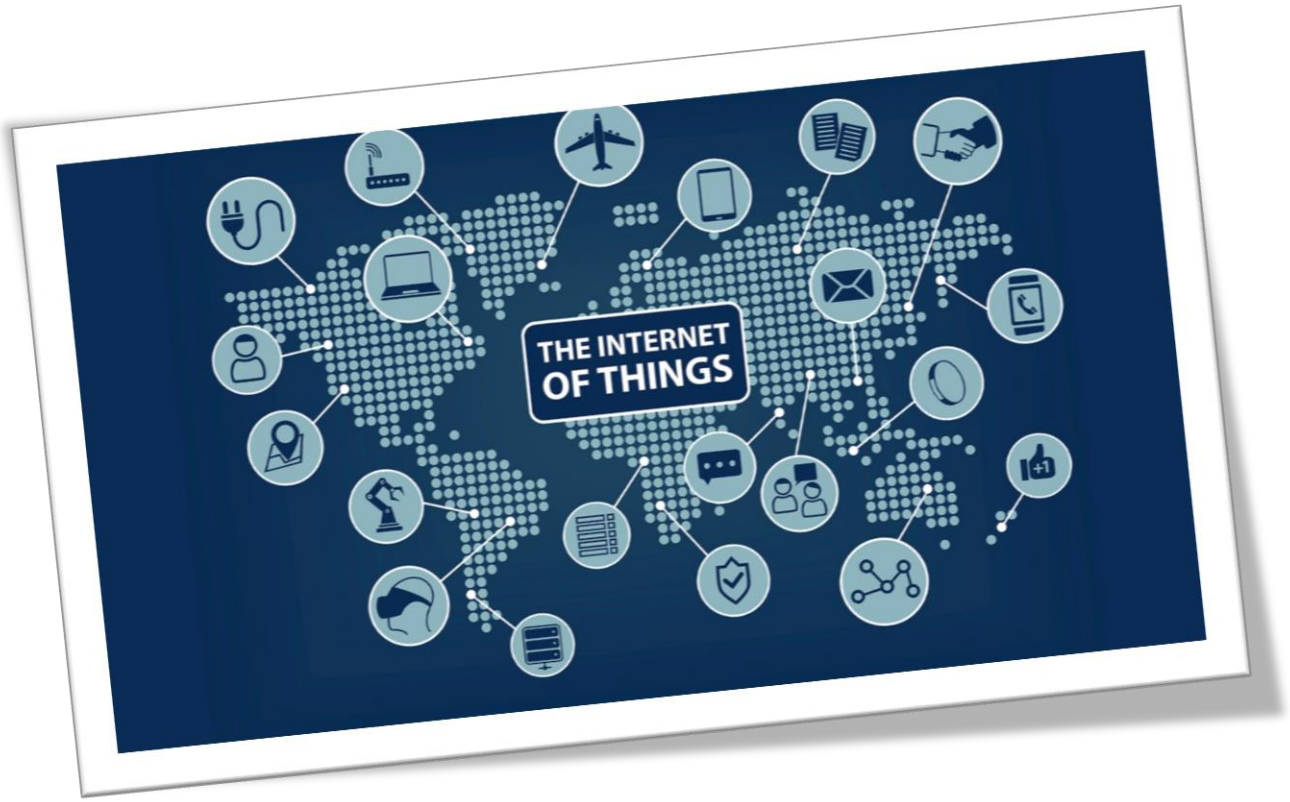
FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

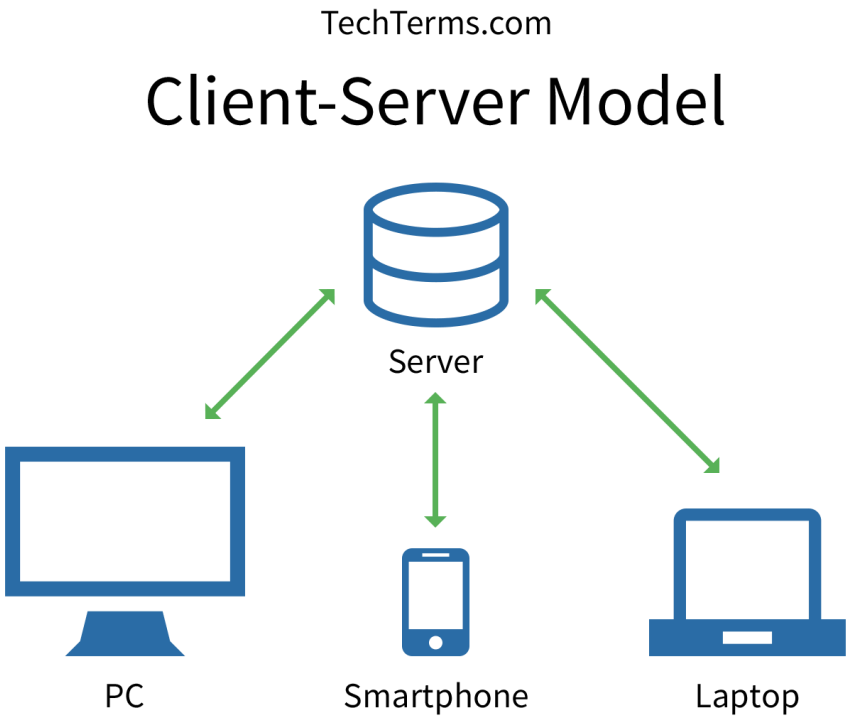
Relatório do Projecto de Introdução Às Redes de Comunicação



Trabalho realizado no âmbito da cadeira de Introdução às redes de comunicação por:

- Miguel Paulo Martins Marques nº2017266263
- Paulo Tomás Falcão de Almeida Cardoso nº2017249716

Índice



Introdução

A entrega deste projecto, elaborado no âmbito da cadeira de Introdução às Redes e Comunicação, possui mais três ficheiros para além deste documento: dois ficheiros de código na linguagem de programação "Python" e uma imagem auxiliar da interface gráfica implementada, obrigatoriamente na mesma pasta que os ficheiros de código para que o programa possa ser executado.

Para fornecer um pouco de contexto, o trabalho que fomos encarregues de produzir funcionaria em sinergia com uma aplicação desenvolvida no Departamento de Engenharia Informática da Universidade de Coimbra, ISABELA, um acrónimo para *IoT Student Advisor and Best Lifestyle Analyser*. Esta aplicação tem como intuito explorar o crescente paradigma da Internet das coisas, recolhendo e tratando dados sobre o estilo de vida dos utilizadores, neste caso estudantes. A característica distintiva desta iniciativa é que a aplicação tem como objectivo tomar uma atitude proactiva relativamente aos dados estatísticos que obtém, ao contrário dos sistemas já existentes, cujo propósito seria o apenas o estudo dos dados, sem nenhuma interferência no estilo de vida dos utilizadores. Assim, a aplicação procura alertar o utilizador para os dados recolhidos, através de um chat bot, de modo a conseguir contribuir para o seu processo de tomada de decisões, e assim contribuir positivamente para o melhoramento do seu estilo de vida.

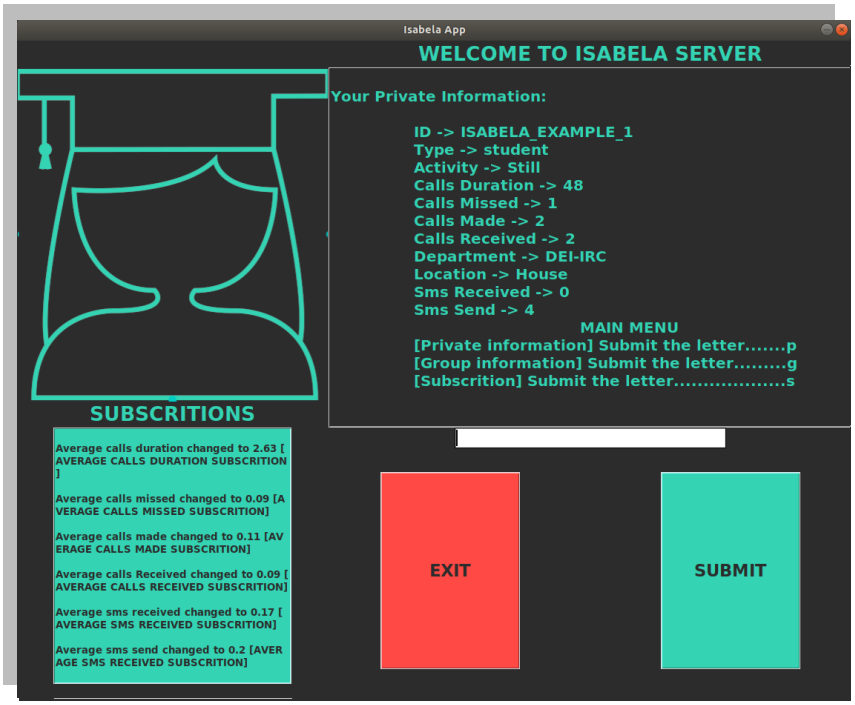
Este projeto incumbia-nos então de estabelecer ligações entre vários clientes e um servidor, onde estes conseguiriam obter os dados relevantes da aplicação. Mas, sendo este servidor privado, cada utilizador poderia aceder apenas à sua informação e à chamada informação de grupo (sendo os dados processados de modo a garantir o anonimato de todos os membros pertencentes ao grupo, enviando apenas uma média das várias categorias disponíveis no http da ISABELA, sem qualquer envio dos dados dos utilizadores). Teríamos também de implementar uma funcionalidade chamada "subscrições", que visaria notificar o utilizador (passado um determinado intervalo de tempo) quando as médias dos parâmetros a que o mesmo se tivesse subscrito se alterassem na nossa base de dados.

Relativamente à Linguagem de programação escolhida, optámos por elaborar o nosso programa em Python, versão 2.7.15rc1, uma linguagem de programação inserida no cânone das "Linguagens Orientadas aos Objectos", criada pelo Holandês Guido van Rossum, e lançada em 1991. Escolhemos esta linguagem, mesmo tendo aprendido os conceitos desta matéria em C nas aulas, pois ambos expressámos uma vontade de aprender este conceitos numa linguagem orientada a objetos, mas também pelo facto de Python ser uma linguagem com uma sintaxe, concernente às redes de comunicação, bastante streamlined em comparação com C ou até Java.

Para além do que era pedido no enunciado, elaborámos também uma interface gráfica (importando a biblioteca Tkinter) de maneira a melhorar a estética do trabalho e de facilitar o uso desta mesma aplicação. Esta interface possui duas caixas de texto: uma para a comunicação entre cliente e servidor e outra que serve unicamente para

mostrar a informação relativa às subscrições, recebendo actualizações periódicas caso algum valor na base de dados seja alterado, não interferindo, deste modo, com a comunicação entre o servidor e o cliente.

Para comunicar então com o servidor, o cliente terá apenas de escrever na barra de texto em branco no lado direito da interface. No momento de inicialização esta caixa estará pronta admitir valores para login, aceitando apenas os ID presentes no http da ISABELA. O utilizador pode executar os seus comandos clicando no botão "SUBMIT" ou pressionando apenas a tecla "Enter". Para sair, a única opção disponível é o botão "EXIT". Por fim, queríamos apontar que todo o projeto foi elaborado e pensado em Linux, pelo que não garantimos que o mesmo funcione sem problemas noutros sistemas operativos (especialmente se estes não tiverem instaladas as utilidades de "Python" necessárias).



Ligação Servidor Cliente

Em primeiro lugar, para uma aplicação deste tipo considerámos obrigatório o uso de uma ligação com um socket TCP (Transmission Control Protocol), uma vez que qualquer perda de informação seria crítica para o bom funcionamento e desempenho do programa. Falhas no envio dos pacotes do Cliente para o Servidor resultariam numa tentativa falhada de login na aplicação, uma vez que o ID recebido pelo servidor poderia estar corrompido. No sentido contrário, poderia resultar no envio de informações erradas aos clientes por parte do servidor, e o processamento de dados falsos.

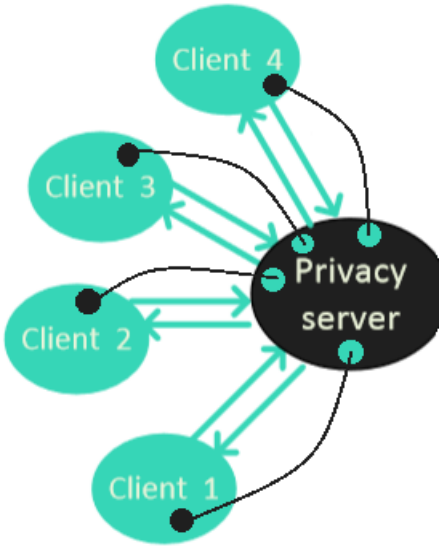
Numa fase inicial do desenvolvimento do servidor, servindo-nos do protocolo "telnet" através da consola de comandos de Linux para testar esta porção do programa. Seguidamente, passamos à fase de desenvolvimento do cliente.

Neste segmento do trabalho, decidimos conectar os sockets de cada cliente a um único socket do servidor, como é prática comum neste tipo de ligações.

O funcionamento específico do código relativo a esta porção do trabalho encontra-se explicado posteriormente na secção do manual do programador.

Subscrições

No que toca às subscrições, a nossa abordagem, ainda que não sendo a mais simples, foi a que considerámos fazer mais sentido no nosso programa, e foi também a implementação sugerida pelo professor. Assim, esta consiste no seguinte:



Dentro de cada cliente é criado um servidor, que vai estar à escuta de conecções de subscrição para estabelecer a ligação desejada, e, desta maneira, não vai interferir com o socket que se liga ao servidor (Privacy server) com os restantes clientes. É de notar que nesta ligação o socket é igualmente do tipo TCP e, no que toca ao porto, este tem que ser diferente dos já utilizados anteriormente. Para tal, guardamos todos os portos já usados numa lista, e geramos um porto aleatório entre 1024 e 49151, pois são estes os portos registados. Após verificar que este novo porto não costa na lista de portos já criados, este é enviado ao Privacy server para que se consiga conectar e é criado finalmente o servidor dentro do cliente.

Deste modo, podemos constatar que o número total de sockets criados para as subscrições é igual ao dobro do número de clientes, um por cliente pois é o socket da subscrição e um socket por cada ligação criado no Privacy server. Mais uma vez, entendemos que poderíamos aplicar esta porção do trabalho de maneira diferente, ligando as sockets de subscrição de cada cliente a uma única socket no servidor, mas como já havíamos seguido esta estratégia para a ligação principal entre cliente e o servidor, decidimos seguir a sugestão do professor e implementar esta solução um pouco mais redundante.

Manual do utilizador

No momento em que o utilizador se encontra perante a interface gráfica, este tem de efectuar o login, escrevendo o seu ID da ISABELA, existente na base de dados do API da aplicação, e validando-o. Assim que valide o seu ID, este depara-se com o menu principal. Neste menu o utilizador tem 3 opções: ver a informação pessoal (escrevendo a letra p), a informação de grupo (escrevendo a letra g), e finalmente a gestão das subscrições (escrevendo a letra s). A submissão das opções já foi explicada anteriormente na secção da interface.

Para terminar o servidor, deve-se utilizar o botão “EXIT”.

Dentro da parte da gestão das subscrições, existe um submenu. Neste, o utilizador depara-se com todas as opções de subscrição. Inicialmente nenhuma das

subscrições está activa (estado “NOT-SUBSCRIBED”). Para ativar estas subscrições, o utilizador terá de seleccionar um número de 1 a 6, correspondente à subscrição desejada, sendo que o número 7 serve unicamente para poder abandonar o submenu das subscrições. Quando o utilizador submeter um dos dígitos, o estado da subscrição respectiva passará a “SUBSCRIBED”, e a partir desse momento receberá um notificação de minuto a minuto, caso o valor médio do dado em questão se alterar. Para retirar subscrições basta premir o mesmo número para obter o estado “NOT SUBSCRIBED”.

O utilizador pode sair do programa sempre que desejar, para tal basta premir o botão vermelho nomeado de “EXIT”.

Manual do programador

isabela_server.py

Funções

- **catch_ctrl_c(sig, frame)**
- **readapi(url,headers,lista)**
- **atualizapi(url,headers)**
- **atuliza_users_api()**
- **subsclientthread(my_user,subsport)**
- **clientthread(connection,addr)**
- **main()**
- **if __name__ == '__main__'**

Classes

- **User**

  #### Funções dentro da classe

  - **__init__()**
  - **do_subscription(self,atual_avg,ssubs)**
  - **setAverage(self,average)**
  - **displaySubscrition(self,reply,connection)**
  - **displayCallSmsInfo(self)**
  - **displayEveryting(self)**

  #### Parâmetros da classe

  - **SUBS_calls_duration**
  - **SUBS_calls_missed**
  - **SUBS_calls_made**
  - **SUBS_calls_received**
  - **SUBS_sms_received**
  - **SUBS_sms_send**

Isabela_client.py

Funções

- **interface()**
- **ignore()**
- **leave()**
- **enter_key(event)**
- **click()**
- **gotoserver()**
- **catch_ctrl_c(sig, frame)**
- **substhreadserver()**
- **if __name__ == '__main__'**

Assim como foi mencionado previamente na introdução, o desenvolvimento deste projeto foi concretizado com a linguagem de programação híbrida, Python. Foram utilizadas as seguintes ferramentas dentro da linguagem:

- Listas
- Dicionários
- Classes
- Threads
- Sockets
- Interface gráfica (Tkinter)
- Sinais

Funcionamento das funções de isabela_server.py

- **if __name__ == '__main__'**

Nesta função, para além de serem declaradas as variáveis globais, é estabelecido o host, no endereço "localhost"(127.0.0.1), da nossa máquina (servidor interno, apenas acessível dentro da mesma). É também criado o socket principal que vai permitir a conectividade entre todos os clientes e o sinal SIGINT é redireccionado para a função catch_ctrl_c.

Por fim são inicializadas algumas listas para utilizar e modificar com os dados relevantes ao longo programa.

- all_sockets -> Para guardar todos os sockets criados no programa;
- used_ports-> Para guardar todos os portos criados no programa;
- users_list -> Para guardar todos os dados presentes na API;
- url -> Contém o link que permite acesso aos dados dos utilizadores;
- headers -> Permite o acesso ao url;

- **main()**

```
while 1:
    connection, addr = s.accept()
    print("Connected with "+ addr[0] +":"+ str(addr[1]))
    start_new_thread(clientthread,(connection,addr, ))
```

Esta função, em primeiro lugar, acede ao endereço url fornecido pelos professores para aceder às bases de dados do API da ISABELA, e coloca em users_list os dados de todos os utilizadores da aplicação Isabela. Este ciclo vai assegurar ligações simultâneas de diversos clientes com a implementação de threads. Assim, o servidor estará à escuta de ligações, e no momento em que ocorrer uma conecção este criará uma thread para lidar com os pedidos específicos desta ligação, não prejudicando outras funcionalidades. Enviará depois a thread para a função "clientthread" que receberá como parâmetros de entrada a "connection", que vai permitir fazer o envio e a receção de mensagens para os clientes, e o "addr", que utilizamos para imprimir na consola um aviso quando este se desconectar.

- **clientthread(connection,addr)**

Resumidamente, esta função vai assegurar a comunicação entre clientes e servidor, respondendo devidamente aos "requests" do cliente. A informação proveniente do socket vem codificada, logo temos de fazer o "decode" da informação. No entanto, Python já possui um método para isto mesmo dentro dos objetos (strings) criados pelo método "recv" de uma conecção. Este objeto possui a informação em bytes que recebe no socket, e, no nosso caso, considerámos suficiente o envio

```
data = connection.recv(1024)
reply = data.decode()[0:-1]
if not data:
    break
```

de 1024 bytes (1Kb). Da informação recebida (já descodificada) retiramos o último elemento (que corresponde ao caracter "_"), deste modo evitamos o envio de informação "vazia" pelo socket.

Algo que pode parecer confuso é o código presente linha 379: a instrução "if" para a mensagem recebida "3478GFDgu4w639". Esta foi a string única que pré-definimos de modo a indicar que o cliente pretende sair do programa, ou seja, que clicou no botão "EXIT".

- **subsclientthread(my_user,subsport)**

Esta função irá receber como parâmetros de entrada: a classe com a informação de um certo cliente, e o porto onde se poderá conectar para realizar as suas subscrições. Sucintamente, esta função estabelece uma conecção com o servidor presente dentro do cliente, que fará a gestão das subscrições, enviando (caso o cliente tenha subscrições ativas), em períodos de um minuto, as informações relativas às suas subscrições.

```
while check:
    new_avg = atualizapi(url,headers)
    check = my_user.do_subscription(new_avg,ssubs);
    time.sleep(60)
```

- **atualiza_users_api()**

Esta função serve para realizar a atualização dos dados presentes na API fornecida, verificando os dados dos clientes, sempre que um deles acede aos dados, certificando-se que a nossa lista de estruturas cliente tem as informações mais recentes.

Esta função retorna assim a lista de informação dos utilizadores atualizada.

- **atualizapi(url,headers)**

Esta função recebe como parâmetros de entrada a url http://socialiteorion2.dei.uc.pt:9014/v2/entities?options=keyValues&type=student&attrs=activity,calls_duration,calls_made,calls_missed,calls_received,department,location,sms_received,sms_sent&limit=100 e um dicionário com os seguintes headers: no-cache, / e socialite. Estes permitem que o nosso programa aceda à base de dados que contêm a informação relativa aos utilizadores da aplicação ISABELA. É de notar que esta função gere apenas com os dados estatísticos (a informação do grupo), e nada mais, devolvendo em "return" um objecto da classe "User" chamado media, com a informação de grupo.

- **readapi(url,headers,lista)**

Função encarregue de obter os dados dos utilizadores provenientes do API da aplicação. Utilizada também dentro da função "atualiza_api()", de modo a compartimentalizar o código e a possibilitar a reutilização desta função. Esta devolve

então em “return” uma lista (users_list) de objectos da classe “user”, que possuem a informação disponível de cada utilizador da aplicação, lista essa inicializada anteriormente dentro da função “main()”.

- **catch_ctrl_c(sig, frame)**

Esta função serve para executar a terminação controlada do terminal (é para esta que redireccionamos o sinal SIGINT). Aqui é utilizada a variável “all_sockets”, presente em memória global, uma lista com todos os sockets criados no decorrer do programa. Desta maneira, temos a certeza de que todos os sockets são fechados devidamente.

- **Funções dentro da classe “user”:**
  - **__init__(…)**

    Construtor da classe. Serve unicamente para inserir as variáveis necessárias dentro da classe (variáveis estas existentes em todos os seus objectos).

  - **do_subscription(self,atual_avg,ssubs)**

    Esta função está encarregue de enviar a informação que diz respeitante às subscrições de cada cliente. Executa em primeiro lugar uma verificação, de modo a averiguar quais as subscrições activas (isto é, quais os dados sobre os quais o utilizador pretende receber notificações), e após essa verificação, confirmará se estes dados sofreram alterações. Caso se verifiquem essas alterações, a função enviará para o cliente os novos valores. Caso contrário, nenhuma informação é enviada. Em return, é devolvido um boolean: True se o cliente ainda estiver ativo, e False se este já se tiver desconectado. Neste último caso, esta função não volta a ser chamada.

  - **setAverage(self,average)**

    Este método da classe user, tem a funcionalidade de atualizar os dados relativos à média, ou, por outras palavras, à informação de grupo.

  - **displaySubscrition(self,reply,connection)/displayCallSmsInfo(self)/ displayEveryting(self)**

    Todas estás funções têm o propósito de enviar ao cliente a informação relativa às subscrições, informação de grupo ou informação individual respetivamente.

Funcionamento das funções de isabela_client.py

- **if __name__ == '__main__'**

Nesta função, além de declararmos as variáveis globais, estabelemos também o host, sendo este um "localhost"(127.0.0.1). É também criado o socket principal permitirá a conectividade entre o servidor e o cliente, o handler responsável por redirecionar o sinal SIGINT para a função catch_ctrl_c.

Por fim são inicializadas algumas varáveis indispensáveis ao bom funcionamento do programa:

- subsport -> Lista onde serão guardados todos os portos utilizados nas subscrições;
- output_info -> string que possui as informações presentes na janela de texto da comunicação entre o servidor e o cliente, visualizada na interface;
- text_input -> Possui a informação de texto que vai ser enviada ao servidor;
- text_subs -> Possui a informação de texto que vai estar na janela de texto das subscrições da interface;
- janela -> É a variável utilizada na criação e funcionamento da interface gráfica.

- **interface()**

Função responsável pela interface gráfica, desde o design aos botões, janelas de texto output e input, imagens e título. Esta função é executada por uma thread.

- **ignore()**

Assim como o nome sugere, é uma função responsável por ignorar algo, neste caso servirá de apoio ao términus do programa, pois o sinal SIGINT será ignorado, e esta será precisamente a função para a qual este será redireccionado.

- **leave()**

Brevemente, esta é a função que estabelece a string de conclusão do programa, previamente explicada ("3478GFDgu4w639_"). É responsável também por terminar o servidor e o cliente.

- **enter_key(event)**

Função de apoio à interface que permitirá que a tecla "enter" tenha a mesma funcionalidade que o botão "SUBMIT".

- **click()**

Esta função está encarregue de transmitir ao servidor tudo o que o cliente escrever na caixa de texto na sua comunicação com o servidor.

- **gotoserver()**

Esta função está encarregue da receção de toda a informação enviada pelo servidor ao cliente (excluindo a informação relativa às subscrições). Esta função está a ser executada pela thread principal, e está constantemente à escuta (sem espera ativa) de informações provenientes do servidor, comunicando-as diretamente à interface com o auxílio da variável output_info.

```
while True:
    reply = s.recv(4096)
    print(reply.decode(),end="")
    if not reply:
        break
    if reply.decode() == "Exit\n":
        s.close()
        sys.exit()
    output_info.insert(END,reply)
```

- **substhreadserver()**

Por fim temos a função gerida pela thread das subscrições, que tem como propósito singular gerir todos os métodos e dados relativos a esta funcionalidade, inclusive a criação do socket que servirá de servidor de subscrição dentro do cliente. Após a sua criação, este esperará que o servidor principal crie uma thread que se ligue com ele. A partir daí, todas as informações recebidas por este socket serão os dados específicos da subscrição na sua incumbência, e assim, este enviará toda a informação relevante para a caixa de texto respectiva na interface do cliente.

Conclusão

Na elaboração deste projecto, a nossa primeira decisão foi a de definir uma estratégia global de rigor na pesquisa da matéria em questão. Procurámos não só desenvolver um trabalho funcional, mas, acima de tudo, experimentar com conceitos novos, priorizando uma compreensão alargada das várias componentes, e, em caso de redundâncias nos vários passos do projecto, procurámos perseguir rotas alternativas, que nos desafiassem a experimentar com soluções diferentes. Exemplos desta estratégia são, por exemplo, a escolha de elaborar o programa em python, uma linguagem que considerámos por nós um pouco esquecida, mas com um enorme potencial. Outro exemplo, foi a solução distinta que encontrámos para implementar as subscrições, optando por ter um socket por cada subscrição dentro do servidor, ao invés da solução já utilizada numa fase anterior do projecto, de ligar os vários sockets dos clientes a um único socket no servidor. A escolha de programar a nossa interface gráfica com recurso ao módulo de python Tkinter é também um destes exemplos, dado que nunca antes havíamos sido neste curso confrontados com esta funcionalidade.

Desta forma, considerámos este projecto especialmente interessante e com a mais uma mais-valia específica: a tangibilidade do nosso progresso, especialmente em comparação com o que experienciámos noutros trabalhos de diferentes áreas de estudo da engenharia informática.

Consideramo-nos satisfeitos com o nosso progresso e resultado final do trabalho apresentado, e prontos para aprofundar os nossos conhecimentos de redes de comunicação no futuro.

Notas importantes

Compilação do código:

python3 isabela_server.py para compilar o servidor

python3 isabela_client.py para compilar o cliente

Para conseguir compilar o cliente deverá ter instalada a livraria da interface gráfica Tkinder, para tal, deverá escrever o seguinte comando no terminal:

sudo apt-get install python3-tk