Sistemi di Calcolo - Modulo 2 (A.A. 2015-2016)

Esonero - 27 Maggio 2016 - Turno 2 - Traccia 3

Tempo a disposizione: 1h 30'.

> *Attenzione*: assicurarsi di compilare il file **studente.txt** e che il codice prodotto non contenga **errori di compilazione**, pena una valutazione negativa dell'elaborato.

Esercizio 1 - Realizzazione di processi padre-figlio che comunicano via doppia pipe
Un processo esegue una fork(). Il processo figlio e quello padre comunicano tramite una coppia di pipe. In particolare, il processo *figlio* genera un numero pseudo-random e lo comunica (sotto forma testuale) al padre. Il processo *padre* per ciascun messaggio ricevuto, estrae il numero x contenuto e invia al figlio un messaggio contenente (sotto forma testuale) il numero y ottenuto come 2*x. I messaggi sono di di lunghezza variabile e contraddistinti dal delimitatore di fine messaggio '\n'. Entrambi i processi terminano dopo che N numeri generati dal figlio sono stati gestiti. Il padre attende esplicitamente la terminazione del figlio.

Obiettivi:

1. Gestione processi figlio
    - Creazione/Attesa terminazione processo figlio
2. Comunicazione via (coppa di) pipe
    - Creazione pipe e chiusura descrittori
    - Lettura dati di lunghezza variabile (con condizione di terminazione)
    - Invio dati

Esercizio 2 - Realizzazione di un processo multi-thread con paradigma prod/cons
Un processo lancia THREAD_COUNT thread, dei quali tutti sono produttori tranne uno, che è un consumatore. 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. Infine, il processo deve rilasciare le risorse che non usa più.
Ogni thread esegue ITERATION_COUNT iterazioni.I thread produttori invocano la funzione *enqueue()* ad ogni iterazione *i*, con un valore pari a *idx * i*. Il thread consumatore invoca la funzione *dequeue()* ad ogni iterazione. Le funzioni *enqueue()* e *dequeue()* implementano rispettivamente la scrittura su e la lettura da un buffer circolare, gestito secondo la semantica *più produttori/singolo consumatore*. Al termine delle iterazioni, ogni thread deve rilasciare le risorse che non usa più.

Obiettivi:

1. Implementazione della semantica più produttori/singolo consumatore
    - Dichiarazione e Inizializzazione dei semafori necessari
    - Uso dei semafori nelle funzioni enqueue() e dequeue()
2. Gestione multi-thread
    - Creazione thread
    - Rilascio risorse allocate

Altro

- **i commenti nel codice contengono molte informazioni utili per lo svolgimento della prova, si consiglia quindi di tenerli in debita considerazione**
- in caso di necessità, nella cartella `backup/` è presente una copia della traccia
- il file `dispensa.pdf` contiene una copia della dispensa *Primitive C per UNIX System Programming* preparata dai tutor di questo corso
- il file `raccomandazioni.pdf` contiene una serie di considerazioni sugli errori riscontrati più di frequente

---

Regole Esame

- Domande ammesse
  Le domande possono riguardare solo la specifica dell'esame e la struttura di alto livello del codice, nessuna domanda può riguardare singole istruzioni.
- Oggetti vietati
  I seguenti oggetti non devono essere presenti sulla scrivania, né tantomeno usati: smartphone, smartwatch, telefonini, tablet, portatili, dispositivi di archiviazione USB, copie cartacee della dispensa, astucci e qualsiasi forma di libri ed appunti. **Chi verrà sorpreso ad usare uno di questi oggetti verrà automaticamente espulso dall'esame.**
- Azioni vietate
  È assolutamente vietato comunicare in qualsiasi modo con gli altri studenti. **Chi verrà sorpreso a comunicare con gli altri studenti per la prima volta verrà richiamato, la seconda volta verrà invece automaticamente espulso dall'esame**.