



1. Descrição geral

A avaliação da cadeira de aplicações distribuídas está dividida em quatro projetos. O projeto 2 é uma continuação do projeto 1 onde os alunos vão resolver três limitações básicas do projeto relacionadas com organização, desempenho e fiabilidade. Para além disso o projeto 2 é uma oportunidade dos alunos resolverem quaisquer problemas pendentes do projeto 1.

O objetivo geral do projeto será concretizar um gestor de exclusão mútua. O seu propósito é controlar o acesso a um conjunto de recursos partilhados num sistema distribuído, garantindo exclusão mútua aos clientes que acedem aos recursos. O gestor será concretizado num servidor escrito na linguagem *Python*.

2. Modificações a aspetos definidos no enunciado 1

A primeira tarefa consiste em efetuar algumas alterações ao projeto 1. Nesse sentido, no projeto 2 a comunicação será serializada (**relembrar aulas TP02 e PL02 sobre serialização**) e as mensagens trocadas entre cliente e servidor seguirão o formato apresentado na Tabela 1, utilizando listas do *Python* como forma de enviar um código de operação bem como os parâmetros da mesma. Em resposta o servidor enviará também uma lista, com um código de operação que será sempre o código enviado pelo cliente acrescido de uma unidade. Além deste, o servidor enviará um valor de resultado que substitui as *strings* “OK”, “NOK” e “UNKNOWN RESOURCE”. Por exemplo se a pessoa que utiliza o cliente com id 15 introduzir o comando “LOCK 20”, o programa enviará a lista [10, 15, 20] e o servidor responderá com [11, None] se o recurso 20 não existir.

Tabela 1 - Lista de operações suportadas pelo servidor e formato das mensagens de pedido e resposta.

Operação	Mensagem enviada pelo cliente	Resposta do servidor
Lock	[10, <id do cliente>, <número do recurso>]	[11, True] ou [11, False] ou [11, None]
Release	[20, <id do cliente>, <número do recurso>]	[21, True] ou [21, False] ou [21, None]
Test	[30, <número do recurso>]	[31, True] ou [31, False] ou [31, None]
Stats	[40, <número do recurso>]	[41, <nº de bloqueios do recurso>] [41, None]

Convém relembrar que o id do cliente deverá ser um número inteiro a definir para cada cliente e que o número do recurso será outro número inteiro entre 0 e N-1, sendo N número de recursos geridos pelo servidor de *Locks*.

Além da serialização e formato das mensagens, os programas cliente e servidor serão reorganizados segundo o modelo de comunicação baseado em RPC (**relembrar aulas TP03 e PL03 sobre RPC**). Assim, além dos ficheiros atuais, neste projeto teremos os ficheiros `lock_stub.py` (contendo o *stub*) do lado do cliente e `lock_skel.py` (contendo o

skeleton) do lado do servidor. No servidor, o ficheiro atual será desdobrado em dois: `lock_server.py` (já existente) e `lock_pool.py` (contendo as definições respeitantes ao conjunto de recursos). A reorganização está ilustrada na figura Figura 1.

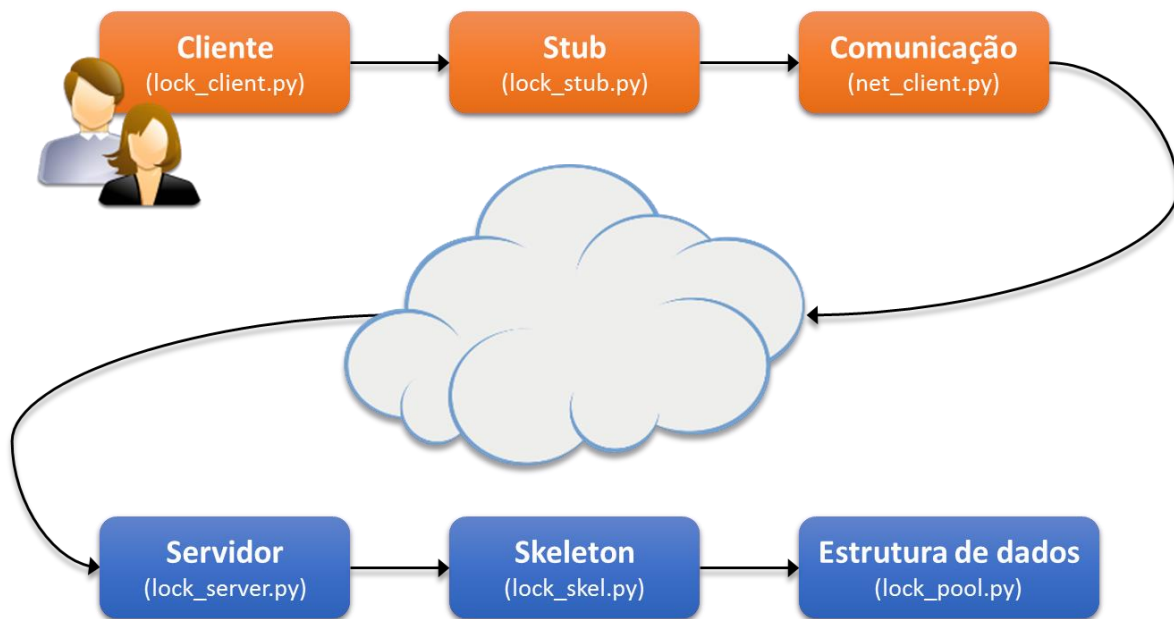


Figura 1 – Reorganização dos programas cliente e servidor.

3. Desempenho: suporte a múltiplas ligações

Nesta segunda fase o código do projeto deve ser modificado para suportar múltiplos clientes com ligações estabelecidas com o servidor. Para este fim, deve-se usar o módulo *select*, (**relembrar as aulas TP04 e PL04 sobre suporte a múltiplos clientes**). É de notar que ao invés do que aconteceu no projeto 1, se pretende que os clientes abram 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 comandos.

4. Fiabilidade: tratamento de erros e mensagens parciais

Nesta segunda fase do projeto 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(N)* devolve o número de bytes *N* que estamos à espera ou uma mensagem completa (caso ela tenha menos de *N* bytes). No entanto, esta função pode retornar menos bytes do que *N*, e portanto, para recebermos uma mensagem completa temos de invocá-la varias vezes até recebermos a mensagem pretendida (**relembrar aulas TP01 e PL01 sobre sockets em Python**). 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*).

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** (por exemplo grupo01 ou grupo23). Juntamente com os ficheiros `.py` deverá ser enviado um ficheiro de texto README.txt (não é .pdf nem .rtf nem .doc nem .docx) 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**. 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 prazo de entrega é sábado, dia 09/04/2016, até às 12:00hs.