

UNIVERSIDADE DE ÉVORA



**REDES DE COMPUTADORES
2015/2016**

Discentes:

Pedro Jacob - 27421

André Figueira - 31626

Introdução

Neste trabalho foi nos pedido para implementar um cliente para o protocolo fornecido pelo docente.

Tendo três alternativas de implementação, **decidimos implementar o Cliente Simples e o Cliente Completo.**

Descrição

Quando um utilizador se conecta ao servidor é imediatamente pedido para introduzir um nome e uma password. Caso a password não seja introduzida o nome é tratado como um nome temporário, isto se o servidor o aceitar. Caso contrário o utilizador introduz a password correspondente ao nick.

Se o utilizador for aceite, este entra no chat. É subscrito automaticamente no canal @global, tendo a hipótese de sair deste ou até mesmo se juntar a outros.

O utilizador pode enviar mensagens, seja privadas ou para canais assumindo que estes existem. Caso contrário lhe é enviada uma mensagem de erro.

O utilizador pode gerir os tópicos de quaisquer canais. Se estes existirem, caso contrário é lhe enviada mensagem de erro

O utilizador pode enviar, receber, recusar, aceitar e cancelar ficheiros.

O utilizador pode adicionar amigos e este receberá notificação deste quando está no chat. Os amigos correspondem ao nome com um *. **Exemplo: “[PM] from friend* : hello”**

O utilizador pode durante a duração da ligação alterar o seu nome em que todos os utilizadores recebem notificação de tal (desde que estes estejam em canais que o utilizador está)

O utilizador pode registar nomes, opcionalmente pode escolher adicionar um email juntamente com a password para conseguir recuperar a password.

O utilizador pode saber quais salas a que pertence, quais os utilizadores em cada sala, etc...

O utilizador pode fazer todas as funcionalidades descritas no protocolo mais uns extras.

Respostas do servidor

O utilizador está sempre a receber mensagens do servidor. Isto acontece através do uso de Threads. Assim o utilizador pode mesmo sem estar a introduzir inputs estar a receber respostas do servidor. Visto que, no nosso ver, é o que achamos mais correcto pois se trata de um Chat.

Todas as respostas do servidor são reenviadas para os utilizadores de uma maneira completamente diferente. Seja relacionada com aceitação ou de erros.

Isto para que um utilizador que desconheça o protocolo saiba com facilidade o erro que foi causado ou mensagem que recebeu ou qualquer outra resposta enviada pelo servidor.

Ou seja para perceber com mais facilidade o que está acontecer.

Esta decisão foi feita para que o chat fosse mais “**user friendly**” e simples de perceber para qualquer utilizador.

Exemplo de entrada numa sala, mensagem privada e para canal e lista de utilizadores num canal:

```
Room name: @global
Room topic: friends

User user2 has entered channel @global
Users in room @global:
    user1
    user2

System: You have received a message, to reply /pm user1 <msg>
[PM] From user1 : hello friend!

System: You have received a message, to reply /msg @global <msg>
[@global] user1: hello friends!

System: Room @global topic has been changed to redes by user1

/inroom
Users in room @global:
    user1
    user2
```

Nota-se que as mensagens são completamente diferentes das que descritas no protocolo.

O que acontece é, faz-se uma verificação de qual a mensagem que o servidor está a enviar e esta é substituída por outra como visto acima.

Implementação

Sempre que se recebe um comando, este é feito um split que remove os espaços em branco e mete todos os outros elementos num array.

Para saber qual o comando certo, a primeira posição do array é sempre o comando,

Se um comando tiver, por exemplo uma password, ou seja, um parâmetro que poder conter várias palavras então percorre-se um ciclo a partir da primeira palavra desse parâmetro ate ao fim do array, adicionando tudo numa String de forma a obter a mensagem/password, etc.. completa, ou seja reconstroi-se o parâmetro, isto para se poder enviar para o servidor com o comando equivalente reconhecido pelo servidor.

```
public void commands(String userInput) throws IOException, InterruptedException
{
    String[] array = userInput.split(" ");

    if(array[0].equals("/help")) // get help
        help();

    else if(array[0].equals("/inroom")) // who is in the room
        seeRoom(array);

    else if(array[0].equals("/leave")) // leave the chat
        quit(array);

    else if(array[0].equals("/leaveroom")) // leave room X
        roomLeave(array);

    else if(array[0].equals("/register")) // register a nickname
        register_name(array);

    else if(array[0].equals("/changetopic")) // change the topic of the room
        changeTopic(array);

    else if(array[0].equals("/register")) // register current nick
        register_name(array);

    else if(array[0].equals("/changenick")) // change the nick
        change_nick(array);

    else if(array[0].equals("/pm")) // send a private message to user
        msg_private(array);
}
```

```

else if(array[0].equals("/enter")) // enter a room
    enter_room(array);

else if(array[0].equals("/roomlist")) // check rooms in chat
    seeEverything(array);

else if(array[0].equals("/userlist")) // check users in chat
    seeEverything(array);

else if(array[0].equals("/registerfriend")) // register a friend
{
    registerFriend(array[1]);
}
else if(array[0].equals("/sendrequest")) // send request
{
    Cenas = new File(array[2]);
    sendrequest(array[1],array[2]);
}

else if(array[0].equals("/send")){ // after accepted, send file
    send(array[1],filetobyte(Cenas));
}
else if (array[0].equals("/accept")){ // accept transfer request
    acceptfile(array[1]);
}
else if (array[0].equals("/end")){ // end of file (not used)
    endfile(array[1]);
}

else if (array[0].equals("/refuse")){ // refuse transfer request
    Refusefile(array[1]);
}
else if (array[0].equals("/abort")){ // cancel file
    Abortfile(array[1],array[2]);
}

else if (array[0].equals("/recover")){ // recover password
    recover_password(array);
}
else if (array[0].equals("/friendlist")){ // user friend list
    System.out.println("\nList of friends: \n");
    for(int i =0; i<Friendlist.size(); i++)
        System.out.println("\t"+Friendlist.get(i));
}
else if(array[0].equals("/belong")) // rooms the user belongs to
    belongsTo(array);
else if (array[0].equals("/msg"))// if nothing then its a message to the channel
    msg_channel(array);
else
    sendstuff(userInput); // any other command goes in here

```

Qualquer comando que não seja um dos possíveis, então é enviado para o servidor que devolve uma resposta.

```

/adeus
System: Error occurred, Invalid command "/adeus"

/redes
System: Error occurred, Invalid command "/redes"

```

Entrar no servidor

Quando um utilizador se conecta ao Chat é introduzido num ciclo infinito e é lhe pedido para introduzir um nome e uma password. Não tendo hipótese de usar outras funcionalidades do Chat até ter introduzido um nome.

Ao introduzir o nome verifica se o nome tem espaços, se tal se verificar pede novamente.

Caso contrário pede a password. A password é opcional, isto é o utilizador pode não introduzir a password juntamente com o nick, mas este nick passará a ser um temporário. Qual seja a decisão, envia-se um pedido ao servidor NICK <NOME> \<PASS> e dependendo da resposta o ciclo infinito ou continua a pedir novamente nome ou password(caso incorrecta) ou termina visto que foi aceite.

O utilizador é automaticamente subscrito no canal @global após entrar no Chat, e esta sala é adicionado à lista de sala a quais o utilizador pertence.

Após tal, O utilizador recebe uma mensagem a dizer quais os comandos e o que pode fazer no chat.

Depois é introduzido num ciclo infinito que pede constantemente inputs e só termina até que o input seja **/leave [<reason>]** ou seja **desconectado por uma outra razão**.

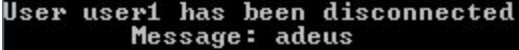
```
Available commands
Chat related:
  /pm <nome> <msg> - sends a private message to user <nome>
  /msg <channel> <msg> - sends message to channel <channel>
Registry related:
  /register [<email>] <password> - registers current name with password, w
ith optional email
  /chanenick <new_name> - changes user's name
  /recover - recovers a password from name sends it to email
  /registerfriend <name> - register <name> as friend
Channel Related:
  /changetopic [<room> <new_topic>] - changes the topic on room < room >,
if no parameters are given then shows a list and ask user to choose
  /enter [<room>] - if no parameter is given then shows list of rooms, pla
yer can choose to enter or create room <room>
  /leaveroom [<room> <message>] - leaves room <room>, if no parameters are
given, shows list and makes user choose
  /inroom [<room>] - returns a list of users in the room <room> if given t
he parameter, else gives the user list in all rooms he belongs to
File Related:
  /sendrequest <name> <file> - send request to <name>
  /send <descriptor> - send file /refuse <descriptor> - refuses the file
being sent to you
  /accept <descriptor> - accepts the file being sent to you
  /abort <descriptor> [<reason>] - aborts the file, you can give a reason
General:
  /userlist - returns a list of all the users in the chat
  /roomlist - returns a list of all channels in the chat
  /friendlist - returns user a list of his friends
  /help - all commands and descriptions
  /belong - returns all the rooms the user belongs to
  /leave [<message>] - leaves the chat with optional message
System: In short, here's what the user can do:
  1 - Register name and recover password
  2 - Change name
  3 - Send, accept, refuse, cancel files
  4 - Send messages to a channel or to a user (via private message)
  5 - Enter, leave rooms, see information about room and change their top
ic
  6 - See all the users in chat
  7 - See all rooms in chat
  8 - See all rooms user belongs to
  9 - Add friends
Above are all the commands and descriptions
```

Saída do servidor

Caso o utilizador não se tenha desconectado por uma outra razão qualquer (por exemplo perdeu a ligação) a forma directa de ser desconectado é usando o comando **/leave [<reason>]**.

Ao usar este comando é enviado um pedido ao servidor QUIT [<reason>] e termina o ciclo infinito causando (mudando o estado da variável booleana) o fecho da conexão ao servidor. O utilizador pode opcionalmente introduzir uma razão.

Todos os utilizadores e o utilizador em si recebem a mensagem:

A screenshot of a terminal window with a black background and white text. It displays two lines of text: "User user1 has been disconnected" and "Message: adeus".

```
User user1 has been disconnected  
Message: adeus
```

(Com mensagem)

Salas

O utilizador pode criar, entrar, sair e mudar o t pico de salas. Se num comando nao for introduzido um @ juntamente com o nome da sala, este   adicionado.

Entrar/Criar uma sala

Se o utilizador quiser entrar numa sala basta introduzir o comando /enter [<room>] Sendo <room> o nome da sala a qual quer entrar. Nota-se que este par metro   opcional, isto porque caso n o introduza uma sala ent o   lhe apresentado uma lista de salas existentes (fazendo o pedido ao servidor RLIST) e pedir que o utilizador introduza uma sala ou crie uma nova sala.

Qualquer que seja o aspecto do comando, se a sala n o existir ent o esta   criada, caso contr rio o utilizador entra na sala.

  enviado ao servidor o comando ENTER <room> que este indica o topic, nome, e lista de utilizadores da sala.

Ao entrar/criar uma sala esta sala   adicionada   lista de salas a qual o utilizador pertence, e todos os utilizadores nessa sala recebem notifica  o

```
User user has entered channel @global
```

Enquanto o utilizador que entrou recebe:

```
/enter Global
Room name: @Global
Room topic:

User user has entered channel @Global
Users in room @Global:

    users
    user
```

Sendo "user" o nome do utilizador.

Sair de uma sala

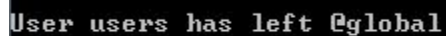
O utilizador pode sair de qualquer sala a que pertence, usando o comando `/leaveroom [<room> <message>]`.

Nota-se que o room e message novamente é opcional. Caso não seja introduzido uma sala nem uma mensagem então é lhe apresentado a lista de salas a que o utilizador pertence e lhe é pedido para escolher uma das salas e o motivo de saída (podendo neste não introduzir nada)..

Qualquer que seja o aspecto do comando, se a sala existir então ele é removido caso contrário recebe uma notificação do servidor com uma mensagem de erro.

É enviado ao servidor `LEAVE <room> [\<message>]`

Se a sala existir, ou seja se a sala pertence a lista de salas a que o utilizador pertence, então esta é removida desta lista. E todos os utilizadores da sala recebem a mensagem:



```
User users has left @global
```

(Sem mensagem)

Sendo “user” o nome da utilizador.

Mudar o tópico de uma sala

Mudar o tópico de uma sala é um processo simples. Usa-se o comando `/changetopic [<room> <topic>]`

Nota-se novamente que tem parâmetro opcionais. Se o utilizador não escolher introduzir a sala e topic então é lhe apresentado uma lista com todas as salas do Chat e lhe é pedido para escolher uma sala e o tópico (podendo neste não introduzir nada).

Qualquer que seja o aspecto, é enviado ao servidor o comando `TOPIC <room> [<topic>]` e todos os utilizador presentes no canal recebem uma mensagem que indica o utilizador que mudou e qual o novo tópico da sala.

```
/changetopic
List of existing rooms:
Room @TicTacToe
    Number of users: 1
    Topic:
Room @global
    Number of users: 2
    Topic:
Room @Default_channel
    Number of users: 1
    Topic:
Room @Geral
    Number of users: 2
    Topic:
Room @Default_R
    Number of users: 1
    Topic:
Room to change topic: global
Topic: redes
System: Room @global topic has been changed to redes by user
```

sendo user o nome do utilizador

Com parâmetro:

```
/changetopic global redes
```

Nota: Se não for introduzido um @ juntamente com a sala é adicionado.

Informação de salas

Ao usar o comando `/inroom [<room>]`, notam-se dois acontecimentos. Se introduzir `<room>` então é lhe apresentado uma lista com os utilizadores da sala, caso contrário é lhe apresentado uma lista com todos os utilizadores de cada sala a que pertence.

```
/inroom @global
Users in room @global:

    user
    users
```

Sem `<room>`:

```
/inroom
Users in room @global:

    user
    users

Users in room @Global:

    users
    user

Users in room @cenas:

    user

Users in room @teste:

    user
```

Ao introduzir o comando `/belong`, é efectuado um ciclo sobre a lista de salas a que o utilizador pertence em que faz um print de todas a salas a que pertence.

```
/belong
List of rooms you are in:

    Room: @global
    Room: @Global
    Room: @cenas
    Room: @teste
```

Mensagens

Existem dois tipos de mensagens de Chat.

Mensagem privada

Para mandar uma mensagem privada a um utilizador (podendo ser ele mesmo) basta introduzir o comando `/pm <nome> <msg>`, sendo `<nome>` o destinatário e `<msg>` a mensagem.

E é enviada uma mensagem ao servidor `MSG <nome> <msg>`.

Quem recebe a mensagem recebe neste formato:

```
System: You have received a message, to reply /pm user <msg>
[PM] From user : hello
```

E quem envia recebe neste formato:

```
/pm users hello
[PM] To users : ello
```

Sendo “user” e “users” nomes dos utilizadores

Mensagem de chat

Semelhante a mensagem privada, só que todos os utilizadores presentes no chat recebem a mensagem

Quem envia, recebe neste formato(pode enviar sem e com @) :

```
/msg @global hello
[@global] user : hello

/msg global hello
[@global] user : hello
```

Quem recebe, recebe neste formato:

```
System: You have received a message, to reply /msg @global <msg>
[@global] user: hello
```

Registo

O utilizador tem a hipótese de registar um nome ou mudar o seu nome actual.

Estes comandos podem ser utilizados a qualquer altura durante a ligação do utilizador.

Registar Nome

O utilizador tem de hipótese de, no momento que entrou no canal registar o seu nome que está a usar de momento de maneira permanente. Para tal executa `/register [e-mail] <password>`

O utilizador tem a hipótese de registar juntamente com a password um e-mail isto para ter a hipótese de recuperar a sua password.

Para saber se contem e-mail ou não, verifica-se se existe o carácter “@” na posição 2 do array ou seja e-mail, se sim então existe um e-mail juntamente com a password.

Se o comando conter menos de 2 argumentos, nesse caso dá erro pois não existem argumentos suficientes para o comando.

Caso tenha o numero de comandos correcto, envia uma mensagem ao servidor `REGISTER <nome> [<e-mail>] <password>`.

Exemplo:

```
/register test123@gmail.com 123456789
System: wat has been registered sucessfully
```

Sendo “wat” o nome do utilizador.

Mudança de nome

Para mudar o nick tem de utilizar o comando `/changenick <New_nome> send <new_nome>` o nome que deseja.

O que este comando faz é enviar um pedido ao servidor `NICK <new_nome>`.

Recover password

Para recuperar a password o utilizador tem de submeter inicialmente um e-mail quando se registou caso contrário receberá mensagem de erro do servidor, mesma situação se aplica se o utilizador não se registou.

Usando o comando `/recover`, é enviado um e-mail para o e-mail do utilizador.

Sem estar registado:

```
/recover  
System: User user1 is not a registered user
```

Após registar com e-mail:

```
/recover  
System: Recover sucessful. Password has been sent to email
```

Listas

Sem contar com `/belong` existem mais dois comandos que retornam listas.

Lista de Utilizadores do Chat

Usando o comando `/userlist`, envia um pedido ao servidor ULIST, que devolve ao utilizador uma lista de todos os utilizadores do Chat.

```
/userlist  
  
User list:  
  
    OneReportMan  
    TicTacBot  
    user1
```

(lista de pessoas presentes no chat naquele dado momento)

Lista de Salas do Chat

Usando o comando `/roomlist`, envia um pedido ao servidor RLIST, que devolve ao utilizador uma lista de todos os canais juntamente com o numero de utilizadores em cada e o topico.

```
/roomlist  
  
Room list:  
Room @TicTacToe  
    Number of users: 1  
    Topic:  
Room @global  
    Number of users: 2  
    Topic:
```


Transferência de Ficheiros

O cliente permite também o envio de ficheiros entre dois utilizadores. O mesmo está dividido em 3 partes: Request->Accept/Refuse->Send/Abort.

Request

Nesta fase, o utilizador que deseja mandar um ficheiro tem de pedir autorização ao utilizador “alvo” para o fazer. Para tal, o utilizador deve usar o seguinte comando:

```
/sendrequest <nome do utilizador> <nome do ficheiro>
```

Exemplo:

```
/sendrequest wat C:/HaxLogs.txt
```

Este comando manda a seguinte mensagem para o servidor:

```
SENDFILE <nickname> <filesize> \<filename>
```

O utilizador que deseja enviar recebe a seguinte notificação:

```
System: Sending request to wat
```

```
System: User wat sending 406c76d7-17f5-4635-9aae-e3559b6442c4 \C:/HaxLogs.txt has been accepted by the server
```

Em sua parte, o utilizador para qual o request foi enviado recebe a seguinte mensagem:

```
System: User Userino is sending file. Details: File Size: 93  
Descriptor To Use:406c76d7-17f5-4635-9aae-e3559b6442c4  
File Name: C:/HaxLogs.txt  
System: To accept /accept <descriptor> to refuse /refuse <descriptor>
```

Accept/Refuse

Nesta fase, o utilizador que recebeu o request tem duas opções. Aceitar ou Recusar.

Para Aceitar deve usar o comando:

`/accept <descriptor>`

Tal que <descriptor> é a String gerada pelo servidor para identificar a operação.

Exemplo:

```
System: To accept /accept <descriptor> to refuse /refuse <descriptor>
User pink has entered channel @global
/accept be445fa3-1f60-4b11-af9f-be64acec0112
```

Assim, é mandada a seguinte mensagem para o servidor:

`ACCEPTFILE <filenum>`

E, o utilizador que envia recebe a notificação:

```
System: File has been accepted
```

Em caso de recusar , o utilizador deve usar o comando:

`/refuse <filenum>`

Por exemplo:

`/refuse be445fa3-1f60-4b11-af9f-be64acec0112`

Send/Abort

Em caso de o request ter sido aceite, o utilizador pode então mandar o ficheiro com

`/send descriptor`

Por exemplo

`/send be445fa3-1f60-4b11-af9f-be64acec0112`

O mesmo vai chamar a função `send`, que por sua vez chama `fileToByte()`, que recebe o ficheiro que pretende enviar, e passa para um array de bytes. A função `send`, então, usa um ciclo para mandar os bytes usando o comando do protocolo `SENDPART`. No final do ciclo, confirma que o ficheiro foi enviado com `ENDFILE`.

O cliente de quem está a receber então, ao receber a notificação de final de ficheiro do servidor, chama a função `byteToFile()` que transforma novamente num ficheiro.

O utilizador que está a receber, pode então abortar a transferência usando `/abort <descriptor>`.

Friendlist

O cliente contém também uma friendlist, guardada em disco num ficheiro chamado Friends.txt. Ao iniciar o cliente, este é carregado para memória, e ao fazer qualquer notificação que envolva nomes de utilizadores, se o mesmo tiver contido na friendlist aparece com um * a frente do nome.

Pode também fazer os seguintes comandos referentes a friendlist

/registerfriend <nome>

Regista um nome como amigo

```
/registerfriend Relatorio
```

```
System: Friend Added! YAY FRIENDSHIP!!
```

/friendlist

Devolve a lista actual dos amigos.

```
List of friends:
```

```
operino
KEKBUR
EPAPRONTIO
KEKBUR
operino
KEKBUR
EPAPRONTIO
KEKBUR
Relatorio
```

Comandos

Como já explorado no resto do relatório, o utilizador tem acesso a um número fixo de comandos, qualquer comando enviado que não é um destes é enviado para o servidor e este retorna uma mensagem de erro.

```
/redes  
System: Error occurred, Invalid command "/redes"
```

Estes comandos estão divididos por secções:

- Chat
- Registo
- Canal
- Ficheiro
- Gerais.

Comandos de chat

/pm <nome> <msg> - Envia uma mensagem privada para o utilizador <nome>

/msg <channel> <msg> - Envia uma mensagem para o canal <Channel>

Comandos de registo

/register [<email>] <password> - Regista o nome actual com a password <password> e com um email opcional

/changenick <new_name> -Muda o nick do utilizador para <new_name>

/recover - recupera a password mandando-a para o email registado anteriormente.

/registerfriend <name> - regista um amigo com o nome <name>

Comandos de Canal

`/changetopic [<room> <new_topic>]` - Muda o tópico do canal `<room>` para `<new_topic>`

`/enter [<room>]` - Muda o utilizador para o canal `<room>`. Se não forem inseridos parâmetros, uma lista é dada e o utilizador escolhe.

`/leaveroom [<room> <message>]` - Permite ao utilizador sair o canal `<room>` e deixar a mensagem opcional de `<message>` com mensagem de saída. Se nenhum canal for especificado nos parâmetros, é dada uma lista para o utilizador escolher

`/inroom [<room>]` - retorna uma lista dos utilizadores no canal `<room>`. Se nenhum canal for especificado nos parâmetros então é dada uma lista para o utilizador escolher

Comandos de Ficheiro

`/sendrequest <name> <file>` - Manda request para o utilizador `<name>` para enviar o ficheiro `<file>`

`/send <descriptor>` - Envia o ficheiro

`/refuse <descriptor>` - Recusa o ficheiro a ser enviado

`/accept <descriptor>` - Aceita o ficheiro a ser enviado

`/abort <descriptor> [<reason>]` - Aborta a transferencia de ficheiro com a razão opcional de `<reason>`

Comandos Gerais

/userlist - retorna a lista de todos os utilizadores

/roomlist - retorna a lista de todos os canais

/friendlist - retorna a lista de amigos

/help - retorna a lista de todos os comandos possíveis e descrições

/belong - retorna a lista de todos os canais nos quais o utilizador entrou

/leave [<message>] - sai do chat com uma mensagem opcional