



**FCTUC** FACULDADE DE CIÊNCIAS  
E TECNOLOGIA  
UNIVERSIDADE DE COIMBRA

## **Relatório**

### **Projeto de IRC**

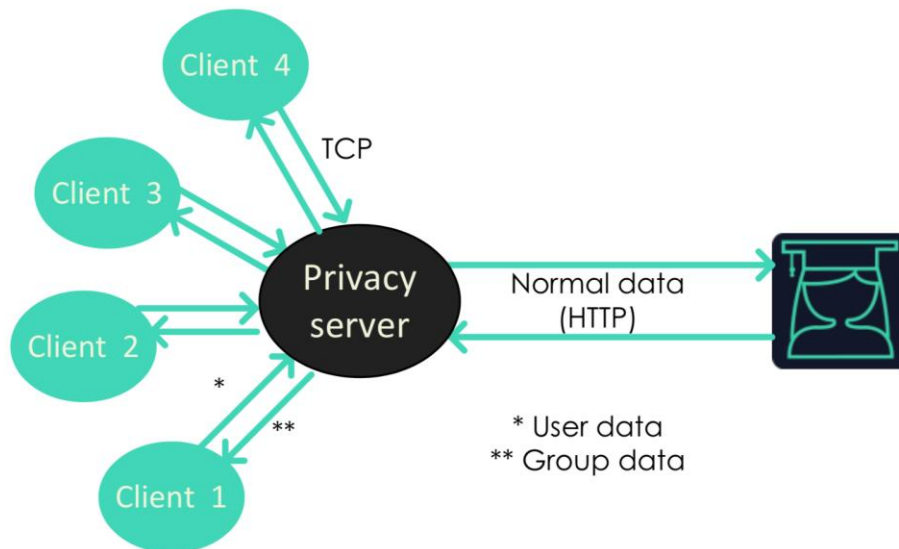
(IoT Student Advisor and Best Lifestyle Analyzer)  
Introdução às Redes de Comunicação  
Licenciatura em Engenharia Informática

Carolina de Castilho Godinho - 2017247087  
Maria Paula de Alencar Viegas - 2017125592

---

## Introdução

A partir do contexto da “Internet das Coisas”, proteção de dados e privacidade do usuário, criámos neste projeto, um servidor capaz de aceder e manipular dados fornecidos pela aplicação ISABELA acerca da qualidade de vida de estudantes usuários. Este servidor também comunica com clientes de forma privada, ou seja, um usuário só é capaz de visualizar os seus próprios dados e os dados gerais do grupo.



## Manual do Utilizador

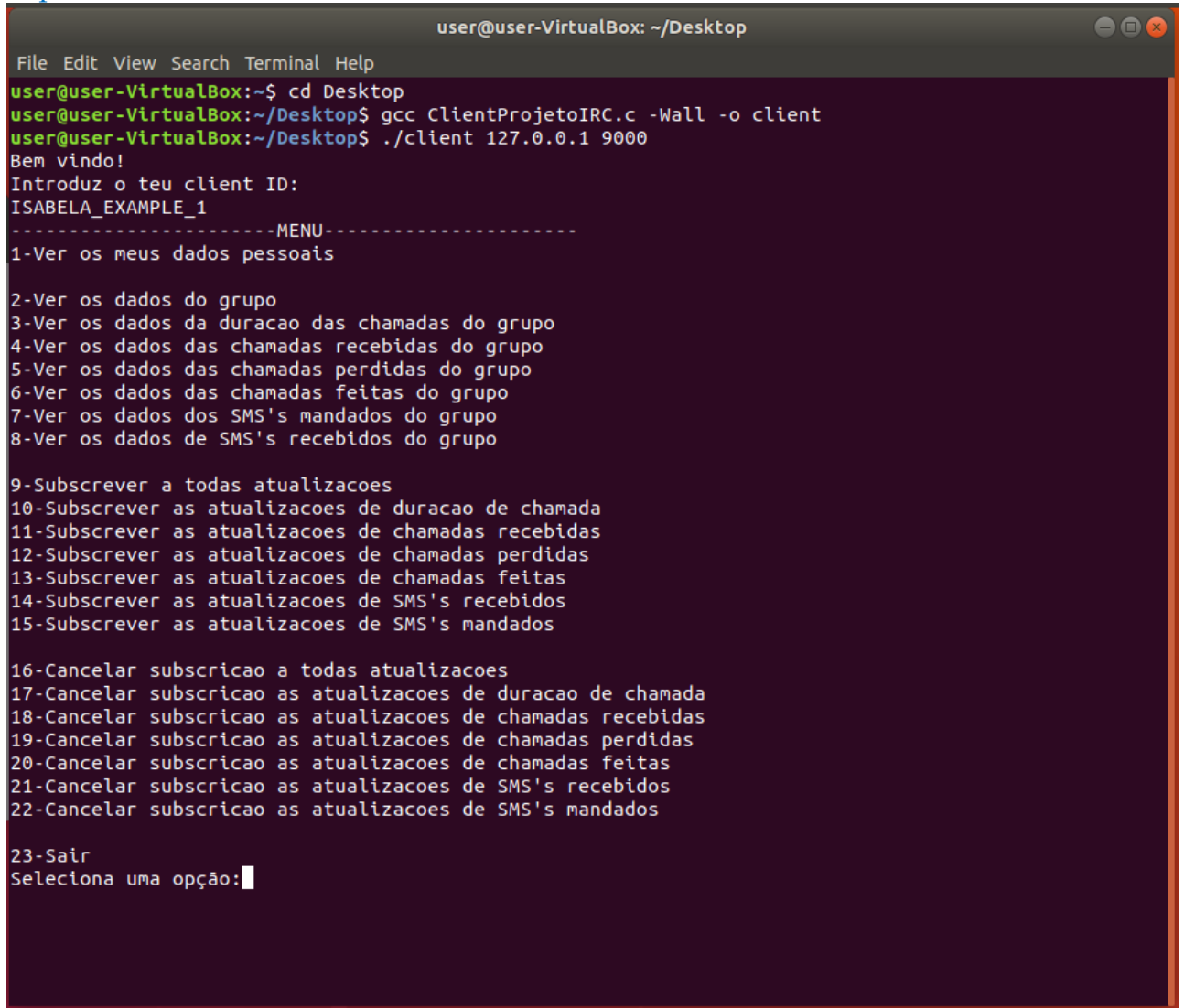
No terminal, um usuário, após se registrar com um ID válido (que se encontre na base de dados da ISABELA), pode consultar dados pessoais (**opção 1**) como atividade, localização, duração e quantidade de chamadas feitas, além da quantidade de chamadas recebidas e perdidas nos últimos 5 segundos, nome do departamento e quantidade de SMS's recebidos e mandados nos últimos 5 segundos. Também pode ver as estatísticas do grupo (**opção 2**) quanto SMS's recebidos e mandados e ligações feitas, recebidas, perdidas e duração.

Caso queira ver apenas as estatísticas de um dado específico, disponibilizamos as **opções 3** (duração das chamadas), **4** (chamadas recebidas), **5**, (chamadas perdidas) **6** (chamadas feitas), **7** (SMS's mandados) e **8** (SMS's recebidos).

Disponibilizamos ainda, a possibilidade do usuário se inscrever a todos os dados (**opção 9**) ou a um específico como duração das chamadas, chamadas recebidas, chamadas perdidas, chamadas feitas, SMS's mandados ou SMS's recebidos (respectivamente, **opção 10, 11, 12, 13, 14, 15**) e de retirar as respectivas subscrições (**opção 16, 17, 18, 19, 20, 21, 22**), sendo atualizado quando as estatísticas do grupo mudam.

A **opção 23** permite que o cliente se retire sem afetar o servidor.

Exemplo de funcionamento:



```
user@user-VirtualBox: ~/Desktop
File Edit View Search Terminal Help
user@user-VirtualBox:~$ cd Desktop
user@user-VirtualBox:~/Desktop$ gcc ClientProjetoIRC.c -Wall -o client
user@user-VirtualBox:~/Desktop$ ./client 127.0.0.1 9000
Bem vindo!
Introduz o teu client ID:
ISABELA_EXAMPLE_1
-----MENU-----
1-Ver os meus dados pessoais
2-Ver os dados do grupo
3-Ver os dados da duracao das chamadas do grupo
4-Ver os dados das chamadas recebidas do grupo
5-Ver os dados das chamadas perdidas do grupo
6-Ver os dados das chamadas feitas do grupo
7-Ver os dados dos SMS's mandados do grupo
8-Ver os dados de SMS's recebidos do grupo
9-Subscrever a todas atualizacoes
10-Subscrever as atualizacoes de duracao de chamada
11-Subscrever as atualizacoes de chamadas recebidas
12-Subscrever as atualizacoes de chamadas perdidas
13-Subscrever as atualizacoes de chamadas feitas
14-Subscrever as atualizacoes de SMS's recebidos
15-Subscrever as atualizacoes de SMS's mandados
16-Cancelar subscricao a todas atualizacoes
17-Cancelar subscricao as atualizacoes de duracao de chamada
18-Cancelar subscricao as atualizacoes de chamadas recebidas
19-Cancelar subscricao as atualizacoes de chamadas perdidas
20-Cancelar subscricao as atualizacoes de chamadas feitas
21-Cancelar subscricao as atualizacoes de SMS's recebidos
22-Cancelar subscricao as atualizacoes de SMS's mandados
23-Sair
Seleciona uma opção:█
```

## Estrutura

Para realizarmos o proposto, optámos a utilização do sistema operativo Linux e compilador gcc. Utilizamos também os documentos fornecidos no GitHub para interagir com a base de dados da ISABELA.

Usando a linguagem de programação C, fizemos os programas **ClientProjetoIRC.c** e **ServerProjetoIRC.c** (deve ser compilado com as flags `-lcurl` e `-ljson-c`).

### ClientProjetoIRC.c

Este é o programa que interage com o utilizador, cria o socket e que se conecta ao servidor para receber informações da base de dados fornecida pela aplicação ISABELA.

A função **miniclient()** é responsável pela criação de um novo socket de cada subscrição e também por receber dados do grupo atualizados.

```
423 //FUNCAO DOS PROCESSOS DE SUBSCRICAO DO CLIENT
424 void miniclient(char *argmc,int port0, int num)
425 {
426     char endServer[100];
427     char buffer[BUF_SIZE];
428     int nread = 0;
429
430     struct sockaddr in addr;
431     struct hostent *hostPtr;
432
433     strcpy(endServer, argmc);
434     if ((hostPtr = gethostbyname(endServer)) == 0)
435         erro("Nao consegui obter endereço");
436
437     bzero((void *) &addr, sizeof(addr));
438     addr.sin_family = AF_INET;
439     addr.sin_addr.s_addr = ((struct in_addr *) (hostPtr->h_addr))->s_addr;
440     addr.sin_port = htons((short)port0);
441
442     if((shared_mem->fd2[num] = socket(AF_INET,SOCK_STREAM,0)) == -1)
443         erro("socket");
444     if( connect(shared_mem->fd2[num],(struct sockaddr *)&addr,sizeof (addr)) < 0)
445         perror("Connect");
446     if(num==0) //Guardar os pid's na shared memory, para mais tarde conseguir eliminar
447         shared_mem->child_pid[0]=getpid();
448     else if(num==1)
449         shared_mem->child_pid[1]=getpid();
450     else if(num==2)
451         shared_mem->child_pid[2]=getpid();
452     else if(num==3)
453         shared_mem->child_pid[3]=getpid();
454     else if(num==4)
455         shared_mem->child_pid[4]=getpid();
456     else if(num==5)
457         shared_mem->child_pid[5]=getpid();
458     else if(num==6)
459         shared_mem->child_pid[6]=getpid();
460
461     //No client os processos de subscricao vao estar sempre a ler o que vem do server
462     while(1)
463     {
464         nread = read(shared_mem->fd2[num],buffer,BUF_SIZE-1);
465         buffer[nread] = '\0';
466         printf("%s\n", buffer);
467     }
```

A função **erro()** disponibiliza no terminal a mensagem recebida e encerra o programa. É utilizada para a deteção de erros. Há uma de funcionamento e nome igual em **ServerProjetoIRC.c**.

```
void erro(char *msg)
{
    printf("Erro: %s\n", msg);
    exit(-1);
}
```

É utilizada memória partilhada para guardar os pid's de cada processo que criámos no array **child\_pid**, também guardamos uma variável **elimina** para referência dos arrays associados às subscrições a cancelar pelo utilizador. Por fim necessitámos do uso do array **fd2** para assegurar uma destruição segura dos sockets estabelecidos nas subscrições.

A decisão da criação de memória partilhada foi necessária para a possibilidade de cancelar as subscrições feitas pelos utilizadores.

```
typedef struct shm{
    pid_t child_pid[8];
    int elimina;
    int fd2[7];
}shm;
int shm_id;

shm* shared_mem;
```

child_pid[0]	pid do processo de subscrição geral
child_pid[1]	pid do processo de subscrição de duração de chamadas
child_pid[2]	pid do processo de subscrição de chamadas recebidas
child_pid[3]	pid do processo de subscrição de chamadas perdidas
child_pid[4]	pid do processo de subscrição de chamadas feitas
child_pid[5]	pid do processo de subscrição de sms's mandados
child_pid[6]	pid do processo de subscrição de sms's recebidos
child_pid[7]	pid do processo principal

Para executá-lo: ./client <host> <porto> (ex: ./client 127.0.0.1 9000).

### ServerProjetoIRC.c

Este é o servidor privado responsável por aceder aos dados da ISABELA, manipulá-los e transmiti-los ao utilizador. Cria-se um socket e liga-se ao cliente.

Foram utilizadas funções, bibliotecas e structs fornecidas no GitHub como writefunc(), init\_string(), struct string, struct json\_object, curl/curl.h, json-c/json.h.

A função **main()** deste ficheiro criará um processo filho (**process\_client**) para cada cliente que queira estabelecer ligação com o servidor. Uma decisão tomada na criação deste novo processo foi a atribuição de um número de portos específicos para cada cliente poder usar para a criação das suas subcrições (linha 180 da imagem abaixo).

```
159 int main() {
160     int fd, client;
161     struct sockaddr_in addr, client_addr;
162     int client_addr_size;
163
164     bzero((void *) &addr, sizeof(addr));
165     addr.sin_family = AF_INET;
166     addr.sin_addr.s_addr = htonl(INADDR_ANY);
167     addr.sin_port = htons(SERVER_PORT);
168
169     if ( (fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) //Cria o socket
170         erro("na funcao socket");
171     if ( bind(fd,(struct sockaddr*)&addr,sizeof(addr)) < 0) //Liga o socket
172         erro("na funcao bind");
173     if( listen(fd, 5) < 0) //Até 5 utilizadores em espera
174         erro("na funcao listen");
175
176     while (1) {
177         client_addr_size = sizeof(client_addr);
178         client = accept(fd,(struct sockaddr *)&client_addr,(socklen_t *)&client_addr_size);
179         if (client > 0) {
180             novo_porto=novo_porto+20; //portos reservados para as subscicoes de cada cliente
181             if (fork() == 0) {
182                 close(fd);
183                 process_client(client,novo_porto);
184                 exit(0);
185             }
186             close(client);
187         }
188     }
189     return 0;
190 }
```

Caso a conexão tenha sido bem estabelecida com o cliente, inicializamos a função **process\_client()**, que manda as informações ao cliente mediante write()'s e read()'s. Também faz cálculos de médias.

A função **existe\_pessoa()** verifica se encontra o id fornecido pelo cliente nos dados da ISABELA.

Para cada subscrição feita será criado um novo processo pelo **process\_client** incrementando +1 ao **novo\_porto**, ou seja, cada subscrição feita terá um porto específico.

```
430 else if(strcmp(option,"9")==0 || strcmp(option,"10")==0 || strcmp(option,"11")==0 || strcmp(option,"12")==0 || strcmp(option,"13")==0 || strcmp(option,"14")==0 || strcmp(option,"15")==0)
431 {
432     if(subscrito[0]==0 && strcmp(option,"9")==0)
433     {
434         printf("Um cliente subscreveu-se as informações do grupo\n");
435         write(client_fd,"Subscreveste-te as informações do grupo!\nSerás notificado cada vez que houver uma alteração",BUF_SIZE-1);
436         subscrito[0]=1;
437         novo_porto++;
438         write(client_fd,buffer,BUF_SIZE-1);
439         if(fork()==0)
440         {
441             miniserver(novo_porto,option, 0); //é mandado o numero 0 que é a posição desta subscrição nos arrays da shared memory
442             exit(0);
443         }
444     }
445     else if(subscrito[0]==1 && strcmp(option,"9")==0)
446     {
447         write(client_fd,"Já te encontras subscrito!",BUF_SIZE-1);
448     }
449     else if(subscrito[1]==0 && strcmp(option,"10")==0)
450     {
```

Quando um cliente pretende retirar a sua subscrição (**opções 16, 17, 18, 19, 20, 21, 22**) uma mensagem é enviada ao utilizador a avisá-lo do ocorrido e alteram-se as variáveis específicas para a eliminação da subscrição e recorreremos à função **quit\_sub()** (função apronfundada mais á frente neste relatório) para a eliminação segura do processo e socket respetivo à subscrição a ser eliminada.

```
561 else if(strcmp(option,"16")==0 || strcmp(option,"17")==0 || strcmp(option,"18")==0 || strcmp(option,"19")==0 || strcmp(option,"20")==0 || strcmp(option,"21")==0 || strcmp(option,"22")==0)
562 {
563     if(subscrito[0]==1 && strcmp(option,"16")==0)
564     {
565         printf("Um cliente cancelou a sua subscricao\n");
566         write(client_fd,"A tua subscricao foi cancelada",BUF_SIZE-1);
567         subscrito[0]=0; //flag de subscricao (serve para verificar se o user está subscrito ou nao)
568         sleep(1); // sleep usado para que o processo do client seja terminado primeiro
569         shared_mem->elimina=0; //indica a posicao desta subscricao nos arrays
570         quit_sub(); //funcao que elimina o processo da subscricao
571     }
572     else if(subscrito[1]==1 && strcmp(option,"17")==0)
573     {
```

A função **miniserver()** será utilizada quando o usuário inserir que deseja subscrever-se. Ela cria um novo socket responsável por retransmitir as estatísticas do grupo atualizadas ao cliente. Calcula as médias, verifica se houve mudanças nelas num intervalo de 10 segundos e se houver, manda a informação atualizada pelo novo socket ao cliente.

A função **quit\_sub()** é usada para a eliminação das subscrições. Esta irá matar o respetivo processo e fechar o socket de uma determinada subscrição.

```
40 //funcao que elimina os processos das subscicoes e fecha os sockets
41 void quit_sub()
42 {
43     pid_t self = getpid();
44     if(shared_mem->child_pid[7]==self)
45     {
46         close(shared_mem->client2[shared_mem->elimina]);
47         kill(shared_mem->child_pid[shared_mem->elimina],SIGKILL);
48     }
49 }
```

Esquema de funcionamento dos processos:

