CPU Scheduling

Lo scheduler di Linux

La politica di scheduling di Linux si propone il raggiungimento dei seguenti obiettivi (molti dei quali sono in contrasto):

- timesharing
- gestione di priorità dinamiche
- avvantaggiare i processi I/O-bound
- tempi di risposta bassi
- throughput elevato per i processi in secondo piano
- evitare starvation

L'algoritmo di scheduling cerca di raggiungere i suddetti obiettivi attraverso

- una suddivisione del tempo in **epoche**
- dipendenza della priorità dal residuo del quanto di tempo
- definizione delle condizioni che determinano l'invocazione dello scheduler

Politica di scheduling

In Linux lo scheduling si basa sul concetto di timesharing, per cui ad ogni processo è assegnato un quanto di tempo massimo per l'esecuzione.

La selezione del prossimo processo da eseguire è basata sul concetto di priorità; questa può essere dinamica o statica. In particolare, la prima introdotta per evitare il fenomeno della *starvation*, mentre la seconda è stata è stata introdotta per consentire la gestione di processi real-time. Ai processi real-time (più precisamente soft real-time, utili ad esempi per riprodurre flussi audio e/o video in applicazioni multimediali) è assegnata una priorità maggiore di quella assegnata ai processi "ordinari".

Di norma Linux prevede uno scheduling con prelazione (*preemptive*), per cui ad

un processo viene tolta la CPU se:

- esaurisce il quanto di tempo a sua disposizione
- un processo a priorità più alto è pronto per l'esecuzione (`TASK_RUNNING`).

La durata del quanto di tempo è tipicamente di 20 ***clock ticks***, o 210 millisecondi. Questo è circa il più grande valore ammissibile senza che sia compromessa la reattività del sistema. I processi possono variare la durata del quanto tramite la funzione `nice(int n)`; tuttavia, gli utenti comuni possono solo diminuirlo, mentre l'utente amministratore può anche aumentarlo.

Algoritmo di scheduling

Il tempo è suddiviso in periodi, detti **epoche**, che si ripetono ciclicamente.

All'inizio di ogni epoca, a ciascun processo viene assegnato un quanto di tempo. Il quanto di tempo definisce un limite superiore al tempo di utilizzo di della CPU nell'epoca in questione; si osservi che, in una stessa epoca, la CPU può essere assegnata ad un processo anche più volte, finché il quanto non è esaurito.

Un'epoca termina quando tutti i processi eseguibili (quelli, cioè, che si trovano nello stato `TASK_RUNNING`) hanno esaurito il loro quanto. Terminata un'epoca, ne inizia un'altra; si calcola quindi il nuovo quanto di tempo da assegnare a ciascun processo. Questo calcolo prende in considerazione il ***base time quantum*** (cioè, il valore predefinito per il quanto di tempo) e l'eventuale residuo dall'epoca precedente; infatti, processi che erano bloccati al momento in cui è terminata l'epoca, potevano non aver esaurito il quanto di tempo. Così facendo si assegna un bonus ai processi I/O-bound; sebbene un processo I/O-bound difficilmente esaurirà il quanto a sua disposizione, questo è utilizzato per determinare la priorità del processo.

Ai processi ordinari, cioè quelli che vengono gestiti con la politica **`SCHED_OTHER`**, è assegnta una priorità statica pari a 0; la priorità dinamica, invece, è determinata come indicato sopra.

Ai processi real-time è assegnata un livello di priorità statica compreso tra 1 e 99, e non cambia mai durante l'esecuzione. Da questo si evince che i processi real-time hanno una priorità statica superiore a quella dei processi ordinari; poiché lo scheduler seleziona il processo a priorità più alta, un processo ordinario può essere eseguito solo se non ci sono processi real-time pronti per l'esecuzione.

I processi real-time possono appartenere ad una di due categorie:

- **SCHED_FIFO**, ovvero processi real-time con quanto di tempo illimitato. Questi processi lasciano la CPU solo se 1) si bloccamo, 2) terminano, 3) un processo a priorità più alto diviene pronto, 4) terminano;
- **SCHED_RR**, ovvero processi real-time soggetti a timesharing, cioè scheduling di tipo Round Robin.

Le informazioni necessarie allo scheduler, comprese quelle relative alla priorità, sono contenute nel descrittore di processo (**task_struct**). In particolare, queste sono:

- `long int` **state**, cioè lo stato del processo;
- `long int` **need_resched**, che indica se è necessaria l'invocazione dello scheduler. Questo campo viene controllato al termine di ogni routine di servizio delle interruzioni;
- `long int` **policy**. Può assumere i valori **SCHED_OTHER**, **SCHED_FIFO**, **SCHED_RR**, **SCHED_YIELD**;
- `long int` **rt_priority**, indica il livello di priorità statica per i processi real-time;
- `long int` **priority**, rappresenta il quanto di tempo (espresso in clock ticks) assegnato ad un processo soggetto a timesharing;
- `long int` **counter**, è il numero di clock ticks residui. La variabile è aggiornata ad ogni interruzione dell'orologio dalla funzione `update_process_times()`.

Le variabili `counter` e `priority` non sono utilizzate per i processi soggetti alla politica **`SCHED_FIFO`**.

Invocazione dello scheduler

Lo scheduler viene invocato tramite la funzione **`schedule()`**, che ha lo scopo di individuare il processo a maggiore priorità fra quelli pronti. La sua invocazione può avvenire in due modi:

- **invocazione diretta** (o ***direct invocation***). Questa modalità viene eseguita ogni volta che il processo si blocca (stati **`TASK_UNINTERRUPTIBLE`** o **`TASK_INTERRUPTIBLE`**);
- **invocazione ritardata** (o ***lazy invocation***). Questa ha luogo quando, al termine dell'esecuzione di una primitiva di sistema, la variabile `need_resched` indica che è necessario eseguire lo scheduler. Questa variabile viene impostata se:
  - durante l'esecuzione della funzione `update_process_times()` si verifica che `counter` <= 0;
  - durante l'esecuzione della funzione **`wake_up_process()`** ci si accorge che è passato nello stato di pronto (`TASK_RUNNING`) un processo a priorità più alta di quello attualmente in esecuzione;
  - il processo chiama la funzione **`sched_yield()`**.

L'esecuzione della funzione `schedule()` comprende i seguenti passi:

- completamento di eventuali system calls in corso (nel caso di kernel non preemptible non è consentita la commutazione di processo mentre il kernel sta eseguendo delle operazioni per conto di un processo) e operazioni di manutenzione del sistema;
- se il processo è di tipo `SCHED_RR` ed ha esaurito il quanto, viene posto in fondo alla coda e gli viene assegnato un nuovo quanto (per questo tipo di processi non vale la suddivisione in epoche);

- Si analizza la coda dei processi pronti per determinare qual'è il processo a priorità maggiore. Questa selezione viene fatta mediante la funzione `goodness()`, descritta nel seguito.
  - Se è in grado di selezionare un processo, allora viene invocato il dispatcher per eseguire l'assegnazione;
  - altrimenti si inizia una nuova epoca.

Per un processo, la funzione `goodness()` può produrre uno dei seguenti risultati:

- G=-1000, ad indicare che il processo non deve essere selezionato;
- G=0, se il processo ha esaurito il quanto;
- 0 < G < 1000, se si tratta di un processo ordinario che non ha ancora esaurito il quanto di tempo. Il valore restituito rappresenta la sua priorità dinamica;
- G >= 1000, per un processo real-time. Se G =1000+g, allora g è il livello di priorità.

Nel caso in cui più processi assumano il valore massimo di goodness, viene selezionato quello che è stato incontrato per primo nella scansione della coda dei processi pronti.

Problemi

- Le prestazioni dello scheduler di Linux degradano al crescere del numero di processi; infatti le priorità (cioè i quanti) devono essere ricalcolate per tutti i processi al termine di ogni epoca.
- Il valore predefinito per il quanto può essere eccessivo nei sistemi con carico elevato.
- Il meccanismo di ***boost*** dei processi I/O-bound non è ottimale; infatti, non tutti i processi comportano un'interazione con l'utente, e quindi potrebbero essere gestiti in maniera diversa (es. database, trasferimento dati via rete, ...).

- Il supporto fornito ai processi real-time è limitato.

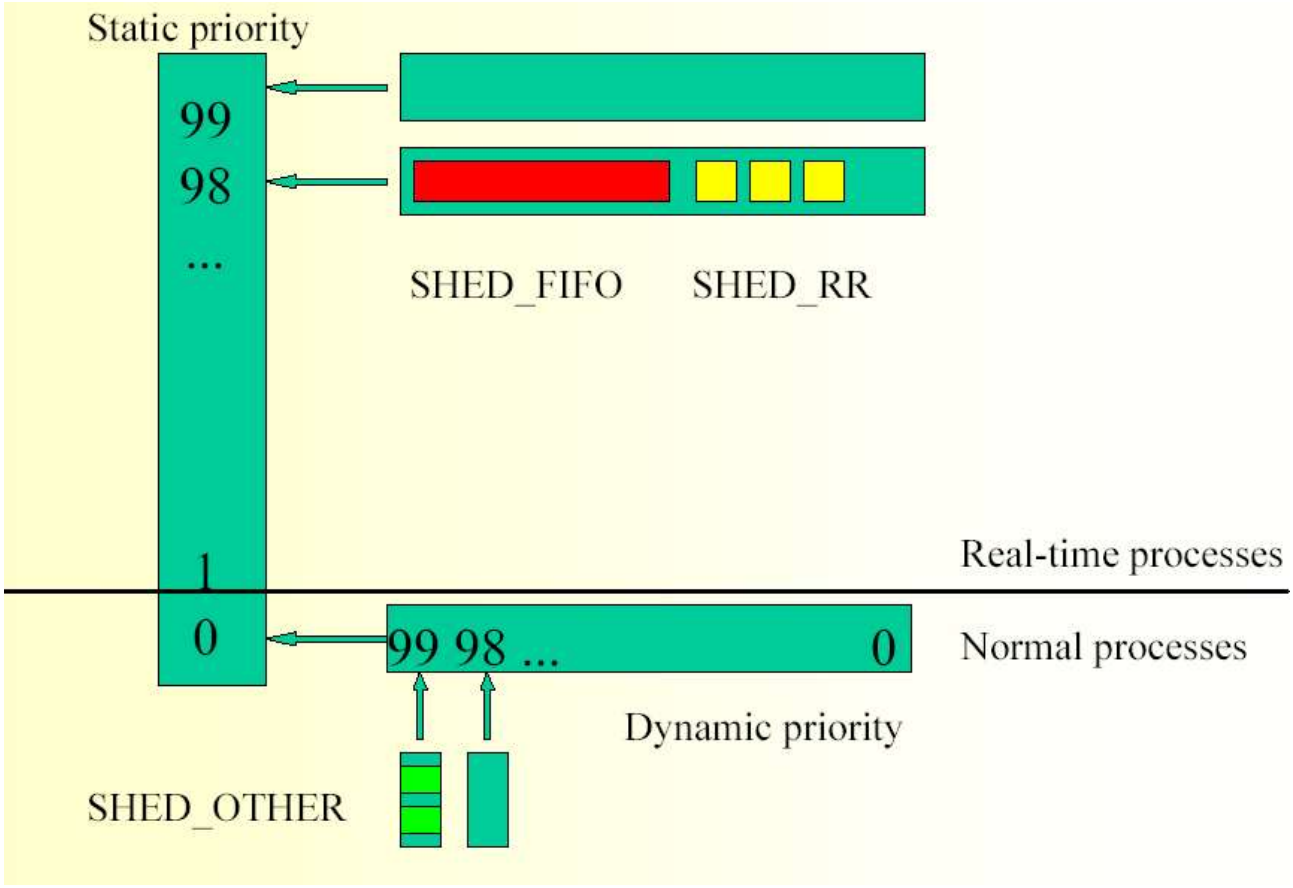

Figura 1: Code per la gestione dei processi nei sistemi Linux.