Sistemi di Calcolo - Modulo 2 (A.A. 2017-2018)

Esercitazione Finale - 25 Maggio 2018

Esercizio 1 - Comunicazione unidirezionale padre/figli via pipe con sincronizzazione
Il processo padre crea `CHILDREN_COUNT` processi figlio e condivide con loro una pipe unica dalla quale lui legge e nella quale i figli scrivono. Affinchè i figli scrivano nella pipe in mutua esclusione, viene impiegato un semaforo named binario. All'inizio, il padre deve assicurare che il semaforo named non esista, successivamente apre il semaforo binario creandolo e passa il puntatore al semaforo appena aperto ai figli. Una volta lanciati i figli il padre chiude il semaforo. Prima di uscire, il padre deve rimuovere il semaforo named per pulizia.
Una volta aperto il semaforo named, il figlio `i`-esimo deve scrivere sulla pipe `MSG_COUNT` messaggi, dove ogni messaggio è un array di interi di dimensione `MSG_SIZE` avente `i` come valore di ogni elemento. Una volta creati i figli, il padre deve leggere dalla pipe `CHILDREN_COUNT*MSG_COUNT` messaggi e verificarne l'integrità. Infine, il padre deve attendere il termine dei figli.

Obiettivi
1. Gestione processi figlio: creazione/attesa terminazione processi figlio
2. Semafori named: creazione, apertura, chiusura, rimozione, uso per mutua esclusione
3. Comunicazione su pipe: invio e ricezione dati di lunghezza fissa

Esercizio 2 - Ricerca multi-thread su array
Un processo crea `THREADS` thread che devono processare il contenuto di un array `array` in parallelo. Ogni thread, caratterizzato da un indice `i`, processa una porzione dell'array e salva il risultato del processamento nella cella `i`-esima di un secondo array `counters` composto da `THREADS` celle.
L'array da processare viene diviso in segmenti di `STEP` elementi. Il primo thread processa dalla cella 0 alla cella `STEP` (esclusa), il secondo inizia dalla cella `STEP` e così via. A riguardo si presti particolare attenzione al blocco di commento inserito nel codice.
Il main thread attende la terminazione degli altri thread, poi aggrega i risultati.

Obiettivi principali
1. Gestione multi-thread: creazione/attesa terminazione thread
2. Gestione degli argomenti dei thread `thread_args_t`

Esercizio 3 - Realizzazione di un server multi-thread con comunicazione su socket.

Un server espone N-1 risorse identificate da un numero compreso tra 1 e N-1. I client possono connettersi al server tramite socket e richiedere l'utilizzo di una risorsa inviando un numero tra 1 e N-1. Il server mantiene un array shared_counters con N contatori, dove il contatore i-esimo tiene traccia del numero di richieste ricevute per la risorsa i-esima.

Ogni client viene gestito con un thread (su cui non è previsto fare join) che esegue la funzione connection_handler(). Questo thread riceve comandi dal client ed invoca la funzione process_resource(), che incrementa il contatore di shared_counters relativo alla risorsa richiesta. L'accesso al contatore i-esimo di shared_counters è protetto dal semaforo i-esimo in semaphores. Es: se il client invia al server "1", viene incrementato il contatore in posizione 1. Qualsiasi altro comando viene considerato come "0", ed in tal caso è il contatore in posizione 0 ad essere incrementato. L'unica eccezione è il comando "QUIT", che fa terminare il thread.

Obiettivi:

1. Gestione semafori
    - Inizializzazione semafori
    - Gestione sezione critica nella funzione process_resource()
2. Gestione multi-thread
    - Creazione thread di gestione connessione client
    - Rilascio risorse allocate
3. Gestione scambio messaggi su socket
    - Invio/ricezione messaggi su socket con gestione interruzioni
    - Chiusura descrittori