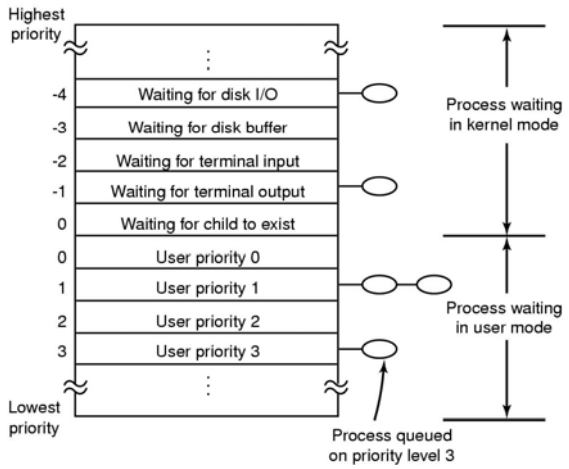
Lo scheduler di UNIX (1)



Lo scheduling a basso livello è basato su una coda a più livelli di priorità

1

Lo scheduler di UNIX (2)

- Si esegue il primo processo della prima coda non vuota per massimo 1 *quanto* (tipicamente 100ms)
- Scheduling round robin fra processi della stessa coda
- Il numero di coda si calcola con priorità/costante
- Una volta al secondo tutte le priorità vengono ricalcolate:

Più è bassa meglio è

priorità = cpu _usage + nice + base

cpu _usage : numero di clock tick per secondo che il processo ha avuto negli ultimi secondi

nice : valore intero nell'intervallo [-20, +20] Bonus che può essere dato:

base : valore intero che dipende da cosa sta facendo il processo

-20 è Bonus buono
+20 è Bonus malus

- ha il valore della priorità precedente se il processo sta eseguendo elaborazione normale in user mode
- ha un valore negativo molto basso se sta effettuando I/O da disco o da terminale Rola interattiva porta via altre al processo

2

Lo scheduler di UNIX (3)

Meccanismo di *aging (invecchiamento)* usato per il calcolo di *cpu _usage* :

- Fissiamo un intervallo di decadimento Δt → Intervallo di ricalcolo in cui aggiorno priorità
- I tick ricevuti mentre il processo P è in esecuzione vengono accumulati in una variabile temporanea *tick*
- Ogni Δt → Aggiornato con algoritmo di aging

  cpu _usage = cpu _usage / 2 + tick / 2 *

  tick = 0 → Ultima volta di quello che gli è stato assegnato
- Il peso dei tick utilizzati decresce col tempo
- La penalizzazione dei processi che hanno utilizzato molta CPU diminuisce nel tempo

3

* Tick precedente pesa $\frac{1}{2}$, precedente ancora $\frac{1}{4}$ ... Decadimento esponenziale

Lo scheduler di Linux (1)

- Vengono schedulati i thread, non i processi
- Tre classi di thread : real-time FIFO (non prerilasciabile), real-time Round Robin, Timesharing → Schedulati in FIFO MOLTO PRIORITARI → Thread normali
- Ogni thread ha → Processi schedulabili usando approccio round robin ma hanno in genere priorità molto alta
  - una *priorità* nell'intervallo [0, +40], generalmente all'inizio la priorità di default è 20
  - un *quanto* (misurato in *jiffy* = 10ms) (data di elaborazione)
- Lo scheduler calcola la *goodness* (*gdn*) di ogni thread pronto come

```
if (class == real-time) gdn = 1000 + priority   (subito schedulati perché hanno goodness alta)
if (class == timeshar && quantum > 0) gdn = quantum + priority   → più alta meglio è
if (class == timeshar && quantum == 0) gdn = 0   → Somma il quanto che gli è rimasto
```

4

Se spende tutto ha goodness 0.

PRIORITÀ È COMUNQUE STATICA, ma ad un certo punto, tutti i thread ready tendono a consumare tutto il loro quanto.

Lo scheduler di Linux (2)

Algoritmo di scheduling :

- Ogni volta viene selezionato il thread con goodness maggiore
- Ogni volta che arriva un tick il quanto del thread in esecuzione viene decrementato
- Un thread viene de-schedulato se si verifica una delle seguenti condizioni
  - il quanto diventa 0
  - il thread si blocca
  - diventa ready un thread con una goodness maggiore

5

Lo scheduler di Linux (3)

Algoritmo di scheduling (cont.):

- Poiché i quanti continuano ad essere decrementati, prima o poi, tutti i quanti dei thread ready andranno a 0. A questo punto lo scheduler ricalcola il quanto di ogni thread (anche se *blocked*) come segue :

  quantum = quantum / 2 + priority

- Se dovesse avvenire un nuovo ricalcolo del quanto, un processo bloccato che non ha ancora ottenuto l'uso della CPU tende ad acquisire un valore del quanto sempre più alto. In questo modo vengono favoriti i thread I/O bound.
- Due thread CPU-bound, con questo sistema, tendono ad occupare la CPU in proporzione alla priorità.

6

Scheduling in Windows 2000 (1)

- Le API Win32 permettono all'utente di specificare :
  - priorità di un processo (6 livelli diversi)
  - priorità di un thread all'interno di un processo (7 livelli diversi)
- Windows 2000 mappa le 42 combinazioni possibili su 32 livelli di priorità
- Windows 2000 non possiede un thread centrale per la schedulazione, è il thread stesso che, quando non può continuare l'esecuzione per qualche motivo, entra in modo kernel, attiva lo scheduler e si sceglie il successore.

A un processo sono associati thread

7

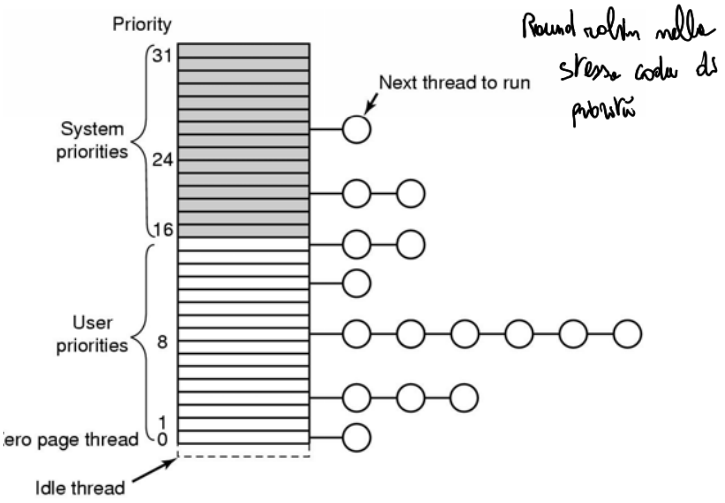
Scheduling in Windows 2000 (2)

		Win32 process class priorities					
		Realtime	High	Above Normal	Normal	Below Normal	Idle
Win32 thread priorities	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

Corrispondenza fra le priorità di Win32 e quelle di Windows 2000

8

Scheduling in Windows 2000 (3)



Windows 2000 fornisce 32 priorità diverse per i thread

9

Scheduling in Windows 2000 (4)

Algoritmo di scheduling :

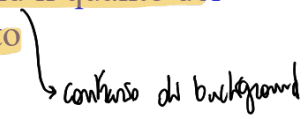
- Si esegue il primo thread della prima coda non vuota per massimo 1 *quanto* (20ms--120ms)
- Scheduling round robin fra thread con la stessa priorità
- Come variano le priorità nel tempo :
  - i thread tipicamente entrano a priorità 8
  - la priorità viene elevata se:
    - viene completata una operazione di I/O (+1 disco, +2 linea seriale, +6 tastiera, +8 scheda audio …)
    - termina l'attesa su un semaforo, mutex, evento (+1 background, +2 foreground)
    - l'input nella finestra di dialogo associata al thread è pronto

10

Scheduling in Windows 2000 (5)

Algoritmo di scheduling :

- Come variano le priorità nel tempo (cont.):
  - la priorità viene abbassata se:
    - un thread usa tutto il suo quanto (-1), fino a ritornare alla priorità base
  - se un thread non ha girato per un tempo maggiore di una soglia fissata, allora passa per 2 quanti a priorità 15
- Quando una finestra va in foreground il quanto dei thread corrispondenti viene allungato



11