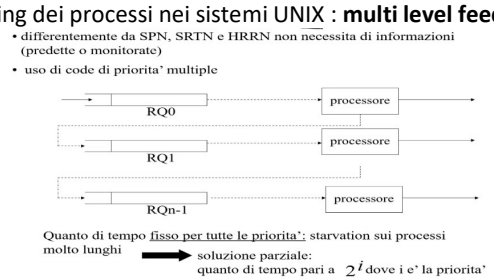
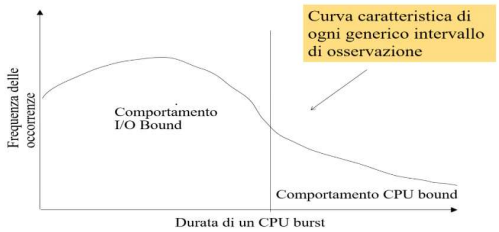
Scheduling 2 UNIX

martedì 28 ottobre 2025 09:26

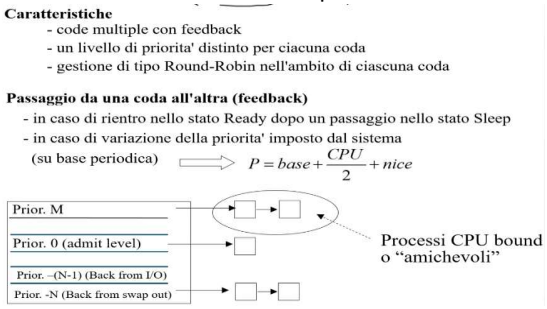
Vediamo ora lo scheduling dei processi nei sistemi UNIX : **multi level feedback queue scheduling**



Le code hanno priorità diverse . Se un processo entro nello stato ready entra nella prima coda secondo FIFO (first in first out) , assegnandogli un quanto di tempo e se arriva interrupt (il processo finisce l'esecuzione), viene spostato all'interno della coda di livello inferiore. Data la priorità (**0 è la più alta**), per passare a eseguire un processo nella coda a priorità più bassa (1,2,...n) bisogna aspettare che la coda a priorità superiore sia vuota. Sorge un problema ? Cosa succede se il processo utilizza meno quanto di quello disponibile?? Il processo viene rimesse nell'ultima coda dove stava : **se utilizzo tutto il quanto il livello si declassa** . In generale nella prima coda si hanno processi i/o bound. Il timer non è a granularità arbitraria (non misuro RR): il timer manda interrupt ogni quanto si vuole . Non si ha un feedback migliorativo in quanto **non è possibile fare risalire il processo alla coda precedente** . Andiamo ora a vedere una variazione di questo algoritmo osservando la curva di carico:



Nota : è possibile che vi sia l'interscambiabilità tra i comportamenti. Andiamo quindi a vedere lo **schema di scheduling tradizionale** :



Attenzione al feedback : i processi possono sia scendere che salire . All'interno di ogni coda si usa il round-robin. Vediamo come funziona il feedback : se parto da ready , vado a sleep e torno a ready , mi riviene assegnata nuova priorità. **In dettaglio : le priorità vanno da +M (cpu bound) fino a –N (circa 40); il livello 0 è intermedio ; quelli inferiori allo 0 sono migliori ; altrimenti peggiori .** Esempio : se applicazioni parte dal livello 0 e esaurisce il quanto , viene schedulate al livello N-1; mentre se applicazione riportata in memoria (swap in), dopo che era stata bloccata e portata fuori(swap out), gli viene assegnata priorità massima : sarà la prossima ad essere eseguita. Se invece da running torno a ready cosa succede ? (gli assegno la priorità -N) Viene assegnata la priorità di riferimento di quel processo secondo la formula : **dove la base (quasi sempre 0) , CPU/2 il tempo di utilizzo della cpu , nice** (valore anche negativo , la quale viene cambiata dalle system call) . Vediamo ora in dettaglio come cambiare questa niceness del processo in esecuzione:

Baseline system call `int nice(int incr)`

The range of the nice value is +19 (low priority) to -20 (high priority). Attempts to set a nice value outside the range are clamped to the range

questa e' la system call che viene invocata dalla shell quando si passa sulla linea di comando il comando **nice [+number] [command]**

Quindi come configuriamo la priorità :

```
#include <sys/time.h>
#include <sys/resource.h>

int getpriority(int which, int who);
int setpriority(int which, int who, int prio);
```

DESCRIPTION
The scheduling priority of the process, process group, or user, as indicated by *which* and *who* is obtained with the getpriority call and set with the setpriority call. *Which* is one of PRIO_PROCESS, PRIO_PGRP, or PRIO_USER, and *who* is interpreted relative to *which* (a process identifier for PRIO_PROCESS, process group identifier for PRIO_PGRP, and a user ID for PRIO_USER). A zero value for *who* denotes (respectively) the calling process, the process group of the calling process, or the real user ID of the calling process. *Prio* is a value in the range -20 to 20 (but see the Notes below). The default priority is 0; lower priorities cause more favorable scheduling

Nota : se si ha un terminale che lancia altre applicazioni (comandi , background ecc ecc) si parla di gruppo , quindi in UNIX si parla di auto-grouping e se si cambia la nice dal terminale, quel cambiamento è locale solo a tutti i processi/thread del gruppo. Vediamo ora degli esempi :

```
void __start(){
        while(1);
}
```

Che compilato con –nostartfile genera il seguente output

```
ripristinato il predefinito 0000000000001000
```

Quindi la prima istruzione che viene eseguita è all'indirizzo sopra

```
renice -n 10 25028
```

Consente di impostare la niceness al valore 10 a quel determinato processo con PID specificato. Se passo nice 0 -> errore , a meno che non do lo faccio con i privilegi di admin (sudo)

```
/*
  try this program with different inputs and by relying on the chrt shell command
*/

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/time.h>
#include <sys/resource.h>

unsigned long spawns;
unsigned long busy_loop;

#define SCALING 1000
//analogo al taskset da terminale
#ifdef AUTO_AFFINITY
#include <sched.h>
ulong CPU_set = 0x1;
cpu_set_t *mask = (cpu_set_t*)&CPU_set;
#endif

void * child_thread(void*p){
        int i;
        unsigned long id;
        int round=0;
        id = (unsigned long)p;
        while(1){
                for (i= 0; i< SCALING*busy_loop; i++);
        }
}
```

```
int main(int argc, char** argv){
   int i;
   int status;
   void **thread_status;
   pthread_t tid;
   int prio;
   int ret;
   if(argc<3){
        printf("usage: command spawns busy-loop-length\n");
        exit(EXIT_FAILURE);
   }

#ifdef AUTO_AFFINITY
        ret = sched_setaffinity(getpid(),sizeof(ulong),mask);
        printf("affinity set returned with code %d\n",ret);
#endif

   spawns = strtol(argv[1],NULL,10);//string to long
   busy_loop = strtol(argv[2],NULL,10);//string to long
   printf("running with spawns set to %lu and busy-loop-length set to %lu\n",spawns,busy_loop);
   for (i=0;i<spawns;i++){
          //pthread_create(&tid,NULL,child_thread,(void*)((unsigned)i));
          pthread_create(&tid,NULL,child_thread,NULL);
   }
   while(1){
        printf("please give me the priority level you would prefer\n");
        scanf("%d",&prio);
        //ret = setpriority(PRIO_PROCESS,0,prio);
        ret = setpriority(PRIO_PGRP,0,prio);
        printf("priority set returns %d\n",ret);
        ret = getpriority(PRIO_PROCESS,0);
        printf("new priority is %d\n",ret);
   }

}
```

Il quale compilato usando –lpthread dà il seguente output e mandato in esecuzione sulla CPU 0-esima :

```
taskset 0x1 ./a.out 2 1000000
```

Si ha che :

```
running with spawns set to 2 and busy-loop-length set to 1000000
please give me the priority level you would prefer
10
priority set returns 0
new priority is 10
```

Dove il valore 0 del return indica che si ha successo