Ciências ULisboa | Informática

Aplicações Distribuídas

2020/21

Projeto 1

1. Descrição geral

A avaliação da disciplina de Aplicações Distribuídas está divida em quatro projetos. Os Projetos 1 e 2 terão continuidade 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**.

O objetivo geral do projeto será concretizar um gestor de pedidos simultâneos a recursos e o processamento destes pedidos em exclusão mútua. O seu propósito é controlar o acesso a um conjunto de recursos partilhados num sistema distribuído, onde diferentes clientes podem requerer de forma concorrente o acesso aos recursos. Um recurso é bloqueado exclusivamente por um só cliente, mas está disponível para bloqueio até a um máximo de **`K`** bloqueios ao longo do tempo, ou seja, findo os **`K`** bloqueios permitidos o recurso fica inativo (i.e., indisponível para novos bloqueios). O gestor permite **`Y`** recursos bloqueados em simultâneo e será concretizado num servidor escrito na linguagem *Python 3*. A **Figura 1** ilustra a arquitetura a seguir no servidor de exclusão mútua (*Lock Server*), bem como a estrutura de dados em *Python* que o suporta.

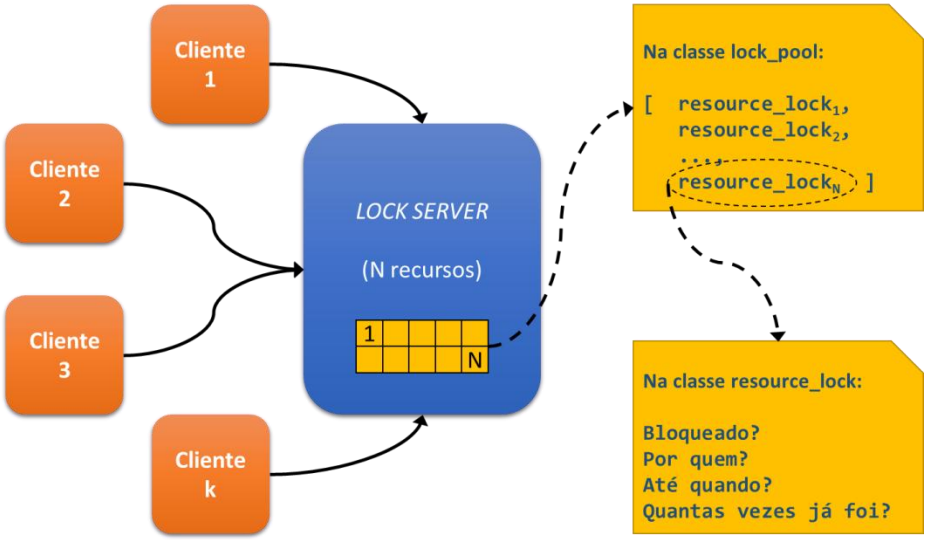

Figura 1– Arquitetura do servidor de *Locks*, e estrutura de dados através de duas classes.

Nesta primeira fase o servidor aceita ligações de clientes, recebe e processa os pedidos, responde ao cliente, e termina a ligação. O cliente efectua o pedido de ligação ao servidor, envia o pedido ao servidor, recebe a resposta e termina a ligação.

Entre outros aspetos, na segunda fase vamos considerar a possibilidade de o servidor atender a múltiplos clientes em simultâneo e de o cliente não necessitar de estabelecer e terminar a ligação sempre que pretende enviar um comando.

2. Descrição específica

A resolução do Projeto 1 está dividida em três passos:

1. Definição do formato das mensagens a ser trocado entre o cliente e o servidor;
2. Definição da estrutura de dados a ser mantida pelo servidor;
3. Concretização dos processos no cliente e no servidor.

2.1. Comandos suportados e formato das mensagens

O sistema será controlado através de 7 comandos possíveis. Uma sequência de comandos compõe uma história de execução que será apresentada para o cliente. O programa cliente fará uma primeira validação do nome de cada comando fornecido e enviará para o servidor os comandos pertinentes através de *strings* com um formato de mensagem específico. O servidor, por sua vez, validará o comando e os argumentos recebidos na mensagem, fará o respetivo processamento e responderá ao cliente também com uma *string* que indica o resultado do processamento do comando.

Dos 7 comandos previstos, 5 deles (listados na Tabela 1) resultarão em envios de mensagens do cliente para o servidor e em respostas do servidor para o cliente.

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

Comando	**Comando recebido pelo cliente**	**Formato da *string* enviada pelo cliente**	**Resposta do servidor**
LOCK	LOCK <número do recurso> <limite de tempo>	LOCK <número do recurso> <limite de tempo> <id do cliente>	OK ou NOK ou UNKNOWN RESOURCE
UNLOCK	UNLOCK <número do recurso>	UNLOCK <número do recurso> <id do cliente>	OK ou NOK ou UNKNOWN RESOURCE
STATUS	STATUS R <número do recurso>	STATUS R <número do recurso>	LOCKED ou UNLOCKED ou DISABLED ou UNKNOWN RESOURCE
	STATUS K <número do recurso>	STATUS K <número do recurso>	<número de bloqueios feitos no recurso> ou UNKNOWN RESOURCE
STATS	STATS Y	STATS Y	<número de recursos bloqueados atualmente>
	STATS N	STATS N	<número de recursos disponíveis>
	STATS D	STATS D	<número de recursos desabilitdados>
PRINT	PRINT	PRINT	<estado de todos os recursos>

Todos os recursos do servidor iniciam a sua execução com o estado `UNLOCKED`, o contador de bloqueios com `0` e o id do cliente que detém o recurso bloqueado com `-1`.

O comando **`LOCK`** bloqueia um determinado recurso durante um tempo de concessão específico (em segundos) para o cliente que está a enviar o pedido. Se o recurso solicitado estiver no estado `UNLOCKED` e os limites de bloqueios **`K`** do recurso e **`Y`** de recursos bloqueados simultaneamente não tiverem ainda sido atingidos, o servidor deve bloquear o

recurso, incrementar o contador de bloqueios, registar o id do cliente que fez o pedido, calcular o limite de tempo de concessão que estará bloqueado e retornar `OK`. Se o recurso solicitado estiver no estado `LOCKED`, o servidor deve verificar se o cliente que está a fazer o pedido é o mesmo que detém o bloqueio do recurso atual. Neste caso, o servidor deverá incrementar o contador de bloqueios (se este não ultrapassar o limite de bloqueios **`K`**), estender o período de bloqueio pelo tempo solicitado e retornar `OK`. Caso o recusrso esteja `LOCKED` por outro cliente, esteja `DISABLED`, ou esteja `UNLOCKED` mas tenha atingido o limite de bloqueios **`K`** ao longo do tempo ou o limite **`Y`** de recursos que podem ser bloqueados simultaneamente, o servidor deve retornar `NOK`. Por fim, se o pedido referir-se a um recurso inexistente (e.g., um número de recurso menor que `0` ou maior que `N-1`), o servidor deverá retornar `UNKNOWN RESOURCE`.

O comando **`UNLOCK`** desbloqueia um determinado recurso para o cliente que está a enviar o pedido. Se o recurso solicitado estiver no estado `LOCKED` e o cliente que o está a bloquear for o mesmo que está a pedir o desbloqueio, então o servidor deve desbloquear o recurso e retornar `OK`. Caso o recurso esteja no estado `UNLOCKED` ou `DISABLED` ou ainda se ele estiver bloqueado por outro cliente, o servidor deve retornar `NOK`. Por fim, se o pedido referir-se a um recurso inexistente (e.g., um número de recurso menor que `0` ou maior que `N-1`), o servidor deverá retornar `UNKNOWN RESOURCE`.

Note que tanto no `LOCK` quanto no `UNLOCK` a diferença entre o comando apresentado ao cliente e a mensagem enviada ao servidor é o `<id do cliente>`, o qual o cliente irá inserir no pedido. Todos os outros comandos possuem o mesmo formato no comando e no pedido.

O comando **`STATUS`** é utilizado para obter informações sobre um determinado recurso e possui 2 formas. Na primeira (`STATUS R`), o servidor retorna o estado do recurso solicitado. Na segunda (`STATUS K`), o servidor retorna o número de bloqueios que já foram efetuados no recurso solicitado. Em ambos os casos, se o pedido referir-se a um recurso inexistente (e.g., um número de recurso menor que `0` ou maior que `N-1`), o servidor deverá retornar `UNKNOWN RESOURCE`.

O comando **`STATS`** é utilizado para obter informações sobre o servidor de locks e possui 3 formas. Na primeira (`STATS Y`), o servidor retorna o número de recursos que se encontram bloqueados atualmente (i.e., no estado `LOCKED`). Na segunda (`STATS N`), o servidor retorna o número de recursos que se encontram disponíveis atualmente (i.e., no estado `UNLOCKED`). Na terceira (`STATS D`), o servidor retorna o número de recursos que se encontram desabilitados atualmente (i.e., no estado `DISABLED`).

O comando **`PRINT`** é utilizado para obter uma visão geral do estado do serviço de locks, onde o servidor retorna uma *string* onde são apresentados os seguintes campos (separados por espaços), numa linha para cada recurso: a letra `R` para indicar um recurso, o id do recurso, o seu estado e o número de bloqueios efetuados no mesmo. Se o recurso estiver no estado `LOCKED`, a linha do recurso ainda incluirá o id do cliente que detém o bloqueio do recurso e o limite de tempo de concessão que indica o término do bloqueio deste recurso (em segundos desde a época, ver a Secção 2.3.3). Um exemplo de resultado do comando `PRINT` de um servidor com 3 recursos pode ser o seguinte:

```
R 0 UNLOCKED 1
R 1 DISABLED 3
R 2 LOCKED 2 5 1612873214
```

O programa cliente será responsável por verificar se o comando apresentado existe, se não possui gralhas no seu nome e se não faltam argumentos conforme a especificação. Caso o comando não exista ou possua gralhas, o cliente apresentará o resultado "`UNKNOWN COMMAND`" e não enviará o pedido ao servidor. Caso faltem argumentos, o cliente apresentará o resultado "`MISSING ARGUMENTS`" e não enviará o pedido ao servidor.

Os outros 2 comandos serão tratados apenas do lado do cliente (i.e., não são enviados pedidos para o servidor), os quais são:

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.

Seguem alguns exemplos de sequências de comandos que poderão ser utilizados para analisar a execução dos programas a serem desenvolvidos:

Tabela 3 – Exemplos de sequências de comandos. Todos os comandos do quarto exemplo contém erros ou argumentos em falta e não deveriam ser enviados ao servidor. Todos os comandos do quinto exemplo utilizam recursos inexistentes (e.g., para `N=3`).

LOCK 0 10	LOCK 0 10	LOCK 0 10	BLOCK 0 10	LOCK 3 10
SLEEP 8	LOCK 1 10	LOCK 0 10	UNBLOCK 0	LOCK -1 10
LOCK 0 10	LOCK 2 10	…	LOCK	LOCK 1000 10
STATUS R 0	LOCK 3 10	LOCK 0 10	LOCK 0	LOCK 3 10
STATUS K 0	LOCK 4 10	UNLOCK 0	UNLOCK	UNLOCK -2
STATS Y	STATS Y	STATUS R 0	STATS X	STATUS R 1000
UNLOCK 0	SLEEP 20	STATUS K 0	STATUS R	STATUS K -2
PRINT	PRINT	STATS D	EXIT	EXIT
EXIT	EXIT	EXIT		

2.2. Estrutura de dados

Os dados a guardar no servidor consistem de informação relacionada com a exclusão mútua no acesso concorrente a um conjunto de **`N`** recursos partilhados. Serão utilizadas duas classes em Python [1] para estruturar a informação e o acesso à mesma.

Sobre cada recurso dever-se-á saber a seguinte informação:

- se está bloqueado, desbloqueado ou desabilitado;
- quantas vezes já foi bloqueado;

- no caso de estar bloqueado:
    - qual o cliente que detém a concessão de bloqueio; e
    - até quando essa concessão é válida.

A definição de um recurso (quanto à exclusão mútua) deverá ser implementada na classe *resource_lock*, mostrada de seguida.

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

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

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

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

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

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

    def __repr__(self):
        output = ""
        # Se o recurso está bloqueado:
        # R <número do recurso> bloquado <id do cliente> <instante limite da
        #concessão do bloqueio>
        # Se o recurso está desbloquado:
        # R <número do recurso> desbloqueado
        # Se o recurso está inativo:
        # R <número do recurso> inativo
        return output
```

Esta classe deve ser definida e instanciada no ficheiro `lock_server.py` fornecido para a implementação do servidor.

O conjunto de **`N`** recursos será definido (quanto à exclusão mútua) pela classe *lock_pool* que deverá ser definida e instanciada no ficheiro `lock_server.py` já mencionado. Esta classe serve também de interface para o acesso aos valores de exclusão mútua de cada recurso, bem como da gestão dos **`K`** bloqueios permitidos por recurso e **`Y`** recursos bloqueados em simultâneo.

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

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

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

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

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

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

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

2.3. Concretização do cliente e do servidor

2.3.1 Cliente

O cliente, a implementar no ficheiro `lock_client.py` fornecido, deverá receber os seguintes parâmetros pela linha de comandos, pela ordem apresentada:

- o id único do cliente
- o ip ou hostname do servidor que fornece os recursos;
- o porto TCP onde o servidor recebe pedidos de ligação;

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

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

O cliente deverá, de forma cíclica:

1. pedir ao utilizador um comando para enviar ao servidor, através da prompt, "`comando >` "
2. estabelecer a ligação ao servidor;
3. enviar o comando e receber a resposta do servidor;
4. terminar a ligação ao servidor;
5. apresentar a resposta recebida.

A implementação do cliente deverá ser feita com base na classe *server* 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:
    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
```

2.3.2 Servidor

O servidor, a implementar no ficheiro `lock_server.py` fornecido, deverá receber os seguintes parâmetros pela linha de comandos, pela ordem apresentada:

- IP ou hostname onde o servidor fornecerá os recursos;
- porto TCP onde escutará por pedidos de ligação;
- número de recursos que serão geridos pelo servidor (**`N`**);
- número de bloqueios permitidos em cada recurso (**`K`**);
- número permitido de recursos bloqueados num dado momento (**`Y`**);

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

```
$  python3 lock_server.py localhost 9999 4 3 2
```

O servidor deverá implementar o seguinte ciclo de operações:

1. aceitar uma ligação;
2. mostrar no ecrã informação sobre a ligação (IP/hostname e porto de origem);
3. verificar se existem recursos com bloqueios cujo tempo de concessão de exclusão mútua tenha expirado, e remover esses bloqueios nos recursos;
4. verificar se existem recursos que já atingiram os **`K`** bloqueios permitidos, e desativar estes recursos em caso positivo;
5. verificar se o número de recursos bloqueados num dado momento já atingiu **`Y`**, e não permitir bloqueios a novos recursos em caso positivo;
6. receber uma mensagem com um pedido;
7. processar esse pedido;
8. responder ao cliente;
9. fechar a ligação;

2.3.3 Detalhes adicionais de implementação e testes

O tempo limite da concessão de um bloqueio deverá ser manipulado através da função `time()` do módulo *time* do *Python* [3]. Esta devolve o número de segundos desde a *época* (uma data de referência à qual se soma um número de segundos para obter a data e hora locais).

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 que um novo comando seja apresentado. 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 o seguinte comando:

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

Para além disso, 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 resposta 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 lock_server.py  localhost 9999 4 3 2 > output_server.log
$  cat commandos.txt | python3 lock_client.py 1 localhost 9999 > output_client.log
$  diff output_server.log output_client.log
```

3. Avaliação

3.1 Checkpoint

O check point do projeto consiste em apresentar ao docente o funcionamento do projeto, demonstrando todas as funcionalidades deste, incluindo tratamento de erro. Assim, sendo o grupo de trabalho deve preparar uma bateria de testes para apresentar e responder às questões colocadas pelo docente. A classificação do projeto depende de ambas as partes, i.e., apresentação e respostas dadas.

O check point é realizado na semana de **17 a 19 de Fevereiro de 2021**, **na aula PL** a qual os elementos de grupo pertencem. Todos os elementos do grupo têm de estar presentes, caso contrário os alunos em falta obtêm classificação zero.

3.2 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 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 cumprimentos destas regras podem anular a avaliação do trabalho.

O prazo de entrega do Projeto 1 é **domingo, 28 de Fevereiro de 2021, às 23:59 (GMT)**.

3.3 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.

4. Bibliografia

[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