CORSO DI LAUREA IN INFORMATICA
INSEGNAMENTO DI LABORATORIO DI SISTEMI OPERATIVI
ANNO ACCADEMICO 2021/2022

Progettazione di un applicazione per la rilevazione di buche su strada

STUDENTI

Caccavale Mariano N86003303
mariano.caccavale@studenti.unina.it

Rossi Mattia N86003211
matti.rossi@studenti.unina.it

PROFESSORI

Grazioso Marco
marco.grazioso@unina.it

Scala Giovanni
giovanni.scala@unina.it

Questa documentazione è stata scritta in Latex

Indice

Capitolo 1

Descrizione del progetto

1.1 Traccia del progetto

Il sistema deve memorizzare e rendere disponibile una lista di eventi generati dai clients riguardanti la rilevazione di una un repentino cambiamento dell' accelerazione lungo l' asse verticale che superi una soglia comunicata dal server in fase di connessione. Ciascun client è identificato da un nickname, scelto dall' utente. Il sistema tramite il client deve offrire agli utenti i seguenti servizi:
1) Permettere all' utente di avviare una sessione di registrazione eventi durante la quale il client si connette al server, riceve i parametri di soglia e comunica al server la posizione e il valore del cambiamento ogni volta che registra un nuovo evento.
2) Mostrare all' utente la lista di tutti gli eventi registrati dal server in un certo raggio dalla propria posizione.

1.2 Guida al Client

Il client della nostra applicazione è stato reso lineare per facilitare i nuovi utenti all'uso immediato dell'app.
La prima schermata che si presenta all'utente ad ogni apertura dell'app è composta da un messaggio di benvenuto e la richiesta all'inserimento di un nickname.

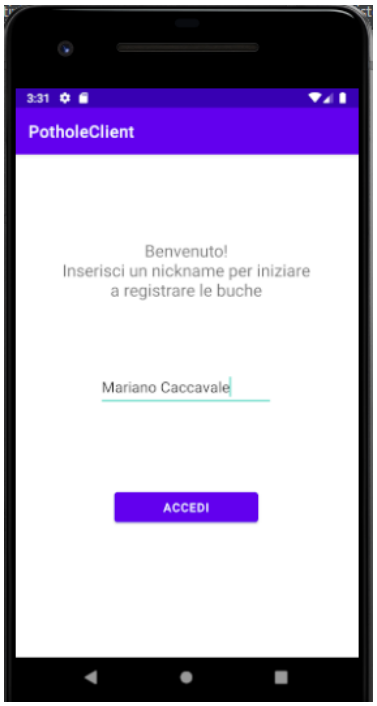

Figura 1.1: Schermata di apertura

Dopo aver digitato un nickname valido e premuto il tasto accedi si passa alla schermata principale dell'app dove si presenta un breve menù dove l'utente può selezionare la funzione da svolgere.

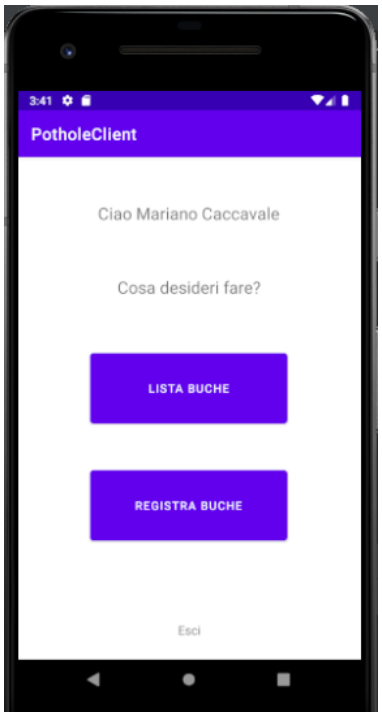

Figura 1.2: Schermata principale

Nel caso si scelga la funzione di registra buche, si aprirà una schermata dove verrà chiesto al client di calibrare il dispositivo per iniziare a registrare, dove veranno usati valori di treshold passati dal server al client.

Figura 1.3: Apertura della funzione di registrazione

Dopo che l'app abbia calibrato il dispositivo con successo, l'utente può iniziare la sessione di registrazione premendo sul pulsante di start.

Figura 1.4: Apertura della funzione di registrazione

Nel caso l'app rilevi una buca verrà visualizzato un piccolo messaggio di avviso per l'utente.

Figura 1.5: Rilevazione di una buca

L'altra funzionalità dell'app è invece la visualizzazione delle buche in un range prefissato dalla posizione del client, questo può dare due possibile risoluzioni della richiesta:

Figura 1.6: Nessuna buca nelle vicinanze

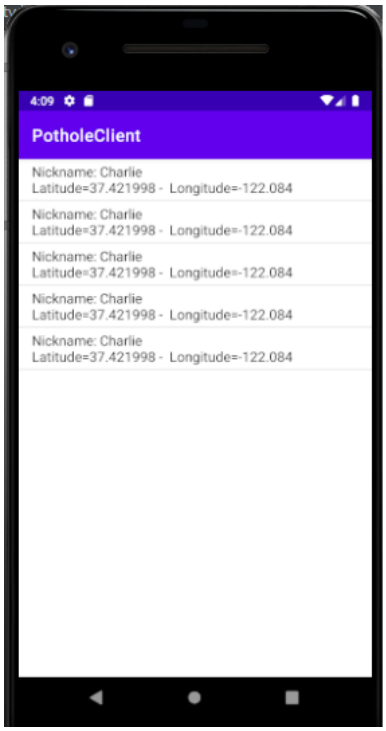

Figura 1.7: Lista di buche nelle vicinanze

1.3 Guida al Server

Il codice del Server (utilizzabile solo su piattaforma Linux) è composto da due cartelle: Client-Server e Database.
Nella prima sono presenti 5 file: initServer.c e initServer.out rispettivamente il codice sorgente e l'eseguibile della prima apertura del Server con creazione (nel caso non sia presente) del database nominato potholes.db. Gli ultimi due file sono server.c e server.out che, come i due file di initServer, sono di codice e di esecuzione del server dalla seconda volta di utilizzo.
Mentre nella cartella Database è presente il file sql.c e il suo relativo backup; tale file è una libreria fatta ad hoc utilizzata nel codice del server per l'interazione tra, appunto, il server e il database. Se il server viene aperto per la prima volta su un dispositivo, è necessario far eseguire il file initServer.out dove nel caso di corretta esecuzione creera il database contenente la tabella di registrazione delle buche incontrate dagli utenti. Dalla seconda esecuzione.

Capitolo 2

Protocollo applicativo

Abbiamo preferito l'uso del protocollo TCP a quello UDP per la presenza di controllo della congestione e affidabilità in termini di consegna messaggi. Per la creazione del protocollo siamo partiti dal Server, in quanto volevamo gettare la basi della comunicazione partendo dalla parte che ascoltasse.
Per via della loro semplicità in C, il Server gestisce le richieste dei Client attraverso Threads, poichè ciò permette al Server di servire diverse richieste in modo concorrente, senza però smettere di ascoltare le connessioni in entrata. Di conseguenza anche il Client adotta un sistema di networking che sfrutta i Threads, considerando anche, ma non solo, l'impossibilità di eseguire operazioni di networking sul Main Thread.
La prima connessione tra Client e Server avviene dopo l'inserimento del nickname, dove il Client manda richiesta al Server di ricevere valori di tolleranza (usati durante la sessione di registrazione buche). Delle tre richieste che il Client fa al Server l'unica diversa è l'acquisizione di buche, dove il Client crea una connessione con il server non all'inizio della sessione di registrazione, ma solo dopo il rilevamento di una buca per impedire che possono avvenire timeout improvvisi durante la registrazione delle buche.

Capitolo 3

Dettagli implementativi

La panoramica sui dettagli implementativi si suddivide in due sezioni: La prima riguarda il codice del Server, con un occhio di riguardo alle system call usate, mentre la seconda ricopre il codice del lato Client e la classe usata per la connessione Client-Server.

3.1 Lato Server

Questo estratto del codice racchiude tutte le system call usate più importanti, quali:

- signal: ci permette di definire un handler per una specifica tipologia di segnale.
- socket: creazione dell'oggetto di tipo socket.
- bind: funzione che assegna ad una socket le informazioni a corredo.
- listen: funzione che permette ad una socket di ascoltare attivamente.
- accept: funzione che accetta le connessioni in entrata.
- pthreadcreate: funzione che assegna alla struttura thread una funzione e la esegue.

3.1.1 connessione

```
1  //Dichiaro l'handler del segnale USR1, in modo da avere un modo 'safe' di
       chiudere il server
2      signal(SIGUSR2,signUSRHandlet);
3
4      int clientSocket, new_sock;
5
6      //Struttura dati sockets
7      struct sockaddr_in server , client;
8
9      char hostname = "0.0.0.0";
10
11     if( ( socketDescriptor = socket(AF_INET, SOCK_STREAM, 0) ) < 0 ){
12
13         perror("Errore nella creazione della socket\n");
14         exit(1);
15
16     }
17
18     //Assegno i valori alla struttura 'server'
19     server.sin_family = AF_INET;
20     server.sin_addr.s_addr = inet_addr(hostname);
21     server.sin_port = htons(PORT_N);
```

```
22
23
24      //Avverto il sistema operativo che voglio che la socket sia '
            riconosciuta' e entri in funzione
25      if( ( bind(socketDescriptor, (struct sockaddr) &server, sizeof(server
            )) ) < 0 ){
26
27          perror("Errore nel binding\n");
28          exit(1);
29
30      }
31
32      printf("[] Binding effettuato con successo\n");
33
34    listen(socketDescriptor, 10);
35
36      printf("[] Sono in ascolto su %s : %d. Attendo nuove connessioni...\n
            ", hostname, PORT_N);
37
38      int c = sizeof(struct sockaddr_in);
39
40
41      while( (clientSocket = accept(socketDescriptor, (struct sockaddr)&
            clientSocket, (socklen_t*)&c) ) > 0){
42
43  if( pthread_create(&thread, NULL, threadFunction, (void*) new_sock) < 0 )
        {
44
45              perror("Errore nella creazione di un thread\n");
46
47          }
48
49  void signUSRHandlet(int signal){
50
51      printf("[] Ricevuto segnale USR, procedo alla chiusura del server.\n"
            );
52      close(socketDescriptor);
53      printf("[] Server out!\n");
54      exit(0);
55
56  }
```

3.1.2 getTolerance

Snippet di codice che viene eseguito se il Server rileva la richiesta della tolleranza da parte del Client.

```
1  else if(strcmp(requestBuffer, "toll") == 0){
2
3                  send(socketDesc, "0.000002;", 8, 0);
4
5          }
```

3.1.3 sendData

Lettura dei dati inviati dal Client con successivo inserimento degli stessi all'interno del Database.

```
oid postRequest(int socketDescriptor){

    char readBuffer[1000];
    int byteLetti = 0, byteScritti = 0;

    //Riferimento al database in cui andr ‘o ad inserire i dati
    sqlite3* database;

    //Leggo fin quando ha dati da mandarmi
    recv(socketDescriptor, readBuffer, 2000, 0);
    //printf("%s\n", readBuffer);


    char* field = strtok(readBuffer, ";");
    char nickname[20];
    char latitude[20];
    char longitude[20];
    int charN = 0;

    while(field != NULL){

        if(charN == 0){
            strcpy(nickname, field);
        }else if (charN == 1){
            strcpy(latitude, field);
        }else if (charN == 2){
            strcpy(longitude, field);
        }
        charN++;

        field = strtok(NULL, ";");

    }

    //Chiedo alla funzione di aprire il database chiamato ’potholes’
    int status = sqlite3_open_v2("potholes.db", &database,
        SQLITE_OPEN_READWRITE | SQLITE_OPEN_NOMUTEX, NULL);

    if(status == SQLITE_OK){

        status = insertPothole(database, nickname, atof(latitude), atof(
            longitude));

        if(status == SQLITE_OK){

            printf("[*] Inserimento avvenuto con successo\n");

        }else{

            printf("[!] Inserimento fallito - %d\n", status);

```

```
50          }
51
52      }else{
53
54          printf("[!] Apertura del database fallita\n");
55
56      }
57
58      //Mi congedo da client
59      printf("[*] Comunicazione terminata, chiudo la connessione\n");
60
61      sqlite3_close(database);
62
63      return;
64
65  }
```

3.1.4 receiveData

Interrogazione che il Server esegue sul Database data la posizione di un Client e conseguente invio dellla eventuale lista di buche eseguito dalla funzione getPotholes in un range fissato dal Server.

```
1
2   void getRequest(int socketDescriptor){
3
4       char readBuffer[1000];
5       int byteLetti = 0, byteScritti = 0;
6
7       //Riferimento al database in cui andr ‘o ad inserire i dati
8       sqlite3* database;
9
10      //Leggo fin quando ha dati da mandarmi
11      recv(socketDescriptor, readBuffer, 2000, 0);
12      //printf("%s\n", readBuffer);
13
14
15      char* field = strtok(readBuffer, ";");
16      char nickname[20];
17      char latitude[20];
18      char longitude[20];
19      int charN = 0;
20
21      while(field != NULL){
22
23          if(charN == 0){
24              strcpy(nickname, field);
25          }else if (charN == 1){
26              strcpy(latitude, field);
27          }else if (charN == 2){
28              strcpy(longitude, field);
29          }
30          charN++;
31
32          field = strtok(NULL, ";");
```

```
33
34      }
35
36      printf("Dati su cui sto andando a fare la ricerca: %s - %f - %f\n",
            nickname, atof(latitude), atof(longitude));
37
38      //Chiedo alla funzione di aprire il database chiamato 'potholes'
39      int status = sqlite3_open_v2("potholes.db", &database,
            SQLITE_OPEN_READWRITE | SQLITE_OPEN_NOMUTEX, NULL);
40
41      if(status == SQLITE_OK){
42
43          status = getPotholes(database, atof(latitude), atof(longitude),
                socketDescriptor);
44
45          if(status == SQLITE_OK){
46
47              printf("[*] Ricerca avvenuta con successo\n");
48
49          }else{
50
51              printf("[!] Ricerca fallita - %d\n", status);
52
53          }
54
55      }else{
56
57          printf("[!] Apertura del database fallita\n");
58
59      }
60
61      //Mi congedo da client
62      printf("[*] Comunicazione terminata, chiudo la connessione\n");
63
64      sqlite3_close(database);
65
66      return;
67
68  }
```

3.2 Lato Client

3.2.1 Lato Client

Funzione che viene invocata dall'utente per ricevere una lista di buche nelle sue vicinanze.

```
1   public static LinkedList<PotholesModel> receiveData(String myNickname,
        double myLatitude, double myLongitude){
2
3           LinkedList<PotholesModel> resultList = new LinkedList<>();
4
5           try {
6               Socket socket = new Socket(Costants.ip, Costants.port);
7
8               String get = "get";
```

```
            socket.getOutputStream().write(get.getBytes());

            BufferedReader stdIn =
                    new BufferedReader(
                            new InputStreamReader(socket.getInputStream()))
                                ;

            String dati = myNickname+";"+myLatitude+";"+myLongitude;
            socket.getOutputStream().write(dati.getBytes());

            Thread.sleep(500);

            String responseBuffer;
            while(stdIn.ready()){
                responseBuffer = stdIn.readLine();
                responseBuffer = responseBuffer.replace("\u0000", "");
                if(responseBuffer.isEmpty()){
                    break;
                }
                String[] fields = responseBuffer.split(";");
                String nickname = fields[0];
                double latitude = Double.parseDouble(fields[1]);
                double longitude = Double.parseDouble(fields[2]);

                resultList.add(new PotholesModel(nickname, latitude,
                    longitude));

            }
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
        return resultList;
    }
```

3.2.2 Lato Client

Questa funzione viene invocata da parte del Client subito dopo l'accesso dell'utente per recuperare i valori di tolleranza dal Server.

```
public static double getTolerance(){

        String getTolerance = "toll";
        String toleranceString = "0.000000";

        try {
            Socket socket = new Socket(Costants.ip, Costants.port);
            BufferedReader stdIn =
                    new BufferedReader(
                            new InputStreamReader(socket.getInputStream()));

            socket.getOutputStream().write(getTolerance.getBytes());

            if(stdIn.ready()){
                toleranceString = stdIn.readLine();
                toleranceString = toleranceString.replace("\u0000", "");
```

```
                toleranceString = toleranceString.replace(";", "");
            }

            socket.close();

        } catch (IOException e) {
            e.printStackTrace();
        }

        return Double.parseDouble(toleranceString);
    }
```

3.2.3 Lato Client

Funzione che viene invocata automaticamente ogni volta che il Client rileva una buca, creando un messaggio contenente i dati convertiti in formato byte.

```
public static void sendData(String nickname, double latitude, double
    longitude){

        //Creo il socket a cui mandare i dati
        try {
            Socket socket = new Socket(Costants.ip, Costants.port);

            //Debug
            //String dati = "Charlie;78.000015;96.369874";

            String post = "post";
            socket.getOutputStream().write(post.getBytes());

            String dati = nickname+";"+latitude+";"+longitude;
            socket.getOutputStream().write(dati.getBytes());

            socket.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```