Relazione progetto Sistemi Operativi

Semplice ticketing system con un modello client-server in C

Quinto Edoardo

Docente: Bilotta Stefano
Università degli studi di Firenze
Corso di laurea triennale in Informatica

Indice

1 Descrizione generale

Il progetto implementa un sistema per la gestione di ticket da parte di un Help Desk.
L'impostazione per gestire la comunicazione è del tipo **AF_INET** (ossia con i *socket* del dominio internet) e si suddivide in una parte dedicata al **server** ed una al **client**.
Il tipo più importante del progetto è chiaramente la struttura ticket che ha i seguenti campi (nel file *server.c* hanno dei nomi leggermente differenti):

- **Id**: permette di identificare in modo univoco un ticket, è un intero crescente da 1 a 100 (che rappresenta il massimo numero di tickets).
- **Titolo**: il titolo del ticket, è un array di al massimo 100 caratteri.
- **Descrizione**: la descrizione del ticket, è un array di al massimo 500 caratteri.
- **Data**: la data di creazione di un ticket, è un array di massimo 20 caratteri. Viene usata la libreria *time.h* per permettere di ottenere la data corrente al momento della creazione di un ticket.
- **Priorità**: la priorità del ticket. Questa è specificata attraverso un *enum* che permette di inserire i seguenti valori:
  - BASSA.
  - MEDIA.
  - ALTA.
  - CRITICA.
  - SENZA PRIORITÀ (quella di default alla creazione).

  Solo un agente autenticato può cambiare il valore di questo campo.
- **Stato**: lo stato del ticket. È specificato attraverso un *enum* che permette di inserire i seguenti valori:
  - APERTO.
  - IN CORSO.
  - CHIUSO.

  Solo un agente autenticato può cambiare il valore di questo campo.
- **Agente**: il nome dell'agente assegnato al ticket. Solo un agente autenticato può cambiare il valore di questo campo.
- **Cliente**: la persona che ha creato il ticket.

1.1 Funzionalità

Le funzionalità offerte a tutti gli utenti sono le seguenti:

- **Effettuare il login**. Ci sono tre categorie di utenti in totale, quella di default ossia *Guest*, *utenti regolari* ed infine *agenti di supporto* (che hanno il massimo livello di permessi). Il login è chiaramente semplificato, in quanto esistono solo quattro combinazioni di *username* e *password* valide:
  - Utente regolare: cliente1 password1.
  - Utente regolare: cliente2 password2.
  - Agente di supporto: agente1 password1.
  - Agente di supporto: agente2 password2.

  Per semplicità queste combinazioni sono salvate in chiaro all'interno di una struttura dati nel file *server.c*.
- **Creare un nuovo ticket**. Permette al client di creare un nuovo ticket inserendo il titolo e la descrizione. Nel caso in cui la descrizione sia *NULL* verrà usato il titolo anche per essa.
- **Visualizzare i propri ticket**. Permette al client di visualizzare solo i propri ticket. Se, per esempio si è autenticati come *cliente1* non si potranno vedere i ticket creati da dei *guest* e viceversa.
- **Cercare un ticket per ID**. Permette di cercare fra i propri ticket uno con l'id corrispondente a quello inserito in input.

Solo dopo essersi autenticati come agenti si può accedere ai seguenti comandi:

- **Assegnare un ticket ad un agente**. Dato l'id del ticket ed il nome dell'agente (non necessariamente agente1 o agente2) assegna il ticket specificato all'agente inserito.
- **Modificare la priorità di un ticket**. Dato l'id del ticket e la nuova priorità modifica il ticket corrispondente.
- **Modificare lo stato di un ticket**. Dato l'id del ticket e il nuovo stato modifica il ticket corrispondente.
- **Eliminare ticket**. Dato l'id del ticket lo elimina.
- **Visualizzare tutti i ticket**. Permette di vedere tutti i ticket presenti nel sistema.

1.2 Struttura del progetto

Il progetto ha una struttura piuttosto semplice:

- Una cartella *src* contenente:
  - *server.c* è l'implementazione del server.
  - *client.c* è l'implementazione del client.
- *Makefile*, per la compilazione e le operazioni di pulizia dopo l'esecuzione
- *tickets.txt*, file per la persistenza dei ticket, ogni operazione manipola questo file in modo da mantenere aggiornati più client concorrenti. Al suo interno vengono salvati i ticket con il seguente formato: ID|Titolo|Descrizione|Data Creazione|Priorità|Stato|Agente Assegnato|Cliente. Ogni client continua comunque ad avere un vettore personale (gestito dal server) dove vengono mantenuti i propri ticket durante l'esecuzione del processo.

1.3 Funzionamento tipico

In generale il sistema funziona nel seguente modo:

1. Si lancia il server, che rimane in ascolto sul **socket** 127.0.0.1 : 8080, in attesa di ricevere qualche comando.
2. Da altri terminali (lasciando in esecuzione il server) si lancia il client.
3. Il client si conette al server in ascolto.
4. Il server crea un processo figlio con una *fork()* per gestire le richieste del client.
5. Il client fa una serie di richieste al server.
6. Si chiude il client.
7. Si chiude il server.

I tickets manipolati e creati rimangono comunque disponibili nel file *tickets.txt*.
Per implementare le funzionalità è stato seguito uno schema nel quale:

- Il *client* si occupa solo di accettare le richieste dell'utente via terminale e di inviarle al server. Chiaramente il client deve anche occuparsi di in qualche modo *tokenizzare* (ossia tradurre in una modalità comprensibile per il server) le richieste. Infatti il server accetta comandi solo in questa forma (sono stringhe):
  - `login <username> <password>`. Per effettuare il login.
  - `crea <titolo>\n<descrizione>`. Per creare un nuovo ticket.

- `lista`. Per visualizzare i propri ticket.
- `cerca <Id>`. Per cercare un ticket dato l'id.
- `listatutti`. Per visualizzare tutti i ticket (solo agenti).
- `assegna <Id> <agente>`. Per assegnare al ticket indicato un agente (solo agenti).
- `stato <Id> <stato>`. Per modificare lo stato del ticket indicato (solo agenti). Lo stato viene inviato come un numero intero e successivamente decodificato dal server in stringa (passando dall'enum precedentemente presentata).
- `priorita <Id> <priorita>`. Per modificare la priorità del ticket indicato (solo agenti). La priorità viene inviata come un numero intero e successivamente decodificato dal server in stringa (passando dall'enum precedentemente presentata).
- `elimina <Id>`. Per eliminare il ticket indicato (solo agenti).

Una volta accettata la richiesta dal terminale essa viene inviata al server attraverso una procedura `void invia_comando(...)` che richiama la *system call send()* e controlla se il server risponde correttamente.

- Il **server** si mette in ascolto ed attende connessioni da parte di client. Quando ne arriva una crea un processo figlio per gestirla che invoca la procedura `void gestisci_client(int client_socket)`, che essenzialmente riceve dal client il comando (nel formato di sopra) e a seconda della richiesta richiama le procedure appropriate. Per la traduzione viene fatto uso delle funzioni:
  - *strcmp()* per controllare il comando iniziale (per esempio *login*).
  - *strtok()* che permette di dividere una stringa in base a dei caratteri divisori (in questo caso lo spazio vuoto e $\backslash n$). Questo ci permette di leggere i parametri passati dal client (per esempio $<$*username*$>$ e $<$*password*$>$).

2 Istruzioni di compilazione

La compilazione è completamente gestita dal *Makefile*. Chiaramente sarà necessario avere installato un compilatore per il linguaggio C (*gcc* o *cc*) ed il comando *make*. Una volta arrivati alla cartella dove si è salvato il progetto basta digitare il comando *make* e verranno compilati i due sorgenti nella cartella *src*.
Per avviare il server si usa *./server* (nella stessa cartella dove si trova il *Makefile*), poi in altri terminali si possono avviare fino a 10 client con il comando *./client*.
Un tipico utilizzo è il seguente:

1. `$ cd path/al/progetto`.
2. `$ make`.
3. `$ ./server`.
4. In altri terminali (nella stessa cartella di prima): `$ ./client`.

Una volta terminato di utilizzare il sistema il *Makefile* offre due comandi di utilità:

- `$ make clean`. Rimuove i file .o creati dalla compilazione.
- `$ make delete_all_tickets`. Rimuove tutti i tickets salvati nel file *tickets.txt*.

3 Gestione degli errori

I programmi sviluppati gestiscono gli errori principali usando:

- *perror* quando necessario (ossia in situazioni critiche) e richiamando successivamente una *exit()* per uscire dal programma. Le situazioni critiche gestite in questa maniera sono:
  - Errori nella gestione del file *tickets.txt*.
  - Errori nella gestione della connessione *client-server*.
- Un semplice meccanismo di ridigitazione nel caso di input erroneo.

In generale, comunque, gli errori critici sono trattati come situazioni non ripristinabili, in modo simile a quello che succede nei linguaggi orientati agli oggetti con il meccanismo delle eccezioni.