Progetto `chatterbox` , per il modulo di laboratorio SOL a.a. 2017-2018

Contents

1 Introduzione

Il modulo di Laboratorio di Sistemi Operativi prevede lo svolgimento di un progetto da sviluppare utilizzando il linguaggio C in ambiente Linux. Il progetto deve essere svolto singolarmente da un solo studente. Questo documento descrive la struttura complessivia del progetto chiamato `chatterbox` . La scadenza per la consegna del progetto coincide con la scadenza dell'appello straordinario (Marzo/Aprile 2018) dell'a.a 2016/2017. Il progetto dovrà essere sviluppato e documentato utilizzando gli strumenti, le tecniche e le convenzioni presentati durante il corso.

1.1 Materiale in linea

Tutto il materiale relativo al corso può essere reperito sul sito web:

`http://didawiki.cli.di.unipi.it/doku.php/informatica/sol/laboratorio18`

Il sito verrà progressivamente aggiornato con le informazioni riguardanti il progetto (ad esempio con update delle specifiche). Il sito è un Wiki e gli studenti possono registrarsi per ricevere automaticamente gli aggiornamenti delle pagine che interessano maggiormente. In particolare consigliamo a tutti di registrarsi alla pagina degli 'Avvisi Urgenti'.

Eventuali chiarimenti possono essere richiesti consultando i docenti del corso durante l'orario di ricevimento e/o per posta elettronica.

1.2 Struttura del progetto e tempi di consegna

Il progetto è valido per l'intero anno accademico (a.a.) e può essere consegnato entro l'appello straordinario dell'a.a. 2017/2018.

Per la consegna del progetto svolto dallo studente, deve essere generato un tarball utilizzando il target `consegna` del `Makefile` contenuto nel kit di sviluppo fornito dai docenti (il nome del file dovrà essere del tipo `NomeCognome_CorsoA_chatty.tar` per gli studenti del corso A e del tipo `NomeCognome_CorsoB_chatty.tar` per gli studenti del conso B). Il tarball generato deve essere consegnato seguendo le istruzioni fornite dai docenti nel sito del progetto a questo indirizzo:

`http://didawiki.di.unipi.it/doku.php/informatica/sol/laboratorio18/progetto`

I tarball consegnati che non contengono tutti i file necessari, che non compilano ed eseguono correttemante sulla macchina virtuale Ubuntu x86_64 fornita durante il corso (vedere pagina del corso), non saranno accettati. I progetti il cui tarball non è stato generato con il target `consegna` del Makefile fornito dai docenti non verranno accettati. **Si prega di controllare che tutti i file necessari alla corretta compilazione ed esecuzione del progetto siano presenti nel tarball prima di consegnarlo.**

La consegna del progetto deve essere fatta almeno 5 giorni prima della data dell'appello che si intende sostenere. Si ricorda, che per sostenere l'esame è condizione necessaria iscriversi.

1.3 Valutazione del progetto

La valuazione del progetto avviene in due fasi:

1. valutazione preliminare per stabilere se lo studente è *Ammesso* o *Non Ammesso* alla prova scritta di teoria;
2. valutazione approfondita del progetto e discussione del progetto con lo studente durante la prova orale.

Condizione necessaria per essere *Ammesso* alla prova scritta è che tutti i test del progetto siano superati con successo.

La valutazione approfondita del progetto è effettuata in base ai seguenti criteri (non in ordine di importanza):

- motivazioni, originalità ed economicità delle scelte progettuali
- strutturazione del codice (suddivisione in moduli, corretto uso del Makefile e librerie etc.)
- efficienza e robustezza del software
- gestione della concorrenza e dei segnali

- modalità di testing
- aderenza alle specifiche
- qualità del codice C e dei commenti
- utilizzo di strutture dati opportune
- chiarezza ed adeguatezza della relazione (vedi Sez. 5.3)

La prova orale tenderà a stabilire se lo studente è realmente l'autore del progetto consegnato e verterà su tutto il programma del corso e su tutto quanto usato nel progetto anche se non fa parte del programma del corso. L'esito della prova orale è essenziale per delineare il voto finale del modulo di laboratorio. In particolare, l'orale potrà comprendere:

- una discussione delle scelte implementative del progetto
- l'impostazione e la scrittura di semplici script bash e Makefile
- l'impostazione e la scrittura di programmi C + POSIX calls non banali (sia sequenziali che concorrenti), incluso il funzionamento delle system calls (anche in relazione al funzionamento del Sistema Operativo in cui operano)
- domande su tutto il programma presentato durante il corso (fare riferimento al sito del corso).

Al progetto viene assegnata una valutazione complessiva da 0 a 30. La *qualità della documentazione allegata* contribuisce fino ad un massimo di 8 punti.

Casi particolari Gli studenti iscritti ai vecchi ordinamenti (nei quali il voto di Lab. 4 contribuiva al voto di SO) avranno assegnato un voto in trentesimi che verrà combinato con il voto del corso di SO corrispondente secondo modalità da richiedere ai due docenti coinvolti.

2 `chatterbox`

2.1 Descrizione generale

Lo scopo del progetto è lo sviluppo in C di un server concorrente che implementa una chat. Gli utenti della chat possono scambiarsi messaggi testuali e/o files collegandosi al server con un programma client.

Un messaggio testuale è una stringa non vuota avente una lunghezza massima stabilita nel file di configurazione del server (vedi Sez. 2.2). Un file di qualsiasi tipo può essere scambiato tra due client purché la size non sia superiore a quanto stabilito nel file di configurazione del server (vedi Sez. 2.2).

Ogni utente della chat è identificato univocamente da un *nickname* alfanumerico. L'utente si connette al server, invia richieste e riceve risposte utilizzando un client la cui struttura base è fornita dai docenti (vedere il file client.c nel kit del progetto). Uno o piú processi client comunicano con il server per inviare richieste di vario di tipo. Le richieste gestite dal server sono almeno le seguenti:

1. registrazione di un nuovo utente (ossia registrazione di un nuovo nickname) (REGISTER_OP);
2. richiesta di connessione per un utente giá registrato (CONNECT_OP);
3. invio di un messaggio testuale (POSTTXT_OP) ad un nickname;
4. invio di un messaggio testuale a tutti gli utenti registrati (POSTTEXTALL_OP);
5. richiesta di invio di un file (POSTFILE_OP) ad un nickname;
6. richiesta di recupero di un file inviato da un altro nickname (GETFILE_OP);

7. richiesta di recupero degli ultimi messaggi inviati al client (GETPREVMSGS_OP);
8. richiesta della lista di tutti i nickname connessi (USRLIST_OP);
9. richiesta di deregistrazione di un nickname (UNREGISTER_OP);
10. richiesta di disconnessione del client (DISCONNECT_OP).

Per ogni messaggio di richiesta il server risponde al client con un messaggio di esito dell'operazione richiesta e con l'eventuale risposta. Ad esempio, se si vuole inviare un messaggio ad un utente non esistente (cioè il cui nickname non è stato mai registrato), il server risponderà con un messaggio di errore opportunamento codificato (vedere file ops.h nel kit del progetto) oppure risponderà con un messaggio di successo (codificato con il codice OP_OK – vedere il file ops.h nel kit del progetto contenente i codici delle operazioni di richiesta e di risposta).

La comunicazione tra client e server avviene tramite socket di tipo AF_UNIX. Il path da utilizzare per la creazione del socket AF_UNIX da parte del server, deve essere specificato nel file di configurazione utilizzando l'opzione `UnixPath` (vedi Sez. 2.2). Per rendere piú agevole lo sviluppo ed il testing dell'applicazione sulle macchine del centro di calcolo per la didattica, le socket create dallo studente con numero di matricola `matricola` potranno avere un nome che rispetta il seguente formato: `chatterbox_sock_matricola`. L'obiettivo è minimizzare possibili interazioni indesiderate fra progetti sviluppati da utenti diversi sulla stessa macchina del laboratorio.

Nello sviluppo del codice del server, lo studente dovrà tenere in considerazione che il server deve essere in grado di gestire efficientemente fino ad alcune centinaia di connessioni di client contemporanee senza che questo comporti un rallentamento evidente nell'invio e ricezione dei messaggi/files dei clients. Inoltre il sistema deve essere progettato per gestire alcune decine di migliaia di utenti registrati alla chat (questo aspetto ha impatto sulle strutture dati utilizzate per memorizzare le informazioni).

Il server al suo interno implementa un sistema multi-threaded in grado di gestire concorrentemente sia nuove connessioni da nuovi client che gestire le richieste su connessioni già stabilite. Un cliente può sia inviare una sola richiesta per connessione oppure, inviare piú richieste sulla stessa connessione (ad esempio: postare un messaggio ad un utente, inviare un file ad un nickname, inviare un messaggio a tutti gli utenti, recuperare file inviati da altri client, etc...). Il numero massimo di connessioni concorrenti che il server `chatterbox` gestisce è stabilito dall'opzione di configurazione `MaxConnections`. Se tale numero viene superato, dovrà essere ritornato al client un opportuno messaggio di errore (vedere la codifiche dei messaggi di errore nel file ops.h). Il numero di thread utilizzati per gestire le connessioni è specificato nel file di configurazione con l'opzioni `ThreadsInPool`. Tale numero specifica la dimensione di un pool di threads, sempre attivi per tutta la durata dell'esecuzione del server e che il server `chatterbox` utilizzerà per gestire le connessioni dei client. Tale valore non può essere 0.

Un thread appartenente al pool gestisce una sola richiesta alla volta inviata da uno dei client. Non appena conclude la gestione della richiesta, controlla se ci sono altre richieste da servire dalla coda delle richieste ed in caso affermativo ne prende una da gestire. Se non ci sono richiesta da servire, il thread si mette in attesa di essere svegliato per gestire una nuova richiesta.

Non appena il server accetta una nuova connessione da un client che ha inviato una richiesta di CONNECT, il server invia al client la lista di tutti gli utenti connessi in quel determinato istante (il formato della lista inviata dal server al client è lo stesso del messaggio di risposta alla richiesta USRLIST_OP).

Il server può essere terminato in ogni momento della sua esecuzione inviando un segnale di `SIGINT`, `SIGTERM` o `SIGQUIT` (vedi Sez. 2.2).

2.2 Esecuzione e file di configurazione

Il server `chatterbox` viene attivato da shell con il comando

```
$ ./chatterbox -f chatty.conf
```

dove `chatty.conf` è il file di configurazione contenente tutti i parametri relativi alla configurazione del server. Il formato del file di configurazione è schematizzato in Fig. 1,

```
# path utilizzato per la creazione del socket AF_UNIX
UnixPath         = /tmp/chatty_socket

# numero massimo di connessioni concorrenti gestite dal server
MaxConnections   = 32

# numero di thread nel pool
ThreadsInPool    = 8

# dimensione massima di un messaggio testuale (numero di caratteri)
MaxMsgSize       = 512

# dimensione massima di un file accettato dal server (kilobytes)
MaxFileSize      = 1024

# numero massimo di messaggi che il server 'ricorda' per ogni client
MaxHistMsgs      = 16

# directory dove memorizzare i files da inviare agli utenti
DirName          = /tmp/chatty

# file nel quale verranno scritte le statistiche
StatFileName     = /tmp/chatty_stats.txt
```

Figure 1: Formato del file testuale `chatty.conf`

Nel momento in cui un client si connette al server (CONNECT_OP) invia il proprio nickname. Se il nickname non è stato precedentemente registrato con una operazione di (REGISTER_OP) il server non accetta il client tra quelli che possono inviare messaggi e ritorna un messaggio di errore (i.e., OP_NICK_UNKNOWN – vedi il file ops.h nel kit del progetto). Se il nickname è registrato, il server risponde al client con la lista di tutti gli utenti in quel momento connessi. La lista degli utenti connessi viene inviata dal server al client anche all'atto della registrazione di un nickname dato che l'operazione di registrazione implica che il client sia connesso. Se il cliet, con *nickname1* vuole inviare un file ad un altro client con *nickname2*, allora preparerà una richiesta di tipo POSTFILE_OP. Il server, riceve il file e lo memorizza nella directory specificata nel file di configurazione con il nome `DirName` ed informa *nickname2* che è disponibile per il download un file inviato da *nickname1*. La notifica avviene inviando come identificatore il nome del file. Se *nickname2* vuole recuperare il file dovrá inviare una richiesta GETFILE_OP inviando nella richiesta il nome del file ricevuto in precedenza.

I messaggi possono essere notificati anche a client non connessi al server. Se ad esempio un client vuole inviare un messaggio a tutti gli utenti registrati (POSTTEXT_ALL), gli utenti che risultano connessi al server al momento dell'arrivo del messaggio (cioé sono online), riceveranno il messaggio "immediatamente", mentre gli utenti che non sono connessi al server potranno ricevere tali messaggi solo dietro esplicita richiesta di tipo GETPREVMSGS_OP. Il server `chatterbox` memorizza una history di al più *MaxHistMsgs* messaggi (testuali o files) per ogni client registrato. I messaggi vanno sempre memorizzati nella history, anche se il messaggio viene consegnato al destinatario (il concetto di history è lo stesso del comando bash `history`).

Un client puó in ogni momento, oltre ad inviare messaggi testuali e file, disconnettersi (DISCONNECT_OP), chiedere la lista degli utenti connessi (USRLIST_OP), deregistrarsi (UNREGISTER_OP) ed in quest'ultimo caso tutti i messaggi pendenti e non ancora inviati al client destinatario dovranno essere eliminati. NOTA: il client fornito dai docenti non implementa in modo esplicito il messaggio di disconnessione. La disconnessione avviene quando il client chiude la connessione verso il server.

2.3 Gestione dei segnali

Oltre ai segnali `SIGINT` o `SIGTERM` o `SIGQUIT` che determinano la terminazione di `chatterbox` , il server gestisce il segnale `SIGUSR1`. Se viene inviato un segnale di tipo `SIGUSR1` il server stampa le statistiche correnti nel file associato all'opzione `StatFileName` presente nel file di configurazione secondo il formato descritto nella sezione seguente. Se tale opzione è vuota o non è presente, il segnale ricevuto viene ignorato.

2.4 Statistiche

Alla ricezione del segnale `SIGUSR1` il server appende nel file associato all'opzione `StatFileName` le statistiche del server. Le stampa nel file deve avere il formato `<timestamp - statdata>` dove `timestamp` è l'istante temporale in cui le statistiche vengono inserite nel file e `statdata` è una sequenza di numeri (separati da uno spazio) che riportano i valori monitorati durante il tempo di vita del server. I dati statistici minimi che devono essere prodotti nel file sono riportati in Fig. 2. Se lo studente lo ritiene, può arricchire le statistiche prodotte documentandole opportunamente nella relazione che accompagna il progetto.

```
n. di utenti registrati
n. di clienti online
n. di messaggi consegnati
n. di messaggi da consegnare
n. di file consegnati
n. di file da consegnare
n. di messaggi di errore
```

Figure 2: Dati statistici minimi che devono essere prodotti nel file associato all'opzione `StatFileName`

2.5 Protocollo client server

Il protocollo di comunicazione per ogni tipo di messaggio di richiesta che il client può inviare al server, deve essere stabilito dallo studente e documentato nella relazione. Per effettuare i test deve essere utilizzato il client (`client.c`) fornito dai docenti che non deve essere modificato in nessuna parte.

Dovranno essere definiti chiaramente i tipi dei messaggi di richiesta e di risposta e come sono gestiti i messaggi di errore.

2.6 Script bash

Deve essere realizzato uno script bash che prende in input come argomenti il file di configurazione del server `chatterbox` ed un numero intero positivo (t). Lo script estrae il nome della directory associato all'opzione *DirName* dal file passato come primo argomento e archivia in un file con estensione `.tar.gz` tutti i files (e directories) contenuti in tale directory che sono piú vecchi di t minuti. Se l'operazione di archiviazione è andata a buon fine, i file e le cartelle archiviati dovranno essere eliminati. Se il parametro t vale 0 (t==0), allora dovranno essere stampati sullo standard output tutti i file contenuti nella directory *DirName*. Lo script deve stampare il messaggio di uso se lanciato senza opzioni o se l'opzione lunga "–help" è presente tra gli argomenti del programma.

È fatto esplicito divieto di usare `sed` e `awk` nella realizzazione dello script.

3 Istruzioni

3.1 Materiale fornito dai docenti

Viene fornito un kit denominato `kit_chatty.tar` contenente i file per lo svolgimento del progetto `chatterbox` . Il kit può essere scaricato all'indirizzo:

`http://didawiki.cli.di.unipi.it/doku.php/informatica/sol/laboratorio18/progetto`

Si prega di leggere con attenzione il `README` contenuto nel kit per una descrizione dei file forniti dai docenti.

3.2 Cosa devono fare gli studenti

Gli studenti devono:

- leggere *attentamente* le specifiche del progetto contenute in questo documento, il README contenuto nel kit e capire il funzionamento del codice fornito dai docenti (in particolare il client.c)
- implementare tutte le funzionalità richieste
- sviluppare semplici test iniziali per verificare la funzionalità base del sistema
- **solo successivamente**, verificare le funzioni con i test forniti dai docenti (attenzione: questi test vanno eseguiti su codice già corretto e ragionevolmente funzionante altrimenti possono dare risultati forvianti o di difficile interpretazione)
- preparare la *documentazione*: vedi Sez. 5.
- generare i progetti finali esclusivamente utilizzando il Makefile fornito dai docenti seguendo le istruzioni riportate nei README.
- creare un tarball contenente tutti i file del progetto ed inviarlo ai docenti seguendo le istruzioni fornite nella Sez. 1.2.

4 Parti Opzionali

Possono essere realizzate funzionalità ed opzioni in più rispetto a quelle richieste (ad esempio per quanto riguarda `chatterbox` , aggiungendo altre opzioni nel file di configurazione aumentando le funzionalità del server, inserendo nuove statistiche, etc.).

Le parti opzionali devono essere spiegate nella relazione e corredate di test appropriati ove opportuno.

4.1 Gestione dei gruppi

Questa parte è opzionale. Per chi intende realizzarla, il sistema deve prevedere la creazione e gestione di gruppi di utenti. Un gruppo ha un nome (*groupname*) e più utenti possono essere registrati ad un gruppo. Devono essere implementate le seguente richieste:

1. richiesta di creazione di un gruppo (CREATEGROUP_OP) con un dato nome (*groupname*);
2. richiesta di aggiunta di un nickname registrato ad un groupname (ADDGROUP_OP);
3. richiesta di cancellazione di un nickname registrato da un groupname (DELGROUP_OP);

Un qualsiasi utente può creare un nuovo gruppo che non sia stato già registrato. Solamente l'untente che ha creato il gruppo può cancellare un *groupname*. All'atto della creazione di un gruppo, il nickname che ne fa richiesta, è automaticamente aggiunto al gruppo. Altri utenti possono aggiungersi e cancellarsi a/da un gruppo in ogni momento con l'operazione ADDGROUP_OP e DELGROUP_OP. Solo un utente registrato al gruppo può inviare e ricevere messaggi al/dal gruppo. E' possibile inviare agli utenti di un gruppo sia messaggi testuali che file di qualsiasi tipo con le stesse regole e restrizioni viste in precedenza per gli utenti.

Lo studente dove implementare la gestione delle tre richieste sopra descritte, la gestione della richiesta di cancellazione di un gruppo è facoltativa.

5 Codice e documentazione

In questa sezione, vengono riportati alcuni requisiti del codice sviluppato e della relativa documentazione.

5.1 Vincoli sul codice

Makefile e codice devono compilare ed eseguire CORRETTAMENTE su versoni recenti di Ubuntu x86_64 ed almeno sulla macchina virtuale (xubuntu) fornita durante il corso. La relazione deve specificare su quali versioni di Ubuntu è possibile far girare correttamente il codice.

La stesura del codice deve osservare i seguenti vincoli:

- la compilazione del codice deve avvenire attraverso il Makefile fornito dai docenti
- il codice deve compilare senza errori o warning gravi utilizzando le opzioni `-Wall`
- NON devono essere utilizzate funzioni di temporizzazione quali le `sleep()` o le `alarm()` per risolvere problemi di race condition o deadlock fra i processi/thread. Le soluzioni implementate devono necessariamente funzionare qualsiasi sia lo scheduling dei processi/thread coinvolti
- DEVONO essere usati dei nomi significativi per le costanti nel codice (con opportune `#define` o `enum`)
- il server non deve generare memory leaks (testare l'esecuzione del server con valgrind)

5.2 Formato del codice

Il codice sorgente deve adottare una convenzione di indentazione e commenti chiara e coerente. In particolare deve contenere

- una intestazione per ogni file che contiene: il nome ed il cognome dell'autore, la matricola e l'indirizzo e-mail; una dichiarazione che il programma è, in ogni sua parte, opera originale dell'autore.
- un commento all'inizio di ogni funzione che specifichi l'uso della funzione (in modo sintetico), l'algoritmo utilizzato (se significativo), il significato delle variabili passate come parametri, eventuali variabili globali utilizzate, effetti collaterali sui parametri passati per puntatore etc.
- un breve commento che spieghi il significato delle strutture dati e delle variabili globali (se esistono);
- un breve commento per i punti critici o che potrebbero risultare poco chiari alla lettura
- un breve commento all'atto della dichiarazione delle variabili locali, che spieghi l'uso che si intende farne (se non ovvio dal contesto)

5.3 Relazione

La documentazione del progetto consiste nei commenti al codice e in una breve relazione (massimo 10 pagine) il cui scopo è quello di descrivere la struttura complessiva del lavoro svolto. La relazione *deve rendere comprensibile il lavoro svolto ad un estraneo, senza bisogno di leggere il codice se non per chiarire dettagli implementativi. In particolare, NON devono essere ripetute le specifiche del progetto contenute in questo documento.* In pratica la relazione deve contenere:

- le principali scelte di progetto: strutture dati principali utilizzate, scelte sulla size delle strutture, come é stata gestita la memoria e l'eventuale accesso a files.
- la strutturazione del codice: logica della divisione su più file, eventuali librerie.
- la struttura dei programmi sviluppati in modo schematico e le principali interazioni tra i vari moduli
- l'interazione fra i processi/threads ed i relativi protocolli di comunicazione: come é stata gestita la terminazione, gestione della concorrenza, gestione dei segnali.
- la struttura dei programmi di test aggiuntivi oltre a quelli forniti dai docenti
- eventuali estensioni che lo studente ha sviluppato oltre a quanto richiesto
- quanto altro si ritiene essenziale alla comprensione del lavoro svolto
- su quali macchine é stato testato il codice
- le difficoltà incontrate e le soluzioni/semplificazioni adottate

La relazione deve essere in formato PDF.