Ciências ULisboa | Informática

Aplicações Distribuídas

2022/23

Projeto 1

1 Notas prévias

A avaliação prática da disciplina de Aplicações Distribuídas está divida em quatro projetos. Os Projetos 1 e 2 têm continuidade entre eles e, por essa razão, é muito importante que consigam cumprir os objetivos do Projeto 1, de forma a não tornar mais difícil o Projeto 2.

fonte: www.quoteinspector.com /

2 Visão Geral

O primeiro projeto visa concretizar um *Ticker* do mercado de ações utilizando o modelo cliente-servidor (uma ação é um título patrimonial correspondente a uma parcela do capital social de uma empresa). Um *Ticker* do mercado de ações permite aos utilizadores subscreverem informação sobre ações específicas e receberem atualizações continuamente ao longo do tempo. O projeto utilizará a arquitetura cliente-servidor, em que os utilizadores usam o cliente para subscreverem informação sobre as ações e o servidor envia continuamente aos clientes a informação sobre as ações subscritas. O servidor a implementar neste projeto possui os últimos dados das ações e transmite as atualizações para todos os clientes que subscreveram a informação sobre essas ações.

Cada cliente pode (1) subscrever ou (2) cancelar a subscrição de múltiplas ações. Quando uma subscrição é feita, ela terá uma duração definida em segundos. A partir do momento da subscrição e durante esse período, o servidor deverá transmitir a informação mais recente da ação em questão ao cliente, uma vez por segundo. Após o período definido, a inscrição será cancelada e as informações deixarão de ser enviadas.

O cliente pode ainda solicitar um vasto leque de informações ao servidor, nomeadamente informações do cliente: (3) saber se está subscrito ou não a uma determinada ação, (4) a lista de ações que ele tem atualmente subscritas e (5) quantas outras ações ainda pode subscrever. Tem também a possibilidade de pedir informação

mais genérica, nomeadamente: (6) o número atual de subscritores de uma dada ação, e (7) a listagem completa do estado de cada uma das ações do servidor.

3 Especificações

3.1 Conceitos:

Ação – também designada por recurso no contexto do projeto – é no essencial uma estrutura de dados e uma lista de funcionalidades intrínsecas. Cada recurso é caracterizado:

- pelo ID único (número natural),
- pela SIMBOLO (*string* de 3 carateres)
- pela NOME (*string* de 7 carateres)
- pelo VALOR (número real).

Um recurso deverá implementar funcionalidades como:

- ser subscrito,
- anular a subscrição,
- retornar se um cliente em particular é subscritor, e
- retornar o número total de subscritores.

Cliente – programa que aceita comandos de texto do Utilizador para que este possa fazer uso das 7 funcionalidades acima referidas, e que envia esses comandos ao programa servidor.

Servidor – programa gestor dos recursos (ações) que responde aos pedidos dos clientes.

3.2 Regras:

- Uma ação pode ser subscrita por múltiplos clientes simultaneamente até a um máximo de N clientes - limite fixado pelo servidor. A estrutura de dados de cada ação *deverá ir anotando* a lista dos clientes subscritores para garantir este limite. Assim sendo, um cliente ao tentar subscrever a uma ação já subscrita por outros N clientes terá o seu pedido negado.
- Por outro lado, um cliente está restrito a subscrever até ao máximo de K ações simultaneamente - limite fixado pelo servidor. Assim sendo, quando o cliente tenta subscrever a ação K+1, o servidor deverá negar esse pedido. O servidor deverá ser capaz de contar quantas ações tem subscritas um dado cliente.
- Uma ação é sempre subscrita por um tempo de concessão decidido pelo cliente. O servidor deverá automaticamente anular a subscrição de um cliente findo esse tempo.

4 Arquitetura do Sistema

4.1 Arquitetura Funcional

Deverá desenvolver uma arquitetura de cliente-servidor com *Sockets TCP*. A Figura 1ilustra a arquitetura a utilizar. O Utilizador deverá introduzir um comando em texto como entrada no programa cliente. Este deverá interpretar o pedido, efetuar o pedido de ligação ao servidor, enviar o pedido ao servidor, receber a resposta ao comando enviado e por fim terminar a ligação. O servidor, por outro lado, aceita ligações de clientes (um de cada vez no projeto 1), recebe e processa os pedidos, responde ao cliente com a informação pedida, e termina a ligação.

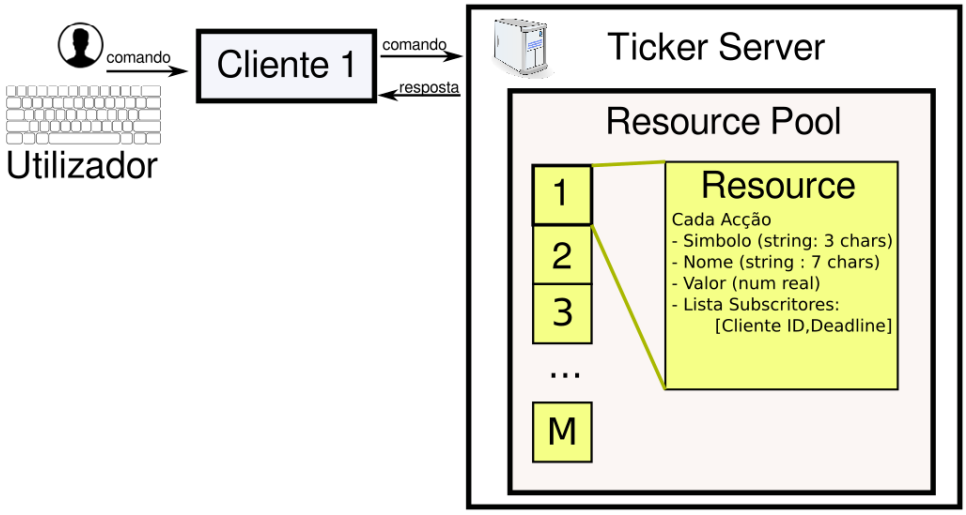

Figura 1- Arquitetura do sistema, desde o utilizador até ao servidor de Ticker, o qual inclui uma estrutura de dados com três classes.

4.2 Fora do âmbito deste Projeto

Entre outros aspetos, no projeto 2 iremos considerar a possibilidade de o servidor atender múltiplos clientes em simultâneo, assim como a possibilidade de o cliente não necessitar de estabelecer e terminar a ligação sempre que pretende enviar um novo comando. O envio de dados das ações segundo a segundo (*stream* de dados) que o cliente subscreve será implementado no projeto 2.

4.3 Arquitetura de Software

Programa Cliente

O programa cliente será implementado em Python3 no ficheiro fornecido, `ticker_client.py`. Este espera receber um ou vários de 7 comandos possíveis. Uma sequência de comandos compõe uma história de execução que será apresentada ao cliente. O programa cliente deverá implementar o seguinte ciclo de execução para cada comando:

1- Pedir ao utilizador um comando, através da *prompt* “comando > ”
2- Ler uma *string* do utilizador no standard input

3- Validar o comando fornecido*
4- Caso o comando pressuponha um pedido ao servidor:
    a. Ligar ao servidor;
    b. Enviar para o servidor o comando respetivo através de uma *string* com um formato de mensagem específico;
    c. Receber a *string* de resposta do servidor;
    d. Apresentar a resposta recebida;
    e. Terminar a ligação com o servidor;
5- Caso o comando seja para processamento local, executá-lo;
6- Voltar a 1.

O programa deverá receber os seguintes parâmetros pela linha de comandos, pela ordem apresentada:

1. o ID único do cliente
2. o IP ou *hostname* do servidor
3. o porto TCP onde o servidor recebe pedidos de ligação

Desta forma, um exemplo de inicialização do cliente é o seguinte:

```
$  python3 ticker_client.py 1 localhost 9999
```

*Validar o comando fornecido: O cliente será responsável por verificar se o comando do utilizador é válido e se não possui gralhas na sintaxe especificada.

- Caso o comando não exista ou possua gralhas, o cliente apresentará o resultado "`UNKNOWN-COMMAND`".
- Caso faltem argumentos, o cliente apresentará o resultado "`MISSING-ARGUMENTS`".

Programa Servidor

O programa Servidor será implementado em Python3 no ficheiro `ticker_server.py` fornecido.

O seu ciclo de operações é o seguinte:

1. Esperar um pedido de ligação;
2. Mostrar no ecrã informação sobre a ligação (IP/*hostname* e porto de origem do cliente);
3. Verificar se existem recursos cujo tempo de subscrição tenha expirado, e remover essas subscrições;
4. Receber uma mensagem com um pedido;
5. Processar esse pedido;
6. Responder ao cliente;
7. Fechar a ligação.

O servidor deverá receber os seguintes parâmetros pela linha de comandos, pela ordem apresentada:

1. IP ou *hostname* onde o servidor fornecerá os recursos;
2. Porto TCP onde escutará por pedidos de ligação;
3. Número de ações/recursos que serão geridos pelo servidor (**M**);
4. Número máximo de ações por cliente (**K**);
5. Número máximo de subscritores por ação (**N).**

Desta forma, um exemplo de inicialização do servidor é o seguinte:

```
$  python3 ticker_server.py localhost 9999 4 3 100
```

O servidor deverá ser composto por 3 classes, tal como representado na Figura 1, nomeadamente:

1. O *ticker_server* – gere as ligações por *sockets* com o cliente,
2. O *resource_pool* – a estrutura que gere o conjunto das ações.
3. O *resource* – a estrutura que representa cada uma das ações disponíveis no servidor.

Cada servidor deverá instanciar **M** recursos, sendo que cada recurso deverá ser inicializado aleatoriamente. Isto é, com Nome e Valor gerados aleatoriamente. Nome deverá ser uma *string* com 7 carateres. Valor deverá ser um número real entre 100 e 200. Símbolo deverá ser uma *string* com 3 carateres iguais aos 3 primeiros carateres de Nome.

A lista de subscritores de cada ação deverá começar vazia.

4.4 Comandos suportados e formato das mensagens

Para fazer face às sete necessidades já apresentadas na Secção 2 - Visão Geral, o utilizador terá disponível sete comandos. Estes comandos resultarão em envios de mensagens do cliente para o servidor e em respostas do servidor para o cliente. O utilizador terá ainda acesso a dois comandos adicionais que não envolvem interação com o servidor e que serão apresentados mais abaixo neste enunciado.

Apresenta-se agora uma caracterização de cada um dos comandos que o utilizador tem acesso.

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

#	Comando	String enviada pelo Utilizador ao Cliente	*String* enviada pelo Cliente ao Servidor	Resposta do Servidor
1	**SUBSCR**	**SUBSCR** <ID do recurso> <limite de tempo>	**SUBSCR** <ID do recurso> <limite de tempo> <id do cliente>	OK ou NOK ou UNKNOWN RESOURCE
2	**CANCEL**	**CANCEL** <ID do recurso>	**CANCEL** <ID do recurso> <ID do cliente>	OK ou NOK ou UNKNOWN-RESOURCE
3	**STATUS**	**STATUS** <ID do recurso>	**STATUS** <ID do recurso> <ID do cliente>	SUBSCRIBED ou UNSUBSCRIBED ou UNKNOWN-RESOURCE
4	**INFOS**	**INFOS M**	**INFOS M** <ID do cliente>	<lista de recursos-ID subscritos pelo cliente>
5		**INFOS K**	**INFOS K** <ID do cliente>	<número de ações que cliente *ainda* pode subscrever>
6	**STATIS**	**STATIS L** <ID do recurso>	**STATIS L** <ID do recurso>	<número de subscritores ativos do recurso-ID> ou UNKNOWN-RESOURCE
7		**STATIS ALL**	**STATIS ALL**	<listagem do estado de cada recurso>

4.4.1 SUBSCR

O comando SUBSCR <recurso ID> <limite tempo> subscreve um determinado recurso, durante um tempo de concessão específico (em segundos) para o cliente que está a enviar o pedido (D*eadline = tempo do relógio atual do servidor + tempo concessão*)

- Em geral, se o recurso existir, o servidor deverá registar o pedido, e retornar OK.
- Se o pedido for para um recurso já subscrito pelo cliente, o novo Deadline deverá ser atualizado, e retornar OK.
- Se o recurso não existir, o servidor deverá retornar UNKNOWN-RESOURCE
- Se o cliente já tiver ativas K subscrições, o servidor deverá retornar NOK.
- Se o recurso já tiver ativas N subscrições, o servidor deverá retornar NOK.

4.4.2 CANCEL

O comando `CANCEL <recurso ID>` remove uma subscrição ativa numa determinada ação/recurso para o cliente que está a enviar o pedido.

- Em geral, se o recurso existir e estiver subscrito pelo cliente, o servidor deverá registar o pedido, e retornar OK.
- Se o recurso não existir, o servidor deverá retornar UNKNOWN-RESOURCE.
- Se o pedido for para um recurso não subscrito pelo cliente, o servidor deverá retornar NOK.

4.4.3 STATUS

O comando `STATUS <recurso ID>` é utilizado para obter o estado atual de um determinado recurso face ao cliente em questão.

- O servidor retorna o estado do recurso solicitado (i.e., SUBSCRIBED, UNSUBSCRIBED).
- Se o pedido se referir a um recurso inexistente, o servidor deverá retornar UNKNOWN-RESOURCE.

4.4.4 INFOS

O comando `INFOS` tem duas variantes, é utilizado para obter informações sobre o cliente no servidor.

- `INFOS M` – deverá retornar o <lista de recursos-ID subscritos pelo cliente>
- `INFOS K` – deverá retornar o <número total de ações a que o cliente ainda pode subscrever>

4.4.5 STATIS

O comando STATIS é utilizado para obter informação genérica do servidor.

- `STATIS L <recurso ID>` – deverá retornar o número de subscritores do recurso em questão.
- `STATIS ALL` – é utilizado para obter uma visão geral do estado do serviço de gestão de ações e seus subscritores. O servidor deverá retornar uma *string* formada por uma linha por cada recurso no servidor. Cada linha é composta por texto com os seguintes campos (separados por espaços):
    1. A letra `R` para indicar um recurso;
    2. O ID do recurso;
    3. O número atual de subscritores desse recurso;
    4. A lista de clientes subscritos (ordenada crescentemente pelo cliente-ID). Não apresentar nada, se não houver clientes subscritos.

Um exemplo de resultado do comando num servidor com M=4 recursos pode ser o seguinte:

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

Note, que os comandos SUBSCR, CANCEL, STATUS e INFOS possuem uma diferença entre o comando escrito pelo Utilizador no cliente e a mensagem que deverá ser enviada ao servidor. A diferença reside no <ID do cliente>. Cada programa cliente é responsável por adicionar este dado nos seus pedidos ao servidor.

4.4.6 SLEEP e EXIT

Os 2 comandos que serão tratados apenas do lado do cliente (i.e., não são enviados pedidos para o servidor) são o SLEEP e o EXIT, tal como listados na Tabela 2.

Tabela 2 - Lista de comandos suportados apenas pelo cliente.

Comando	**Comando recebido pelo cliente**
SLEEP	SLEEP <limite de tempo>
EXIT	EXIT

O comando SLEEP faz com que o cliente adormeça durante um determinado tempo (em segundos). Ou seja, o cliente esperará este tempo antes de interpretar o próximo comando.

O comando EXIT encerra a execução do cliente. Pode-se assumir que todas as execuções terão um comando EXIT no final.

4.4.7 Exemplos

Seguem alguns exemplos de sequências de comandos que o utilizador poderá introduzir no programa cliente. Os comandos dos primeiros três exemplos correm normalmente, enquanto os comandos do quarto exemplo contêm erros ou argumentos em falta e não deverão ser enviados ao servidor. Todos os comandos do quinto exemplo poderão ser enviados ao servidor, pese embora se refiram a recursos inexistentes.

Tabela 3 – Cinco exemplos de sequências de comandos do Utilizador. M=6 (servidor com 6 Ações), N=5 (um cliente tem até 5 ações subscritas) e K=3 (cada ação tem até 3 subscritores).

ex°1	ex°2	ex°3	ex°4	ex°5
SUBSCR 1 30	SUBSCR 1 10	SUBSCRB 1 5	SUBXRB 1 30	SUBSCR 7 -1
SLEEP 5	SUBSCR 2 10	SLEEP 2	CAMCEL 1	SUBSCR -1 10
STATUS 1	SUBSCR 3 10	…	CANCEL	SUBSCR 100 10
INFOS M	SUBSCR 5 10	SUBSCRB 1 5	SUBSCR 1	SUBSCR 0 10
INFOS K	STATIS L 1	SLEEP 5	CANCEL 1	CANCEL -2
STATIS L 1	SLEEP 20	INFOS K	STATIS X	STATUS 100
STATIS ALL	STATIS ALL	STATIS L 1	STATUS	STATIS L 8
EXIT	EXIT	EXIT	EXIT	EXIT

5. Implementação Python 3

5.1 Servidor

A definição de um recurso deverá ser implementada na classe `resource` mostrada de seguida. Esta classe deve ser definida e instanciada no ficheiro `ticker_server.py` fornecido para a implementação do servidor.

```
class resource:
    def __init__(self, resource_id):
        pass # Remover esta linha e fazer implementação da função

    def subscribe(self, client_id, time_limit):
        pass # Remover esta linha e fazer implementação da função

    def unsubscribe (self, client_id):
        pass # Remover esta linha e fazer implementação da função

    def status(self, client_id):
        pass # Remover esta linha e fazer implementação da função

    def __repr__(self):
        output = ""
        # R <resource_id> <list of subscribers>
        return output
```

O conjunto de **`M`** recursos será gerido pela classe `resource_pool`, a qual deverá ser definida e instanciada também no ficheiro `ticker_server.py` já mencionado. Esta classe serve também de interface para o acesso aos dados relacionados a cada recurso.

```
class resource_pool:
    def __init__(self, N, K, M):
        pass # Remover esta linha e fazer implementação da função

    def clear_expired_subs(self):
        pass # Remover esta linha e fazer implementação da função

    def subscribe(self, resource_id, client_id, time_limit):
        pass # Remover esta linha e fazer implementação da função

    def unsubscribe (self, resource_id, client_id):
        pass # Remover esta linha e fazer implementação da função

    def status(self, resource_id, client_id):
        pass # Remover esta linha e fazer implementação da função

    def infos(self, option, client_id):
        pass # Remover esta linha e fazer implementação da função

    def statis(self, option, resource_id):
        pass # Remover esta linha e fazer implementação da função

    def __repr__(self):
        output = ""
        # Acrescentar no output uma linha por cada recurso
        return output
```

5.2 Cliente

A implementação do cliente deverá ser feita com base na classe `server_connection` definida no ficheiro `net_client.py`, fornecido para o efeito. Por sua vez esta classe deverá usar o módulo `sock_utils.py` sugerido nas aulas práticas para a implementação relacionada com *sockets* [2].

```
class server_connection:
    def __init__(self, address, port):
        pass # Remover esta linha e fazer implementação da função

    def connect(self):
        pass # Remover esta linha e fazer implementação da função

    def send_receive(self, data):
        pass # Remover esta linha e fazer implementação da função

    def close(self):
        pass # Remover esta linha e fazer implementação da função
```

5.3 Detalhes adicionais de implementação e testes

O tempo limite da concessão de uma subscrição deverá ser manipulado através da função `time()` do módulo *time* do *Python* [3]. Esta devolve o número de segundos desde a *época* (*Unix Epoch* – 00h00m00s de 1 janeiro de 1970 [4]).

A receção dos comandos no ciclo do programa cliente deverá ser feito através da função `input()`. Esta interrompe a execução do programa e fica à espera de que um novo comando seja apresentado pelo utilizador. A vantagem de utilizar esta função é que ela serve tanto para obter comandos de forma interativa com um utilizador, como receber comandos enviados através do *pipe* para o programa. Para este último, basta gravar a lista de comandos que compõe uma história de execução num ficheiro de texto (e.g.: `comandos.txt`) e executar a seguinte instrução:

```
$   cat comandos.txt | python3 ticker_client.py 1 localhost 9999
```

IMPORTANTE: Tanto o programa cliente quanto o servidor devem imprimir todas as mensagens enviadas e recebidas. No caso do cliente, serão todos os pedidos enviados e respostas recebidas. No caso do servidor, serão todos os pedidos recebidos e as respostas a serem enviadas. O esperado é que a saída do cliente seja muito semelhante à saída do servidor, ao ponto que permita fazer a comparação com a ferramenta `diff`.

Por fim, um exemplo de teste será executar as seguintes linhas na consola:

```
$   python3 ticker_server.py localhost 9999 4 3 > output_server.log
$   cat commandos.txt | python3 ticker_client.py 1 localhost 9999 > output_client.log
$   diff output_server.log output_client.log
```

6. Avaliação

6.1. Entrega

A entrega do Projeto 1 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 informar 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 pode anular a avaliação do trabalho.

O prazo de entrega do Projeto 1 é **domingo, 5 de março de 2022, às 23:59 (Lisboa)**.

6.2. Plágio

Não é permitido aos alunos dos grupos partilharem código com soluções, ainda que parciais, de alguma parte do projeto com alunos de outros grupos; seja através do Fórum da disciplina, seja 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

[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/time.html

[4] https://en.wikipedia.org/wiki/Unix_time