Relazione progetto Paroliere

Carlo Tarabbo [654342] - Appello Estivo Laboratorio II

File e librerie

Il progetto e' diviso in due main files `server.c` e `main.c`, che rispettivamente contengono il codice di gestione del server e dei client, e vari file di librerie implementate per la gestione della game logic e della comunicazione server/client tramite socket.

Il progetto e' quindi diviso nel seguente modo:

- main files : `server.c` e `client.c`
- game logic : `matrix.h` e `matrix.c`
- server/client communication : `paroliere.h` e `paroliere.c`

1. Game Logic - Char Matrix e controllo di parole

➡️ File di riferimento : matrix.h e matrix.c

Per prima cosa da avere nel gioco del paroliere e' una matrice di caratteri (in questo caso 4×4) da cui poter comporre le parole. La matrice puo' venire generata randomicamente usando un seed usando la funzione `generate_letters` o caricata volta per volta da un file usando `load_matrix_fromfile`, che scorre riga per riga il file contenente le matrice.

Il carattere speciale
qu viene salvato nella matrice come il singolo carattere q. Nel momento in cui la matrice deve essere stampata viene aggiunta la u. Nel controllo di una parola nella matrice viene eliminata la u nel caso sia preceduta da una q.

1.1 Ricerca di una parola nella matrice

Per poter attribuire punti ad una parola e' necessario controllare se la parola é contenuta all'interno della matrice e del dizionario. Per il controllo della parola all'interno della matrice vengono usate le funzioni `adjacentSearch` e `isinmatrix`.

`adjacentSearch`

Questa funzione è quella che effettua la ricerca ricorsiva della stringa all'interno della matrice.

1. **Base Case**:

- Se il carattere da cercare (`find[index]`) è il carattere di fine stringa (`\0`), significa che la stringa è stata trovata completamente. La funzione ritorna `true`.

2. **Controllo dei limiti**:

- Vengono controllate le coordinate (`i`, `j`) per evitare di accedere ad elementi fuori dalla matrice. Inoltre, si verifica se il carattere nella matrice (`matrix[i][j]`) corrisponde al carattere da cercare (`find[index]`).

3. **Marcatura e ricerca ricorsiva**:

- Per evitare di controllare lo stesso elemento più volte, si segna temporaneamente l'elemento della matrice con un asterisco (`'*'`).
- La funzione chiama se stessa ricorsivamente per cercare il carattere successivo della stringa in 4 direzioni: sopra (`i + 1`, `j`), sotto (`i - 1`, `j`), sinistra (`i`, `j - 1`), destra (`i`, `j + 1`). In ogni chiamata ricorsiva vengono incrementati sia l'indice del carattere da cercare (`index + 1`) che indica il prossimo carattere nella stringa, sia le coordinate per spostarsi all'elemento successivo nella direzione scelta.

4. **Ripristino e return**:

- Dopo le chiamate ricorsive, si ripristina il valore originale dell'elemento della matrice (`matrix[i][j]`) con il carattere che era presente prima.
- Infine, la funzione ritorna il risultato ottenuto dalle chiamate ricorsive (`found`).

`isinmatrix`

Questa funzione controlla se una stringa (`find`) è presente all'interno di una matrice di caratteri (`matrix`).

1. **Rimozione della 'u' precedente alla 'q'**:

- La funzione presuppone che la stringa da cercare non possa contenere la lettera "q" senza la successiva "u".
- Si scorre la stringa da cercare (`find`) e si copia ogni carattere in `newstring` tranne la "u" che precede una "q".

2. **Controllo preliminare**:

- Viene verificata la lunghezza della stringa (`len`). Se è maggiore di 16 (4 righe * 4 colonne), la funzione ritorna `false` indicando che la stringa non può stare nella matrice.

3. **Ricerca nella matrice**:

- Si effettua una doppia iterazione su ogni elemento della matrice.
- Per ogni elemento si chiama la funzione `adjacentSearch` passando la stringa modificata (`newstring`), le coordinate dell'elemento (`i`, `j`) e l'indice del carattere da cercare (`index`) che inizialmente è 0.
- Se `adjacentSearch` ritorna `true`, significa che la stringa è stata trovata nella matrice e la funzione `isinmatrix` ritorna `true`.

1.2 Ricerca di una parole nel dizionario - le Trie

Per controllare se una parola e' all'interno del dizionario fornito usa la data structure chiamata Trie. Una **trie** è una struttura dati ad albero utilizzata per memorizzare e cercare stringhe o sequenze di caratteri in modo efficiente.

Come funziona una trie:

- Ogni nodo nell'albero rappresenta un carattere nella stringa. Se il nodo e' flaggato come terminale, vuol dire che il cammino dalla radice a quel nodo e' una parola nel dizionario
- I rami che collegano i nodi rappresentano le possibili scelte di caratteri successivi.
- La radice dell'albero non ha un carattere associato.

```
typedef struct trienode {
  struct trienode *children[NUM_CHAR];
  bool terminal ;
} trienode ;
```

- Una stringa viene memorizzata inserendo i suoi caratteri uno alla volta, creando un nuovo nodo per ogni carattere non ancora presente nel percorso.

Carico quindi il dizionario in una trie usando `trieinsert` su ogni parola contenuta nel dizionario e la funzione `isintrie` per controllare se la parola data rappresenta nella trie un path dalla radice a un nodo *terminale*.

Sono poi contenute funzioni di utility per il convertire la matrice in una stringa di caratteri per poterla spedire come messaggio `matrix_to_char`, una funzione di stampa per la trie e matrice

---

2. Comunicazione server/client e gestione dei giocatori.

➡️ File di riferimento paroliere.h e paroliere.c

2.1 struct messaggio, write e read di messaggi su una socket

Per implementare il protocollo di comunicazione richiesto viene usato il type `messaggio` per definire il pacchetto di informazioni da scrivere o leggere tramite socket.

```
typedef struct messaggio {
    char type ;
    unsigned int length ; //0 nel caso in cui il campo dat
    char * data ; //dati effettivi
} messaggio ;
```

Le due funzioni `write_message` e `read_message` implementano la comunicazione e, rispettivamente, creano il messaggio a partire dai suoi dati o leggendolo dalla socket.

Ci sono poi due versioni "silenziose"di queste due funzioni per il client, con l'unica differenza di non avere prints

2.1.1 `void write_message(int socket_fd , char type , char * data)`

1. **Calcolo della lunghezza del messaggio:**
   - Viene dichiarata una variabile `length` per memorizzare la lunghezza del messaggio.
     - Si controlla se il puntatore `data` è `NULL`. Se è `NULL`, significa che non ci sono dati da inviare, quindi la lunghezza viene impostata a `0`.
2. **Formattazione del messaggio in formato CSV:**
   - Viene dichiarata una variabile buffer `buffer` per contenere il messaggio formattato.
   - Si utilizza la funzione `sprintf` per formattare il messaggio in formato CSV all'interno del buffer. Il messaggio avrà la forma: `tipo,lunghezza,dati`.
3. **Scrittura del messaggio sulla socket:**
   - Viene dichiarata una variabile `rv` per memorizzare il valore di ritorno della funzione `write`.
   - Si utilizza la macro `SYSC` per chiamare la funzione `write` in modo sicuro e gestire eventuali errori. `write` tenta di scrivere il contenuto del buffer `buffer` sulla socket identificata da `socket_fd`. La dimensione del buffer da scrivere viene specificata da $(\text{strlen}(\text{buffer}) + 1) * \text{sizeof}(\text{char})$, usata per scrivere solamente i dati significativi

2.1.2 `messaggio read_message(int socket_fd)`

1. **Lettura del messaggio dalla socket:**
   - Si utilizza la macro `SYSC` per chiamare la funzione `read` che tenta di leggere dati dalla socket identificata da `socket_fd` e li memorizza nel buffer `buffer`.
   - Il valore di ritorno della funzione `read` viene memorizzato in `rv`.
2. **Elaborazione del messaggio:**
   - Si controlla il valore del secondo carattere nel buffer `buffer[2]`. Questo carattere rappresenta la lunghezza del messaggio (`length`).
     - Se `buffer[2]` è uguale a 0, significa che il messaggio ha lunghezza zero e non contiene dati.
       - In questo caso, si imposta il tipo del messaggio (`msg.type`) al primo carattere del buffer (`buffer[0]`).
       - Si impostano la lunghezza (`msg.length`) a 0 e il puntatore ai dati (`msg.data`) a `NULL`.
     - Se `buffer[2]` è diverso da '0', significa che il messaggio ha una lunghezza maggiore di zero e contiene dati.
       - Si utilizza la funzione `strtok` per tokenizzare il buffer separando i campi del messaggio in base alla virgola (",").

2.2 Gestione dei giocatori

Ai client che si collegano al server viene associato un “profilo” definito e salvato in una variabile di tipo `Player`. I diversi Players sono collegati tra di loro come una coda di Players, per poter permettere un numero variabile di giocatori e rendere l’aggiunta e rimozione di giocatori semplice.

Questo profilo contiene diversi dati utili del client:

- il file descriptor della sua socket,
- il nome e score → utilizzato poi dallo **scorer thread** a per calcolare i risultati a fine partita
- l’array di parole che sono gia’ state usate da quel giocatore e l’index per scorrere questa lista
- il puntatore al prossimo giocatore.

Nel file sono poi definite diverse funzioni di utility per l’aggiunta `add_player`, aggiornamento `add_score,clear_players_data` e rimozione `delete_player` di un profilo dalla lista di giocatori.

---

3. Client

➡️ File di riferimento : client.c

Il processo client prende dalla riga di comando i dati richiesti per collegarsi alla socket del processo server.

Dopo essersi collegato si divide in due threads:

- Il thread principale che riceve gli inputs dall’utente e manda messaggi al server
- Il thread `message_reader` che si occupa di leggere e gestire i messaggi ricevuti dal server.

Entrambi i thread usano un semaforo `prompt_sem` per sincronizzarsi e stampare il prompt **[PROMPT PAROLIERE]** nel momento corretto.

3.1 main thread

Il thread principale continua quindi definendo due buffer dove salvare gli input dell’utente letti dallo STDIN e inizializza il semaforo `prompt_sem` che verra’ usato per sincronizzare la print del prompt *dopo* aver stampato i messaggi letti dal server.

Si entra quindi nel main game loop dove l’utente puo’ interagire e proporre diversi comandi per ricevere aiuto(`aiuto`), vedere la matrice di gioco e il tempo rimanente(`matrice`), vedere la classifica a fine partita(`classifica`) e uscire dalla partita(`fine`).

I diversi comandi hanno effetti diversi a seconda se l’utente e’ loggato o no, indicato dalla variabile globale `logged`, come per esempio se un utente non e’ loggato non puo’ proporre parole o richiedere la matrice di gioco, ma puo’ chiedere aiuto o uscire lo stesso.

3.2 message_reader thread

Il thread message_reader invece legge i messaggi del server dal file descriptor della socket che gli viene passato alla sua creazione. Il main loop legge il messaggio tramite la funzione `read_message` definita in paroliere.h e definisce il messaggio ricevuto dal server. A seconda del tipo di messaggio ricevuto il thread stampa a video i diversi tipi di messaggi ricevuti dal server.

---

4. Server

➡️ File di riferimento : server.c

Il processo server e’ diviso su piu’ threads per le diverse funzioni che svolge.

4.1 Main function

La funzione `main` per prima cosa inizializza variabili utili per la lettura dei passati dalla riga di comando. Per fare questo usa la funzione `getpot` della libreria getopt.h, definisce la lista delle opzioni che possono essere passate dalla riga di comando, se richiedono un argomento e associa loro un char identificativo. Estrae dalle opzioni passate il corrispondete char, o ‘?‘ nel caso in cui sia una opzione non conosciuta, e tramite uno switch case va a modificare le variabili globali corrispondenti con i dati richiesti dall’utente,

Dopo aver definito le specifiche delle partite creo la **trie** del dizionario di gioco e, se non gia’ caricata da un file, la matrice di gioco `game_matrix`.

Usando la la funzione `sigaction` vado ad associare la funzione custom `alarm_handler` per gestire l’alarm che viene fatto partire per separare la fase di gioco dalla fase di pausa. Dato che sto usando delle funzioni *blocking* come le read/write setto la flag `SA_RESTART` per fare ripartite le chiamate che erano state interrotte dalla interrupt causata dalla `SIGALRM`.

Viene quindi definita e aperta la socket con un numero massimo di utenti sulla listen di 32, numero arbitrario e modificabile dato che la struttura dati che gestisce i giocatori non ha un dimensione fissa essendo una coda. I client che vengono accettati vengono gestiti da un thread ognuno che viene creato subito dopo la accept, thread che li gestira’ fino alla loro disconnessione.

4.2 thread `client_handler`

Il thread che gestisce il client entra subito in un loop di ascolto di messaggi mandati dal client attraverso la socket che gli viene passata come argomento.
A seconda del tipo di messaggio ricevuto, letto con la `read_message`, si ha una gestione diversa.

- MSG_REGISTRA_UTENTE
  L'handler prova ad aggiungere il giocatore con il nome fornito dal client e a seconda del valore restituito dalla funzione di aggiunta risponde al client.
  Se il nome non e' ancora stato usato viene associato il puntatore del Player nella lista dei giocatori al profilo del client nel thread, a cui vengono associati il client_fd e inizializzati i dati del giocatore. Se il nome non e' disponibile o troppo lungo viene inviato un messaggio di errore
- MSG_MATRICE
  Il messaggio matrice ha delle risposte diverse a seconda del
  gamestate in cui ci si trova.
  Se si sta giocando (
  gamestate = 0) viene mandata la matrice corrente al client
  Se si e' in pausa (
  gamestate = 1) viene cambiato il tipo del messaggio a MSG_TEMPO_PARTITA per ottenere il tempo rimanente
- MSG_TEMPO_PARTITA
  Se si sta giocando viene mandato il tempo rimanente nella partita con il messaggio MSG_TEMPO_PARTITA
  Se non si sta giocando viene mandato il tempo rimanente nella pause con MSG_TEMPO_ATTESA
  - Il tempo mancante si ottiene con la chiamata `alarm(0)` che resetta il timer e restituisce il tempo mancante, che viene usato per fare ripartire subito il timer con il tempo rimanente e per mandare il messaggio al client.
- MSG_PAROLA
  Se si e' in partita l'handler controlla la parola in 3 modi: se la parola e' stata gia' usata → parola nella matrice → parola nella trie-dizionario. Nel caso in cui la parola passi tutti i controlli viene calcolato il punteggio guadagnato, mandato al client e aggiunto al profilo del giocatore. Se la parola e' stata gia' usata vengono dati 0 punti, se sbagliata manda un messaggio di errore.
- MSG_CLIENT_QUIT
  Se il client si vuole disconnettere , se loggato viene cancellato dalla lista di giocatori , e il thread termina.
- MSG_ALARM
  Tipo di messaggio aggiuntivo creato che viene mandato dai client quando finisce una partita. Il thread rimane in attesa sulla barriera `barrier` fino a che il thread `scorer` non finisce di calcolare la classifica finale, per poi comunicarla ai diversi client e resettare i score e parole usate dal client.

4.3 alarm_handler

L'alarm handler gestisce il segnale `SIGALRM`.
Nel caso in cui sia finita una pausa carica la nuova matrice, la manda a tutti i clients (con il tempo di durata della partita) e fa ripartite il timer della durata della partita.
Nel caso in cui sia finita una partita inizializza una barriera che blocca
$\text{num\_players} + 1$ threads , fa parite il timer per la fine della pausa e fa partire il thread `scorer` per calcolare la classifica finale.

4.4 thread `scorer`

Lo scorer thread va a recuperare i dati dei diversi giocatori dalla coda di Player, li avvisa della fine della partita con un `MSG_ALARM` e usa un bubble sort per riordinare in ordine decrescente i players per definire la classifica. Finito di calcolare si mette in attesa insieme agli altri threads sulla barriera, facendola "aprire".

---

5 Compilazione e Test Run

- comando `make` per compilare automaticamente i due eseguibili `paroliere_srv` e `paroliere_cl`

5.1 Server Logs

```
normalset@normalPC:~/Lab2/progetto_finale$ make
gcc -Wall -g -pedantic -pthread server.c -o paroliere_srv
gcc -Wall -g -pedantic -pthread client.c -o paroliere_cl
normalset@normalPC:~/Lab2/progetto_finale$ ./paroliere_srv localhost 2001 --matrici matrix.txt --durata 40
Loading custom matrix from : matrix.txt
*** Data for server ***
Server name: localhost
Server port: 2001
Using custom matrix filename: 1
Seed: -1 (default if not provided)
Duration: 40 seconds (default 3 min if not provided)
Dictionary file: dict.txt
Starting Matrix:
|qu|a |n |t |
|a |b |c |o |
|a |b |c |d |
|a |b |c |d |
New Client connected 5! <--Primo client si connette
Client handler here
```

```
---------------------
[ ] Read Message : R 6 tester <-- messagio di registrazione dal client
Trying to add player : tester
[ ] Searching for player : tester
[ ] Wrote Message : K 0 (null) <-- aggiunto correttamente, messaggio OK al client
[ ] Added player tester
Player List
        tester score : 0 fd : 0
---------------------
[ ] Read Message : M 0 (null) <-- Messaggio di richiesta della matrice dal client
[ ] Wrote Message : M 16 qantabcoabcdabcd <-- Riposta matrice
---------------------
[ ] Read Message : T 0 (null) <-- Richiesta del tempo dal client
[ ] Wrote Message : T 2 34 <-- Riposta tempo di gioco
---------------------
[ ] Read Message : W 6 quanto <-- Proposta parola quanto
[ ] Wrote Message : P 1 5 <-- Prima volta usata e corretta, 5 punti
---------------------
[ ] Read Message : W 6 quanto <-- Proposta parola quanto
[ ] Wrote Message : P 1 0 <-- Corretta ma gia' usata, 0 punti
---------------------
[ ] Read Message : W 4 test <-- Proposta parola test
[ ] Wrote Message : E 16 Wroing choice :( <-- non presente nella matrice/dict
---------------------
New Client connected 6!
Client handler here
---------------------
[ ] Read Message : R 7 tester2
Trying to add player : tester2
[ ] Searching for player : tester2
[ ] Wrote Message : K 0 (null)
[ ] Added player tester2
Player List
        tester2 score : 0 fd : 0
        tester score : 5 fd : 5
---------------------
[ ] Read Message : M 0 (null)
[ ] Wrote Message : M 16 qantabcoabcdabcd
---------------------
[ ] Read Message : T 0 (null)
[ ] Wrote Message : T 2 20
---------------------
[ ] Read Message : M 0 (null)
[ ] Wrote Message : M 16 qantabcoabcdabcd
---------------------
[ ] Read Message : T 0 (null)
[ ] Wrote Message : T 2 19
---------------------
ALARM RANG!
[DEGUB] Number of players when alarm : 2
------ Inizio pausa ------
[ ] scorer started
List players in scorer thread:
Player List
        tester2 score : 0 fd : 6
        tester score : 5 fd : 5
[DEBUG] Writing msg to client 6 <-- Scrive ai client per farsi mandare la richiesta della classifica
[ ] Wrote Message : Z 0 (null)
[DEBUG] Writing msg to client 5
[ ] Wrote Message : Z 0 (null)
Pre Sort Arrays:
(0) tester2 | 0
(1) tester | 5
Post Sort Arrays:
(1) tester | 5
Post Sort Arrays:
(2) tester2 | 0
[ ] Read Message : Z 0 (null)
[ tester2 ] Waiting on barrier <-- client 6 aspetta sulla barrier
adding (0) tester : 5; to the leaderboard <-- compila la classifica
adding (1) tester2 : 0; to the leaderboard
[ ] Read Message : Z 0 (null)
```

```
[ tester ] Waiting on barrier <-- client 5 aspetta sulla barrier
[ tester2 ] Exited barrier <-- tutti sulla barrier, si apre
[ ] scorer ended
[ tester ] Exited barrier
[ ] Wrote Message : F 31 (0) tester : 5;(1) tester2 : 0;
---------------------
[ ] Wrote Message : F 31 (0) tester : 5;(1) tester2 : 0;
---------------------
[ ] Read Message : F 0 (null)
[ ] Wrote Message : F 31 (0) tester : 5;(1) tester2 : 0;
---------------------
[ ] Read Message : Q 0 (null) <-- richiesta di uscita del client 5
[ ] Player 5 left
Player List
        tester2 score : 0 fd : 6 <-- client 5 eliminato dai giocatori
[ ] Read Message : Q 0 (null) <-- richiesta di uscita del client 6
[ ] Player 6 left
Player List <-- Non ho giocatori
^C <-- chiudo il server
```

5.2.1 Player 1

```
normalset@normalPC:~/Lab2/progetto_finale\
$ ./paroliere_cl localhost 2001
[ PROMPT PAROLIERE ]--> registra_utente tester
[ ] Now logged in
[ ] Matrice di gioco :
|qu|a |n |t |
|a |b |c |o |
|a |b |c |d |
|a |b |c |d |
[TIME] Tempo rimanente nella partita : 34
[ PROMPT PAROLIERE ]--> p quanto
[POINTS] Got 5 points for that word!
[ PROMPT PAROLIERE ]--> p quanto
[POINTS] Got 0 points for that word!
[ PROMPT PAROLIERE ]--> p test
[ERR] Wroing choice :(
[ PROMPT PAROLIERE ]-->
------ TIME'S UP ! ------

[FINAL] Classifica:
(1) tester : 5
(2) tester2 : 0
[ PROMPT PAROLIERE ]--> classifica

[FINAL] Classifica:
(1) tester : 5
(2) tester2 : 0
[ PROMPT PAROLIERE ]--> fine
[ ] client exiting
```

5.2.2 Player 2

```
normalset@normalPC:~/Lab2/progetto_finale\
$ ./paroliere_cl localhost 2001
[ PROMPT PAROLIERE ]--> registra_utente tester2
[ ] Now logged in
[ ] Matrice di gioco :
|qu|a |n |t |
|a |b |c |o |
|a |b |c |d |
|a |b |c |d |
[TIME] Tempo rimanente nella partita : 20
[ PROMPT PAROLIERE ]--> matrice
[ ] Matrice di gioco :
|qu|a |n |t |
|a |b |c |o |
|a |b |c |d |
|a |b |c |d |
[TIME] Tempo rimanente nella partita : 19
[ PROMPT PAROLIERE ]-->
------ TIME'S UP ! ------

[FINAL] Classifica:
(1) tester : 5
(2) tester2 : 0
[ PROMPT PAROLIERE ]--> fine
[ ] client exiting
```