Laboratorio di Sistemi Operativi
Progetto d'Esame AA 2010/11

Versione 1.0

Corso di Laurea in Informatica Applicata

Maggio 2011

1 Introduzione

Oltre ad un compito scritto, che copre il modulo teorico, il corso di Sistemi Operativi e Laboratorio (SOL, 277AA) prevede lo svolgimento di un progetto finale. Il progetto consiste nello sviluppo di un software, e della relativa documentazione, che implementi un semplice sistema di

FILE SHARING,

in cui più client condividono e si scambiano file interagendo con un server.
Il progetto può essere svolto **singolarmente** oppure in gruppi di al più **due (2) studenti**.

1.1 Precedenti anni accademici

Gli studenti iscritti prima dell'anno 2004/2005 possono svolgere il progetto per mutuare Laboratorio di Programmazione di Sistema (LPS, AA012), ma sono tenuti ad indicare al docente la loro situazione al momento dell'inizio del progetto. Analogamente, gli studenti possono svolgere il progetto per mutuare Laboratorio di Programmazione Concorrente e di Sistema (LCS, AA538).

Gli studenti che dovessero mutuare sia LPS che Laboratorio di Programmazione Concorrente (LPC, AA011), o sia LPS che Laboratorio di Linguaggi di Sistema (LLS, AA536), sono tenuti a realizzare le istruzioni aggiuntive indicate nelle sezioni 3.4.2 e 4.4.2.

Studenti in situazioni diverse da quelle indicate devono preventivamente consultarsi con il docente.

1.2 Materiale in linea

Tutto il materiale relativo al corso di Laboratorio di Sistemi Operativi dell'Anno Accademico 2010–2011 è disponibile all'indirizzo web:

`http://www.di.unipi.it/~cardillo/labso`.

A quell'indirizzo web sarà possibile leggere le risposte alle più frequenti domande di chiarimento sul testo del progetto.

1.2.1 Specifiche del progetto (questo documento)

Il documento che descrive le specifiche del progetto (cioè questo documento) potrebbe essere modificato durante lo svolgimento del progetto con il solo obiettivo di fornire una descrizione più chiara delle sue parti che si riveleranno ambigue. Gli studenti sono invitati a verificare periodicamente la presenza di nuove versioni sulla pagina web indicata sopra (Sez. 1.2). Il numero di versione associato a questo documento, presente nel titolo – versione corrente: 1.0 – verrà incrementato in caso di modifiche.

1.3 Consegna: entro il 31/01/2012

Il progetto deve essere consegnato tramite posta elettronica, con un messaggio che contiene il software sviluppato e la relativa documentazione(consultare la sez. 6). Il progetto dovrà essere **consegnato *improrogabilmente* entro il**

31 gennaio 2012,

(quindi entro le ore 23:59, ora italiana, del 31 gennaio 2012).

1.3.1 Discussione del progetto

Il progetto verrà discusso in una data *vicina* a quella della consegna e, comunque, non oltre l'inizio del secondo semestre dell'anno accademico 2011-2012. Nel caso in cui il progetto venga realizzato in gruppo, alla sua discussione devono presenti *tutti* gli studenti del gruppo. La data della discussione verrà concordata via eMail tra il docente e lo studenti/gli studenti.

1.4 Valutazione del progetto

Al **progetto base**, se accettato, è assegnata una valutazione iniziale da 18 a 24, di cui 6 punti esclusivamente determinati dalla *qualità della relazione allegata*. Nel caso del **progetto avanzato** la valutazione può arrivare fino a 28. Uno studente può scegliere di implementare uno tra i seguenti progetti:

PROGETTO BASE Tutte le sezioni meno 3.4 e 4.4.

PROGETTO AVANZATO–1 Tutte le sezioni meno 3.4.2, 4.4.2, 4.4.3. La descrizione delle funzionalità avanzate da implementare è contenuta nelle sezioni **3.4.1** e **4.4.1**.

PROGETTO AVANZATO–2 Tutte le sezioni meno 3.4, 4.4.1, 4.4.2. La descrizione delle funzionalità avanzate da implementare è contenuta nella sezione **4.4.3**.

In qualsiasi progetto scelto tra i tre sopra elencati, gli studenti che devono mutuare LLS devono implementare anche le funzionalità descritte nelle sezioni 3.4.2 e 4.4.2.

La valutazione iniziale è assegnata in base ai seguenti criteri:

- motivazioni e chiarezza delle scelte progettuali;
- strutturazione del codice (suddivisione in moduli, uso di makefile e librerie, ecc.)
- efficienza e robustezza del software;
- aderenza alle specifiche;
- qualità del codice e dei commenti;
- adeguatezza e chiarezza della documentazione.

La discussione del progetto, che potrà includere un qualsiasi argomento del programma del corso – anche se non utilizzato all'interno del progetto consegnato, servirà principalmente a valutare l'apporto individuale dei componenti del gruppo.

Se la discussione evidenzia uno scarso apporto alla realizzazione del progetto o lacune relative ai concetti di base del corso da parte di uno solo dei componenti del gruppo, a questo soltanto viene applicato un *malus* di penalizzazione, che potrebbe includere anche la richiesta di un progetto integrativo basato su quello presentato. Ovviamente, nel caso in cui si evidenzino carenze e lacune in entrambi i componenti del gruppo, quanto specificato nel caso precedente per un singolo studente si applica ad entrambi. ***Il voto finale sarà espresso in trentesimi e sarà individuale***, ciò significa che nel caso di un progetto di gruppo i voti assegnati ai due studenti potranno essere differenti.

2 Progetto: Specifiche

Lo scopo del progetto è lo sviluppo di un sistema software per gestire la condivisione e lo scambio di file tra più utenti attraverso un server centrale che tiene traccia dell'insieme dei file condivisi da tutti i client e offre la possibilità di effettuare ricerche all'interno di questo insieme. Il sistema è costituito da due processi concorrenti:

fsclient il processo client (uno per ogni utente che desidera condividere e scaricare file) che si occupa di interagire con il server per:

- pubblicare sul server `fsserver` l'elenco dei file condivisi;
- effettuare ricerche sul server `fsserver`;
- effettuare il download (da un altro `fsclient`) di un file;
- effettuare l'upload (verso un altro `fsclient`) di un file

fsserver il processo server interagisce con i client per gestire le informazioni relative ai file condivisi. In particolare, il server `fsserver`:

- memorizza per ogni client `fsclient` l'elenco dei file condivisi e la password, specificata dai singoli `fsclient`, necessaria per avviare un download;
- gestisce le ricerche dei client `fsclient` restituendo le informazioni necessarie per scaricare un particolare file;
- mantiene traccia dei client `fsclient` connessi.

3 Il server `fsserver`

Il server `fsserver` viene attivato da shell con il comando:

```
./fsserver [-l log-file] [-s sck-name]
```

dove:

- l'opzione `-l` specifica il nome del file di log nel quale il server deve *aggiungere* i messaggi di log.
- l'opzione `-s` specifica il nome del socket sul quale il server si porrà in ascolto delle richieste dei client.

3.1 Avvio del server

Al momento dell'avvio il server:

- apre in scrittura il file di log specificato oppure, se non è stato specificato, sceglie un nome `N` per il file di log, lo apre, e stampa `N` (cioè percorso assoluto del file di log) sullo standard output;
- apre il socket `sck-name` specificato da riga di comando oppure, se non è stato specificato, sceglie un nome `N` per il socket, lo apre e stampa `N` (cioè il percorso assoluto del socket) sullo standard output.
- Si mette in attesa di richieste da parte dei client sul socket creato al passo precedente.

3.2 Gestione delle richieste

Quando il server riceve una richiesta da parte di un client delega la sua gestione a un nuovo thread `clth`. Il thread creato termina subito dopo avere restituito le informazioni richieste al client oppure in caso di errore. Un thread `clth`:

- in caso di ricerche da parte di client `fsclient` remoti, accede alle stutture dati del server e restituisce i nomi dei file che soddisfano la query di ricerca ricevuta. Per ogni file che soddisfa la query, `clth` deve restituire al client `fsclient` anche i dati necessari per scaricare il file dai processi `fsclient` che lo hanno condiviso.
- in caso di richieste di pubblicazione di file in condivisione, il thread `clth` deve aggiornare le strutture dati che contengono l'insieme dei client connessi e dei relativi file in condivisione.

3.3 Scrittura del log

La gestione del file di log viene assegnata ad un thread specifico `logfth` del processo `fsserver`. Le modalità di interazione tra i thread `clth` e il thread `logfth` devono essere decise dallo studente e discusse nella documentazione del progetto.

3.4 Sezione relativa alle funzionalità extra del server

3.4.1 Aggiornamento dei client connessi + vincolo sul massimo numero di comandi da parte di un singolo client

Si noti che non viene richiesto un login / logout esplicito ai client. Come conseguenza di questa scelta, se un client `C` termina, il server conserva in memoria i riferimenti ai file condivisi da `C`. Tali riferimenti potrebbero essere inviati ai client che effettuano ricerche anche dopo la disconnessione di `C`. Aggiungere nell'implementazione del server `fsserver`, un thread che periodicamente verifichi quali client siano ancora in linea aggiornando di conseguenza le strutture dati in memoria.

Inserire nel server un vincolo sul massimo numero di richieste da parte di un singolo client che possono essere gestite contemporaneamente.

3.4.2 Consentire l'unsharing di file - per LLS

Offrire la possibilità a un client di rimuovere uno o più file dall'insieme dei file condivisi. Si noti che devono essere implementate le funzionalità necessarie anche nel programma dei client (sez. 4.4.2).

Oltre alla funzionalità di unsharing, inserire nel server un vincolo sul massimo numero di richieste da parte di un singolo client che possono essere gestite contemporaneamente. Questa parte è in comune con il PROGETTO AVANZATO – 1.

4 Il client `fsclient`

Il client `fsclient` viene attivato da shell con il comando:

```
./fsclient -s server-sck -f shared-folder -p passwd [-l log-file] [-s sck-name] [-d
                                    download-folder]
```

dove:

- il parametro (obbligatorio) `-s` indica il nome del socket del server a cui collegarsi.
- il parametro (obbligatorio) `-f` indica il nome della directory all'interno della quale si trovano i file che il client vuole condividere. Non è necessario gestire file nelle sottodirectory della directory condivisa: nel caso in cui lo studente decida di farlo deve però gestire opportunamente la condivisione di file con lo stesso nome contenuti in directory diverse.
- il parametro (obbligatorio) `-p` indica la password che gli altri client devono comunicare per avviare una sessione di download. Questa password verrà comunicata al server, il quale la invierà ai client quando restituisce, tra i risultati di una ricerca, almeno un file condiviso da questo client.
- l'opzione `-l` specifica il nome del file di log nel quale il client deve *aggiungere* i messaggi di log.
- l'opzione `-s` specifica il nome del socket sul quale il client si porrà in ascolto delle richieste di download da parte di altri client.
- l'opzione `-d` specifica il nome della directory in cui memorizzare i file scaricati.

4.1 Avvio del client

Quando il client viene avviato:

- apre in scrittura il file di log specificato oppure, se non è stato specificato, sceglie un nome `N` per il file di log, lo apre, e stampa `N` (cioè percorso assoluto del file di log) sullo standard output;
- apre il socket `sck-name` specificato da riga di comando oppure, se non è stato specificato, sceglie un nome `N` per il socket e lo apre.
- elenca i file da condividere contenuti nella directory `shared-folder` specificata da riga di comando. Se l'opzione non è stata specificata, usa un valore di default.
- si connette al server sul socket `server-sck` per pubblicare l'elenco dei file condivisi, insieme alla password necessaria per scaricare i file;
- genera un thread per gestire l'interazione da tastiera con l'utente;
- si mette in ascolto sul socket creato.

4.2 Il client in esecuzione

Il client ha sempre almeno tre thread in esecuzione. Il primo `inth` gestisce l'interazione da riga di comando con l'utente. Il secondo `sckth` si mette in ascolto sul socket pubblico in attesa di richieste di download. Il terzo si occupa del file di log (si veda la sezione 4.3). Il thread `inth` legge da tastiera i comandi per interagire con il server. La sintassi dei comandi viene stabilita dallo studente e deve essere discussa nella relazione.

L'esecuzione dei comandi non deve essere bloccante: il thread `inth` deve accettare nuovi comandi anche durante l'esecuzione dei comandi inseriti in precedenza.

Il thread `sckth` rimane in attesa di richieste di download da parte di altri client. Quando riceve una richiesta:

- genera un thread per la gestione della richiesta. Il nuovo thread verifica che la password sia corretta e, in questo caso, avvia il trasferimento del file verso il client.

4.3 Scrittura del log

La gestione del file di log viene assegnata ad un thread specifico `logfth` del processo `fsclient`. Le modalità di interazione tra i vari thread del client devono essere decise dallo studente e devono essere discusse nella documentazione del progetto.

4.4 Sezione relativa alle funzionalità extra del client

4.4.1 Aggiornamento dei client connessi

In questo caso il client riceverà anche le richieste inviate dal server per verificare se il client è ancora in esecuzione. Aggiungere nel client le funzioni necessarie per implementare questa funzionalità.

4.4.2 Consentire l'unsharing di file – per LLS

Oltre ad offrire un opportuno comando per richiedere l'unsharing di file, il client dovrà includere un thread che osservi continuamente la directory condivisa e rilevi la cancellazione di eventuali file. Una volta rilevata la cancellazione, il thread dovrà interagire con gli altri thread per inviare al server la richiesta di unsharing. Devono essere implementate anche le funzioni corrispondenti nel server (sez. 3.4.2).

4.4.3 Download simultaneo dello stesso file da più client

Nel caso in cui uno stesso file `F` sia condiviso da più client `C1`, `C2`, ..., `CK`, un client può decidere di scaricare segmenti diversi del file `F` da più client `c1`, ..., `cm`, con `m` $\leq$`K`. Implementare una modalità di download "parallela" di un file condiviso da più utenti. Prestare attenzione a:

- dimensione del segmento, in particolare come indicare l'inizio e la fine;
- gestire il fallimento del download di un singolo segmento in modo da non scaricare segmenti già ricevuti (completamente o parzialmente).

È ovviamente possibile gestire il caso di file con lo stesso nome (si consulti il manuale di `md5`): gli studenti sono invitati a gestire questo caso che rimane, però, facoltativo: verrà considerata completa la soluzione priva di tale gestione.

5 Note sull'implementazione

5.1 Socket

Usare socket nel dominio `AF_UNIX`.

5.2 Protocollo di comunicazione

Lo studente deve stabilire e discutere nella relazione la sintassi dei messaggi scambiati tra client e server e tra due client. Nei paragrafi successivi verranno mostrate alcune possibilità. In alternativa, è possibile inviare variabili con tipi (ad esempio, strutture C) specificati nel protocollo di comunicazione.

5.2.1 Client $\Longrightarrow$ Server

Pubblicazione dei file condivisi:

``ADD:SOCKET:PASSWD:FILENAME''

per comunicare al server la condivisione del file `FILENAME`, con la password `PASSWD`, da parte del client in ascolto sul socket `SOCKET`. Scegliendo questa sintassi il client dovrà inviare al server un messaggio per ogni file condiviso. In alternativa è possibile usare un singolo messaggio contenente tutti i nomi dei file condivisi:

``ADD:SOCKET:PASSWD:FILENAME[:FILENAME]*''.

Ricerca per filename (esatta):

``EXACT:<string>''

Ricerca per prefisso:

``PREFIX:<string>''

5.3 File condivisi

Per semplificare l'implementazione, due file vengono considerati simili se hanno lo stesso nome. Ovviamente, nel caso in cui venga implementata la parte avanzata, se due file differenti vengono condivisi con lo stesso nome potrebbero verificarsi errori nel download dei segmenti dei due file e nella loro successiva fusione in un singolo file: non è richiesta la gestione degli errori legati a questa situazione, ma gli studenti sono invitati ad aggiungere la gestione di questo caso (che, si ribadisce, rimane facoltativa).

Il sistema di file sharing può essere limitato alla condivisione di **soli file di testo**. Nel caso in cui il progetto consenta la condivisione di altri tipi di file, nella documentazione deve essere esplicitamente indicato un metodo per verificare se il file ricevuto è "corretto".

5.4 Interazione con l'utente

La sintassi dei comandi che il client offre può essere molto semplice. Il comando inserito dall'utente non deve essere bloccante: l'utente deve poter eseguire un nuovo comando durante l'esecuzione dei comandi inseriti in precedenza. Ad esempio, se l'utente effettua una ricerca sul server, deve essere in grado di chiedere il download di un file (il cui riferimento è stato restituito da una precedente ricerca) anche prima che la ricerca appena lanciata sia terminata.

5.5 Tempi

Il download di un file da un client all'altro sarà, in generale, molto veloce. Rallentare il sistema sospendendo i vari thread durante il download. La durata della sospensione, in secondi, può essere prefissata oppure scelta casualmente in un range predeterminato.

6 Codice e documentazione

Questa sezione riporta i requisiti minimi sul codice sviluppato e sulla relativa documentazione.

6.1 Vincoli sul codice

La scrittura del codice deve osservare i vincoli riportati di seguito.

6.1.1 Leggibilità

Il codice deve essere facilmente leggibile. Variabili, costanti simboliche, enumerazioni, nomi di funzione, ... **devono avere nomi significativi** (non usate nomi come `b`, `pippo`, `pluto`, `fun1()` ...). Nel caso in cui il docente riterrà che nella scrittura del codice sia stata prestata poca attenzione a questo aspetto, il progetto verrà respinto e dovrà essere modificato in modo da risultare leggibile.

6.1.2 Compilazione

I programmi dovranno essere compilati senza warning (ed errori) usando il compilatore `gcc` con le seguenti opzioni:

```
-Wall -pedantic -std=c89.
```

Nel caso in cui uno studente non riesca a rimuovere un warning è invitato a contattare il docente con un messaggio di posta elettronica che contenga il segmento di codice (che causa il warning) e il messaggio di warning relativo.

Questo documento potrebbe essere modificato per includere un insieme di warning che verranno considerati irrilevanti ai fini della valutazione finale.

Il codice sorgente **deve** essere accompagnato da un `Makefile` in modo che possa essere compilato usando la utility `make`.

6.1.3 Gestione degli errori

Gli errori devono essere gestiti appropriatamente. Per ogni errore e situazione anomala, sia il client che il server devono stampare sullo standard error un messaggio significativo, utilizzando quando possibili anche la funzione `perror`.

6.1.4 Stringhe e memoria

Non devono essere utilizzate funzioni per la manipolazione di stringhe che *non* limitano il massimo numero di caratteri copiati. Ad esempio, la `strcpy()` deve essere evitata a favore della `strncpy()`, in cui è possibile specificare il massimo numero di caratteri copiati (per le motivazioni si consulti il manuale in linea).

Tutta la memoria allocata nello heap deve essere esplicitamente deallocata.

6.1.5 Concorrenza

Non devono essere usate funzioni di temporizzazione, come, ad esempio, `sleep()` e `alarm()` per risolvere problemi di *race condition* o *deadlock*. Le soluzioni implementate devono funzionare correttamente qualsiasi sia lo *scheduling* dei thread e dei processi coinvolti.

6.1.6 Formato del codice

Il codice deve seguire una convenzione di tabulazione chiara e coerente, includendo:

- una intestazione per ogni file che includa il nome e il cognome dell'autore/i, la versione e il nome del programma, una dichiarazione che il programma è, in ogni sua parte, opera dell'autore/i;
- un breve commento che spieghi il significato delle strutture dati e delle variabili globali;
- un breve commento per i punti critici o che potrebbero risultare di difficile lettura;
- un breve commento all'atto della dichiarazione delle variabili locali, che ne spieghi l'uso;
- un commento all'inizio di ogni funzione che ne specifichi in sintesi l'uso, l'algoritmo utilizzato (se significativo), il significato dei parametri, le variabili globali utilizzate, gli effetti laterali sui parametri passati per riferimento, ecc.

6.2 Documentazione

La documentazione del progetto consiste nei commenti al codice e in una breve relazione (massimo 10 pagine, formato A4, font 12px, margini ragionevoli, usare font Times o Arial) che descrive la struttura complessiva del lavoro svolto.

La relazione *deve rendere comprensibile il lavoro svolto ad un estraneo, senza che sia costretto a leggere il codice se non per chiarire dettagli implementativi.*

La relazione dovrà contenere:

- le principali scelte di progetto (strutture dati principali, algoritmi fondamentali, ecc.)
- la strutturazione del codice (logica della suddivisione in più file, librerie, ecc.)
- la struttura del server e del client, nonché i protocolli relativi alla loro interazione;
- le difficoltà incontrate e le soluzioni adottate;
- istruzioni per l'utente su come compilare ed eseguire ed utilizare il codice (per esempio, in quale directory deve essere attivato, quali sono le assunzioni fatte, ecc.). Queste istruzioni possono essere incluse nella relazione oppure in un file `README.txt` allegato al progetto.
- tutte le altre informazioni che si ritengono essenziali per la comprensione del progetto.

6.2.1 Struttura della relazione

Un possibile schema della relazione è il seguente:

1. Introduzione
   - 1.1 Strutture dati
   - 1.2 Principali algoritmi
2. Struttura del codice
   - 2.1 Suddivisione in file
   - 2.2 Codice condiviso tra client e server
   - 2.3 Librerie

3. Istruzioni
   - 3.1 Compilazione (breve commento sul makefile)
   - 3.2 Avvio del sistema
   - 3.3 Esecuzione (includendo anche la sintassi e la semantica dei comandi che un utente può inserire da tastiera per interagire con il sistema)
4. Protocollo di comunicazione
5. Server
   - 5.1 Descrizione dei vari thread
   - 5.2 Problemi di accesso concorrente ai dati
   - 5.3 Esempio di sessione (connessione di un client, ricerca dei file, ...)
6. Client
   - 6.1 Descrizione dei thread
   - 6.2 Problemi di accesso concorrente ai dati
   - 6.3 Esempio di sessione (connessione al server, ricerca e download di un file, gestione dell'upload, ...)
7. Conclusioni
   - 7.1 Difficoltà principali
   - 7.2 Soluzioni adottate
   - 7.3 Possibili miglioramenti

Lo schema è solo un suggerimento: gli studenti sono liberi di organizzare la relazione come preferiscono.

7 Formato e nomi dei file da consegnare

1. Il codice e la documentazione (⟵ si noti *codice* – cioè i file sorgente – e la *documentazione* cioè la relazione più eventuali altri file di documentazione il cui contenuto dovrà descritto nella relazione ⇒ **non devono essere consegnati i file binari e i file condivisi usati in fase di test**) del progetto dovranno essere inviati in un file `.zip` oppure `.tar` oppure `.tgz`. Qualsiasi altro formato non verrà accettato, in particolare, non verranno accettate eMail con i singoli file del progetto in attachment.
2. La relazione dovrà essere consegnata nel formato `PDF`. Qualsiasi altro formato non verrà accetatto.
3. I file dovranno avere nomi intelligibili *e* che possano far capire senza ambiguità il loro contenuto.
4. Si consiglia di archiviare i file in directory differenti. Ad esempio, il codice sorgente nella directory `src/`, la relazione nella directory `doc/`, ecc.

8 Frequently Asked Questions

Nessuna per ora.

9 Come contattare il docente

Per qualsiasi chiarimento su questo documento e, più in generale, sul progetto, inviare una eMail al seguente indirizzo:

`cardillo@di.unipi.it`,

usando come prefisso dell'oggetto del messaggio la stringa `[SOL]`.

Versioni - Modifiche

Data dell'ultima modifica: 30 maggio 2011.

v1.0 Prima versione, questo documento.