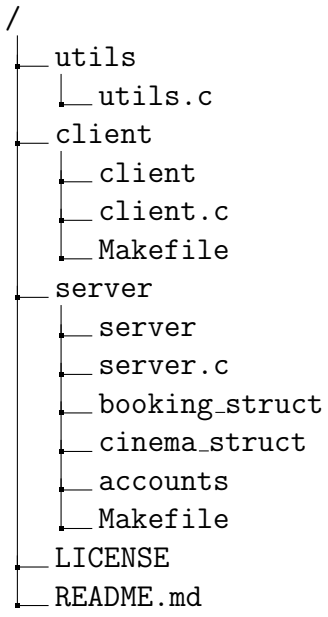
Progetto Sistemi Operativi - Cinema

Team Imola-Finocchi

October 21, 2022

Introduzione

Lo scopo del progetto è quello di gestire una sala cinematografica in modo tale da poter prenotare e/o disdire postazioni all'interno della stessa; il tutto viene gestito tramite un codice univoco generato randomicamente per l'utente, che avrà bisogno di accedere al suo account per poter eseguire operazioni.
Il progetto è stato realizzato per sistemi operativi UNIX, utilizzando quindi API Posix. Si compone di undici elementi suddivisi in 3 cartelle, come segue:



Il client, connettendosi al server, una volta che esegue il ***sign in*** (nel caso di utente già esistente) o ***sign up*** (nel caso di nuovo utente), può decidere se prenotare dei posti o disdire una precedente prenotazione. Nel primo caso, il client riceve sempre la mappa dei posti aggiornata e deciderà, in base a questa, gli eventuali posti da prenotare ottenendo un codice identificativo delle postazioni. Nel secondo caso, invece, l'utente potrà disdire una precedente prenotazione, tramite il codice della stessa.

Scelte realizzative

Si è deciso di suddividere il progetto in più sottocartelle, ognuna contenete dei file appositi per la corretta gestione dell'applicazione.

Utils

In `utils`, è presente un file header contenente librerie, macro e codice condiviso tra il client ed il server: si è utilizzato l'*include guard* per far si che non vengano ridefiniti simboli e tipi. Al suo interno si trova sia una funzione per gestire input numerici di tipo `long` che una per input di tipo `char *`, in maniera tale che qualunque inserimento non valido non venga accettato.

File di configurazione

Per il salvataggio dei dati, al posto di un database, si è deciso di utilizzare tre file, i quali verranno consultati ed aggiornati rispettivamente all'inizializzazione e allo shutdown del server.
Il primo è **cinema_struct**, il quale contiene una mappa delle postazioni prenotate all'interno della sala. Il formato è il seguente

$$n;\ m;\ x_{1,1}x_{1,2}\dots x_{1,m};\dots;x_{n,1}x_{n,2}\dots x_{n,m}$$

$$\begin{aligned} \text{n} &= \#\text{rows} \\ \text{m} &= \#\text{cols} \end{aligned}$$

$$\text{x}_{i,j} = \begin{cases} 0 & \text{se libero} \\ 1 & \text{se occupato} \end{cases}$$

il file verrà inizializzato al primo startup del server, momento nel quale il file ancora non esiste, con valori di righe e colonne richiesti al bootstrap e tutti 0 a indicare che il cinema non ha ancora posti prenotati.

Il secondo è **booking_struct**, nel quale ogni riga rappresenta un posto e nel caso in cui non sia vuota, questa contiene il codice della prenotazione durante la quale è stato occupato, altrimenti il carattere `\n` a significare la disponibilità del posto ad essere prenotato. Il formato è il seguente

$f(x_{1,1})$
$f(x_{1,2})$
$\dots$
$f(x_{1,m})$
$f(x_{2,1})$
$\dots$
$f(x_{2,m})$
$\dots$
$f(x_{n,m})$

$$f(x_{i,j}) = \begin{cases} \texttt{\textbackslash n} & \text{se il posto è libero} \\ g(x_{i,j})\texttt{\textbackslash n} & \text{se il posto è prenotato} \end{cases}$$

dove la funzione g è una funzione che genera il codice randomico della prenotazione contenente il posto $x_{i,j}$. Si osservi che se i posti $\underline{a}$ e $\underline{b}$ fanno parte della stessa prenotazione, allora $g(\underline{a}) = g(\underline{b})$.

Il terzo è **accounts**, nel quale ogni riga rappresenta un utente con le sue rispettive prenotazioni. In ordine, le informazioni, sono disposte nel seguente formato:

$\text{nickname}_1;\ \text{email}_1;\ \text{password}_1;\ g(a_{1,1});\ g(a_{1,2}); \dots ; g(a_{1,l_1})\texttt{\textbackslash n}$
$\text{nickname}_2;\ \text{email}_2;\ \text{password}_2;\ g(a_{2,1});\ g(a_{2,2}); \dots ; g(a_{2,l_2})\texttt{\textbackslash n}$
$\dots$
$\text{nickname}_2;\ \text{email}_2;\ \text{password}_2;\ g(a_{i,1});\ g(a_{i,2}); \dots ; g(a_{i,l_i})$

dove la funzione g è la funzione già definita, che genera il codice randomico della prenotazione.

Server

Inizializzazione Al lancio del server, vengono inizializzate tutte le variabili e le strutture dati utili alla comunicazione. Per prima cosa verranno definite le regole di comportamento da adottare alla cattura dei vari segnali: tutti quei segnali che possono rappresentare l'avvenimento di un malfunzionamento a tempo d'esecuzione verranno gestiti tramite un handler che eseguirà delle funzioni utili al corretto spegnimento del server, i rimanenti verranno ignorati.

L'handler, per prima cosa, sincronizzerà i file affinché contengano le informazioni aggiornate sulla mappa e le prenotazioni, successivamente proseguirà con la chiusura del descrittore del socket di connessione e il rilascio del semaforo utilizzato per la sincronizzazione tra i client. Affinché l'handler possa fare ciò, si sono dichiarate globali tutte le variabili utilizzate al suo interno, un flag `in_critical_section` specifico per ogni utente, che controlla se al momento dell'evento l'utente si trova all'interno della sezione critica per la cancellazione delle prenotazione; infine tre flag globali che controllano: `connected` se al momento dell'evento la comunicazione è ancora in atto (e quindi bisogna chiuderla), `in_booking_critical_section` se ci si trova all'interno della sezione critica per la prenotazione (perciò bisogna rilasciare il token del semaforo) e `in_signup_critical_section` se ci si trova all'interno della sezione critica per la creazione di un account (perciò bisogna rilasciare il token del semaforo).
Prima di inizializzare le strutture dati per il cinema, si esegue un check sulla presenza/assenza dei file di configurazione: l'assenza di uno solo di questi comporta un incongruenza all'interno del sistema, affrontata eliminando i file rimanenti: questa scelta è dovuta al fatto che così si semplifica la gestione dei dati, considerandoli idealmente come assenti. Successivamente vengono consultati i file per popolare le strutture dati, ovvero le aree di memoria mappate per la condivisione di informazioni tra i threads che eseguiranno le richieste da parte dei client e la lista di accounts; in caso di assenza dei suddetti file, quest'ultimi vengono creati con un contenuto di default, rappresentante il cinema vuoto.
A seguire vi è lo startup della connessione per la valorizzazione del *listening socket*.

Comunicazione Da questo punto in poi, tramite la **syscall** `accept`, si potrà proseguire con la connessione da parte dei client, instaurando un *connection socket* e creando un figlio che si preoccupi di gestire l'interazione.
In base alla richiesta del client, vengono svolte operazioni differenti

1. nel caso di **Sign IN**, aspetteremo dal client le informazioni utili per l'accesso al servizio quali: e-mail e password, sui quali verranno eseguiti dei controlli formali di correttezza lato client

2. in caso di **Sign UP**, oltre a tutto ciò che viene eseguito al *Sign IN*, il server riceverà anche lo username desiderato dall'utente

Una volta eseguito l'accesso, le scelte che un utente può fare sono le seguenti:

1. nel caso di **prenotazione**, verrà inviata al client la mappa del cinema e ci si aspetterà il numero dei posti da voler prenotare, seguito dall'insieme degli

identificativi delle sedie; una volta che la richiesta sarà correttamente terminata, verrà inviato al client un codice generato randomicamente dalla seguente funzione:

```
return ((int) rand() % (RAND_MAX - 1000000000) + 1000000000);
```

con lo scopo di avere un codice sempre da 10 cifre. Per accedere in maniera mutualmente esclusiva alle informazioni, si è inserita una sezione critica in modo tale che la mappa dei posti risulti al client sempre aggiornata. Inoltre, viene aggiornata anche la lista di accounts, aggiungendo all'utente che ha eseguito l'operazione, il codice della prenotazione appena eseguita.

2. in caso di **disdetta**, ci si aspetterà il codice della prenotazione da voler cancellare, con tutti i controlli del caso. Non è necessaria la semaforizzazione, grazie al fatto che non vi è presente alcuna sezione critica, poiché non è un problema se più client nello stesso istante disdicono delle prenotazioni. Ogni utente può disdire solamente le sue prenotazioni; però a causa della possibilità di poter eseguire il Sign IN sullo stesso account da più clients, si è inserita una sezione critica per far si che un client alla volta possa rimuovere le prenotazioni eseguite dall'account con cui si è loggato.

Chiusura Il server può terminare solamente tramite segnalazione di tutti quegli eventi che suggeriscono un'anomalia nell'esecuzione del server come l'accesso non consentito ad un area di memoria (SIGSEGV), la rottura di un socket (SIGPIPE), la chiusura della shell o l'arresto del computer (rispettivamente SIGHUP e SIGTERM) oppure più semplicemente tramite la sequenza di escape `CTRL+C`.

Client

Quando un cliente vuole accedere al servizio del cinema deve specificare da linea di comando porta e indirizzo IP del server, dei quali verrà controllata la correttezza al momento della lettura: per la porta è più semplice essendo che basta effettuare controlli sul valore di ritorno della `strtol`, oltre a quelli del range poiché il tipo di porta deve essere o ephemeral o non-privileged; differentemente, l'indirizzo IP verrà controllato tramite l'utilizzo della regex:

```
^(([0-9]|[1-9][0-9]|1[0-9]2|2[0-4][0-9]|25[0-5])\.)
    3([0-9]|[1-9][0-9]|1[0-9]2|2[0-4][0-9]|25[0-5])$
```

grazie a questa non solo il formato dell'indirizzo IP deve essere rispettato, ma ci si accerta anche che ognuno dei 4 interi di cui è composto rientri nel range $[0, 255]$. L'indirizzo IP non specificato fa reindirizzare in *localhost* (`127.0.0.1`), mentre la porta non specificata viene assegnata con la numero 4444.
In generale, qualunque altro input da parte dell'utente viene controllato affinché rispetti il formato richiesto e rientri nel range di valori accettato, altrimenti vengono presi accorgimenti; ad esempio per l'e-mail si è ricorso all'utilizzo di una regex del formato

```
^[A-Za-z0-9_\-\.]+@([A-Za-z0-9_-]+\.)+([A-Za-z0-9_-])2,4$
```

mentre per la password si sono eseguito controlli sulla lunghezza che, come per email e username, dovrà essere minore della macro di pre-processore `MAX_INPUT_SIZE=255` caratteri. Se anche solo uno dei due input non viene rispettato, vengono prese decisioni lato client sulla continuazione dell'utilizzo dell'applicazione. Questo vale anche per gli eventi verso i quali, seguendo la logica del server, viene adottato un comportamento consono a seconda della tipologia; ciò non si applica per i segnali:

- `SIGALRM`, che viene usato per gestire un timeout, scaduto il quale l'utente viene automaticamente disconnesso dal server (per evitare che un client blocchi completamente l'accesso al server)
- `SIGUSR1` utilizzato quando l'utente cambia idea al momento dell'inserimento della quantità di posti

Infatti, nel caso in cui l'utente voglia effettuare una prenotazione, riceve dal server la mappa dei posti della sala; quelli occupati verranno contrassegnati con una `X` colorata in rosso, così che possa scegliere quelli di suo gradimento tra i disponibili. Una volta presa la decisione, l'utente inserisce prima il numero delle postazioni da prenotare e

poi le postazioni effettive; sarà il client stesso a provvedere alla consistenza di questi input, ad esempio, si possono incontrare situazioni in cui l'utente voglia prenotare più posti di quelli disponibili, oppure voglia prenotare posti assegnati.

Manuale d'uso

Il progetto si presenta con 3 cartelle: `utils, server` e `client`:

- in `./utils` troviamo un file con estensione `.h` contenente librerie comuni tra i file sorgenti dell'applicazione. Inoltre sono presenti al suo interno costanti di pre-processore e metodi comuni.
- in `./server` sono contenuti: il *MakeFile*, il file sorgente del server, due file denominati `booking_struct` e `cinema_struct` che servono per registrare rispettivamente lo stato delle prenotazioni e lo stato della sala cinematografica; infine è presente il file eseguibile denominato `server`.
- in `./client` sono contenuti: il *MakeFile*, il file sorgente del client e il file eseguibile denominato `client`.

Lato server, per la compilazione, bisogna posizionarsi da terminale nella cartella contenente il file sorgente (e quindi anche il MakeFile) ed eseguire il comando `make`; per l'esecuzione bisogna lanciare l'eseguibile precedentemente creato, specificando l'eventuale parametro della porta con cui si vuole accettare connessioni, nel seguente formato:

```
./server [-p <PORT>]
```

Lato client, per la compilazione, bisogna posizionarsi da terminale nella cartella contenente il file sorgente (e quindi anche il MakeFile) ed eseguire il comando `make`; per l'esecuzione bisogna lanciare l'eseguibile precedentemente creato, specificando i parametri della porta e dell'indirizzo ip del server con cui ci si vuole connettere, nel seguente formato:

```
./client [-p <PORT>] [-a <SERVER_IP_ADDR>]
```

Per lo shutdown del server, si può proseguire utilizzando la sequenza di escape `CTRL + C`, o qualunque altro segnale inviabile tramite comando `kill` non ignorato. Riguardo al client, una volta eseguito, può terminare completando la prenotazione/ disdetta, tramite segnalazione, oppure al termine dei 180 secondi dalla scelta dell' operazione da eseguire.