Sistemi di Calcolo 2 (A.A. 2019-2020)

Esercizi preparatori in vista degli appelli

Nota: i file common.h possono contenere vecchie macro per la gestione degli errori, utilizzati nelle tracce e nelle soluzioni. Si prega di ignorarli, *non saranno disponibili negli appelli.*
Nello sviluppo delle vostre sezioni fate uso solamente di handle_error() (quando la funzione setta *errno*) o handle_error_en().

Esercizio 1 - Realizzazione di un processo multi-thread con paradigma prod/cons
Un processo lancia un numero pari THREAD_COUNT di thread, dei quali metà (THREAD_COUNT / 2) sono produttori e metà sono consumatori. Ogni thread ha un identificativo (*idx*, che varia tra 0 e THREAD_COUNT-1) ed un ruolo (*role*, che può valere PROD_ROLE oppure CONS_ROLE). Una volta lanciati, il processo si mette in attesa della loro terminazione. Inoltre, il processo deve rilasciare le risorse che non usa più.

Ogni thread produttore esegue ITERATION_COUNT iterazioni del proprio ciclo, così come ogni thread consumatore esegue ITERATION_COUNT iterazioni. I thread produttori scrivono nel buffer ad ogni iterazione *i* un valore prodotto univoco. I thread consumatori invece ad ogni iterazione leggono un valore dal buffer.

All'interno delle funzioni *producer_routine* e *consumer_routine* deve essere completata l'implementazione rispettivamente della scrittura su e della lettura da un buffer circolare, gestito secondo la semantica *più produttori / più consumatori*. Al termine delle iterazioni, ogni thread deve rilasciare le risorse che non usa più.

Obiettivi
1. Implementazione della semantica più produttori/più consumatori
    - Dichiarazione/Inizializzazione/Rilascio dei semafori necessari
    - Uso dei semafori nelle funzioni *producer_routine* e *consumer_routine*
2. Gestione multi-thread: creazione thread; rilascio risorse allocate

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 - 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 4 - Realizzazione di comunicazione bidirezionale via pipe tra due processi

Due processi padre-figlio condividono due pipe per realizzare un canale di comunicazione bidirezionale. Il figlio genera N numeri interi casuali e li invia al padre come stringhe di lunghezza variabile, usando '\n' come delimitatore di fine messaggio. Il padre stampa a video ogni messaggio ricevuto, raddoppia il valore numerico ricevuto e lo invia al figlio, sempre come stringa di lunghezza variabile con '\n' come delimitatore di fine messaggio. Il figlio stampa a video ogni messaggio ricevuto. Dopo aver gestito tutti i messaggi, i processi terminano, con il padre che attende la terminazione del figlio.
Obiettivi

1. Gestione processi figlio
    a. Creazione/Attesa terminazione processo figlio
2. Comunicazione via pipe
    a. Creazione pipe e chiusura descrittori
    b. Lettura messaggi di lunghezza variabile (con condizione di terminazione)
    c. Gestione degli errori di invio e ricezione messaggi, compreso il caso di messaggio troppo lungo
    d. Invio messaggi