Sistemi Operativi CANALE M-Z
Compito Scritto del 28 Giugno 2017

Cognome=_____, Nome=_____
Matricola=_____

Votazione massima 30/30
Soglia per superare la prova 15/30
Durata 3 ore

Domanda 1: max 5 punti

Si supponga di avere un Sistema Operativo che implementa una Multilevel Queue Scheduling (MQS). Si supponga che esistano due code diverse gestite rispettivamente con le politiche:

1) Coda 1: Scheduling Shortest Job First (SJF); Priorità BASSA assegnata alla coda 1
2) Coda 2: Scheduling RR con quanto di tempo q=5; Priorità ALTA assegnata alla coda 2

Le due code vengono gestite a priorità fissa, ossia prima si serve la coda con priorità alta e solo quando questa è vuota si serve la coda con priorità bassa. Si suppone che non vi sia Preemption tra le code, ossia nel caso in cui la CPU stia eseguendo un processo a bassa priorità e arriva un processo nella coda ad alta priorità, il processo corrente non viene tolto dalla CPU ma viene eseguito fino alla sua terminazione.

Si supponga che lo scheduler riceva i 6 job, A, B, C, D, E, F con le seguenti caratteristiche:

Processo	**Durata CPU−burst**	**Tempo di arrivo**	**Priorità**
A	15	0	BASSA
B	11	4	ALTA
C	5	3	ALTA
D	8	10	BASSA
E	7	16	ALTA
F	13	14	BASSA

Si descriva le sequenza di esecuzione dei processi tramite un **diagramma di Gantt**, si valuti il tempo di attesa di ogni processo, e il tempo medio di attesa dei processi.

Risposta:

Eventualmente si continui sul retro del foglio……..

Domanda 2: max 5 punti

Si realizzino due processi padre e figlio.
Il processo padre invia il segnale SIGUSR1 al processo figlio. Il processo figlio cattura il segnale SIGUSR1 tramite la funzione di Signal Handler. La funzione Signal Handler deve contenere del codice tale che essa reagisca al segnale SIGUSR1 inviando un segnale SIGUSR2 al processo padre.
Il processo padre riceve il segnale SIGUSR2 inviato dal figlio; tale segnale deve essere gestito da un'altra funzione di Signal Handler il cui codice può essere scritto a piacere. Appena ricevuto il segnale SIGUSR2, il processo padre deve terminare il figlio.

CONSEGNA: *Si scriva il programma in un file denominato SIGNAL_COGNOME_NOME.c*

Domanda 3: max 20 punti

Si voglia realizzare un sistema Client/Server con un Server e diversi Client. Si supponga di definire una struttura dati composta da:

```
typedef struct {
  float vettore1[MAX], vettore2[MAX]; //MAX è una costante numerica scelta a piacere
  char fine; //può assumere solo due valori: 's' per finire e 'n' per continuare.
} message;
```

Ogni Client invia un messaggio di tipo message, avendo riempito i due campi vettore1 e vettore 2 (vettori di float). Il Server riceve questo messaggio, e moltiplica ogni elemento di un vettore per il corrispondente elemento del secondo vettore. Il risultato viene messo nel primo vettore. Ad esempio se il Client invia i due vettori:

- vettore1=[1,2,3]
- vettore2=[4,5,6]

il vettore risultato sarà:

- vettore1=[4,10,18]

Il messaggio così ottenuto viene ritornato al Client che stampa a video il contenuto del vettore1, ossia del risultato.
Se il Client vuole terminare la connessione, allora, quando deve spedire l'ultimo messaggio al Server, inserisce il valore '**s**' nel campo fine (che normalmente deve essere messo a '**n**'). Se il server legge il valore '**s**' nel campo fine, allora anche lui termina la connessione con il client.

Si definisca un programma Server.c e un programma Client.c che realizzi quanto detto utilizzando i **Sockets** e i **threads**. Si deve supporre che per ogni client connesso, il server crei un thread apposito che gestisce la comunicazione con il client. Quando il client si disconnette, il thread finisce. **Per evitare che il Server debba invocare la pthread_join, si deve realizzare ciascun thread in modo detached. Questo è molto importante perché in questo modo le risorse del thread vengono rilasciate dal SO alla terminazione del thread.**

Il Server deve essere lanciato in modo background. Il Client in modo foreground. Si può supporre anche di aprire diversi terminali (applicativo Terminal) e da ogni terminale aperto si può lanciare il programma client sempre in foreground. In questo modo si possono eseguire tanti programmi client. Ricordarsi alla fine di killare il processo Server (kill %1).

CONSEGNA: Si scrivano due file denominati:
Server_COGNOME_NOME.c
Client_COGNOME_NOME.c