Scheduling della CPU

Schedulazione della CPU

- Introduzione allo scheduling della CPU
- Descrizione di vari algorimi di scheduling della CPU
- Analisi dei criteri di valutazione nella scelta di un algoritmo per un particolare sistema

Concetti di base

- La multiprogrammazione cerca di ottenere la massima utilizzazione della CPU.
- CPU burst e I/O burst – l'esecuzione di un processo consiste in un **ciclo** d'esecuzione della CPU ed in un'attesa di I/O.
- Distribuzione dei CPU burst.

Sequenza alternata di CPU burst e di I/O burst

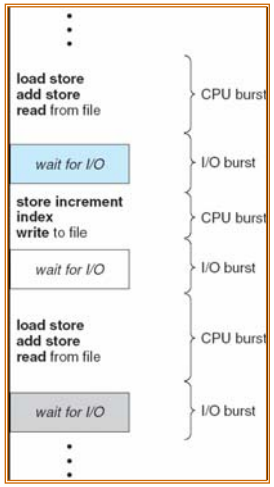
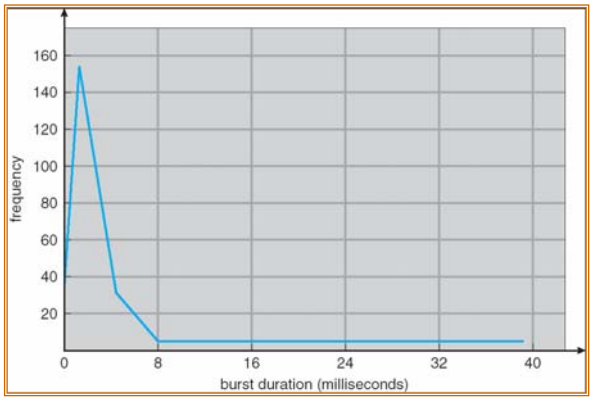


Diagramma delle durate dei CPU burst



Lo scheduler della CPU

- Il SO sceglie fra i processi in memoria che sono pronti per l'esecuzione ed assegna la CPU ad uno di essi.
- Lo scheduling della CPU può avvenire quando un processo:
  1. Passa dallo stato di esecuzione allo stato di attesa.
  2. Passa dallo stato di esecuzione allo stato di pronto.
  3. Passa dallo stato di attesa allo stato di pronto.
  4. Termina.
- La schedulazione nei punti 1 e 4 è detta **nonpreemptive** (senza diritto di *prelazione*).
- Tutte le altre schedulazioni sono dette **preemptive**.

Dispatcher

- Il **dispatcher** è il modulo del SO che dà il controllo della CPU ad un processo selezionato dallo scheduler. Questa funzione comprende:
  - cambio di contesto;
  - passaggio alla modalità utente;
  - salto alla corretta locazione nel programma utente per ricominciarne l'esecuzione.
- **Latenza del dispatcher** – tempo necessario al dispatcher per fermare un processo e cominciarne un altro.

Criteri di scheduling

- **Utilizzo della CPU** – mantenere la CPU il più impegnata possibile.
- **Frequenza di completamento** (*throughput*) – numero di processi completati per unità di tempo.
- **Tempo di completamento** (*turnaround time*) – intervallo che va dal momento dell'immissione del processo nel sistema al momento del completamento.
- **Tempo di attesa** (*waiting time*)– somma dei tempi spesi in attesa nella coda dei processi pronti.
- **Tempo di risposta** – tempo che intercorre dalla formulazione della prima richiesta fino alla produzione della prima risposta, **non** l'output (per gli ambienti di time-sharing).

Criteri di ottimizzazione

- Massimizzare l'utilizzo della CPU.
- Massimizzare il throughput.
- Minimizzare il turnaround time.
- Minimizzare il tempo di attesa.
- Minimizzare il tempo di risposta.

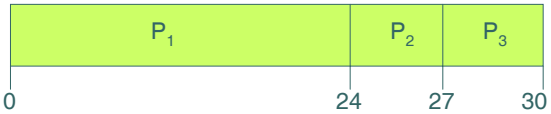
Solitamente si ottimizzano i valori medi. A volte è oppurtuno ottimizzare i valori minimi o massimi. Per i sistemi interattivi andrebbe minimizzata la varianza

Algoritmi di scheduling

First-Come First-Served (FCFS)

Processo	CPU burst
P_1	24
P_2	3
P_3	3

- Se i processi arrivano nell'ordine: P_1 , P_2 , P_3
  si ottiene il risultato mostrato nel seguente diagramma di Gantt:



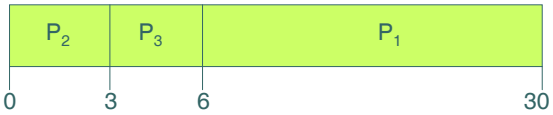
- Tempo di attesa per P_1 = 0; P_2 = 24; P_3 = 27
- Tempo di attesa medio: (0 + 24 + 27)/3 = 17

Caratteristiche di FCFS

Se i processi arrivano nell'ordine:

P_2 , P_3 , P_1

- Il diagramma di Gantt è:

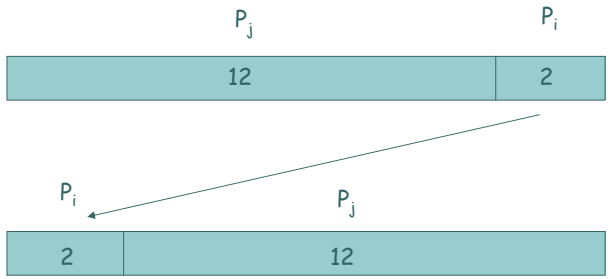


- Tempo di attesa per P_1 = 6; P_2 = 0; P_3 = 3
- Tempo di attesa medio: (6 + 0 + 3)/3 = 3
- Molto meglio del caso precedente.
- C'è un effetto di ritardo a catena (*convoy effect*) mentre tutti i processi attendono che quello grosso rilasci la CPU.

Shortest-Job-First (SJF)

- Schedula il processo con il prossimo CPU burst più breve.
- L'algoritmo SJF può essere:
  - **nonpreemptive** – quando un processo ha ottenuto la CPU, non può essere prelazionato fino al completamento del suo cpu-burst.
  - **preemptive** – quando un nuovo processo è pronto ed il suo CPU-burst è minore del tempo di cui necessita ancora il processo in esecuzione c'è prelazione. Questa schedulazione è anche detta *shortest-remaining-time-first*.
- SJF è ottimale – fornisce il minor tempo di attesa medio per un dato gruppo di processi.

SJF Ottimale

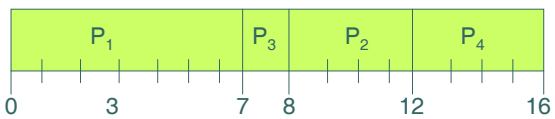


Spostando un processo breve prima di un processo lungo, il tempo di attesa del processo breve diminuisce più di quanto aumenti il tempo d'attesa per il processo lungo. Di conseguenza il tempo d'attesa medio diminuisce.

SJF Non-Preemptive

Processo	Tempo di arrivo	CPU burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



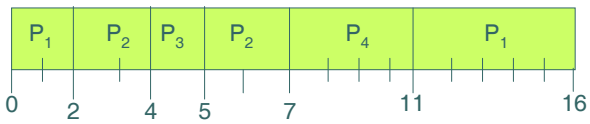
- Tempo di attesa medio = (0 + 6 + 3 + 7)/4 = 4

8-2 7-4 12-5

SJF Preemptive

Processo	Tempo di arrivo	CPU burst
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Tempo di attesa medio = (9 + 1 + 0 +2)/4 = 3

11-2 ...

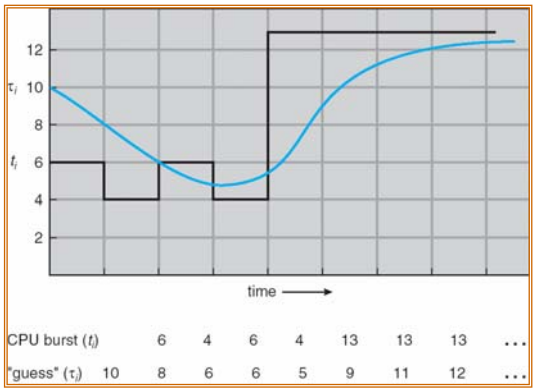
Stima del prossimo CPU burst

- É possibile fare solo una stima della lunghezza.
- Può essere fatta utilizzando la lunghezza dei precedenti CPU burst, usando una media esponenziale.

t_n = valore reale dell'ennesimo CPU burst
τ_{n+1} = valore previsto per il prossimo CPU burst
α = parametro con valore $0 \leq \alpha \leq 1$
τ_1 = previsione 1° CPU burst (valore di default)

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\, \tau_n$$

Previsione della durata del prossimo CPU burst



Esempi di media esponenziale

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - La storia recente non ha nessun effetto.
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Conta solo il CPU burst più recente.
- Se sviluppiamo la formula otteniamo:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\,\alpha\, t_{n-1} + \ldots + (1-\alpha)^j \alpha\, t_{n-j} + \ldots + (1-\alpha)^{n-1} \alpha\, t_1 + (1-\alpha)^n \tau_1$$

- Poichè sia α che $(1-\alpha)$ sono minori o uguali a 1, ciascun termine successivo ha un peso inferiore rispetto a quello precedente.

Scheduling a priorità

- Si associa una priorità numerica a ciascun processo.
- La CPU viene allocata al processo con priorità più alta (più piccolo è il numero ≡ più alta è la priorità).
  - preemptive
  - nonpreemptive
- SJF è un algoritmo a priorità dove la priorità è il prossimo picco (previsto) di CPU.
- Problema ≡ Blocco indefinito (*starvation*) – processi a bassa priorità potrebbero non essere mai eseguiti.
- Soluzione ≡ Invecchiamento (*aging*) – accrescere gradualmente le priorità dei processi nel sistema.

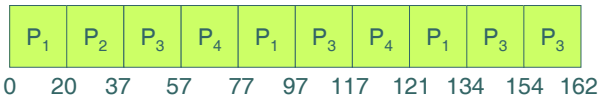
Round Robin (RR)

- Ogni processo riceve la CPU per una piccola unità di tempo (*time quantum*), generalmente 10-100 millisecondi. Se entro questo arco di tempo il processo non lascia la CPU, viene interrotto e rimesso nella coda dei processi pronti.
- Se ci sono n processi nella coda dei processi pronti e il quanto di tempo è q, allora ciascun processo ottiene $1/n$ del tempo di CPU in parti lunghe al più q. Ciascun processo non deve attendere più di $(n-1) \times q$ unità di tempo.
- Prestazioni:
  - q grande ⇒ FIFO
  - q piccolo ⇒ q deve essere grande rispetto al tempo di un context switch, altrimenti l'overhead diventa troppo elevato.

Esempio di RR con quanto di tempo pari a 20 millisecondi

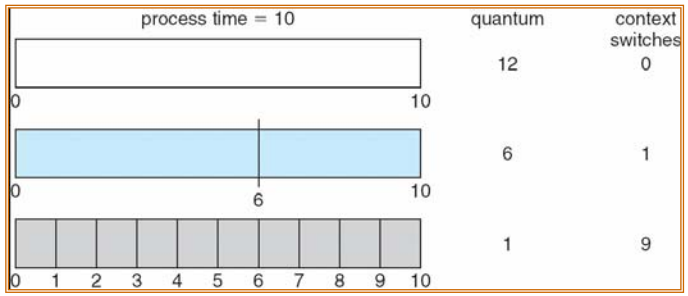
Processo	CPU burst
P_1	53
P_2	17
P_3	68
P_4	24

- Il diagramma di Gantt è:

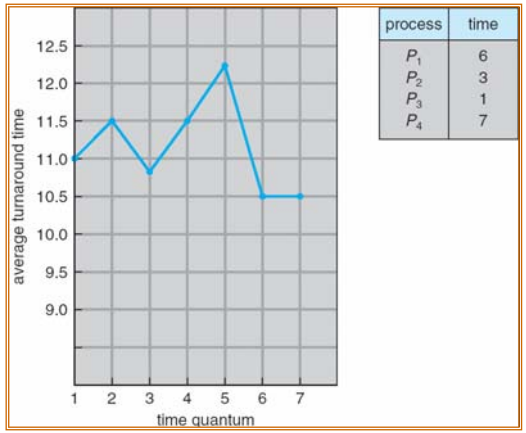


- Tipicamente, il tempo medio di turnaround è più alto di SJF, ma il tempo di risposta è più breve.

Quanto di tempo e context switch



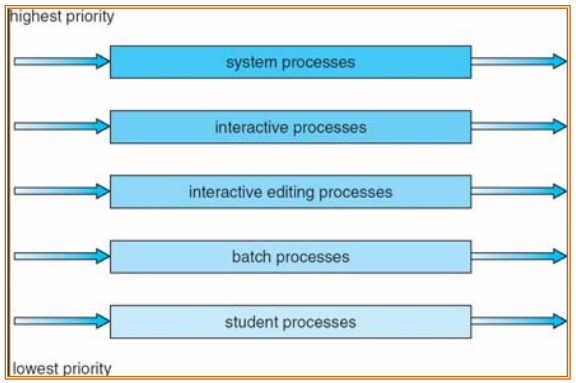
Variazione del tempo medio di turnaround in funzione del quanto di tempo



Scheduling a code multiple

- **La coda dei processi pronti è partizionata in code separate:**
  ***foreground* (interattivi),**
  ***background* (*batch* – sullo sfondo).**
- **Ciascuna coda ha il proprio algoritmo di scheduling:**
  - **foreground – RR**
  - **background – FCFS**
- **Ci deve essere una schedulazione tra le code**
  - ***A priorità fissa*; (e.g., tutti i processi in foreground, poi quelli in background). Possibilità di starvation.**
  - ***Time slice* – ciascuna coda ha una certa quantità di tempo di CPU, che può schedulare fra i processi in essa contenuti;e.g., 80% del tempo di CPU per la coda foreground (RR), 20% background in FCFS.**

Scheduling a code multiple



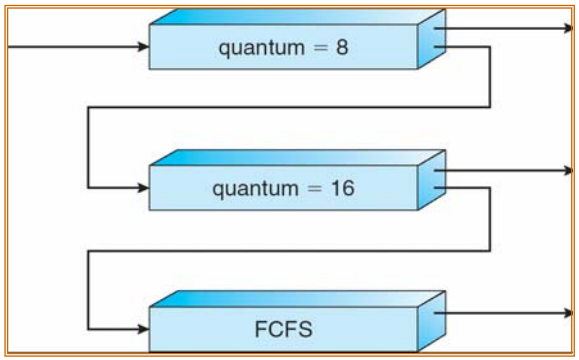
Code multiple con feedback

- Un processo può muoversi tra le varie code; l'aging potrebbe essere implementato in questo modo.
- Uno schedulatore a code multiple con feedback è definito dai seguenti parametri:
  - numero di code;
  - algoritmo di schedulazione per ciascuna coda;
  - metodo utilizzato per "far salire" un processo verso una coda a priorità più alta;
  - metodo utilizzato per spostare un processo in una coda a più bassa priorità;
  - metodo utilizzato per determinare in quale coda entrerà un processo quando avrà bisogno di un servizio.

Esempio di code multiple con feedback

- Tre code:
  - Q_0 – RR con quanto di tempo: 8 millisecondi
  - Q_1 – RR con quanto di tempo: 16 millisecondi
  - Q_2 – FCFS
- Schedulazione
  - Un nuovo processo entra nella coda Q_0. Quando schedulato ottiene la CPU per 8 millisecondi. Se non termina in 8 millisecondi, viene spostato nella coda Q_1.
  - In Q_1 il processo, quando schedulato, riceve la CPU per 16 millisecondi. Se non completa entro I 16 millisecondi, viene spostato nella coda Q_2.

Code multiple con feedback



Scheduling nei sistemi multiprocessore

Scheduling nei sistemi multiprocessore

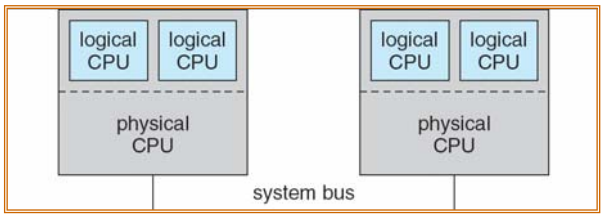
- Se sono disponibili più CPU il problema dello scheduling diviene più complesso.
- **Processori omogenei** all'interno di un multiprocessore.
- **Multiprocessing asimmetrico** *(asymmetric multiprocessing)* – solo un processore accede alle strutture dati del sistema, riducendo così la necessità della condivisione dei dati

Scheduling nei sistemi multiprocessore simmetrici

- **Predilezione dei processori.** Significa tentare di riallocare ad un processo la stessa CPU su cui era in esezione prima. Usata per sfruttare le cache al meglio (i.e., dati del processo già presenti) – *soft/hard affinity*
- **Coda dei processi.** Condivisa o code locali per ogni processore
- **Bilanciamento del carico.** Occorre distribuire uniformemente i processi tra i processori disponibili
  - *migrazione guidata (push migration)*
  - *migrazione spontanea (pull migration)*

Symmetric Multithreading
(Hyperthreading Technology per Intel)

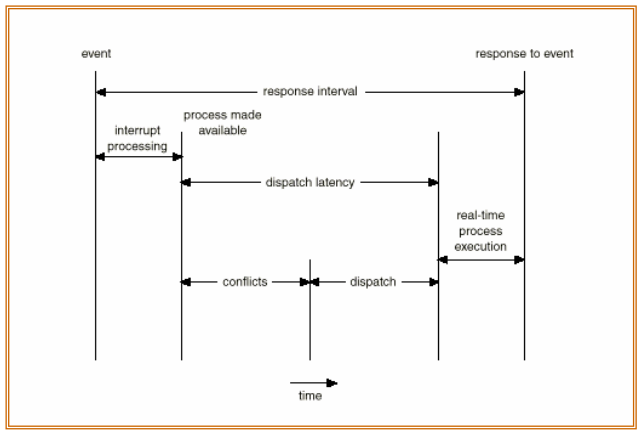
- Più processori logici su ogni processore fisico
- Supporto per SMT offerto dall'hardware non dal software
- Lo scheduler cerca prima di usare processori logici su processori fisici diversi



Schedulazione nei sistemi real-time

- Sistemi **hard real-time** – devono completare un'operazione critica entro una quantità di tempo garantita.
- Computazione **soft real-time** – richiede che i processi critici ricevano priorità su quelli meno importanti.

Latenza di dispatch



Scheduling dei thread

- **Locale - Process-contention scope (PCS).** Schema che specifica come i thread utente debbano essere schedulati dalla libreria sugli LWP (modelli molti-a-uno e molti-a-molti).
- **Globale - System-contention scope (SCS).** Schema che specifica come tutti i thread del sistema debbano essere schedulati dal kernel sulla CPU.
- Tipicamente **PCS** avviene in base alla priorità.
- I sistemi che utilizzano il modello uno-a-uno usano solo SCS.

API Pthread

Pthread permette due differenti politiche di schedulazione

- PTHREAD_SCOPE_PROCESS (utilizza la schedulazione PCS)
- PTHREAD_SCOPE_SYSTEM (utilizza la schedulazione SCS)

L'uso della politica PTHREAD_SCOPE_SYSTEM su un sistema che implementa i thread secondo il modello molti-a-molti, forza il kernel a mappare i thread utente secondo il modello uno-a-uno

Algoritmi di scheduling specifici sono

- SCHED_RR
- SCHED_FIFO
- SCHED_OTHER

API Pthread: esempio

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

int main(int argc, char *argv[])
{ int i;
   pthread_t tid[NUM_THREADS];
   pthread_attr_t attr;

   /* ottiene i valori di default degli attributi */
   pthread_attr_init(&attr);

   /* sceglie l'algoritmo di schedulazione: PROCESS oppure SYSTEM */
   pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);

   /* sceglie la plitica di schedulazione: FIFO, RT oppure OTHER */
   pthread_attr_setschedpolicy(&attr, SCHED_OTHER);

   /* crea i threads */
   for (i = 0; i < NUM_THREADS; i++)
       pthread_create(&tid[i],&attr,runner,NULL);
```

Scheduling nell'API Pthread

```
   /* attende ciascun thread */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_join(tid[i], NULL);
}

/* Ogni thread inizierà il controllo in questa funczione */
void *runner(void *param)
{
   printf("Io sono un thread\n");
   pthread_exit(0);
}
```

Scheduling dei thread in Java

Lo scheduling dipende dalla JVM. Le specifiche non indicano se RR deve essere implementato o se c'è prelazione

Il thread corrente resta in esecuzione finchè:

1. termina il suo quanto
2. si blocca per I/O
3. esce dal metodo run()

Se l'algoritmo di schedulazione implementato è preemptive, il thread corrente può essere prelazionato se ne arriva uno con maggiore priorità

Multitasking cooperativo

Poichè la JVM non assicura il time-slicing, è possibile ricorrere al metodo yield():

```
while (true) {
        // segue un compito con uso intensivo della CPU
        . . .
        Thread.yield();
}
```

Il metodo yield() suggerisce che il thread intende passare il controllo ad un altro thread.

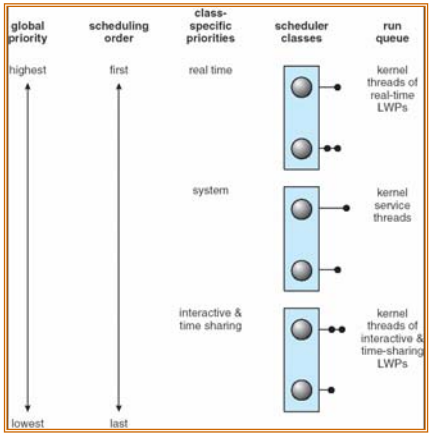
Priorità dei thread

Priorità	**Commento**
Thread.MIN_PRIORITY	La priorità minima dei thread
Thread.MAX_PRIORITY	La priorità massima dei thread
Thread.NORM_PRIORITY	La priorità di default dei thread

Le priorità possono essere modificate usando il metodo *setPriority()*:

```
setPriority(Thread.NORM_PRIORITY + 2);
```

Scheduling in Solaris



Scheduling in Solaris

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Scheduling in Windows XP

Usa un algoritmo preempitive con priorità a 32 livelli.
Le priorità son divise in 2 classi: variabile e real-time

	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

Scheduling in Linux

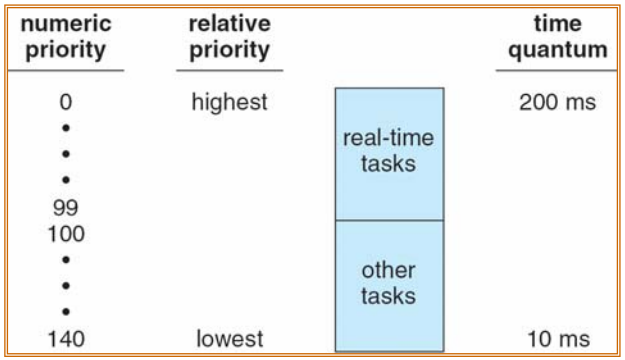
Lo scheduler di linux è preemptive e basato su priorità. Usa due range di valori di priorità – **real time** [0,99] e **nice** [100, 140].

I valori di questi due range sono mappati in un valore di priorità globale. Numeri bassi →priorità alta.

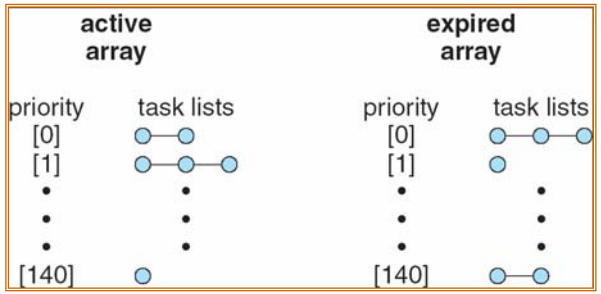
Il kernel mantiene una lista di tutti i task in una **runqueue**

La runqueue contiene due array di priorità – **active array** e **expired array**

Relazione tra le priorità e la lunghezza del time-slice



Liste dei task indicizzate in base alla priorità



Valutazione degli algoritmi di scheduling

- Modellazione deterministica – prende un particolare carico di lavoro e definisce le prestazioni di ciascun algoritmo per quel carico.
- Reti di code.
- Simulazione.
- Implementazione.

Valutazione mediante simulazione