Sistemi di Calcolo - Modulo 2 (A.A. 2014-2015)

Terzo appello - 15 settembre 2015

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.

Realizzazione di un'applicazione server per il processamento di immagini

L'obiettivo di questa prova è completare il codice del modulo server di un'applicazione per il processamento remoto di immagini. Il modulo client, fornito sotto forma di eseguibile pre-compilato, invia una immagine in formato PGM al server; quest'ultimo processa l'immagine applicando un filtro e la invia al client, che la memorizza in un nuovo file.

Il server processa le richieste in ingresso in modo seriale, ossia una per volta; tuttavia, per velocizzare il processamento di immagini di medio-grandi dimensioni, esso utilizza un pool di *working threads* inizializzati al momento dell'avvio del server e che di volta in volta vengono istruiti per processare una nuova immagine.

La sincronizzazione del main thread con i working threads avviene attraverso un array di semafori, uno per thread: in particolare, per ciascuna immagine da processare un working thread - caratterizzato da un indice *i* - si mette in attesa sul semaforo binario *i*-esimo, elabora la porzione di immagine di input che gli compete e segnala l'avvenuto completamento al main thread incrementando il semaforo *i*-esimo.

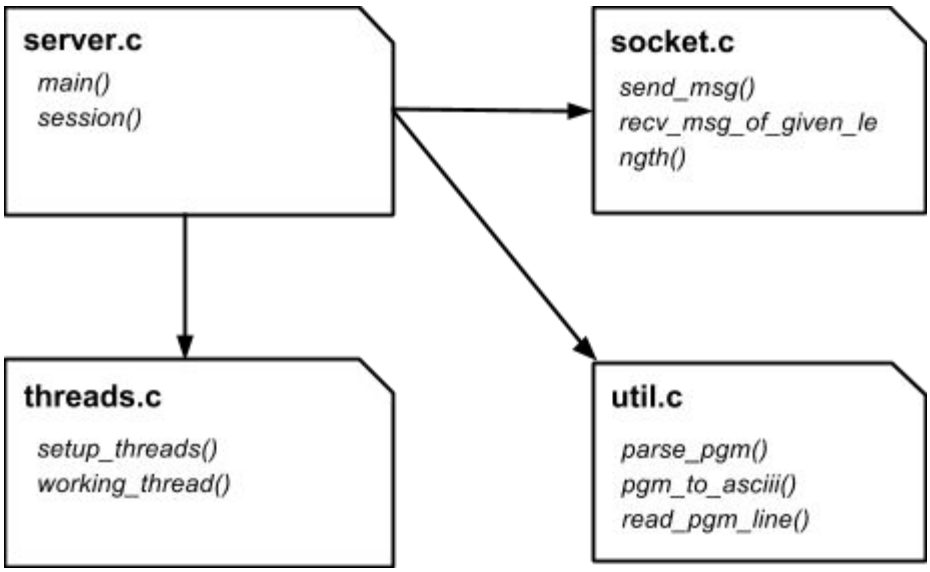
La comunicazione tra client e server è regolata da un semplice protocollo in cui il client invia un primo messaggio (di lunghezza fissa) contenente la dimensione dell'immagine, a cui segue l'invio dell'immagine stessa. Il server, una volta ricevuto il primo messaggio, alloca un buffer della dimensione specificata e riceve l'immagine all'interno del buffer stesso. Lo schema si ripete in modo speculare per l'invio al client della dimensione - tipicamente differente da quella dell'immagine in input - e quindi del contenuto dell'immagine elaborata dal server.

Descrizione del codice del server

Per ottimizzare la stesura e la leggibilità del codice, la componente server è organizzata in modo modulare, con due file di header `.h` e quattro file contenenti i sorgenti `.c` veri e propri:

common.h	Definizione delle costanti e delle macro in uso nei sorgenti `.c`
methods.h	Contiene i prototipi dei metodi definiti in ciascun modulo
server.c	Modulo principale per le gestione delle richieste in ingresso
socket.c	Funzioni per lettura e scrittura su socket

threads.c	Funzioni per creazione e gestione dei working threads
util.c	Metodi ausiliari per la manipolazione delle immagini PGM
Makefile	Compilazione automatizzata del codice



Architettura del codice sorgente. Una freccia indica che il modulo di origine utilizza uno o più metodi definiti nel modulo di destinazione.

Obiettivi

Vi viene richiesto di completare porzioni di codice - contrassegnate anche da blocchi di commenti nei rispettivi file - nei sorgenti C per realizzare le seguenti operazioni:

(1) All'interno della funzione `setup_threads()` del modulo `threads.c`, si richiede di:

A. creare `NUM_THREADS` semafori anonimi con valore iniziale `0`, utilizzando in particolare per il semaforo *i*-esimo la cella *i*-esima dell'array definito tramite variabile globale `sem_t semaphores[NUM_THREADS]`
B. creare `NUM_THREADS` thread che invochino il metodo `working_thread()` definito nello stesso file, tenendo presente che:
   - il metodo in questione prende in input un puntatore ad una struttura dati di tipo `thread_arg_t` contenente l'indice di creazione del thread
   - gli indici di dei thread sono definiti in `{0, 1, …, NUM_THREADS-1}`
   - il programma non prevede operazioni di sincronizzazione sui thread creati
   - per la creazione degli oggetti `pthread_t` va utilizzato l'array definito tramite variabile globale `pthread_t threads[NUM_THREADS]`

(2) All'interno della funzione `working_thread()` del modulo `threads.c`, si richiede di controllare l'accesso alla sezione di codice rappresentata dall'intero corpo del `while`; in particolare, prima di entrare nella sezione il thread *i*-esimo si deve sincronizzare con il main thread mettendosi in attesa sul semaforo *i*-esimo, per poi incrementarlo a fine iterazione.

(3) All'interno della funzione `session()` del modulo `server.c`, si modifichi la sezione evidenziata nei blocchi di commento per implementare la sincronizzazione con i working threads. In particolare, il main thread notifica a ciascun working thread la presenza di una nuova immagine da processare incrementando il semaforo associato al working thread. Una volta completato il ciclo di notifica, il main thread si pone in attesa che ciascun working thread completi il lavoro ed incrementi il semaforo ad esso associato come dal punto (2).

(4) Si implementi il corpo delle funzioni presenti nel modulo `socket.c` per la comunicazione tramite socket tra client e server, in particolare:

A. Per la funzione `void send_msg(int sock_fd, const char *msg, int msg_len)` si richiede di inviare sul descrittore `sock_fd` il contenuto del messaggio `msg`, di lunghezza pari a `msg_len` bytes. L'implementazione deve assicurarsi che in assenza di errori tutti i byte siano stati scritti. Si gestiscano eventuali interruzioni.
B. Per la funzione `int recv_msg_of_given_length(int sock_fd, char* msg, int msg_len)` si richiede invece di ricevere dal descrittore `sock_fd` verso il buffer `msg` un numero di bytes pari a `msg_len`. L'implementazione deve assicurarsi che in assenza di errori tutti i byte siano stati letti. Se il client ha chiuso la connessione, il metodo deve restituire `0`, altrimenti il numero di bytes letti. Si gestiscano inoltre eventuali interruzioni.

Testing

La compilazione è incrementale (ossia avviene un modulo per volta) ed automatizzata tramite `Makefile`, pertanto in assenza di errori di compilazione un binario `server` verrà generato quando si esegue `make`. Per lanciare il modulo server, digitare da terminale:

```
./server <port_number>
```

Dove `<port_number>` è un numero di porta valido per ospitare un servizio (es. `1025`). Con il server in esecuzione, è possibile lanciare da un altro terminale il modulo client con:

```
./client <port_number> <input_file> <output_file>
```

Dove per `<port_number>` va specificato lo stesso numero di porta in uso al server, mentre `<input_file>` è una immagine PGM esistente ed `<output_file>` è il file di destinazione per l'immagine prodotta dal server (se già presente, quest'ultimo verrà sovrascritto).

Per una verifica preliminare (*nonché parziale!*) del codice implementato vi vengono fornite una immagine di input denominata `colosseo.pgm` e la corrispondente `soluzione.pgm` che ci si attende venga generata dal server. Supponendo che il server sia in funzione sulla porta `1025`, è possibile processare l'immagine di input con:

```
./client 1025 colosseo.pgm output.pgm
```

E quindi confrontare il file prodotto `output.pgm` con quello fornito come riferimento:

```
cmp soluzione.pgm output.pgm
```

Nel caso in cui i file siano identici, il comando non mostrerà alcun messaggio, altrimenti verrà riportato il primo byte in cui i due file differiscono. In caso di errori è inoltre possibile confrontare i due file anche visivamente, aprendoli dal file manager dell'interfaccia grafica.

Altro

- in caso di necessità, nella cartella `backup/` è presente una copia di della traccia

- il file `dispensa.pdf` contiene una copia della dispensa *Primitive C per UNIX System Programming* preparata dai tutor di questo corso

Raccomandazioni

Seguono alcune considerazioni sugli errori riscontrati più di frequente tra gli elaborati dei partecipanti, nonché delle raccomandazioni per un buon esito delle prove al calcolatore:

- dato un buffer `buf`, `sizeof(buf)` non restituirà il numero di byte presenti (i.e., scritti) attualmente nel buffer, bensì il numero di byte occupati da un puntatore (4 su IA32, 8 su x86_64) se allocato dinamicamente, o il numero di byte per esso allocati staticamente; si noti che nel caso in cui `buf` contenga una stringa NULL-terminated, è possibile utilizzare `strlen(buf)` per conoscerne l'effettiva lunghezza
- quando si deve invocare una funzione per la programmazione di sistema:
    - fare attenzione se i singoli parametri siano o meno dei puntatori
    - verificare se può essere soggetta ad interruzioni, e in tal caso gestirle
    - verificare il comportamento della funzione in caso di errori: inserire codice idoneo a trattare errori gestibili, utilizzare la macro opportuna per gli altri
    - per funzioni della libreria pthread, si vedano il capitolo 2 e l'appendice A.2 della dispensa di UNIX system programming
- quando viene acquisita una risorsa (socket aperta, area di memoria allocata, etc.), questa deve poi essere rilasciata in maniera opportuna
- utilizzare puntatori non inizializzati porta ragionevolmente ad un Segmentation fault!
- i cast di puntatori da e verso `void*` sono impliciti in C, renderli espliciti o meno è una scelta di natura puramente stilistica nella programmazione
- fare molta attenzione nell'aggiornamento dell'indice (o del puntatore) con cui si opera su un buffer: disallineamenti ±1 sono tra le cause più comuni di errori
- quando si vuole passare un puntatore ad una funzione, aggiungere al nome della funzione una coppia di parentesi tonde equivale invece ad invocarla!
- **i commenti nel codice contengono molte informazioni utili per lo svolgimento della prova, si consiglia quindi di tenerli in debita considerazione**

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, 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**.