Ciências ULisboa | Informática

Aplicações Distribuídas

2017/18

Projeto 1

1. Descrição geral

A avaliação da cadeira de Aplicações Distribuídas está divida em quatro projetos. Os projetos 1 e 2 terão continuidade, 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 processamento destes 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, ou seja, findo os k bloqueios permitidos o recurso fica inativo/indisponível para bloqueio. Também, o gestor permite y recursos bloqueados em simultâneo. O gestor será concretizado num servidor escrito na linguagem *Python*. A Figura 1 ilustra a arquitetura a seguir pelo servidor de exclusão mútua (*Lock Server*), bem como a estrutura de dados em *Python* que o suporta.

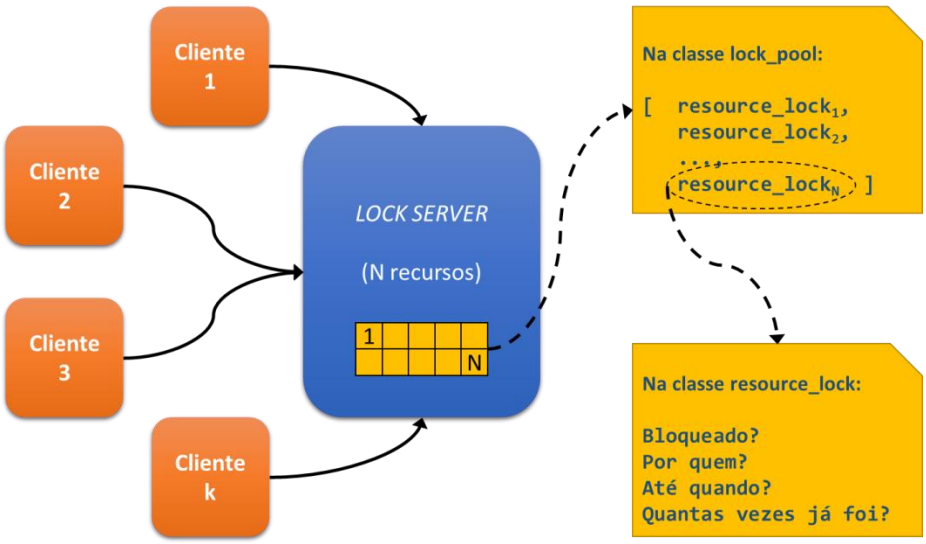

Figura 1– Arquitetura do servidor de *Locks*, e estrutura de dados através de duas classes.

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 do servidor atender múltiplos clientes em simultâneo e de o cliente não necessitar de estabelecer 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 clientes e servidor;
2. Definição da estrutura de dados a ser mantida pelo servidor;
3. Concretização dos processos cliente e servidor.

2.1. Formato das mensagens

O servidor suportará 6 comandos que serão enviados pelo cliente através de *strings* com um formato específico e responderá também com uma string que indica o resultado do processamento do comando. O formato das strings com os comandos bem como o das respostas possíveis do servidor estão detalhados na Tabela 1 . Para outros tipos de mensagens que o servidor receba, a resposta deverá ser a string "UNKNOWN COMMAND".

Note que não se pretende que resolva neste projeto o problema gerado pela possibilidade de uma mensagem ser fragmentada pelo protocolo TCP. Assume-se que a transmissão será ideal, *string* a *string*.

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

Comando	Formato da *string* enviada pelo cliente	Resposta do servidor
LOCK	`LOCK <id do cliente> <número do recurso>`	OK ou NOK ou UNKNOWN RESOURCE
RELEASE	`RELEASE <id do cliente> <número do recurso>`	OK ou NOK ou UNKNOWN RESOURCE
TEST	`TEST <número do recurso>`	LOCKED ou UNLOCKED ou DISABLE ou UNKNOWN RESOURCE
STATS	`STATS <número do recurso>`	<número de vezes que o recurso já foi bloqueado em k> ou UNKNOWN RESOURCE
STATS-Y	`STATS-Y`	<número de recursos bloqueados em y>
STATS-N	`STATS-N`	<número de recursos disponíveis>

O <id do cliente> será um número inteiro distinto para cada cliente. O número do recurso será outro número inteiro entre 1 e N, sendo N o número de recursos geridos pelo servidor de *Locks*. O K será o número máximo de bloqueios permitidos para cada recurso. O Y será o número máximo permitido de recursos bloqueados num dado momento.

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 inativo;
- 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):
        """
        Define e inicializa as características de um LOCK num recurso.
        """
        pass # Remover esta linha e fazer implementação da função

    def lock(self, client_id, time_limit):
        """
        Bloqueia o recurso se este não estiver bloqueado ou inativo, ou mantém o
        Bloqueio se o recurso estiver bloqueado pelo cliente client_id. Neste caso
        Renova o bloqueio do recurso até time_limit.
        Retorna True se bloqueou o recurso ou False caso contrário.
        """
        pass # Remover esta linha e fazer implementação da função

    def urelease(self):
        """
        Liberta o recurso incondicionalmente, alterando os valores associados
        ao bloqueio.
        """
        pass # Remover esta linha e fazer implementação da função

    def release(self, client_id):
        """
        Liberta o recurso se este foi bloqueado pelo cliente client_id,
        retornando True nesse caso. Caso contrário retorna False.
        """
        pass # Remover esta linha e fazer implementação da função

    def test(self):
        """
        Retorna o estado de bloqueio do recurso ou inativo, caso o recurso se
        encontre inativo.
        """
        pass # Remover esta linha e fazer implementação da função

    def stat(self):
        """
        Retorna o número de vezes que este recurso já foi bloqueado em k.
        """
        pass # Remover esta linha e fazer implementação da função
```

```
    def disable(self):
        """
        Coloca o recurso inativo/indisponível incondicionalmente, alterando os
        valores associados à sua disponibilidade.
        """
        pass # Remover esta linha e fazer implementação da função
```

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 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, T):
        """
        Define um array com um conjunto de locks para N recursos. Os locks podem
        ser manipulados pelos métodos desta classe.
        Define K, o número máximo de bloqueios permitidos para cada recurso. Ao
        atingir K, o recurso fica indisponível/inativo.
        Define Y, o número máximo permitido de recursos bloqueados num dado
        momento. Ao atingir Y, não é possível realizar mais bloqueios até que um
        recurso seja libertado.
        """
        pass # Remover esta linha e fazer implementação da função

    def clear_expired_locks(self):
        """
        Verifica se os recursos que estão bloqueados ainda estão dentro do tempo
        de concessão dos bloqueios. Remove os bloqueios para os quais o tempo de
        concessão tenha expirado.
        """
        pass # Remover esta linha e fazer implementação da função

    def lock(self, resource_id, client_id, time_limit):
        """
        Tenta bloquear o recurso resource_id pelo cliente client_id, até ao
        instante time_limit.
        O bloqueio do recurso só é possível se o recurso estiver ativo, não
        bloqueado ou bloqueado para o próprio requerente, e Y ainda não foi
        excedido. É aconselhável implementar um método __try_lock__ para
        verificar estas condições.
        Retorna True em caso de sucesso e False caso contrário.
        """
        pass # Remover esta linha e fazer implementação da função
```

```
def release(self, resource_id, client_id):
    """
    Liberta o bloqueio sobre o recurso resource_id pelo cliente client_id.
    Retorna True em caso de sucesso e False caso contrário.
    """
    pass # Remover esta linha e fazer implementação da função

def test(self,resource_id):
    """
    Retorna True se o recurso resource_id estiver bloqueado e False caso
    esteja bloqueado ou inativo.
    """
    pass # Remover esta linha e fazer implementação da função

def stat(self, resource_id):
    """
    Retorna o número de vezes que o recurso resource_id já foi bloqueado, dos
    K bloqueios permitidos.
    """
    pass # Remover esta linha e fazer implementação da função

def stat_y(self):
    """
    Retorna o número de recursos bloqueados num dado momento do Y permitidos.
    """
    pass # Remover esta linha e fazer implementação da função

def stat_n(self):
    """
    Retorna o número de recursos disponíneis em N.
    """
    pass # Remover esta linha e fazer implementação da função

def __repr__(self):
    """
    Representação da classe para a saída standard. A string devolvida por
    esta função é usada, por exemplo, se uma instância da classe for
    passada à função print.
    """
    output = ""
    #
    # Acrescentar na output uma linha por cada recurso bloqueado, da forma:
    # recurso <número do recurso> bloqueado pelo cliente <id do cliente> até
    # <instante limite da concessão do bloqueio>
    #
    # Caso o recurso não esteja bloqueado a linha é simplesmente da forma:
    # recurso <número do recurso> desbloqueado
    # Caso o recurso não esteja inativo a linha é simplesmente da forma:
    # recurso <número do recurso> inativo
    #
    return output
```

2.3. Concretização do servidor e do cliente

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 ip do servidor que fornece os recursos;
- a porta TCP onde o servidor recebe pedidos de ligação;
- o id único do cliente

De forma cíclica deverá:

1. pedir ao utilizador um comando para enviar ao servidor, através da prompt,
   `comando >`
   (se o comando introduzido for *exit*, o cliente deverá terminar a execução);
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:
    """
    Classe para abstrair uma ligação a um servidor TCP. Implementa métodos
    para estabelecer a ligação, para envio de um comando e receção da resposta,
    e para terminar a ligação
    """
    def __init__(self, address, port):
        """
        Inicializa a classe com parâmetros para funcionamento futuro.
        """
        pass # Remover esta linha e fazer implementação da função

    def connect(self):
        """
        Estabelece a ligação ao servidor especificado na inicialização do
        objeto.
        """
        pass # Remover esta linha e fazer implementação da função

    def send_receive(self, data):
        """
        Envia os dados contidos em data para a socket da ligação, e retorna a
        resposta recebida pela mesma socket.
        """
        pass # Remover esta linha e fazer implementação da função

    def close(self):
        """
        Termina a ligação ao servidor.
        """
        pass # Remover esta linha e fazer implementação da função
```

2.3.2 Servidor

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

- porta TCP onde escutará por pedidos de ligação;
- número de recursos que serão geridos pelo servidor (N);
- número de bloqueios permitidos de cada recurso (K);
- número permitido de recursos bloqueados num dado momento (Y);
- tempo de concessão (em segundos) dos bloqueios.

O tempo limite da concessão de um bloqueio deverá ser manipulado através das funções *time()* e *ctime()* do módulo *time* do *Python* [3]. A primeira 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 segunda recebe como argumento o número de segundos desde a época e retorna uma string com a data e hora locais (útil na implementação de *lock_pool.__repr__()*).

O servidor deverá implementar o seguinte ciclo de operações:

1. aceitar uma ligação;
2. mostrar no ecrã informação sobre a ligação (IP e porta 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;
5. verificar se o número de recursos bloqueados num dado momento já atingiu Y, e não permitir bloqueios a novos recursos;
6. receber uma mensagem com um pedido;
7. processar esse pedido;
8. responder ao cliente;
9. fechar a ligação;
10. mostrar no ecrã informação sobre o estado dos recursos.

3. Check Point

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 **05 a 09 de março, na aula da PL** onde os elementos de grupo pertencem, ou a maioria dos elementos de grupo. Todos os elementos do grupo têm de estar presentes, caso contrário os alunos em falta obtêm classificação zero.

4. Bibliografia

[1] https://docs.python.org/2/tutorial/classes.html

[2] https://docs.python.org/2.7/library/socket.html

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