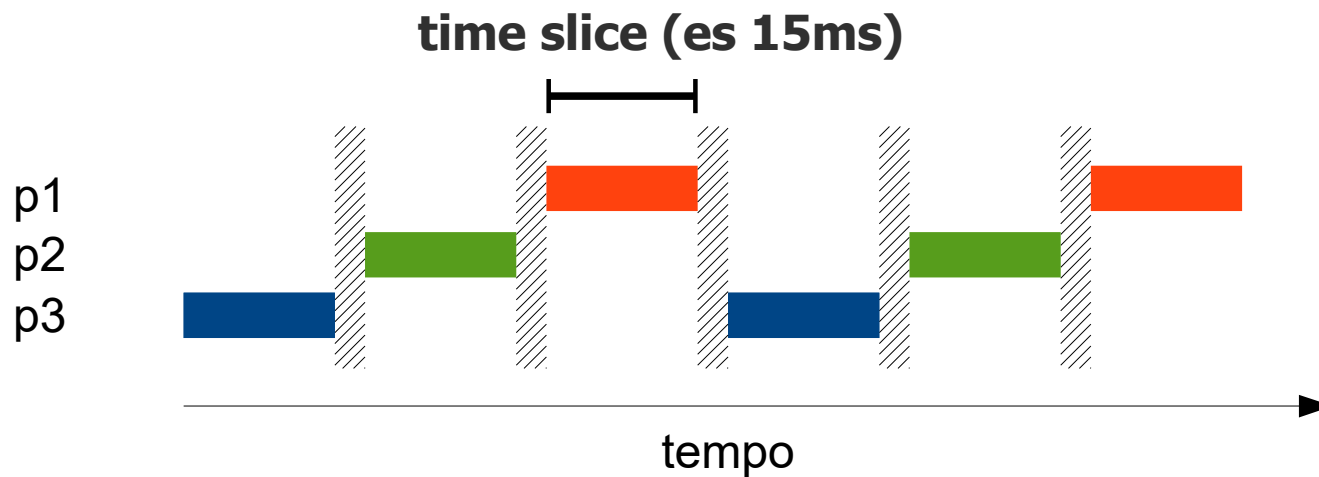
- esempio: posso lanciare la compressione di una cartella di 1GB e nel frattempo riprodurre un file audio, lavorare con un text editor, ecc...
- I primi sistemi operativi per PC non erano multitasking (es **DOS = Disk Operating System**). Ciò significa che per iniziare l'esecuzione di un nuovo programma occorreva attendere la fine del precedente.

Realizzazione del multi-tasking

- Osservando il numero di processi attivi su un sistema (*) si nota che esso è ben superiore al numero di CPU del computer
- Come può il S.O eseguire contemporaneamente tutti questi processi?
- Il S.O crea l'**illusione** di esecuzione contemporanea, pur disponendo di un numero di processori ben inferiore al numero di attività in corso
- Per creare questa illusione il S.O alterna ciclicamente l'esecuzione di brevi porzioni di codice di ogni task
- Ad esempio: esegue il processo1 per 10ms, poi lo sospende temporaneamente ed esegue il processo2 per 10ms, e così via

Realizzazione del multi-tasking

- Consideriamo il caso di una sola CPU e 3 processi da eseguire



/// = tempo impiegato dal sistema operativo per passare da un task all'altro (**cambio di contesto**)

Quanto di tempo (time slice)

- Il S.O concede al processo l'utilizzo del processore (cioè assegna la CPU al processo) per un piccolo intervallo di tempo
- Questo intervallo è detto **quanto di tempo** o **time slice**, e rappresenta il tempo di esecuzione massimo concesso ad un processo prima che esso venga interrotto temporaneamente
- La tecnica descritta è detta **time-sharing** (ripartizione di tempo)
- Il valore del time slice è un parametro del sistema operativo
- Valore "ragionevole": 10-20 ms
- Per effettuare il passaggio da un processo all'altro il S.O deve salvare (in memoria principale) lo stato del processo interrotto e ripristinare lo stato del processo entrante. Questa operazione occupa un certo tempo ed è detta **cambio di contesto (context switch)**
- La scelta del valore del time slice è frutto di un compromesso: con un valore troppo piccolo si spreca molto tempo nei cambi di contesto; con un valore troppo grande il sistema non è reattivo

Cambio di contesto (context switch)

- Ogni volta che un processo viene sospeso temporaneamente (per consentire l'esecuzione di un altro processo) il S.O deve salvare in memoria lo stato della CPU in quell'istante
- Lo **stato della CPU**, detto anche **contesto** del processo, è l'insieme dei valori assunti dai registri della CPU in un dato istante
- Salvare lo stato è indispensabile poichè quando il processo sarà rimesso in esecuzione, esso dovrà trovare la CPU esattamente nello stato in cui l'aveva lasciata immediatamente prima di essere sospeso
- Il S.O quindi, prima di rimettere in esecuzione un processo, deve recuperare dalla memoria le informazioni di contesto che aveva salvato in precedenza, e usarle per ripristinare lo stato della CPU

Dispatcher

- La parte di codice del S.O che si occupa di eseguire il cambio di contesto è detto **dispatcher**
- **Esempio:** supponiamo che il processo A venga fermato per riprendere l'esecuzione del processo B

il dispatcher deve:

- copiare il contenuto attuale di tutti i registri della CPU (cioè contesto del processo A) in memoria
- recuperare dalla memoria il contesto del processo B e usarlo per reimpostare tutti i registri della CPU
- avviare il processo B

Descrittori di processo (PCB)

La struttura dati usata da un S.O per memorizzare tutte le informazioni reattive ad un processo è detta **descrittore di processo** o **process control block (PCB)**

Contenuto tipico di un PCB:

- Identificatore del processo (PID)
- Utente che ha creato il processo
- Stato del processo (ready, running, waiting,..)
- Livello di priorità assegnato al processo
- Informazioni sul contesto di esecuzione
- Risorse associate al processo:
 - Aree di memoria assegnate al processo
 - File aperti dal processo
 - Dispositivi di I/O in uso

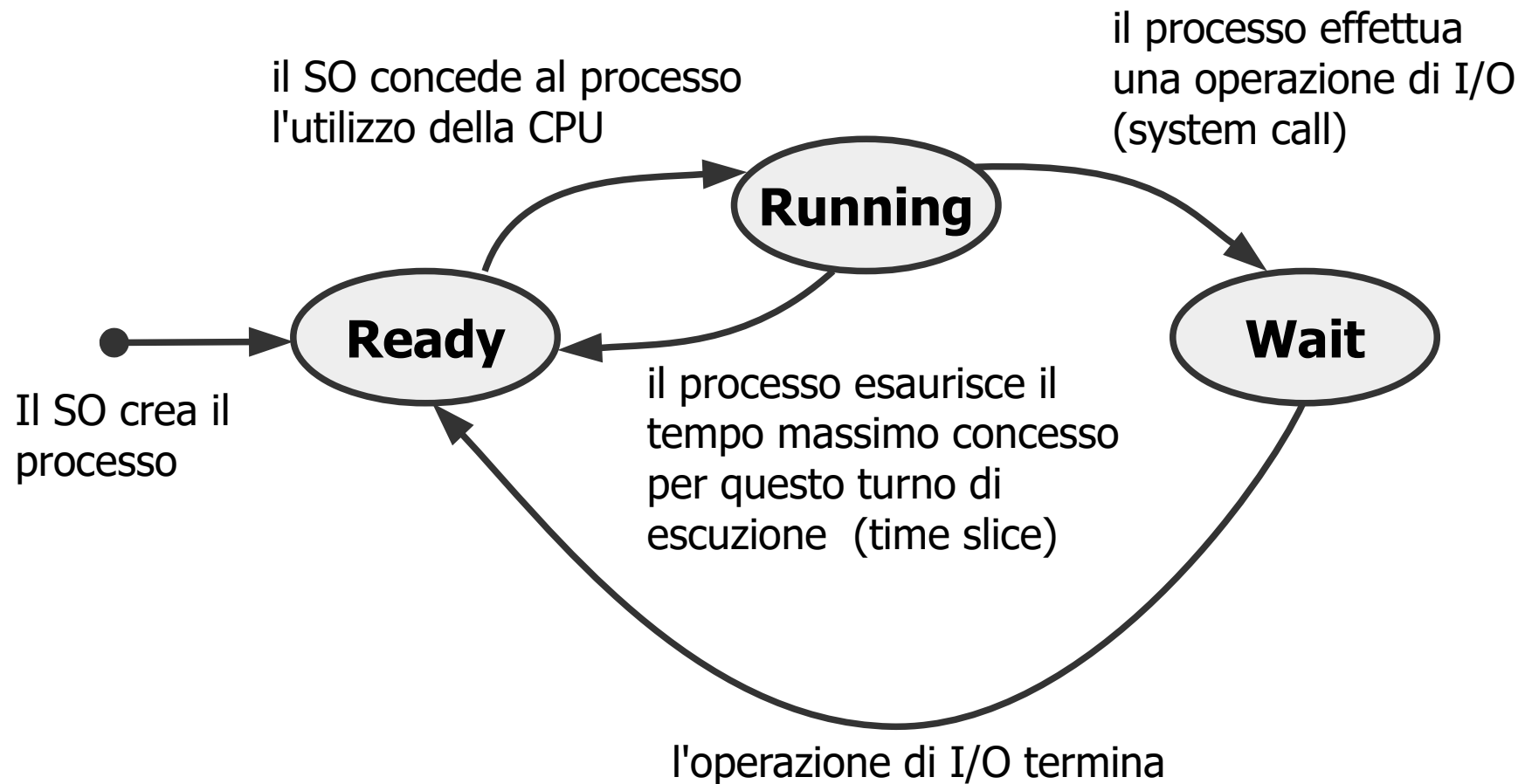
Descrittori di processo (PCB)

In pratica, poichè la maggior parte dei S.O è scritta in C, possiamo immaginare che il PCB corrisponda ad una **struct** definita nel codice del S.O:

```
struct PCB {  
    int pid;  // process id  
    int priority;  
    int uid;  // user id  
    int status;  
    ....  
}
```

Ogni volta che un nuovo programma è avviato, il S.O deve aggiungere un elemento di questo tipo ad una lista di PCB, mantenuta in memoria principale. Questo elemento conterrà tutte le informazioni sul processo appena creato.

Stati di un processo



Stati di un processo

Durante il suo "ciclo di vita" un processo può attraversare (più volte) i seguenti stati:

- **Ready**: il processo è pronto per essere eseguito (l'unica risorsa di cui necessita per proseguire è la CPU) e quindi attende che il S.O gli conceda l'utilizzo di una CPU. In ogni istante, più processi possono trovarsi "in coda" nello stato Ready, in attesa del loro turno di esecuzione.
- **Running**: il processo è in esecuzione, cioè sta utilizzando la CPU (una CPU sta eseguendo il suo codice). In ogni istante possono esservi solo N processi in stato running (dove N è il numero di CPU del sistema). In generale il numero N è molto inferiore al numero totale di processi presenti.
- **Wait**: il processo ha richiesto (tramite una chiamata di sistema) un'operazione di I/O e sta attendendo il termine dell'operazione. Ad esempio, il processo sta attendendo dati dal disco o dalla tastiera. In ogni istante, più processi possono essere in stato di wait.

Transizioni di stato

Le uniche transizioni di stato consentite sono le quattro seguenti:

- **Ready → Running** si verifica quando il S.O decide che è giunto il turno di esecuzione del processo. Il S.O **assegna** una CPU al processo.
- **Running → Ready** si verifica quando un processo esaurisce proprio *quanto di tempo* e quindi il S.O sottrae forzatamente la CPU al processo. Il processo è "rimesso in coda" nello stato di Ready, e per proseguire deve attendere che giunga di nuovo il proprio turno di esecuzione. La coda è detta "**ready queue**".
- **Running → Wait** si verifica quando un processo richiede un'operazione di I/O (sia di lettura che di scrittura) o comunque effettua una chiamata di sistema che comporta un'attesa. Il processo non può proseguire finchè l'operazione non si è conclusa, e quindi cede spontaneamente la CPU.

Transizioni di stato

- **Wait → Ready** si verifica quando l'operazione di I/O richiesta dal processo è terminata (ad esempio, è disponibile un dato proveniente dalla tastiera che era stato richiesto dal processo, oppure è stata completata una scrittura su file o sul monitor). Il processo è rimesso nella ready queue, e per proseguire deve attendere che giunga di nuovo il proprio turno di esecuzione.

Notare che:

- 1) NON è consentita la transizione diretta da Wait a Running
- 2) un processo esce dallo stato Running anche se ha terminato la propria esecuzione (cioè dopo che è stata eseguita la sua ultima istruzione).

Processi CPU-bound e I/O-bound

- I processi che effettuano molte operazioni di I/O, e quindi passano la maggior parte del proprio tempo nello stato di wait, sono detti **processi I/O-bound**
- Viceversa, i processi che utilizzano prevalentemente la CPU (poichè effettuano poche operazioni di I/O) sono detti **processi CPU-bound**
- L'uso delle risorse è particolarmente efficiente se nel sistema vi è un buon bilanciamento tra processi I/O-bound e processi CPU-bound
- Motivo: mentre un processo I/O-bound sta aspettando il completamento di un'operazione, il processo CPU-bound può usare la CPU (che altrimenti rimarrebbe inutilizzata)

Scheduling della CPU

- **CPU Scheduling** = pianificazione del lavoro della/e CPU: stabilire a quali processi, in quali istanti, e per quanto tempo assegnare la/le CPU ai vari processi
- Ogni volta che un processo esce dallo stato *running* il SO deve selezionare un nuovo processo da mandare in esecuzione
- Lo **scheduler della CPU** è il componente (modulo) del S.O che decide:
 - quale processo, tra quelli nello stato *ready*, deve essere portato nello stato *running*
 - in quali casi un processo può uscire dallo stato *running*

Scheduling preemptive e cooperativo

- Uno scheduler si dice **preemptive** se può sottrarre forzatamente la CPU ad un processo
- Ciò si verifica quando il processo non cede spontaneamente la CPU prima del termine del time slice
- Nei S.O con scheduler **non-preemptive** (detto anche **cooperativo**) non è consentita la transizione running->ready
- In questi S.O un processo non viene mai interrotto forzatamente, può solo cedere spontaneamente la CPU
- Nelle prime versioni di Windows lo scheduler era cooperativo
- Attualmente gli scheduler di tutti i principali S.O sono di tipo preemptive

Algoritmi di scheduling della CPU

- Per effettuare le scelte, lo scheduler implementa un particolare algoritmo di scheduling
- Sono stati proposti moltissimi algoritmi (ogni S.O usa un algoritmo leggermente diverso):
 - **Round-Robin (RR)**
 - **Scheduling con gestione delle priorità ...**

Scheduling Round-Robin (RR)

Funzionamento:

- Quando vi è una CPU disponibile, lo scheduler RR seleziona il processo che si trova da più tempo nella ready queue (politica FIFO)
- Ad ogni processo posto in stato running è assegnato un tempo massimo di esecuzione (detto **time slice** o **quanto di tempo**). Se il processo non termina o non richiede un'operazione di I/O entro questo tempo, lo scheduler sottrae la CPU al processo (**preemption**) e pone il processo in fondo alla ready queue
- Nuovi processi (appena creati) sono posti in fondo alla ready queue
- Non esistono priorità (nessun processo può "saltare la fila")

Caratteristiche di Round-Robin

- la coda dei processi pronti è gestita in modo FIFO: il prossimo processo da estrarre (first-out) è sempre quello arrivato da più tempo (first-in) nella coda
- Si dice anche che la coda è gestita in modo **circolare** (un processo che esce dalla coda deve "rifare il giro" per poter usare di nuovo la CPU)
- È un algoritmo **preemptive**
- Lo scheduler è attivato in risposta ad un interrupt, generato da un timer hardware
- Il dispositivo "timer" genera l'interrupt allo scadere di ogni time slice
- Non utilizza le priorità (tutti i processi trattati in modo equo, non è possibile concedere più tempo a processi "critici")

Scheduling con priorità

- In alcuni casi è utile assegnare una priorità maggiore a determinati processi
- ad esempio, alcuni S.O danno una priorità maggiore ai processi in **foreground** (primo piano, cioè i processi con cui l'utente sta interagendo direttamente) rispetto ai processi in **background** (secondo piano)
- Alcuni processi del S.O sono critici per il buon funzionamento del sistema e necessitano di una priorità maggiore dei programmi utente
- Uno scheduler potrebbe tener conto della priorità dei processi in questo modo: selezionare sempre dalla ready queue il processo con la priorità maggiore e porlo in esecuzione
- Problema: così facendo alcuni processi con bassa priorità potrebbero non venir mai eseguiti (questo fenomeno è detto **starvation**)

Thread

- Un **thread** è un flusso di esecuzione all'interno di un processo, cioè una sequenza di istruzioni che può essere eseguita "in parallelo" rispetto ad altri thread dello stesso processo
- Un thread può esistere solo all'interno di un processo che lo contiene
- Un processo può creare più thread al proprio interno (applicazione multi-thread)
- I thread dello stesso processo condividono molte delle risorse associate al processo (aree di memoria, file aperti, ecc..)
- I thread sono detti anche **processi leggeri** perchè:
 - occupano meno risorse per essere creati
 - il context switch tra i thread è più rapido di quello tra processi