Comunicação entre Processos

CK0154 – Sistemas Distribuídos - Trabalho II

Javam Machado - Eduardo Rodrigues - Jose Serafim Filho

1 Introdução

O modelo cliente-servidor (em inglês client/server model), em computação, é uma estrutura de aplicação distribuída que distribui as tarefas e cargas de trabalho entre os fornecedores de um recurso ou serviço, designados como servidores, e os requerentes dos serviços, designados como clientes.

Geralmente os clientes e servidores comunicam-se através de uma rede de computadores em computadores distintos, mas tanto o cliente quanto o servidor podem residir no mesmo computador.

Existem dois tipos de protocolos de transporte, TCP(*Transport Control Protocol*) e UDP(*User Datagram Protocol*). TCP é um protocolo confiável, orientado a conexão. UDP é um protocolo de datagramas que não garante confiabilidade de transmissão. Os sockets UDP e TCP são a interface provida pelos respectivos protocolos na interface da camada de transporte.

2 Especificação

2.1 Objetivo

Implementar uma aplicação distribuída simples entre dois ou mais processos(cliente/servidor) através de sockets usando os protocolos de transmissão UDP e TCP.

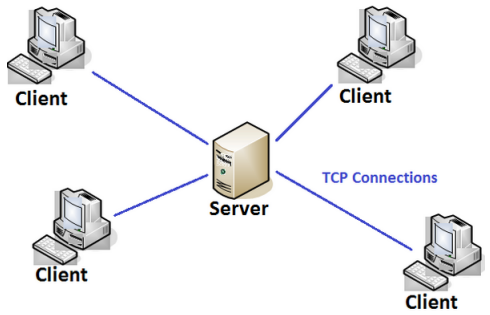

Figure 1: Aplicação distribuída

2.2 Resumo das atividades

Os exercícios a seguir deverão ser feitos usando **UDP**:

1. Crie um cliente que lê como entrada do usuário uma mensagem(sequência de caracteres) e a envia ao servidor. O cliente deve configurar um timeout no socket tal que possa detectar quando o servidor não responder. O servidor deverá retornar a mensagem de forma invertida.

   - O usuário deve especificar qual porta o servidor deve utilizar:
     Ex: UDPserver 7777
   - Para executar a aplicação cliente o usuário deve especificar o IP do servidor, a porta vinculada aquele serviço e a mensagem a ser invertida.
     Ex: UPDClient [*IP*] [*porta*] [*msg*]
   - O cliente deve mostrar a mensagem invertida que recebeu do servidor no terminal. Caso o timeout seja alcançado(servidor fora do ar), uma mensagem de erro deve ser mostrada pelo cliente informando que o timeout foi alcançado.
   - Exemplo caso de sucesso:
     Execute o servidor: UDPserver 7777
     Execute o cliente: UPDClient 127.0.0.1 7777 hello
     Output do cliente: olleh
   - Exemplo caso de falha servidor:
     Execute o cliente: UPDClient 127.0.0.1 7777 hello
     Output do cliente: Timeout

2. Altere o servidor para tratar de solicitações concorrentes, onde para cada mensagem recebida, deverá ser criada uma thread para atendê-la.

Os exercícios a seguir deverão ser feitos usando **TCP**:

1. Crie um servidor que executa operações de soma, subtração, multiplicação e divisão. O servidor retorna para o cliente o resultado dessa operação. O cliente envia ao servidor dois números inteiros e uma string indicando a operação desejada. O cliente deve utilizar serialização no envio da mensagem ao servidor, no qual um objeto da classe Requisicao será enviado na mensagem.

```
class Requisicao {
      int num1;
      int num2;
      String operacao;
        /* Valores possíveis de operacao:
           'soma','sub','div','multi'
        */
}
```

- Utilize a linha de comando para executar o servidor. O usuário deve especificar a porta.
  Ex: TCPServer 7777
- Da mesma forma o usuário deve especificar o endereço do servidor e a requisição:
  Ex: TPCClient 127.0.0.1 7777 10 5 soma
  Nesse caso o cliente faz uma requisição para o servidor no IP 127.0.0.1 na porta 7777 para somar 10 e 5.
- O cliente deve mostrar no terminal o resultado da operação.

2. Altere o servidor para tratar de solicitações concorrentes, onde para cada mensagem recebida, deverá ser criada uma thread para atendê-la.

3. Altere o servidor para que ele salve todas as requisições dos clientes em um log(arquivo de texto). Uma linha do log segue o padrão: [*requisição*][*IP do cliente*]

   Ex:
   [10 5 soma][127.0.0.1]
   [15 3 div][127.0.0.1]
   [8 10 multi][127.0.0.1]

2.3 Critério de avaliação

- Implementar Cliente/Servidor que se comunicam através de UDP(25%).
- Implementar Cliente/Servidor que se comunicam através de TCP(45%).
- Concorrência(30%).
- Para conseguir a totalidade de que cada um dos itens acima, será analisado clareza, organização, corretude e completude.

Para facilitar a correção, cada aluno deverá fazer um documento especificando a matrícula e nome, explicando também o funcionamento de seu programa que deve seguir os padrões apresentados nos exemplos acima.

2.4 Equipe

- Trabalho em dupla;

2.5 Linguagens de programação e API

O trabalho poderá ser realizado nas seguintes linguagens de programação:

- C++
- C
- Java
- Python

2.6 Entrega

O trabalho deverá ser enviado até 23:59 do dia 10/09/17 (domingo) ao Bitbucket.

References

[1] Distributed Systems: Concepts and Design, 5th Edition. George Coulouris, Cambridge University. Jean Dollimore, Formerly of Queen Mary, University of London.

[2] https://pt.wikipedia.org/wiki/Cliente-servidor

[3] https://en.wikipedia.org/wiki/Transmission_Control_Protocol

[4] https://wiki.python.org.br/SocketBasico

[5] https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

[6] https://pt.wikipedia.org/wiki/Serializacao