Progetto Farm Lab II - CorsoA – a.a. 21/22

Progetto riservato agli studenti che hanno superato le prove in itinere

Si chiede di realizzare un programma C, denominato ***farm***, che implementa lo schema di comunicazione tra processi e thread mostrato in Figura 1.

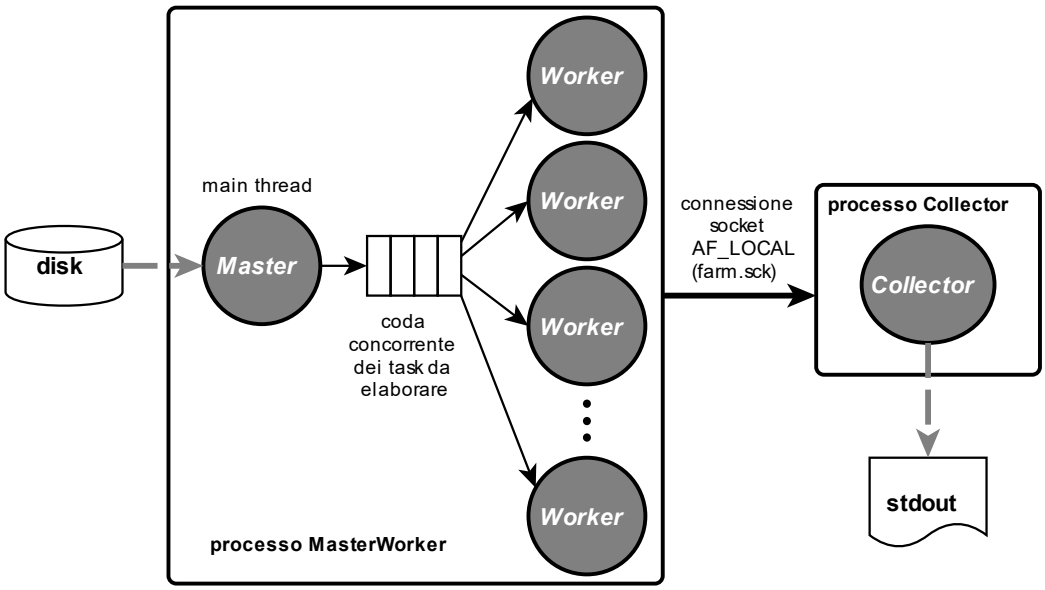

Figura 1 Architettura logica di connessione tra i processi *MasterWorker* e *Collector*

farm è un programma composto da due processi, il primo denominato *MasterWorker* ed il secondo denominato *Collector*. *MasterWorker*, è un processo multi-threaded composto da un thread *Master* e da '*n*' thread *Worker* (il numero di thread Worker può essere variato utilizzando l'argomento opzionale '-n' – vedere nel seguito). Il programma prende come argomenti una lista di file binari contenenti numeri interi lunghi (long) ed un certo numero di argomenti opzionali (opzioni '-n', '-q' e '-t'). Il processo *Collector* viene generato dal processo *MasterWorker*. I due processi comunicano attraverso una connessione socket AF_LOCAL (AF_UNIX). Viene lasciata allo studente la scelta di quale tra i due processi fa da processo master per la connessione socket, così come la scelta se usare una sola connessione o più connessioni socket. Il socket file "***farm.sck***", associato alla connessione AF_LOCAL, deve essere creato all'interno della directory del progetto e deve essere cancellato alla terminazione del programma.

Il processo *MasterWorker* legge gli argomenti passati alla funzione *main* uno alla volta, verificando che siano file regolari, e passa il nome di ogni file (con eventuali altri parametri) ad uno dei thread *Worker* tramite una coda concorrente condivisa (denominata "coda concorrente dei task" in Figura 1). Il generico thread *Worker* si occupa di leggere il contenuto del file ricevuto in input e di fare un calcolo sugli elementi in esso contenuti e quindi di inviare il risultato ottenuto, unitamente al nome del file di input, al processo *Collector* tramite la connessione socket precedentemente stabilita.

Il processo *Collector* attende di ricevere i risultati dai *Worker* e stampa i valori ottenuti sullo standard output nel formato:

risultato nomefile

Il calcolo effettuato dal *Worker* per ogni file ricevuto in ingresso tramite il segmento condiviso è il seguente:

$$result = \sum_{i=0}^{N-1} (i * file[i])$$

dove N è il numero di interi lunghi contenuti nel file e *result* è l'intero lungo che dovrà essere inviato al *Collector*. Ad esempio, supponendo che il file *fileX.dat* passato in input come argomento del *main* abbia dimensione 24 bytes, con il seguente contenuto (si ricorda che i *long* sono codificati con 8 bytes in sistemi Linux a 64bit):

3
2
4

il risultato calcolato dal *Worker* sarà: N=3, $result = \sum_{i=0}^{3-1}(i * file[i]) = (0 * 3 + 1 * 2 + 2 * 4) = 10$.

Gli argomenti che opzionalmente possono essere passati al processo *MasterWorker* sono i seguenti:

- -n <nthread> specifica il numero di thread *Worker* del processo *MasterWorker* (valore di default 4)
- -q <qlen> specifica la lunghezza della coda concorrente tra il thread *Master* ed i thread *Worker* (valore di default 8)
- -t <delay> specifica un tempo in millisecondi che intercorre tra l'invio di due richieste successive ai thread *Worker* da parte del thread *Master* (valore di default 0)

Il processo *MasterWorker* deve gestire i segnali SIGHUP, SIGINT, SIGQUIT, SIGTERM. Alla ricezione di uno di questi segnali il processo deve completare i task eventualmente presenti nella coda dei task da elaborare e quindi terminare dopo aver atteso la terminazione del processo Collector ed effettuato la cancellazione socket file. Il processo Collector maschera i segnali gestiti dal processo *MasterWorker*. Il segnate SIGPIPE deve essere gestito opportunamente dai due processi.

Note

La dimensione dei file in input non è limitata ad un valore specifico. Si supponga che la lunghezza del nome dei file sia non superiore a 255 caratteri.

Materiale fornito per il progetto

Il materiale fornito dai docenti è il seguente:

- Testo del progetto (file *progettoFarm.pdf*)
- Un programma *generafile.c* per generare i file per i tests
- Uno script Bash (test.sh) contenente alcuni semplici test che il programma deve superare.

Consegna del compito

Il compito viene assegnato via il portale Classrom del corso. La consegna deve avvenire su Classroom. Tutti i file che implementano il compito (insieme ai file forniti dai docenti) devono essere consegnati in un unico file zip (o tgz) avente il seguente nome:

NomeCognome-Matricola.zip (o *NomeCognome-Matricola.tgz*).

I docenti verificheranno la funzionalità del compito consegnato compilando ed eseguendo il programma sulla macchina virtuale del corso (laboratorio2.di.unipi.it) ed eseguendo lo script Bash test.sh.

Lo studente dovrà implementare il codice del programma *farm.c*, ed il *Makefile* per la sua compilazione. Inoltre dovrà essere fornita una breve relazione (massimo 2 pagine) in formato PDF che descrive le principali scelte implementative.

Esempi di possibili esecuzioni

> ./farm -n 4 -q 4 file1.dat file2.dat file3.dat file4.dat file5.dat file6.dat file7.dat

```
103453975   file2.dat
153259244   file1.dat
293718900   file3.dat
380867448   file5.dat
584164283   file4.dat
```

> valgrind --leak-check=full ./farm -n 8 -q 4 -t 200 file* (dopo circa 1 secondo viene inviato **SIGINT** al processo MasterWorker)

```
==37245== Memcheck, a memory error detector
==37245== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==37245== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==37245== Command: ./farm -n 8 -q 4 -t 200 file1.dat file2.dat file3.dat file4.dat file5.dat file6.dat file7.dat
==37245==
     64834211   file100.dat
   1146505381   file10.dat
   1884778221   file111.dat
    258119464   file116.dat
^C  380867448   file5.dat
==37246==
==37246== HEAP SUMMARY:
==37246==     in use at exit: 0 bytes in 0 blocks
==37246==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==37246==
==37246== All heap blocks were freed -- no leaks are possible
==37246==
==37246== For lists of detected and suppressed errors, rerun with: -s
==37246== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==37245==
==37245== HEAP SUMMARY:
==37245==     in use at exit: 0 bytes in 0 blocks
==37245==   total heap usage: 18 allocs, 18 frees, 2,888 bytes allocated
==37245==
==37245== All heap blocks were freed -- no leaks are possible
==37245==
==37245== For lists of detected and suppressed errors, rerun with: -s
==37245== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```