Elaborato 2: Processi e IPC.

Consegna entro: 30/6/2016.

Testo dell’elaborato

Si vuole realizzare un programma in C che utilizzi le system call (IPC), ove possibile, per implementare lo scheletro di un simulatore di calcolo parallelo. Il progetto deve essere commentato in formato Doxygen, e corredato da un file di configurazione per Doxygen, e da uno script Makefile per la compilazione. Inoltre si devono allegare al progetto anche eventuali file di supporto.

Il programma dovrà leggere un file di configurazione, contenente:

1. Il numero di processi di calcolo parallelo.
2. I dati di computazione.

Il file di configurazione avra' la seguente struttura:

- La prima riga conterra' un intero, che corrisponde al numero di processi paralleli da creare, *NPROC*.
- Le altre righe avranno il seguente formato: <*id*> <*num1*> <*op*> <*num2*>, dove:
    1. *id* e' il numero di processo parallelo a cui far eseguire l'operazione, se e' maggiore o uguale a 1. Se e' zero, indica che l'operazione dovra' essere eseguita dal primo processo libero. Si assuma che *id* corrisponda sempre ad un valore corretto.
    2. *num1* e *num2* sono due numeri interi, corrispondenti ai dato sui quali eseguire l'operazione.
    3. *op* e' un carattere rappresentante l'operazione da eseguire sui due numeri. Caratteri validi per op sono: '+', '-', '*', '/', rappresentanti rispettivamente la somma, la sottrazione, la moltiplicazione e la divisione.

Ad esempio, un file di configurazione potrebbe essere:

```
5
3 2/9
3 234 + 653
4 3444 + 153
0 34 * 1873
3 0 - 21534
```

Ogni *processo* eseguirà una routine di questo tipo:

1. Attesa su un semaforo per eseguire un calcolo.
2. Ricezione dei dati e dell'operazione da eseguire, da parte del processo padre.

    1. Se riceve il comando di terminazione, termina la sua esecuzione. Il comando di terminazione e' indicato dal carattere 'K'.
3. Esecuzione del calcolo.
4. Invio del risultato al padre, e segnalazione al padre che ha finito di calcolare.
5. Attesa bloccante sullo stesso semaforo al punto 1, per attendere il comando successivo.

Il processo padre comunichera' con i processi figlio tramite memoria condivisa.

Il processo padre eseguira' in questo modo:

1. Setup della simulazione, leggendo dal file di configurazione il numero di processori da simulare, creandone i processi relativi, e creando ed inizializzando le eventuali strutture di supporto quali semafori e memoria condivisa. Inoltre, verra' creato un array in cui memorizzare i risultati delle operazioni eseguite. La lunghezza di tale array sara' ricavata dal numero di operazioni lette dal file di configurazione.
2. Entrata in un ciclo che per ogni operazione da simulare effettua quanto segue:
    1. Se il comando ha *id* diverso da zero, attende che il processo numero *id* sia libero, salva il risultato dell'eventuale calcolo precedente nell'array dei risultati, e poi interagisce con lui passandogli il comando da simulare. Il processo padre *non deve attendere* che il processore abbia completato la simulazione dell'operazione passata al figlio.
    2. Se l'istruzione ha *id* 0, trova il primo processore libero ed in caso interagisce con esso, come al punto 1. Altrimenti attende che almeno un processore sia libero, e poi interagisce con esso come al punto 1.
3. Passati tutti i comandi ai figli, attende che i processi figlio abbiano eseguito tutti i loro calcoli.
4. Salva nell'array dei risultati gli ultimi risultati computati, e fa terminare i processi figlio passando il comando di terminazione 'K'.
5. Attende che tutti i figli abbiano terminato.
6. Stampa su un file di output tutti i risultati.
7. Libera eventuali risorse.
8. Esce.

Si aggiungano ai processi delle stampe a video per poter seguirne l'esecuzione.

Per ogni chiamata ad una system call, si deve controllare che tale funzione abbia successo.

I dettagli su come utilizzare la memoria condivisa o quanta allocarne, o quanti semafori, sono lasciati liberi.

Quando i processi devono attendere, non devono fare attese attive: si devono bloccare.

Ove possibile, si devono usare le system call al posto delle equivalenti chiamate a funzioni di libreria.

Tutte le stampe a video, le letture e le scritture su file devono avvenire tramite system call (quindi ad esempio non si possono utilizzare *printf*, *fprintf*, *scanf*, *fscanf*, *perror*).

FAQ

1. *E’ possibile inserire il codice su file separati?*

   Questa decisione e' lasciata allo studente, che puo' scegliere il modo piu' opportuno. L'importante e' che il codice sia ben strutturato e leggibile.

2. *Si possono usare funzioni quali la printf, fscanf, etc.?*

   No. L'elaborato richiede di usare le system call ove possibile, e tali funzioni sono rimpiazzabili tramite le system call *open*, *write*, etc. In generale, si cerchi di utilizzare il piu' possibile le system call. Invece, si possono usare funzioni quali la *sprintf*, perche' non ha una system call equivalente.

3. *Alcune cose non sono ben specificate nell'elaborato. Cosa faccio?*

   Alcune cose non sono specificate apposta per lasciare liberta' agli studenti di implementarle come preferiscono. In caso di dubbi, si possono comunque contattare i docenti per eventuali chiarimenti.

4. *Quale dei due stili presentati a lezione bisogna usare per il Makefile?*

   Lo studente puo' usare quello che preferisce.

N.B.: Tutto quanto non esplicitato in questo documento può essere implementato liberamente.