Ciências ULisboa | Informática

Aplicações Distribuídas

2022/23

Projeto 2

1. Notas Prévias

A avaliação da disciplina de Aplicações Distribuídas está divida em quatro projetos. O Projeto 2 é a sucessão direta do Projeto 1. A Visão Geral e Especificações (Conceitos e Regras) são as mesmas. Contudo, os alunos deverão agora resolver três limitações do Projeto1, nomeadamente a nível da **organização**, **desempenho** e **fiabilidade** do mesmo. Para além disso, o Projeto 2 é uma oportunidade de os alunos resolverem algum problema legado do Projeto 1.

2. Nova Organização

A primeira tarefa consiste em alterar a forma e conteúdo das mensagens trocadas entre cliente e servidor, relativamente ao Projeto 1.

Forma: a comunicação é obrigatoriamente serializada (ver a PL02 sobre serialização) e as mensagens trocadas entre o cliente e o servidor seguirão o formato apresentado na Tabela 1, utilizando Listas[1].

Conteúdo*:* O cliente envia uma lista contendo o código da operação que pretende que o servidor realize, bem como os parâmetros/argumentos da mesma. Em resposta, o servidor envia também uma lista com um código de resposta de operação[2] acompanhado de alguns parâmetros.

Tabela 1 - Lista de comandos suportados e formato das mensagens de pedido e resposta.

#	Comando	*String* enviada pelo Utilizador ao Cliente	Pedido-Lista enviada pelo Cliente ao Servidor	Resposta-Lista do Servidor
10	**SUBSCR**	**SUBSCR <ID do recurso> <limite de tempo>**	[10, <ID do recurso>, <limite de tempo>, <id do cliente>]	[11, True] ou [11, False] ou [11, None]
20	**CANCEL**	**CANCEL <ID do recurso>**	[20, <ID do recurso> , <ID do cliente>]	[21, True] ou [21, False] ou [21, None]
30	**STATUS**	**STATUS <ID do recurso>**	[30, <ID do recurso>, <ID do cliente>]	[31, 'SUBSCRIBED'] ou [31, 'UNSUBSCRIBED'] ou [31, None]
40	**INFOS**	**INFOS M**	[40, <ID do cliente>]	[41, <lista de recursos-ID subscritos pelo cliente>]
50		**INFOS K**	[50, <ID do cliente>]	[51, <número de ações que cliente *ainda* pode subscrever>]
60	**STATIS**	**STATIS L <ID do recurso>**	[60, <ID do recurso>]	[61, <número de subscritores ativos do recurso-ID>] ou [61, None]
70		**STATIS ALL**	**[70]**	[71, <lista de (estado de) todos recursos>]

[1] Atenção: não enviar *strings.*

[2] O código de resposta é o código enviado pelo cliente acrescido de uma unidade.

Notemos em primeiro lugar que a interface do Utilizador com o Cliente é inalterada face ao Projeto 1 (terceira coluna). Em segundo lugar, assinala-se que as respostas "`OK`" e "`NOK`" e "`UNKNOWN RESOURCE`" do Projeto 1 correspondem agora a `True`, `False` e `None`.

Por exemplo, se um cliente com um identificador `15` receber do utilizador o comando "`SUBSCR 20 100`", o programa cliente enviará a lista `[10, 20, 100, 15]` e o servidor responderá `[11, None]` se o recurso `20` não existir.

Relativamente ao INFOS M, se por exemplo o cliente tiver subscrito aos recursos 4, 8 e 14, deveremos esperar a resposta do servidor como uma lista que contem o código 41 seguido da lista de recursos:

```
[41, [4,8,14] ]
```

Relativamente ao STATIS ALL, o servidor deverá agora enviar uma resposta que é uma lista cujo primeiro elemento é o código 71, seguido de M elementos – um por cada recurso existente no servidor. Cada um destes elementos é uma lista de subscritores. Segue um exemplo abaixo.

```
[71, [1, 2, 3, 4, 5], [None] , [3 4 5], [1 2 3] ]
```

Este exemplo é equivalente ao exemplo do Projeto 1:

```
R 1 5 1 2 3 4 5
R 2 0
R 3 3 3 4 5
R 4 3 1 2 3
```

Além de serializar e seguir o formato das mensagens, os programas cliente e servidor serão reorganizados segundo o modelo de comunicação baseado em RPC (**ver a PL03 sobre RPC**). Assim, além dos ficheiros atuais, neste projeto teremos os ficheiros `ticker_stub.py` (contendo o *stub*) do lado do cliente e `ticker_skel.py` (contendo o *skeleton*) do lado do servidor. No servidor, o ficheiro atual será desdobrado em dois: `ticker_server.py` (já existente) e `ticker_pool.py` (contendo as definições respeitantes ao conjunto de recursos). No cliente, o ficheiro `net_client.py` conterá as definições respeitantes à comunicação do cliente com o servidor, podendo utilizar alguns métodos do ficheiro `sock_utils.py`. A reorganização está ilustrada na Figura 1 (o ficheiro atual `sock_utils.py` não é ilustrado na figura, mas pertence ao projeto).

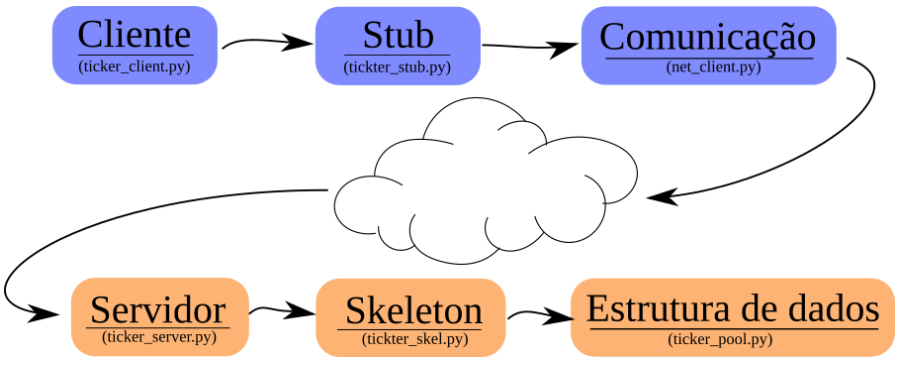

Figura 1 – Reorganização dos programas cliente e servidor.

3. Desempenho

A segunda tarefa consiste em suportar múltiplas ligações simultaneamente: múltiplos clientes e um servidor. Para este fim, deve-se usar o módulo *select* (ver as aulas TP03 e PL04 sobre suporte a múltiplos

clientes com o select). É de notar que, ao invés do que aconteceu no Projeto 1, pretende-se que os clientes estabeleçam a ligação ao servidor e que esta se mantenha aberta até que o programa cliente seja terminado. Durante a ligação, o cliente poderá enviar múltiplos pedidos. Vários clientes podem agora enviar pedidos "simultaneamente" .

4. Fiabilidade

Nesta terceira tarefa o foco está no tratamento de erros e mensagens parciais. Os alunos devem certificar-se de que os seus servidores não falham por problemas nos clientes. Além disso, os clientes devem falhar de forma "organizada", *i.e.*, devem mostrar uma mensagem de erro legível e não "rebentar" com mensagens pouco inteligíveis para um utilizador leigo.

Mais concretamente, nesta fase do projeto os alunos devem escrever código para:

1) **Tratar os possíveis erros em todas as chamadas ao sistema.** Isso pode ser feito através do uso de declarações `try/except` para lidar com condições anormais nos programas e realizar as ações para tratá-las de forma limpa.

2) **Ser capaz de receber mensagens fragmentadas.** Durante o Projeto 1 assumimos que a função `socket.recv(L)` devolve o número de bytes `L` que estamos à espera ou uma mensagem completa (caso ela tenha menos de `L` bytes). No entanto, esta função pode retornar menos bytes do que `L`. Portanto, para recebermos uma mensagem completa temos de invocá-la várias vezes até recebermos a mensagem com o tamanho pretendido (**ver aulas TP01 e PL01 sobre *sockets*, e TP02 e PL02 sobre serialização**). Os alunos devem concretizar no ficheiro `sock_utils.py` a função "*receive_all*" e usá-la no programa em substituição de `recv()` (que apenas será usada na concretização de `receive_all()`).

3) Os alunos devem certificar-se de que as mensagens introduzidas pelos clientes sejam validadas por estes antes do seu envio ao servidor, devolvendo uma mensagem de erro. Também, as mensagens recebidas pelo servidor devem ser validadas.

5. Entrega

A entrega do Projeto 2 consiste em colocar todos os ficheiros `.py` do projeto numa diretoria cujo nome deve seguir exatamente o padrão `grupoXX` (onde `XX` é o número do grupo). Juntamente com os ficheiros `.py` deverá ser enviado um ficheiro de texto `README.txt` (é em TXT, não é PDF, RTF, DOC, DOCX, etc.) onde os alunos podem relatar a informação que acharem pertinente sobre a sua implementação do projeto (por exemplo, limitações). A diretoria será incluída num ficheiro ZIP cujo nome deve seguir exatamente o padrão `grupoXX.zip` (novamente `XX` é o número do grupo). Esse ficheiro será submetido num recurso a disponibilizar para o efeito na página de AD no moodle da FCUL. Note que a entrega deve conter apenas os ficheiros `.py` e o ficheiro `README.txt`, qualquer outro ficheiro vai ser ignorado. O não cumprimento destas regras podem anular a avaliação do trabalho.

O prazo de entrega do Projeto 2 é **domingo, 26 de março de 2023, às 23:59 (Lisboa)**. Há um desconto de 5 pontos na entrega depois da hora, e de 100 pontos para entregas com 8 ou mais horas de atraso.

6. Plágio

Não é permitido aos alunos dos grupos partilharem código com soluções, ainda que parciais, a nenhuma parte do projeto com alunos de outros grupos (nem através do Fórum da disciplina, nem por qualquer outro meio). Além disso, todos os códigos serão testados por um sistema verificador de plágio. Caso alguma

irregularidade seja encontrada, os projetos de todos os alunos envolvidos serão anulados e o caso poderá ser reportado aos órgãos responsáveis na Ciências@ULisboa.

7. Referências Úteis

- https://docs.python.org/3/tutorial/classes.html
- https://docs.python.org/3/library/socket.html
- https://docs.python.org/3/library/exceptions.html
- https://docs.python.org/3/library/time.html
- https://docs.python.org/3/library/select.html
- https://docs.python.org/3/library/pickle.html