The "Real" Paroliere online

Progetto Finale del corso di Laboratorio 2 A

Patrizio Dazzi e Luca Ferrucci

Appello autunnale 2023-24
v. 1.1

Premessa

Il progetto descritto in questo documento è valevole per il superamento della prova scritta del corso di Laboratorio 2 - A. Il testo deve essere letto e compreso nella sua interezza da tutti gli studenti, con la sola eccezione della Sezione 3, denominata "Addendum per chi non ha superato un numero congruo di prove in itinere", necessaria solo per coloro i quali non avessero superato un numero congruo (3) di prove in itinere (anche considerando la prova di recupero). In essa sono dettagliate le richieste in merito all'implementazione di un certo numero di funzionalità aggiuntive.

1 Descrizione generale

Si realizzi un progetto denominato "Paroliere" che permette ad un insieme di utenti di giocare online ad una versione **modificata** gioco de "il Paroliere"[1]. Il gioco consiste nel comporre parole utilizzando le lettere posizionate su di una matrice quadrata di 16 lettere (4x4). Ogni parola deve essere formata da almeno 4 caratteri e può essere composta utilizzando le lettere della matrice una sola volta. Le caselle utilizzate devono essere contigue e composte in senso orizzontale, verticale ed anche diagonale, in questa versione dell'assegnamento. Non è possibile passare sopra una casella più di una volta per la medesima parola.

A	T	L	C
I	O	Qu	A
D	V	E	S
I	S	B	I

Table 1: Esempio di Matrice di Caratteri

[1] `https://it.wikipedia.org/wiki/Il_Paroliere`

Ad esempio data la matrice riportata in Tabella 1, sono legittime, tra le altre, le parole `CASI, CASE, BEVO, VOTA`, `BEVI, VOTI, VOLA`.

Suggerimento: per determinare la leicità di una parola, aldilà dell'effettiva presenza nella matrice generata, si utilizzi il dizionario incluso nel file dizionario.txt.

2 Struttura del gioco

Il gioco è organizzato secondo uno schema client-server. Client e server si scambiano messaggi utilizzando un protocollo di comunicazione ben definito, riportato di seguito. **Non sono ammesse variazioni o estensioni del protocollo definito, nè i tipi di messaggio, nè la struttura dello stesso.**

2.1 Protocollo di Comunicazione

I server e i client interagiscono tramite socket INET. I client si connettono al server col quale si scambiano messaggi. Il server risponde sulla stessa connessione. Sia la richiesta che la risposta includono 3 campi: `type`, `length`, `data`. Il campo `type` è un `char` (8 bit) che contiene il tipo del messaggio spedito, e può assumere i seguenti valori:

```
#define MSG_OK ‘K’
#define MSG_ERR ‘E’
#define MSG_REGISTRA_UTENTE ‘R’
#define MSG_MATRICE ‘M’
#define MSG_TEMPO_PARTITA ‘T’
#define MSG_TEMPO_ATTESA ‘A’
#define MSG_PAROLA ‘W’
#define MSG_PUNTI_FINALI ‘F’
#define MSG_PUNTI_PAROLA ‘P’
#define MSG_SERVER_SHUTDONW ’B’
#define MSG_POST_BACHECA ‘H’
#define MSG_SHOW_BACHECA ‘S’
```

Il campo `length` è un `unsigned int` che indica il numero dei dati significativi all'interno del campo data. Il campo `length` vale `0` nel caso in cui il campo `data` non sia significativo. Il campo `data` è una stringa contenete il messaggio di risposta effettiva. Per ogni messaggio si richiede di inviare solo i byte significativi del messaggio e non un buffer di lunghezza fissa.

2.2 Server

Il processo server si articola nelle seguenti attività:

- All'avvio il server genera una matrice `4x4` e apre un socket INET su di una porta specificata come parametro da riga di comando, sulla quale resta in attesa.
- Il server deve essere in grado di generare matrici sia in modo casuale (non è necessario verificare che la matrice contenga parole valide) che leggendo in sequenza da un file passato in input al server, come parametro da riga di comando. Il file riporta su ogni riga 16 lettere diverse, ognuna separata dalla successiva mediante uno spazio. L'unica eccezione è costituita dalla coppia di lettere `Qu` che vanno trattate come singolo carattere, sebbene nel file vadano riportate entrambe (come nell'esempio). Il suggerimento è di trattare `Qu` come carattere speciale, mantenendo caratteri come singoli elementi della matrice. U potrebbe comparire senza la Q ma la Q non può comparire senza U (ovvero la lettera Q appare solo nella forma Qu). A titolo di esempio si consideri che la matrice riportata in Tabella 1 come il risultato della lettura della seguente riga (terminata da `\n`).

  `A T L C I O Qu A D V E S I S B I`
- Quando riceve una richiesta di connessione da un client, il server genera un thread dedicato alla gestione di tale client. Il thread rimane attivo fino alla disconnessione del client gestito.
- Dopo essersi collegato, il client registra il proprio nome, che deve essere univoco. Il tipo di messaggio utilizzato dal client è `MSG_REGISTRA_UTENTE`. Il server risponde `MSG_OK` se il nome non è già in uso e `MSG_ERR` se, al contrario, lo è già. Qualora fosse già in uso, il client, a seguito della ricezione di un messaggio di errore `MSG_ERR`, deve proporre un nome differente. Il processo si ripete fino a quando il client non propone un nome univoco.
- Il client viene quindi aggiunto ai giocatori, che sono mantenuti dal server utilizzando un'opportuna struttura dati, scelta dallo studente. Suggeriamo di definire una costante (es. `MAX_NUM_CLIENTS`) e di utilizzare quella come riferimento. Si scelga un valore ragionevole, ad esempio 32. Il server comunica al client la matrice di caratteri (tramite il messaggio `MSG_MATRICE`) e la durata residua della partita in corso (utilizzando il messaggio `MSG_TEMPO_PARTITA`) o del tempo restante prima dell'inizio della partita successiva (utilizzando il messaggio `MSG_TEMPO_ATTESA`). In entrambi i casi i tempi sono espressi in secondi.
- Ogni client può proporre parole, una alla volta (utilizzando il messaggio `MSG_PAROLA`). Il server, dopo aver controllato la correttezza della parola, assegna al client un punteggio, pari al numero delle lettere che compongono la parola stessa (la coppia 'Qu' conta per 1). **La correttezza della parola è tale quando lo è rispetto all'effettiva esistenza nella matrice di gioco, l'opportuna lunghezza, ed all'esistenza nel dizionario di riferimento. Il dizionario deve necessariamente essere caricato in memoria all'avvio del gioco, utilizzando una struttura dati di tipo**

Trie[2]**.** Se la parola è corretta il server risponde con il messaggio `MSG_PUNTI_PAROLA` indicando i punti assegnati alla parola; se la parola non è corretta il server risponde con `MSG_ERR`. Questo punteggio viene attribuito al client solo la prima volta che propone una parola che non è stata precedentemente suggerita, dallo stesso client, nella stessa partita. Ergo, le parole ripetute dallo stesso client, nella medesima partita, non determinano un aumento del punteggio. In questo caso il server risponde al client con il messaggio `MSG_PUNTI_PAROLA` indicando 0 punti.

- Ogni partita ha una durata, determinata da uno specifico parametro passato da riga di comando all'avvio del server. Tra due partite diverse deve essere garantita la pausa di un minuto. Durante la pausa il server risponde ad alcune richieste dei client ma non avvia nuovi giochi. In particolare, il server risponde positivamente (senza restitutire errori) ai seguenti messaggi: `MSG_REGISTRA_UTENTE`, `MSG_MATRICE` (che in questo caso determina soltanto la comunicazione del `MSG_TEMPO_ATTESA`), `MSG_PUNTI_FINALI`.

- Al termine della partita, ogni thread (che gestisce un client) spedisce un messaggio contenente il punteggio del client su di una coda che è condivisa tra i diversi thread. Un ulteriore thread, che chiameremo `scorer`, recupera i messaggi dalla coda e stila la classifica del gioco, infine nominando il vincitore. Nel condurre questa attività il thread `scorer` deve opportunamente sincronizzarsi con gli altri threads, i quali, dopo che il vincitore è stato nominato, comunicano ai rispettivi client il risultato finale. NOTA BENE: **non è compito del thread scorer la comunicazione ma dei singoli thread che gestiscono i client.** La comunicazione ai client del risultato finale avviene tramite il messaggio `MSG_PUNTI_FINALI`. Il contenuto del messaggio è in formato CSV (comma-separated values) nel quale si alternano nomi utente e loro punteggio, terminato dal carattere `\0`.

- Ogni giocatore viene considerato partecipante alla partita successiva se ancora connesso all'avvio del nuovo gioco. **Va ricordato che i giocatori possono iscriversi in qualunque momento del gioco, sia esso in pausa o in svolgimento.**

- Se un client si disconnette in seguito ad un errore o viene chiuso il socket col client per un qualsiasi motivo, il giocatore deve essere cancellato dal gioco corrente e chiuso il relativo thread.

- Il server termina quando viene premuto `CTRL-C` sulla shell dalla quale è stato lanciato (ovvero viene inviato il segnale `SIGINT` al processo server). **Il server prima di terminare la propria esecuzione, invia ad ogni client, tramite i thread di gestione, il messaggio** `MSG_SERVER_SHUTDOWN` **per informarli della propria imminente terminazione.**

[2] `https://it.wikipedia.org/wiki/Trie`

- Il server ospita una bacheca messaggi. I client possono pubblicare messaggi (ognuno di massimo 128 caratteri) su di essa con un apposito comando `msg testo_messaggio` (che determina l'invio del messaggio `MSG_POST_BACHECA`). La bacheca memorizza soltato gli ultimi 8 messaggi inviati. Il client possono visualizzarne il contenuto tramite il comando `show-msg` (che determina l'invio del messaggio `MSG_SHOW_BACHECA`). Questi comandi sono utilizzabili anche durante una pausa tra un gioco e il successivo. Il server risponde ad un messaggio di tipo `MSG_POST_BACHECA` con un messaggio di tipo ok (`MSG_OK`) o di tipo errore (`MSG_ERR`) a seconda dell'esito dell'operazione. Invece, ad un messaggio di tipo `MSG_SHOW_BACHECA` il server risponde con un messaggio il cui conteuto è in formato CSV (comma-separated values) nel quale si alternano nomi utente e i singoli messaggi inviati `\0`. Anche in questo caso, le strutture dati da utilizzare e la metodologia di gestione della bacheca sono lasciate allo studente.

Sintassi

Il processo server deve necessariamente essere lanciato mediante il seguente comando:

```
> ./paroliere_srv nome_server porta_server [--matrici data_filename]
      [--durata durata_in_minuti] [--seed rnd_seed] [--diz dizionario]
```

dove:

- `paroliere_srv` è il nome dell'eseguibile;
- `nome_server` è il nomer del server sul quale sarà fatto partire il server (suggerimento: potete testare il progetto sulla vostra macchina indicando `127.0.0.1` o `localhost`)
- `porta_server` è il numero della porta sulla quale far partire il processo server (suggerimento: usate porte con numero maggiore di 1024)
- il parametro opzionale `--matrici` è seguito dal nome del file dal quale caricare le matrici (altrimenti generate tramite l'utilizzo di un generatore di numeri pseudocasuali).
- il parametro opzionale `--durata` permette di indicare la durata del gioco in minuti. Qualora non espresso la durata di default va considerata di 3 minuti.
- il parametro opzionale `--seed` permette di indicare il seed da usare per la generazione dei numeri pseudocasuali.
- il parametro opzionale `--diz` permette di indicare il dizionario da usare per la verifica della leicità delle parole ricevute dal client. Il file di dizionario è un file di testo nel quale su ogni riga è indicata una parola. Ogni riga termina con un `\n`.

Suggerimento: per il parsing dei parametri si consiglia l'utilizzo di `getopt_long`

2.3 Client

Il processo client, **da realizzare necessariamente come programma multithread**, si articola nelle seguenti attività:

- All'avvio, il client, da riga di comando, riceve dall'utente l'indicazione del nome del server e della porta alla quale collegarsi.
- Il client avvia una sessione interattiva con l'utente. L'utente può specificare un certo numero di comandi per interagire con il server al fine di *registrarsi e giocare*. I comandi di gioco devono permettere all'utente di *avere informazioni sulla matrice di caratteri e proporre parole*.

  1. Il primo passo da compiere riguarda la registrazione dell'utente, da effettuarsi dal client mediante il comando `registra_utente` seguito da un nome utente di al più 10 caratteri alfanumerici (ci si limiti alle cifre e ai caratteri dell'alfabeto italiano). **Non è ammissibile considerare tutte le lettere ASCII.** Come anticipato durante la descrizione del server, tale comando determina l'invio da parte del client al server di un messaggio di tipo `MSG_REGISTRA_UTENTE`.
  2. Se il nome non è in uso, il server restituirà un messaggio di tipo `MSG_OK`, altrimenti il server risponde con un messaggio di errore `MSG_ERR`. Fino a quando un utente non ha completato la procedura di registrazione, tutte le altre operazioni, tranne l'uscita dal gioco, gli sono precluse. In altre parole, fintanto che l'utente non si è correttamente registrato, l'invocazione di qualsiasi altro comando, ad eccezione di `fine`, `aiuto`, `registra_utente` determina un messaggio di errore.
  3. Quando il gioco è in corso, il client può cominciare a proporre le parole. Ogni parola è proposta utilizzando il messaggio `MSG_PAROLA`. Se la parola è corretta, il server risponde con un messaggio di tipo `MSG_PUNTI_PAROLA`, se non lo è risponde con un messaggio `MSG_ERR`.
  4. Per capire se il gioco è in corso ed, eventualmente, quale sia la matrice corrente, il client può effettuare una richiesta al server invocando il comando `matrice`. Tale comando determina l'invio di un messaggio `MSG_MATRICE` dal client al server. Se il gioco è in corso il server risponde prima con il messaggio `MSG_MATRICE` indicando la matrice di gioco e successivamente con il messaggio `MSG_TEMPO_PARTITA`. Se il gioco non è in corso il server risponde con il messaggio `MSG_TEMPO_ATTESA`.

- Al termine di ogni partita, il server comunica al client il vincitore e la classifica dei punteggi di tutti i giocatori. A tale scopo, il server invia al client un messaggio `MSG_PUNTI_FINALI`. Il client mostra a video all'utente i risultati ricevuti. Come già specificato nella presentazione del server, il contenuto del messaggio è in formato

CSV (comma-separated values) nel quale si alternano nomi utente e loro punteggio, terminato dal carattere `\0`.

- Nel caso in cui il server inviasse al client un messaggio `MSG_SERVER_SHUTDOWN` il client deve assicurare una terminazione controllata.
- I client possono pubblicare messaggi (ognuno di massimo 128 caratteri) sulla bacheca usando il comando `msg testo_messaggio` (che determina l'invio del messaggio `MSG_POST_BACHECA`). Il client possono visualizzarne il contenuto tramite il comando `show-msg` (che determina l'invio del messaggio `MSG_SHOW_BACHECA`). Questi comandi sono utilizzabili anche durante una pausa tra un gioco e il successivo.
- Quando l'utente specifica il comando `fine`, il client si disconnette dal server e la sessione termina. In questo caso il nome utente deve essere rimosso dal sistema. Anche in caso di disconnessione accidentale o non richiesta del client, il nome utente deve essere rimosso dal sistema.

Sintassi dei comandi del client

Avvio client:
```
> ./paroliere_cl nome_server porta_server
```

dove:

- `paroliere_cl` è il nome dell'eseguibile;
- `nome_server` è il nome del server al quale collegarsi;
- `porta_server` è il numero della porta alla quale collegarsi;

L'avvio del client darà come risultato:
```
[PROMPT PAROLIERE]-->
```

Ciò permetterà di accedere ad un insieme di comandi con i quali interagire con il server. Nello specifico:

`aiuto` → comando utilizzato per mostrare a video i comandi disponibili del client e la loro sintassi. Non determina l'invio di alcun messaggio al server.

`registra_utente nome_utente` → comando utilizzato per registrare un nuovo utente. Per farlo il client invia un messaggio di tipo `MSG_REGISTRA_UTENTE` al server. Qualora il nome indicato non fosse univoco il server risponderebbe con un messaggio di errore `MSG_ERR`. Altrimenti il server risponde con un messaggio di conferma `MSG_OK`.

`matrice` → comando utilizzato per richiedere al processo server la matrice corrente. Se il gioco è in corso il server risponde con l'elenco delle 16 lettere che compongono la matrice e il tempo residuo di gioco, espresso in secondi. Se il gioco non è in corso il server risponde indicando il tempo mancante all'avvio di un nuovo gioco, sempre in secondi. Il tipo di messaggio associato a questo comando è `MSG_MATRICE`.

`p parola_indicata` → comando utilizzato per sottoporre al processo server una parola. Se è corretta e non è stata ancora proposta dall'utente durante la medesima partita, il server risponde con il punteggio assegnato per tale parola. Se corretta ma già proposta in precedenza, il server risponde con zero punti. Se la parola non è corretta, il server risponde con un messaggio d'errore.

`fine` → comando utilizzato per uscire dal gioco.

3 Addendum per chi non ha superato un numero congruo di prove in itinere

In aggiunta a quanto dettagliato sopra, agli studenti che non avessero superato un numero congruo di prove in itinere, è richiesto di sviluppare un certo numero di funzionalità aggiuntive. Tali funzionalità sono descritte in questa sezione e richiedono l'introduzione di alcuni ulteriori tipi di messaggi.

```
#define MSG_CANCELLA_UTENTE 'D'
#define MSG_LOGIN_UTENTE 'L'
```

- La cancellazione di un utente dal server non deve essere automatica, a fronte di una disconnessione ma deve essere gestita tramite un opportuno comando fornito dal client `cancella_registrazione`. Tale comando deve determinare l'invio del messaggio `MSG_CANCELLA_UTENTE`.
- Deve essere possibile riaccedere ad un gioco con un nome utente registrato in precedenza e non ancora cancellato. Il comando che il client deve fornire a tale scopo è `login_utente` e determina l'invio del messaggio `MSG_LOGIN_UTENTE`
- Registrare in un file di log quali sono stati i processi di registrazione e cancellazione utente, le parole inviate da ogni utente, i punteggi di ogni gioco. La decisione sul formato da utilizzare è lasciato allo studente.
- Un client che non comunica con il server per un certo periodo di tempo (espresso in minuti, da specificare come parametro da riga di comando al server tramite `[--disconnetti-dopo tempo_in_minuti]`), viene espulso dalla partita corrente e

non considerato più giocante. Il relativo thread deve essere terminato dopo la disconnessione e il giocatore deregistrato per la partita corrente. Il nome dell'utente NON viene però cancellato dal server.

4 Relazione:

La documentazione del progetto consiste nei commenti al codice e in una breve relazione (massimo 8 pagine) il cui scopo è quello di descrivere la struttura complessiva del lavoro svolto. NON devono essere ripetute le specifiche contenute in questo documento. La relazione deve contenere:

- le principali scelte di progetto (strutture dati principali, algoritmi fondamentali e motivazioni alla base);
- la strutturazione del codice (logica della divisione su più file, librerie etc.);
- la struttura dei programmi sviluppati;
- la struttura e la logica dei programmi di test;
- istruzioni su come compilare/eseguire ed utilizzare il codice

La relazione deve essere in formato PDF.

Note finali:

- La scelta delle strutture dati da utilizzare e la strutturazione concorrente dei processi server e client sono lasciate allo studente.
- Tutti gli esercizi vanno realizzati in autonomia, senza alcun tipo di collaborazione, pena l'annullamento della prova.
- Il codice sviluppato deve essere in linguaggio C conforme POSIX. È consigliabile testarlo sulla macchina di laboratorio `laboratorio2.di.unipi.it` per assicurasi della conformità POSIX.
- per la consegna, al termine della prova d'esame, si proceda a creare un file compresso denominato `progetto-lab2-estate24-<NUM_MATRICOLA>.zip` (indicando al posto di `<NUM_MATRICOLA>` la propria matricola studente).
- Scompattando l'archivio, dovrà essere possibile compilare il codice tramite `make` (con target di default). L'archivio dovrà anche contenere la relazione.

5 Modalità di consegna

I progetti vanno consegnati tramite classroom, entro scadenza prevista. Inoltre è richiesto di inviare notifica via mail ad entrambi i docenti, dopo aver effettuato la consegna.