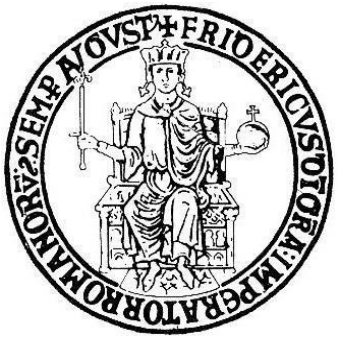
Università degli Studi di Napoli Federico II
Scuola Politecnica e delle Scienze di Base
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione



Corso di Laurea in Informatica

Laboratorio di Sistemi Informativi
Anno accademico 2020/2021

Specifica, progettazione e implementazione della traccia "Quiz d'aritmetica"

Consegna: gennaio

Autore:

Ivan Capasso N86002587
Cristofor Doarme N86002762

Gruppo N.26

Docente:

Faella Marco

1.Guida alla compilazione e manuale d’uso

Segue una descrizione approfondita dell’applicativo, senza entrare nei dettagli tecnici, che permette la comprensione e l’utilizzo del programma.

Innanzitutto, scaricare l’applicativo dal seguente link:

È necessario compilare i due file .c presenti all’inizio della directory: Server.c e Client.c.

Comandi gcc:

```
1. Server: gcc Server.c Question/Question.c -oServer -w -pthread
```

```
2. Client: gcc Client.c -oClient -pthread
```

Si saranno creati due file, rispettivamente Server.out e Client.out, avviare **rispettivamente** il server e il client normalmente come eseguibili shell:

```
1. ./Server
2. ./Client
```

Il client cerca di collegarsi al server all’indirizzo specificato nella funzione *Connection/setServAdress* , **ricordarsi di sostituire l’indirizzo** presente con quello IP PvP4 pubblico del server (Il server hosta sulla porta 20000).

L’applicativo server si presenta con un messaggio di attesa di una prima connessione da parte di un client.

Quando il primo client seleziona “Start game”, il server crea una partita, dove viene aggiunto il client che ha fatto richiesta e dove i successivi client, possono connettersi alla partita e giocare insieme agli altri client.

I client possono uscire prima dell’inizio della partita, ma quest’ultima inizierà uguale.
Una volta iniziata la partita, il server mostra la domanda che attualmente i partecipanti dovranno rispondere. Il client oltre a mostrarla richiede di dare una risposta entro 60 secondi, rappresentabile da un numero da 1 a 4 altrimenti 0.

I client possono connettersi anche quando una partita è già iniziata!
Basterà che selezionino “Start game” e aspettare che si passi alla domanda successiva, ovvero che si arrivi alla fine dei 60 secondi della domanda corrente.

Elenco delle domande:

L’elenco di domande è preso dal file “domande.txt”, questo file può essere modificato solamente prima dell’avvio del server.

Le domande verranno mostrate nell'ordine in cui sono inserite all'interno del file: per rendere il tutto più intuitivo, c'è un ID che va posto all'inizio della domanda che rappresenta il numero di domande presenti nel file.
Il server seguirà sempre l'ordine delle domande del file, **è compito dell'utilizzatore del server assicurarsi che l'ID "X" corrisponda alla domanda "n.X" effettiva.**

Si possono aggiungere, modificare e rimuovere domande da quest'elenco, l'importante è rispettare la forma in cui è scritta la domanda nel file. Segue la strutturazione di una domanda e un relativo esempio a confronto.

Formattazione nel file:	**Esempio:**
Numero.Domanda:	2.Quanto fa (30/5)*10?:
1)<Risposta N.1°>	1)35
2)<Risposta N.2°>	2)-54
3)<Risposta N.3°>OK	3)60OK
4)<Risposta N.4°>	4)70

Nell'esempio, abbiamo la domanda N.2 la cui risposta corretta è la 3, data la presenza di "OK" alla fine della risposta *(Vedere struttura domanda per dettagli implementativi)*.

L'utilizzatore del server per poter aggiungere domande deve usare questa formattazione **in coda** al file delle domande(separare le domande da una riga vuota!).

La modifica di una domanda deve sempre rispettare la formattazione della domanda nel file, mentre se si vuole rimuovere una domanda, bisogna prestare attenzione:

1. A lasciare una sola riga vuota tra una domanda e l'altra.
2. A modificare l'ID delle domande successive a quella eliminata (Se elimino la domanda N.8, la domanda N.9 deve diventare la N.8 e così per le successive).

Quando il client manda una risposta a una domanda, il server si occupa subito di riceverla e di aggiornare di conseguenza la classifica dei giocatori.

Classifica dei giocatori:

Il client prima di connettersi/iniziare la partita può consultare la classifica, inoltre, verrà mostrata in automatico alla fine di ogni partita.

È consultabile lato server solo dal file "classifica.txt".

Il calcolo della media è molto semplice: se la domanda è **corretta**, si prende il tempo impiegato a rispondere a quella domanda e si fa media con il numero di risposte corrette totali

Segue la formattazione del file e un esempio a confronto

Formattazione nel file:	**Esempio:**
IndirizzoIP(Num.Domande)=Media	127.0.0.1(2)=30

Nell’esempio troviamo un indirizzo IP locale, ogni macchina client che gioca ne ha uno diverso, pertanto, è utilizzato come identificativo univoco dei client.

La risposta corretta a due domande la cui media del tempo impiegato a rispondere è 30 secondi *(Per intenderci: a una domanda potrebbe averci messo 20 secondi l’altra 40, oppure ad entrambe ci ha messo 30 secondi ecc…).*

Ogni 60 secondi il server sarà pronto a ricevere richieste per la domanda successiva, e il client si occuperà di mandare tale richiesta; si continua così finché non ci sono più domande nel file e a quel punto la partita finisce, e sarà necessario riavviare i due applicativi per svolgere un’altra partita (Classifica e domande ovviamente persistono).

2. Protocollo applicativo tra client e server

Scopriamo come avviene la comunicazione tra client e server, senza discutere del linguaggio e degli strumenti utilizzati.

Quando Client e Server comunicano, la prima problematica che ci si è presentata è stata far capire, sia lato server che client come distinguere le tipologie di chiamate fatte (Es: Visualizzazione classifica, inizio nuova partita etc…).

Il protocollo attuato risolve egregiamente entrambe le problematiche, garantendo che le richiesta di un client vengono accolte dal server se e solo se lo stato della partita lo permette.

É necessario sapere se il server è capace di soddisfare qualsiasi tipo di richiesta (Numero giocatori massimi raggiunto, troppe richieste in coda), per far ciò il server manda una risposta affermativa (1) o negativa (0) in base alle necessità.

Successivamente, per ogni azione che il client può richiedere al server, quest’ultimo, verifica se la richiesta è pertinente rispetto alla situazione, rispondendo positivamente (1) o negativamente (0).

Quindi, il client riceve 1 se il server è connesso e quindi procede specificando l’azione che vorrebbe svolgere. Se anche quest’ultima richiesta riceve 1, il client aspetta il risultato del servizio richiesto, altrimenti manda un errore su STDOUT.

Ci sono 4 tipi di azioni che il client può chiedere al server:

1. *Next question* (*N*): richiedere la prossima domanda del gioco; l’azione è richiesta anche quando si tratta della prima domanda.
   Il server risponde positivamente se ci sono domande nel file ancora da far rispondere, altrimenti se risponde negativamente e termina la partita.

2. *Send answers*(*A*): mandare la risposta e il tempo impiegato per rispondere alla domanda corrente.

Se il client non risponde, verrà comunque mandata questa richiesta, con risposta "0".

3. *View leaderboard*(*C*): il client visualizza la classifica, il server risponde mandando la Classifica corrente.

4. *Start game*(*S*): Il Client si connette alla partita .
   Il server risponde positivamente se nella lobby ci sono posti disponibili (Massimo 10), negativamente altrimenti.
   Se già precedentemente è stata richiesta quest'azione, significa che la lobby è già stata creata, di conseguenza, il server restituirà il tempo rimanente per poter iniziare a giocare, oppure, se la partita è già in corso, restituisce il tempo rimanente per poter ricevere la domanda successiva (che per quel client sarà la prima).

Il "comando" S è più facile da spiegare con un esempio, dato che il suo output può significare più cose:

Supponiamo di avere 3 client che vogliono giocare una partita:

Il primo client richiede *S*, il server restituisce 60 (secondi), cioè il tempo necessario per far iniziare la partita, dopo 20 secondi un secondo client richiede il servizio *S*, il server risponderà 40 (secondi), cioè il tempo necessario per far iniziare la partita.

Infine, dopo 55 secondi da quest'ultima richiesta, un terzo client richiede *S*, il server risponde 45 (secondi), cioè il tempo che i client devono aspettare per ricevere la domanda successiva (per il terzo client sarà la prima, per i primi due è la seconda, poiché stanno già rispondendo alla prima, essendo la partita già iniziata).

Se l'utente finale è interessato, si può capire di essere entrato in una partita già in corso banalmente guardando il numero della prima domanda che riceverà, che per forza di cose sarà diverso da 1.

3.System call e dettagli implementativi

Server e client sono scritti in linguaggio C (std gnu11, gcc v.10.2.0), i due *"main",* uno client, l'altro server, sono rispettivamente Client.c e Server.c.

Come richiesto, questi programmi utilizzano soltanto librerie standard c.

Tuttavia, per rendere il codice più leggibile, nonché cercare di mantenere un livello di astrazione top-down, sono state scritte parti di codice sotto forma di cartelle da importare.

Un esempio è *la* libreria *"Question*", **importata sia lato client che server**, racchiude la definizione di una struct utilizzata per definire la domanda, accompagnata da metodi che

aiutano a creare, distruggere e visualizzare questa struttura, in modo da rendere chiaro, sia lato client che server, quando si ha a che fare con una determinata domanda.

QUESTION.C

Definizione della `struct domanda` :

```
1. Question* createQuestion(){
2.     Question* domanda=malloc(sizeof(Question));
3.     domanda->domanda=malloc(sizeof(char)*(MAX_QUESTION_LENGHT+1));
4.     domanda->N1=-1;
5.     domanda->N2=-1;
6.     domanda->N3=-1;
7.     domanda->N4=-1;
8.     domanda->N_OK=-1;
9.     domanda->num=-1;
10.      return domanda;
11.  }
```

Formato da : Il testo della domanda , risposta N1 – N2 – N3 – N4, la risposta esatta tra queste quattro ed infine il numero della domanda.

Segue una descrizione delle funzioni e system call ritenute più interessanti sia lato server che client:

- **Server:**

La prima funzione ad essere eseguita è `setupConn()`:

```
1. int setupConn(){
2.     struct sockaddr_in andress;
3.     andress.sin_family=AF_INET;
4.     andress.sin_port=htons(20000);
5.     andress.sin_addr.s_addr=htonl(INADDR_ANY);
6.     int sd;
7.     if((sd=socket(PF_INET,SOCK_STREAM,0))<0){
8.         fprintf(stderr,"Socket Errore: %s \n ",strerror(errno));
9.         exit(EXIT_FAILURE);
10.      }
11.      if(bind(sd,(struct sockaddr*)&andress,sizeof(andress)) <0){
12.          fprintf(stderr,"Socket Errore: %s \n ",strerror(errno));
13.          exit(EXIT_FAILURE);
14.      }
15.      if(listen(sd,PENDING_CONNECTION_QUEUE_LENGHT)<0){
16.          fprintf(stderr,"Socket Errore: %s \n ",strerror(errno));
17.          exit(EXIT_FAILURE);
18.      }
19.      return sd;
20.  }
```

La funzione definisce una `sockaddr_in`, una variabile che definisce una socket da utilizzare per il protocollo TCP/IP.

Dopodiché si usa la funzione `socket()`, che crea *una unbound socket* del dominio *PF_INET* (TCP/IP) e restituisce un *file descriptor* che ci permetterà di usare il *metodo bind*,

che associa al socket locale definito dal file descriptor un socket descriptor definito da una struttura come sockaddr_in; infine, con il metodo `listen()`, il server si mette in attesa di richieste da parte del client.

Il resto del programma si svolge all'interno della funzione `gestisciConnesione()`:

Quando un client richiede un servizio a un server, quest'ultimo accetta la richiesta con il metodo `accept()` e crea un processo figlio a cui affidare la specifica del tipo di richiesta (N,A,C o S) e relativa risoluzione.

Dopodiché, il processo padre risale nel ciclo, permettendo così ad altri client di richiedere servizi e non dover necessariamente aspettare la fine del precedente, magari fatto da un altro client.

```
1. void gestisciConnesione(int sd,…){
2. while(1){
3.          if((client_sd=accept(sd,(struct sockaddr*
4.          )&client_addr,(socklen_t*)&client_len))<0) {
5.              fprintf(stderr,"Socket Errore: %s \n ",strerror(errno));
6.              exit(EXIT_FAILURE);
7.          }
8.          int pid;
9.          int fd[2];
10.           pipe(fd);
11.           if((pid=fork())<0){
12.               fprintf(stderr,"Errore:",strerror(errno));
13.           }
14.           else if(pid==0) {
15.                //Processo figlio
16.                close(sd);
17.                close(fd[0]);
18.
19.                //Specifica del tipo di richiesta
20.                //Risoluzione
21.
22.                 close(client_sd);
23.                 close(fd[1]);
24.                 exit(EXIT_SUCCESS);
25.           }
26.           //Il padre deve solo aspettare richieste
27.           close(client_sd);
28.           close(fd[0]);
29.  }
30.  }
```

Da notare *la* `pipe` *fd*, creata prima della `fork`, in modo tale da creare un canale di comunicazione tra processo padre e figlio, dopo le opportune chiusure allo *stream.*

- **Client:**

Possiede 4 librerie :

- Question.h
- Answer.h
- Connection.h
- Game_Aritmetica.h

Le librerie "`Question.h`" e "`Answer.h`" vengono usate per la gestione delle domande e le risposte. Mentre "`Connection.h`" contiene i dati del Server come indirizzo, porta e gestisce la connessione al server. Infine abbiamo "`Game_Aritmetica.h`" , la libreria che gestisce le principali azione dell'utente.

ANSWER.C

Qui avviene la richiesta della domanda. Il fulcro sono principalmente due thread che vengono creati , `tid_askAnswer` e `tid_checkLimitTime`. Rispettivamente uno attende la risposta dall'utente e l'altro termina l'attesa della risposta una volta raggiunto il tempo limite (`WAIT_TIME`).

```
1. pthread_t tid_askAnswer, tid_checkLimitTime;
2. clock_t clockT;
3.
4. <...>
5.
6. int en; int tempo;
7.      if (en = pthread_create(&tid_askAnswer, NULL, askAnswer, NULL)!=0)
8.          errno = en , perror("create Thread") , exit(1);
9.      if(en = pthread_create(&tid_checkLimitTime,
10.                             NULL,
11.                             checkLimitTime,
12.                             NULL)     !=0 )
13.           errno = en , perror("create Thread") , exit(1);
14.
15.       pthread_join(tid_askAnswer,NULL);
16.       pthread_join(tid_checkLimitTime,NULL);
17.       tempo = (int) clockT;
```

CONNECTION.C

Utilizza una libreria dedicata alla Connessione al Server . Qui viene impostato la creazione del socket con `SOCK_STREAM` per un canale di trasmissione bidirezionale :

```
1. int setSocket(int *sd){
2.      *sd = socket(AF_INET, SOCK_STREAM, 0);
3.      return (*sd != -1 )? true : false ;
4. }
```

Viene specificata anche la struttura del `sockaddr_in` :

```
1. void setServAdress(struct sockaddr_in *srv_addr){
2.     *srv_addr = (struct sockaddr_in){
3.         .sin_addr.s_addr = inet_addr("93.144.54.26"), //indirizzo server
4.       .sin_family = AF_INET, //tipologia famiglia
5.       .sin_port = htons(20000) //porta del server
6.     };
7. }
```

Ci sarà anche una funzione `int syncro()` che ci permetterà di sincronizzare il Client con il Server ad ogni comunicazione.

Si prepara la connessione secondo questi passaggi:

1. Imposta il sockaddr
2. Creazione della Socket
3. Connect()

Ecco la funzione principale della libreria :

```
1. //Comunicazione con Server pronta
2. int startConnection(int* firstTime){
3.     int flag = false;
4.     if(*firstTime){
5.         printf("\nTentativo Connessione... ");
6.         fflush(stdout);
7.     }
8.     setServAdress(&srv_addr); // seetting dati del Server
9.     //creazione socket
10.      if(!setSocket(&sd)){
11.          printf(" Socket non create ");
12.          return flag; //esci con flag -->false
13.      }
14.      //collegamento con Server
15.      if(setConn(sd,&conn,srv_addr)){
16.          if(*firstTime){
17.              printf("Connessione eseguita");
18.              printf("\nSincronizazzione Server... ");
19.          }
20.          flag = syncro(); //sincronizzazione con Server
21.          if(*firstTime){
22.              (flag) ? printf("Server connesso\n"): printf("Server non
   connesso\n");
23.          }
24.      }else{
25.          printf("Connessione fallita\n");
26.          fflush(stdout);
27.      }
28.      *firstTime = 0;
29.      return flag;
30.  }
```

GAME_ARITMETICA.C

Il gioco viene gestito in questa libreria, dove ci sono i controlli per l'esecuzione delle principali azioni che l'utente può svolgere durante il gioco.

Viene creato un processo figlio nel quale viene fatta partire la connessione al Server per ogni azione Principale (N , A , C , S) che va comunicata al Server:

```
1.      int pid;
2.      if((pid=fork())<0){
3.          perror("\nFork error\n");
4.          exit(EXIT_FAILURE);
5.      }
6.
7.      if(pid != 0){
8.          //PADRE
9.          wait(NULL);
10.       }else{
11.           //FIGLIO
12.           int temp = 0;
13.           int flag = 1;
14.           int start = 1;
15.           int idQ;
16.
17.           do{
18.               if (startConnection(&start))
19.                   Azione(&action,&idQ,&temp,&flag);
20.               else {
21.                   printf("Connessione fallita\n");
22.                   action = EXIT;
23.               }
24.               closeConnection();
25.           }while(action != EXIT);
26.           exit(1);
27.       }
```

All'interno del ciclo verrà quindi chiamata la funzione `Azione(…)` dove viene poi eseguita l'Azione suddivisa nella seguente maniera:

```
1. void Azione(char* azione,int* idQ,int* temp,int* flag){
2.      switch (*azione){
3.          case MENU :
4.              getAction(azione);
5.              break;
6.          case START :
7.              startGame(temp);
8.              *azione = NEXT;
9.              break;
10.           case NEXT :
11.               doQuestion(idQ, azione,*temp,flag);
12.               break;
13.           case ANSWER:
14.               startAnswer(*idQ, temp);
15.               *azione = NEXT;
16.               break;
17.           case CLASSIFICA :
18.               doClassifica();
19.               if(*flag) *azione = MENU;
20.               else *azione = EXIT;
21.               break;
22.       }
23.  }
```

In ordine di come sono elencati i casi :

- Visualizzazione menu (`MENU`): viene chiesto all'utente cosa vuole fare : iniziare la partita, vedere la classifica, uscire
- Inizio partita (`START`) : fa partire il gioco ed imposta la prossima azione alla richiesta della prima domanda.
- Prossima domanda (`NEXT`) : Chiede al server la prossima domanda . Se esiste un'altra domanda la fa visualizzare. Altrimenti, se finiscono le domande , termina la partita e imposta come prossima azione la visualizzazione della Classifica.
- Rispondere alla domanda (`ANSWER`): viene fatto partire il tempo che l'utente ha per dare la risposta alla domanda che poi verrà inviata al server e fatta richiesta di prossima domanda.
- Visualizzazione Classifica (`CLASSIFICA`): stampa in STDOUT il file "classifica.txt".
  Se la partita è finita, stampa la classifica e termina il programma.

Divisione dei ruoli:

L'applicativo server e la libreria "Question" sono stati scritti interamente da Capasso Ivan.
L'applicativo client e le librerie "answer", "Connection", "Game_Aritmetica" sono state scritte interamente da Cristofor Doarme.