UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Docente : Marco Faella
Numero Gruppo : 5
Componenti : Annarita Della Rocca N86002186

Quiz Game

1 Guida alla compilazione e all'uso

1.1 Applicativo Server

L'applicativo server è composto dai seguenti file contenuti nella cartella "Server":

- server.c
- serverUtility.h
- serverUtility.c
- list.h
- list.c

Per poter ottenere un corretto funzionamento del server è opportuno utilizzare un file di testo che deve essere necessariamente chiamato "questions.txt". Tale file contiene le domande che verranno poste agli utenti durante il gioco.

Al fine di stampare correttamente le domande il file di testo deve essere formattato nel seguente modo:

Domanda
Risposte Multiple
~RispostaEsatta

Il file non deve contenere spazi aggiuntivi dopo l'ultima domanda.

```
Quale tra i seguenti non è un Jedi durante la trilogia originale di Star Wars?
1) Anakin Skywalker
2) Yoda
3) Palpatine
4) Kylo Ren
~2
Chi ha vinto il premio di miglior regista agli Oscar 2020?
1) Quentin Tarantino
2) Sam Mendes
3) Martin Scorsese
4) Bong Joon ho
~3
Qual e' la mascotte di Nintendo?
1) Mario
2) Sonic
3) Pacman
4) Crash
~1
```

Per la compilazione del server procediamo in questo modo:

```
Gcc –o server server.c serverUtility.c list.c -pthread
```

Una volta compilato procediamo ad eseguire:

```
./server
```

Nell'eventualità in cui dovrebbero verificarsi degli errori che porterebbero alla chiusura del server, il server procederà a stampare l'errore che ha causato la sua terminazione

1.1 Applicativo Client

L'applicativo client è composto dai seguenti file nella cartella "client":

- client.c
- clientUtility.h
- clientUtility.c

Per compilarlo si procede in questo modo:

```
gcc –o client client.c clientUtility.c
```

Una volta compilato, per poter avviare un client è necessario specificare due parametri in ingresso:

Si procede quindi in questo modo:

```
./client  ipServer  portaServer
```

- **ipServer**: ip del server a cui ci si vuole connettere
- **portaServer**: porta del server a cui ci si vuole connettere, deve essere composta da 4 numeri.

Di seguito vediamo un esempio:

./client 192.168.10.100 49154

2. Protocolli di comunicazione dell'applicativo

2.1 Protocollo client->server

Una volta avviato l'applicativo, al client verrà richiesto di inserire un intero nel momento in cui è pronto a giocare

```
**********************************************************************************************
Benvenuto in questo gioco, ti verranno poste una serie di domande a risposta multipla
- per ogni risposta corretta avrai +1 punto
- per ogni risposta errata/non data ti verrà decurtato 1 punto
Il gioco avrà inizio quando ci saranno almeno due sfidanti connessi
**********************************************************************************************
se sei pronto digita un numero e premi invio
```

Il client invia al server l'intero mediante una `write` sul `socket`

Una volta inserito l'intero ci sono due scenari che possono verificarsi:

- C'è un solo client connesso: in tal caso l'applicativo rimarrà in attesa di almeno un altro giocatore
- Ci sono più client connessi: il gioco può iniziare e un n+1esimo client può aggiungersi in ogni momento.

N.B. al client n+1esimo verranno poste le domande dalla prima all'ultima rispettando sempre il limite di tempo.

Esempio di client in attesa:

```
**********************************************************************************************
Benvenuto in questo gioco, ti verranno poste una serie di domande a risposta multipla
- per ogni risposta corretta avrai +1 punto
- per ogni risposta errata/non data ti verrà decurtato 1 punto
Il gioco avrà inizio quando ci saranno almeno due sfidanti connessi
**********************************************************************************************
se sei pronto digita un numero e premi invio
35
```

Dopo aver inviato al server l'intero richiesto al fine di iniziare il gioco, il client, i riceve dal server, mediante una `read`, prima la lunghezza della domanda da stampare all'utente e poi la domanda stessa.

Procede a stampare, mediante una `write` su standard output, la domanda che ha ricevuto.

Dopo aver stampato la domanda, stampa la classifica:

- con una `read` riceve dal server il numero di giocatori

- in un ciclo `for`, che cicla per numero di giocatori, riceve lunghezza dell'ip del giocatore, l'ip stesso e il punteggio associato a tale ip

Tramite una `sprintf` e una `write` il client stampa all'utente la classifica già ordinata. N.B. l'ordinamento è gestito nel server.

La ricezione della risposta utente prevede un tempo massimo, a disposizione dell'utente, per digitare la propria risposta.

L'utente ha un minuto a disposizione.

- Se risponde correttamente il suo punteggio verrà incrementato di un punto.
- Se non risponde correttamente, inserisce un numero che non compare tra le risposte possibili, o non risponde, il suo punteggio verrà decrementato di un punto.

La risposta dell'utente con timeout è stata gestita in un'apposita funzione nominata "`getAnswerWithTimeout`".

Nel caso in cui l'utente non risponda entro il tempo limite stabilito, l'applicativo, mediante una `write` su `standard output` stampa un messaggio d'errore e decurta un punto.

2.2 Protocollo server->client

Quando il gioco inizia, il server riceve l'intero inserito dall'utente che è pronto ad iniziare, incrementa la variabile globale "`readyPlayers`".

Il server controlla, attraverso la funzione "`gameCannotStart`", che il numero dei giocatori pronti sia almeno 2.

- Se il numero è minore di due, il `thread` viene messo in `wait`.

- Se il numero è maggiore o uguale a due, si procede a schedulare la prima sveglia ed ad impostare il timeout sul `socket` attraverso la funzione "`setTimeoutOnSocket`". In seguito si procede con l'apertura del file "`questions.txt`" formattato come detto in precedenza.

La lettura del file avviene in questo modo: tramite una struttura iterativa (`do-while`) il server legge tutto il contenuto del file, quando il file legge il carattere speciale "~" procede a salvare in una variabile ("`correctAns`") il contenuto del file che segue immediatamente il carattere "~". Tale variabile conterrà quindi la risposta corretta alla domanda corrente.

Il server a questo punto invia al client i seguenti dati:

- il numero di caratteri letti dal file, e cioè la lunghezza della domanda sommata a quella delle risposte multiple
- i caratteri letti

Essi vengono inviate con delle `write` sul `socket`.

Viene poi ordinata la classifica attuale e inviata al client secondo il seguente schema:

- viene inviata la dimensione della lista
- in un ciclo while vengono inviati:
  - lunghezza dell'ip corrente
  - indirizzo ip
  - punteggio associato all'indirizzo ip

Il server riceve dal client la risposta dell'utente, la confronta con la risposta corretta, memorizzata nella variabile locale "`correctAns`" e procede ad assegnare il punteggio corretto.

3. Dettagli implementativi

3.1 Gestione della classifica mediante liste thread-safe

La classifica è memorizzata in una lista.
Le operazioni effettuate sulla lista da più thread possono generare race condition, per questo la lista è stata implementata come segue:

```
typedef struct Node Node;
struct Node {
    char ip[17];
    int points;
    Node *next;
};

typedef struct List {
    Node *head;
    int dim;
    pthread_mutex_t *mutex;
} List;
```

Essa al suo interno contiene il mutex che viene bloccato all'inizio di ogni operazione e sbloccato opportunamente alla terminazione di essa. Questo facilita la gestione della lista da parte delle funzioni che la utilizzano.

La lista contiene quindi i giocatori e il loro punteggio che viene assegnato in base alla risposta data nella funzione "`setPlayerScore`". La classifica viene quindi ordinata nella funzione "sortLeaderboard" e poi inviata al client tramite la "printLeaderboard"

Esempio di funzione thread safe che utilizza la lista:

```
/**
 * Calcola il punteggio del giocatore in base alla risposta data
 * @param list: lista che contiene le informazioni associate al
giocatore
 * @param clientIp: ip del client corrente
 * @param increment: flag che indica se aggiungere o decurtare
punti. Se param è 1 allora verrà
 * aggiunto +1 al punteggio del giocatore, viceversa se è 0 verrà
decurtato 1 punto
 */
void setPlayerScore(List* list, char* clientIp, bool increment){
    pthread_mutex_lock(list->mutex);
    Node* head = list->head;
    while(head != NULL) {
        if(strcmp(head->ip, clientIp) == 0) {
            if(increment) {
                head->points++;
            } else{
                head->points--;
            }
            pthread_mutex_unlock(list->mutex);
            return;
        }
        head = head->next;
    }
    pthread_mutex_unlock(list->mutex);
}
```

3.2 Gestione del timeout

3.2.1 Lato server

Il timeout è gestito nella funzione "setTimeoutOnSocket"

```
/**
 * Setta il timeout, cioè il tempo il client ha a disposizione
 * @param fd è il file descriptor
 * @param timeOutInSeconds tempo espresso in secondi
 */
void setTimeoutOnSocket(int fd, int timeOutInSeconds) {
    struct timeval tv;
    tv.tv_sec = timeOutInSeconds;
    tv.tv_usec = 0;
    setsockopt(fd, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof
tv);
}
```

- **struct timeval**

È una struttura contenuta nell'header <sys/time.h> che contiene i seguenti campi:

- tv_sec: è il campo che contiene il numero di secondi
- tv_usec: è il campo che contiene una porzione di secondo

Si noti che il numero di secondi in questo caso è dato dal parametro in input "timeOutInSeconds".
In questo caso è 0 poiché il server aspetta 0 secondi prima di leggere la risposta dell'utente.
Al fine di poter stampare una nuova domanda ogni 60 secondi, il server fa uso del segnale SIGALARM nella funzione "signalHandler".

```
/**
 * Handler di SIGALRM, sveglia i thread in attesa e imposta la prossima
sveglia
 * */
void signalHandler(int receivedSignal){
    if(receivedSignal == SIGALRM) {
        setAllThreadsToWakeup();
        pthread_cond_broadcast(&responsePlayerCondition);
        alarm(ALARM_TIME);
    }
}
```

- **setsockpt**

Funzione contenuta nell'header <sys/socket.h>.
Setta l'opzione specificata dall'argomento "SO_RCVTIMEO" al valore puntato dalla "struct timeval", al livello di protocollo specificato dall'argomento "SOL_SOCKET".
SOL_SOCKET permette di settore l'opzione sul socket stesso.

3.2.2 Lato client

Il timeout è gestito nella funzione "getAnswerWithTimeout"

```
/**
 * Riceve la risposta dell'utente entro un limite di tempo
 * ritorna la risposta dell'utente, -1 altrimenti
 */
int getAnswerWithTimeout() {
//Inizializzo le proprietà del file descriptor
    fd_set read_fds, write_fds, except_fds;
    FD_ZERO(&read_fds);
    FD_ZERO(&write_fds);
    FD_ZERO(&except_fds);
    FD_SET(STDIN_FILENO, &read_fds);

    struct timeval timeout;
    timeout.tv_sec = MAX_TIME;
    timeout.tv_usec = 0;
    int givenAns = -1;

//Aspetto l'input dell'utente o il timeout; il primo parametro è
// uno in più del più grande file descriptor
    if (select(STDIN_FILENO + 1, &read_fds, &write_fds, &except_fds, &timeout)
== 1) {
        char buff[256] = { '\0' };
        int length = read(STDIN_FILENO, buff, 4);
        buff[length - 1] = '\0'; // removes the '\n'
        if(strIsNumber(buff)) {
            givenAns = atoi(buff);
        } else {
            write(STDOUT_FILENO, "La risposta inserita non e' un numero, ti
verrà decurtato un punto\n", 68);
        }
    }
    else {
        write(STDOUT_FILENO, "\nATTENZIONE: Il tempo a disposizione per
rispondere e' scaduto, ti verrà decurtato un punto\n\b", 94);
    }
    return givenAns;
}
```

- **Funzione select**

La funzione select permette di controllare contemporaneamente lo stato dei descrittori degli insiemi specificati, opportunamente inizializzati.

Per l'inizializzazione dei descrittori sono state utilizzate le macro:

- FD_ZERO: inizializza l'insieme di descrittori di set con l'insieme vuoto
- FD_SET: aggiunge fd all'insieme di descrittori set, mantenendo a 0 il via relativo a fd.

Il primo parametro in ingresso, numfds, è il numero massimo di descrittori controllati dalla funzione select.

Dato maxd il massimo descrittore usato, numfds = maxd+1

I suoi parametri sono quindi dei descrittori.

In particolare `read_fds`, `write_fds` e `except_fds` sono puntatori a variabili di tipo fd_set.

fd_set è il tipo di dato che rappresenta l'insieme dei descrittori.

- `read_fds`: si occupa delle operazioni di lettura
- `write_fds`: si occupa delle operazioni di scrittura
- `except_fds`: verifica l'esistenza di eccezioni

In particolare modo il parametro "timeout" specifica il valore che la funzione `select` attenderà per individuare un descrittore pronto.

Si noti il campo "timeout.tv_sec" della "`struct timeval timeout`" è stato inizializzato a `MAX_TIME`. Come richiesto da traccia `MAX_TIME` è 60 secondi, per modificare la durata del timeout è necessario modificare solo tale valore.

La funzione in seguito legge la risposta dell'utente da standard input, controlla che l'utente abbia inserito effettivamente un intero per poi convertire il contenuto del buffer in un int.