Ciências ULisboa | Informática

Aplicações distribuídas

2024/25

Projeto 2

1 Notas prévias

A avaliação da disciplina de Aplicações Distribuídas está dividida em três projectos. O Projeto 2 é a sucessão direta do Projeto 1. A Visão Geral e as Especificações (Conceitos e Regras) são as mesmas. No entanto, os alunos têm agora de resolver três limitações do Projeto 1, nomeadamente a **nível da organização** (RPC Stub & Skeleton - 25%), **desempenho (**multi-utilizador - 30%**)** e **fiabilidade** (serialização Pickle - 30%). Para além disso, o Projeto 2 é uma oportunidade para os alunos resolverem algum problema herdado do Projeto 1. Assim, a implementação correta de todas as funcionalidades vale **15%**.

2 Nova organização

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

Forma: a comunicação deve ser serializada (ver PL02 sobre serialização) e as mensagens trocadas entre o cliente e o servidor seguirão o formato apresentado nas Tabelas 1 e 2, utilizando listas[1].

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

Um utilizador (ou o gestor) tem acesso aos **mesmos** comandos do Projeto 1. As respostas do servidor, OK e NOK do Projeto 1, são agora transformadas em valores booleanos Verdadeiro e Falso, respetivamente, como se mostra nas Tabelas 1 e 2. Além disso, vários clientes podem estabelecer ligação ao servidor simultaneamente.

[1]Atenção: não enviar cadeias de caracteres (strings).

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

Tabela 1 - Lista de comandos do gestor, suportados pelo cliente e servidor, e formato das mensagens de resposta.

Código	Nome do comando	Lista pedido a enviar pelo programa cliente (para o gestor, ID = 0)	Lista resposta a enviar pelo programa servidor
10	ADD_ASSET	[10,asset_symbol,asset_price, available_supply,ID]	[11,*True*] (#OK) ou [11,*False*] (#NOK)
20	GET_ALL_ASSETS	[20,ID]	[21,*True*,a1,a2,…,am] (#OK) ou [21,*False*] (#NOK)
30	REMOVE_ASSET	[30,asset_symbol,ID]	[31,*True*,asset_symbol] (#OK) ou [31,*False*] (#NOK)
40	EXIT	[40,ID]	[41,*True*] (#OK) ou [41,*False*] (#NOK)

O utilizador participante terá à sua disposição o conjunto de comandos apresentados na Tabela 2.

Tabela 2 - Lista de comandos do utilizador, suportados pelo cliente e servidor, e formato das mensagens de resposta.

Código	Nome do comando	Lista pedido a enviar pelo programa cliente (para o utilizador, ID > 0)	Lista resposta a enviar pelo programa servidor
50	GET_ALL_ASSETS	[50,ID]	[51,*True,a1,a2,…,am*] (#OK) ou [51,*False*] (#NOK)
60	GET_ASSETS_BA-LANCE	[60,ID]	[61,*True*,balance,a1,a2,…,am] (#OK) ou [61,*False*] (#NOK)
70	BUY	[70,asset_symbol_quantity, ID]	[71,*True*] (#OK) ou [71,*False*] (#NOK)
80	SELL	[80,asset_symbol,qunatity, ID]	[81,*True*] (#OK) ou [81,*False*] (#NOK)
90	EXIT	[90,ID]	[91,*True*] (#OK) ou [91,*False*] (#NOK)

Para 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 PL03 sobre RPC). Assim, além dos ficheiros atuais, neste projeto teremos os ficheiros coincenter_stub.py (contendo o stub) no

lado cliente e coincenter_skel.py (contendo o skeleton) no lado servidor. A reorganização é ilustrada na Figura 1.

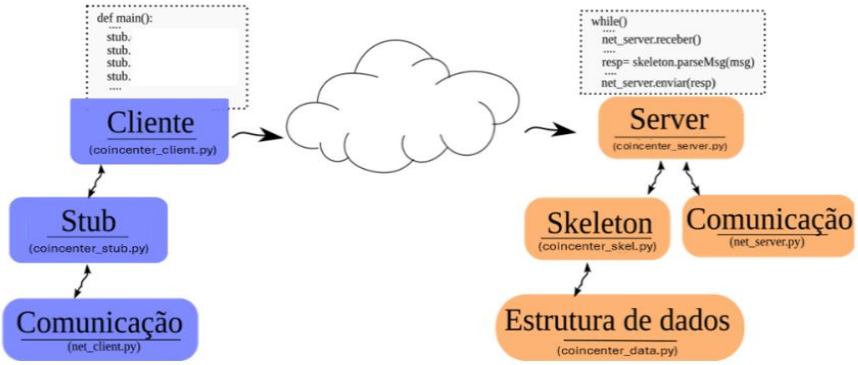

Figura 1- Estrutura de uma aplicação distribuída, seguindo o princípio RPC.

3 Desempenho

A segunda tarefa consiste em suportar múltiplas ligações em simultâneo: múltiplos programas cliente e um programa servidor. Para este efeito, deve ser utilizado o módulo select (ver aulas TP03 e PL04 sobre o suporte de múltiplos clientes com o select). Note-se que, ao contrário do Projeto 1, a intenção é que os clientes estabeleçam a ligação ao servidor e que esta se mantenha aberta até o programa cliente terminar. Durante a ligação, o cliente pode enviar vários pedidos. Agora, vários clientes podem enviar pedidos "simultaneamente".

4 Fiabilidade

Nesta terceira tarefa, o foco é o tratamento de erros e mensagens parciais. Os alunos devem garantir que os seus servidores não falham devido a problemas nos clientes. Além disso, os clientes devem falhar de forma "ordenada", ou seja, devem apresentar uma mensagem de erro legível e não terminar anormalmente com mensagens difíceis de compreender para um utilizador leigo.
Mais especificamente, nesta fase do projeto, os alunos devem escrever código para:

1) **Tratar possíveis erros em todas as chamadas do sistema.** Isto pode ser feito utilizando instruções try/except para tratar condições anómalas nos programas e executar ações para as tratar 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 (se tiver menos de L bytes). No entanto, esta função pode devolver menos bytes do que L. Assim, para recebermos uma mensagem completa temos de a invocar várias vezes até recebermos a mensagem com o tamanho desejado (**ver aulas TP01 e PL01 sobre *sockets* , e TP02 e PL02 sobre serialização**). Os alunos devem implementar a *função "receive_all" no* ficheiro sock_utils.py e usá-la no programa em vez de `recv()` (que só será usada na execução de `receive_all()`).
3) Os alunos devem certificar-se de que as mensagens introduzidas pelos utilizadores são validadas pelo programa cliente antes de serem enviadas para o servidor, devolvendo uma

mensagem de erro caso sejam introduzidas mensagens incorretas. Além disso, as mensagens recebidas pelo servidor também devem ser validadas.

5 Avaliação e entrega

5.1 Avaliação

A avaliação do projeto 2 seguirá uma metodologia similar à descrita para o projeto 1. Para que a avaliação possa ser executada de forma expedita, os alunos devem reler e cumprir a secção 7 do enunciado do projeto 1.

5.2 Entrega

A entrega do Projeto 2 consiste em colocar **todos os `ficheiros python (.py)`** do projeto numa **diretoria** cujo nome deve seguir exatamente o padrão `ad2425_projeto2_aluno-XX` (`XX` é o número de aluno), por exemplo **`ad2425_projeto1_aluno-12345`**.

A diretoria será *compactada* num ficheiro ZIP com o nome seguindo o padrão **`ad2425_projeto1_aluno-XX.zip`** (`XX` é o número de aluno), por exemplo **`ad2425_projeto1_aluno-12345.zip`**).

Quaisquer outros ficheiros serão ignorados. O não cumprimento destas regras pode invalidar a avaliação do trabalho. A data limite para submissão do Projeto 2 é **30 de março de 2025, às 23:59 (Lisboa)**.

5.3 Plágio

Os alunos não estão autorizados a partilhar código com soluções, mesmo que parciais, de qualquer parte do projeto com outros alunos; seja através do Fórum do curso, de repositórios git públicos, ou outros. Para além disso, todos os códigos serão testados por um sistema de verificação de plágio. Se for detetada alguma irregularidade, os projetos de todos os alunos envolvidos serão cancelados e o caso poderá ser reportado aos órgãos responsáveis da Faculdade.

6 Referências úteis

[1] https://docs.python.org/3/tutorial/classes.html

[2] https://docs.python.org/3/library/socket.html

[3] https://docs.python.org/3/library/select.html