Progetto Sistemi Operativi con Laboratorio

'Visualizzazione di N numeri'
In C e in ambiente Unix/Linux

Prof. Arturo Carpi
Prof. Laboratorio Fabio Rossi

realizzato da

Matteo Volpi

341116

Dipartimento di Matematica e Informatica
Università degli Studi di Perugia

Panoramica

OBIETTIVO: Realizzazione di un'applicazione, in C ed in ambiente Unix/Linux, denominata Visualizzazione di N numeri.

BREVE DESCRIZIONE: L'applicazione deve richiedere all'utente il numero di Visualizzatori da creare ed un numero N, il Coordinatore deve avviare i Visualizzatori e comunicargli tramite 'IPC' quando prendere il controllo del flusso di lavoro; il Visualizzatore in attesa passiva, una volta chiamato in causa, deve lavorare sulla memoria condivisa senza che vengano generate inconsistenze e rimettersi in attesa una volta terminato.

OUTPUT ATTESO: Una corretta sincronizzazione/comunicazione tra processi, tramite 'IPC', che permetta di lavorare sulla memoria condivisa in maniera concorrente senza generare inconsistenze; due file testuali 'coord_pid.txt' e 'vis_pid.txt' che tengano traccia delle operazioni eseguite sulla memoria condivisa da parte del Coordinatore e dei Visualizzatori.

Indice

CHAPTER 1

Script di Lancio: Visualizzazione-N-numeri.sh

1.1 Descrizione di Visualizzazione-N-numeri.sh

Questo script di lancio automatizza il processo di compilazione ed esecuzione del programma "Visualizzazione di N numeri". Di seguito è riportato il contenuto dello script con una spiegazione dei suoi componenti principali:

Script di Lancio

```
#!/bin/bash

# Rimuovi i file se gia' esistono
if [ -f "coord_pid.txt" ]; then
    rm coord_pid.txt
fi

if [ -f "vis_pid.txt" ]; then
    rm vis_pid.txt
fi

# Compilazione del Coordinatore
gcc -o outCoordinatore Coordinatore.c -lrt -pthread
# Compilazione del Visualizzatore
gcc -o outVisualizzatore Visualizzatore.c -lrt -pthread

# Messaggio di richiesta inserimento dati
echo "Applicazione: Visualizzazione di N numeri"
echo "Inserisci il numero di visualizzatori (max 10):"
read num_visualizzatori
echo "Inserisci il valore massimo N:"
read N

# Controllo se compilazione OK
if [ $? -eq 0 ]; then
    echo "Compilazione OK!"

    # Esecuzione del programma
    ./outCoordinatore "$num_visualizzatori" "$N"
else
    echo "Errore durante la compilazione del programma."
```

```
fi
```

1.2 Descrizione delle operazioni:

- **Rimozione dei file esistenti**: Lo script inizia controllando l'esistenza dei file 'coord_pid.txt' e 'vis_pid.txt' e li rimuove se presenti.
- **Compilazione dei programmi**: Utilizza 'gcc' per compilare i file 'Coordinatore.c' e 'Visualizzatore.c', creando gli eseguibili 'outCoordinatore' e 'outVisualizzatore'.
- **Richiesta di input all'utente**: Vengono richiesti all'utente il numero di visualizzatori e il valore massimo 'N'.
- **Controllo della compilazione**: Verifica se la compilazione è andata a buon fine controllando il valore di ritorno '$?'.
- **Esecuzione del programma**: Se la compilazione è riuscita, esegue 'outCoordinatore' con i parametri forniti dall'utente.
- **Gestione degli errori**: Se la compilazione fallisce, viene visualizzato un messaggio di errore.

CHAPTER 2

Coordinatore.c

2.1 Coordinatore

Il **coordinatore** è il componente principale dell'applicazione "Visualizzazione di N numeri". Questo programma è responsabile della gestione e coordinazione dei processi visualizzatori, garantendo una corretta sincronizzazione e comunicazione con essi.

2.2 Memoria Condivisa e Semafori

Il coordinatore crea una regione di **memoria condivisa**, (`numero_shm`), inizializzata con un intero che rappresenta il numero corrente da elaborare. I processi visualizzatori accedono a questa memoria condivisa per ottenere il numero da visualizzare.

Per garantire l'accesso esclusivo alla memoria condivisa e sincronizzare l'accesso tra il coordinatore e i visualizzatori, vengono utilizzati due **semafori**:

1. Il **semaforo sulla memoria condivisa** (`semaforo_shm`) viene utilizzato per garantire l'accesso esclusivo alla memoria condivisa. Prima di accedere alla memoria condivisa, il coordinatore e i visualizzatori devono acquisire questo semaforo e rilasciarlo dopo aver completato le operazioni sulla memoria condivisa.

2. Il **semaforo di sincronizzazione** (`semaforo_sincro`) viene utilizzato per sincronizzare l'avvio dell'elaborazione tra il coordinatore e i processi visualizzatori. Il coordinatore attende su questo semaforo che tutti i visualizzatori siano pronti prima di iniziare a comunicare loro l'avvenuta estrazione.

2.3 Gestione dei Segnali

Il coordinatore gestisce diversi segnali per controllare il flusso di esecuzione e rispondere a eventi specifici:

1. L'**handler** per **SIGALRM** viene utilizzato per gestire la comunicazione tra il coordinatore e i visualizzatori. Il coordinatore invia **sigalrm** ai visualizzatori per avvisarli dell'avvenuta estrazione e che devono avviarsi.

2. Gli **handlers** per **SIGUSR1** e **SIGUSR2** vengono utilizzati per sospendere e riprendere l'attività di visualizzazione. Quando il coordinatore riceve il segnale SIGUSR1, sospende l'elaborazione dei numeri. Al contrario, quando riceve il segnale SIGUSR2, riprende l'elaborazione.

3. L'**handler** per **SIGTERM** viene utilizzato per gestire la terminazione dell'applicazione in modo pulito. Quando il coordinatore riceve questo segnale, termina tutti i processi visualizzatori e rilascia tutte le risorse allocate.

4. L'**handler** per **SIGINT** ignora tale segnale.

Il flag `SA_RESTART` è stato utilizzato nell'impostazione degli handler dei segnali per garantire un comportamento coerente e prevedibile del programma in presenza di segnali. Quando un handler è configurato con questo flag e una chiamata di sistema viene interrotta da un segnale, il sistema operativo si occupa automaticamente di ripetere la chiamata di sistema interrotta anziché restituire `EINTR` al processo.

CHAPTER 3

Visualizzatore.c

3.1 Visualizzatore

Il visualizzatore è un processo che ha come scopo principale quello di leggere i dati dalla memoria condivisa, aumentare l'intero presente nella memoria condivisa e comunicare tramite (`semaforo_shm`) la fine delle operazioni.

Il codice del visualizzatore è composto principalmente da tre sezioni:

- **Inizializzazione e Collegamento**: Il visualizzatore inizia collegandosi alla memoria condivisa e ai semafori necessari per la sincronizzazione con il coordinatore. Utilizza le funzioni di sistema `shm_open()`, `mmap()`, `sem_open()` per stabilire la connessione con la memoria condivisa e i semafori.

- **Gestione dei Segnali**: Il visualizzatore gestisce tre segnali principali: `SIGALRM`, `SIGTERM`, e `SIGINT`. L'hanlder per `SIGALRM` viene utilizzato per eseguire l'elaborazione sulla memoria condivisa. L'handler per `SIGTERM` viene utilizzato per terminare il visualizzatore in modo pulito, mentre l'handler per `SIGINT` ignora tale segnale.

- **Esecuzione Principale**: Una volta ricevuto il segnale `SIGALRM`, il visualizzatore entra nella sezione critica in cui legge i dati dalla memoria condivisa, aggiorna il numero e scrive i risultati sul file di output 'vis_pid.txt'. Successivamente, invia `SIGALRM` al coordinatore che lo avvisa di aver terminato le operazioni.

CHAPTER 4

Script Invio Segnali: InvioSegnali.sh

4.1 Invio segnali

Questo script Bash è progettato per inviare i segnali al processo `outCoordinatore`, che consentono all'utente di controllarne il comportamento durante l'esecuzione.

Il coordinatore può essere influenzato da quattro segnali distinti:

1. **SIGUSR1**: Questo segnale provoca la sospensione del coordinatore, sospendendo temporaneamente le operazioni di visualizzazione dei dati.
2. **SIGUSR2**: Questo segnale riattiva il coordinatore dalla sospensione, consentendo la ripresa delle operazioni di visualizzazione.
3. **SIGINT**: Questo segnale viene ignorato dal coordinatore.
4. **SIGTERM**: Questo segnale provoca la terminazione del coordinatore, terminando l'applicazione in modo sicuro e pulito.

L'utente interagisce con lo script attraverso un loop che consente di inserire il numero corrispondente al segnale desiderato. Una volta selezionato il segnale, lo script lo invia al coordinatore utilizzando il comando `kill`, fornendo il PID del coordinatore e il numero del segnale come argomenti.

Script di Invio Segnali

```
#!/bin/bash

# Funzione per inviare segnali
invio_segnale() {
    local COORD_PID=$1
    case $2 in
        1)
            kill -SIGUSR1 $COORD_PID
            echo "Inviato SIGUSR1 al processo $COORD_PID (sospendi)"
            ;;
        2)
            kill -SIGUSR2 $COORD_PID
            echo "Inviato SIGUSR2 al processo $COORD_PID (riprendi)"
            ;;
        3)
            kill -SIGINT $COORD_PID
```

```
            echo "Inviato SIGINT al processo $COORD_PID (ignora)"
            ;;
        4)
            kill -SIGTERM $COORD_PID
            echo "Inviato SIGTERM al processo $COORD_PID (termina)"
            exit 0
            ;;

        *)
            echo "Scelta non valida. Inserisci 1, 2, 3 o 4."
            ;;
    esac

# Trova il PID del processo outCoordinatore
COORD_PID=$(pgrep -f './outCoordinatore')
if [ -z "$COORD_PID" ]; then
    echo "Errore: processo ./outCoordinatore non trovato."
    exit 1
fi

echo "Trovato processo ./outCoordinatore con PID $COORD_PID."

# Loop per inserimento da tastiera
while true; do
    echo "Inserisci il numero del segnale da inviare:"
    echo "1 — Sospendi (SIGUSR1)"
    echo "2 — Riprendi (SIGUSR2)"
    echo "3 — Ignora (SIGINT)"
    echo "4 — Termina (SIGTERM)"
    read -r scelta

    invio_segnale "$COORD_PID" "$scelta"
done
}
```