Server multi-threaded per prenotazioni in spiaggia

Progetto del corso di Laboratorio Sistemi Operativi a.a. 2017/2018

1 Introduzione

Lo scopo di questo progetto è creare un sistema client server per la gestione delle occupazioni dei lettini di una spiaggia, in grado di fornire informazioni sulla disponibilitàdegli ombrelloni, e impegnarli o disdirli. L'architettura è di tipo client server ed allo studente è chiesto di creare il server ed il client. L'obiettivo principale del progetto è di familiarizzare lo studente con socket e programmazione concorrente, oltre che con architetture client/server.

La valutazione del progetto avviene durante una relazione orale che lo studente svolge all'esame, in cui gli verrà chiesto di presentare il suo codice, i suoi risultati e di essere in grado di discuterli criticamente. Può anche succedere che, durante la discussione, vengano proposte modifiche. In tal caso lo studente deve dimostrare, nei limiti del ragionevole, di essere in grado di applicare tali modifiche durante l'esame.

2 Specifiche del server

Il server è strutturato come segue:

- un thread master, che all'avvio legge un file di configurazione in cui sono descritti gli ombrelloni disponibili ed il loro stato. Quando il thread master viene chiuso (anche tramite segnali di tipo SIGTERM, SIGINT, SIGQUIT) è responsabilità del thread master stesso garantire che i dati sul disco siano sempre aggiornati all'ultima operazione avvenuta in memoria. Ci sono chiaramente diversi approcci per raggiungere questa consistenza: se ne scelga uno e lo si implementi. Si sia pronti a discutere anche gli altri modi
- alcuni thread worker: questi thread sono responsabili di:
  - accettare il socket ottenuto dalla accept() e continuare la comunicazione con il client
  - decodificare il messaggio ricevuto dal client
  - verificare quello che il client chiede
  - dare risposta al client e contestualmente modificare le disponsabilità

Evidentemente questa non è l'unica architettura possibile. Se si decide di implementare una diversa architettura, si motivi durante la relazione orale, il perchè. Si consideri per esempio la possibilità di usare un thread pool: https://github.com/Pithikos/C-Thread-Pool

3 Gestione dei dati

L'architettura proposta favorisce chiaramente l'uso di un database per la gestione dei dati. Per questo progetto comunque, preferiremo non usare un database "vero" per non appesantire la scrittura del codice. Invece, gestiremo le informazioni necessarie al server in modo più semplice, usando un file.

Mentre non viene data alcuna specifica per il formato in cui scrivere sul disco le informazioni, esse devono essere:

- numero ombrellone
- stato (libero, occupato, temporaneamente prenotato)
- data di "scadenza" (cioe' quando si libera, se è occupato)

Il consiglio è di tenere il formato semplice.

Alternativamente si può pensare di usare un sistema di journaling per la gestione dei dati. Si veda https://it.wikipedia.org/wiki/Journaling e si chieda a lezione se si è interessati a maggiori dettagli. Idealmente si potrebbe scrivere un log con tutti i cambiamenti da applicare al file e ad intervalli regolari di tempo applicare i cambiamenti. Il vantaggio è che nel caso di incosistenza dello stato del file, basta riapplicare i cambiamenti che sono segnati nel log (questa tecnica per esempio è usata nei file system journaled, ma nulla vieta di applicarla ad altri casi).

4 Protocollo di comunicazione client-server

Il client comunica con il server nel seguente modo, una volta stabilita la connessione al socket:

- il client chiede di poter occupare un ombrellone: trasmette "BOOK"
- il server risponde con "OK" nel caso in cui sia pronto per la prenotazione oppure con "NOK" nel caso in cui non lo sia. Nel secondo caso la comunicazione è terminata. Se non ci sono ombrelloni disponibili, il server risponde con "NAVAILABLE"
- il client procede a richiedere un ombrellone specifico: "BOOK $OMBRELLONE" dove $OMBRELLONE è un indentificativo valido per un ombrellone (dipende sostanzialmente da come vengono identificati gli ombrelloni nel server che si è creato)
- il server verifica la disponibilità. Se l'ombrellone è disponibile il server cambia lo stato dell'ombrellone in "temporaneamente prenotato" e comunica la disponibilità al client: "AVAILABLE". Se non è disponibile, il server risponde con "NAVAILABLE" e chiude la connessione
- quando il client riceve "AVAILABLE" decide se vuole confermare la prenotazione. In tal caso risponde con "BOOK $OMBRELLONE DATE" dove DATE è la data di scadenza della prenotazione. Se invece non vuole confermare la prenotazione trasmette "CANCEL"
- nel caso in cui il client desideri prenotare un ombrellone nel futuro (attenzione alle date), può trasmettere "BOOK $OMBRELLONE DATESTART DATEEND". Il server verifica la disponibilità e risponde nel modo convenzionale indicato poco sopra

Oltre a questo flusso di comunicazione esiste anche la possibilità di mandare altri messaggi al server, per il client:

- il client può inviare una richiesta di tipo “AVAILABLE” per sapere se ci sono ombrelloni disponibili, e quanti sono. Il server risponde “NAVAILABLE” nel caso in cui non ci siano ombrelloni disponibili e con “AVAILABLE $NUMERO” dove $NUMERO è il numero di ombrelloni disponibili
- *opzionale – vale 1 punto in più* il client può inviare una richiesta del tipo “AVAILABLE 1” per richiedere tutti gli ombrelloni disponibili nella prima fila. Ovviamente questo significa che nel formato in cui queste informazioni sono salvate nel file questa informazione deve essere registrata in qualche modo
- il client può chiedere di cancellare una prenotazione inviando “CANCEL $OMBRELLONE”. Nel qual caso il server risponde con “CANCEL OK” e ripone l’ombrellone $OMBRELLONE in stato disponibile

5 Specifiche del client

Si scriva un piccolo client che esegua la connessione e trasmetta un singolo comando, da accettarsi alla linea di comando.
Per chiarezza: il client deve essere chiamato a linea di comando con un argomento che sia l’argomento da trasmettere al server. Da quel punto si gestisca il flusso di comunicazione in modo interattivo.
Volendo è anche possibile usare telnet come client, collegandosi alla porta del server direttamente. Nel qual caso si veda la seguente sezione a riguardo dell’uso di expect.

6 Testing

Si può automatizzare il testing del server e del client usando un tool di scripting chiamato expect: http://expect.sourceforge.net/

Expect permette di usare uno linguaggio di scripting (derivato da tcl) per interagire con eseguibili che sono interattivi, come il client descritto nella sezione precedente.