Servidor de Mensagens

Trabalho Prático Individual

Valor: 25 pontos

Data de entrega: 24 de Outubro de 2022, segunda-feira, até às 23:59.

Introdução

Uma empresa provedora de infraestrutura em núvem deseja reduzir a complexidade das operações de monitoramento e troca de equipamentos dos seus *data centers*. Para isto, deseja-se implementar uma solução que permita a instalação, desistalação e monitoramento de equipamentos de comutação de rede (switches) através de um software. Os components deste *data center*, portanto, são:

Switch: é um equipamento de rede que permite a todos dispositivos do *data center* trocarem informações entre si, por onde trafegam os pacotes de rede.

Os switches instalados em um Rack podem ser de diferentes tipos (tabela 01):

Swtich ID	**Tipo**
01	Switch Topo de Rack
02	Switch de Agregação
03	Switch Virtual
04	Switch de Acesso

Tabela 01 - Descrição dos IDs e tipos de cada equipamento Switch

Rack: racks são estruturas em formato de gabinete, geralmente feitas de metal, que permitem organizar e armazenar diferentes equipamentos de rede como switches, servidores e cabos.

Agente (cliente): Este sistema executa em um servidor e é capaz de solicitar a instalação e desinstalação de equipamentos, e também ler dados de utilização dos equipamentos. Realiza solicitações ao Controlador (servidor), descrito a seguir.

Controlador (servidor): sistema centralizado que recebe e responde às solicitações do agente (cliente). Este software controlador é quem interage diretamente com os equipamentos do centro de dados e realiza instalação, desisntalação, leitura de dados e listagem de equipamentos.

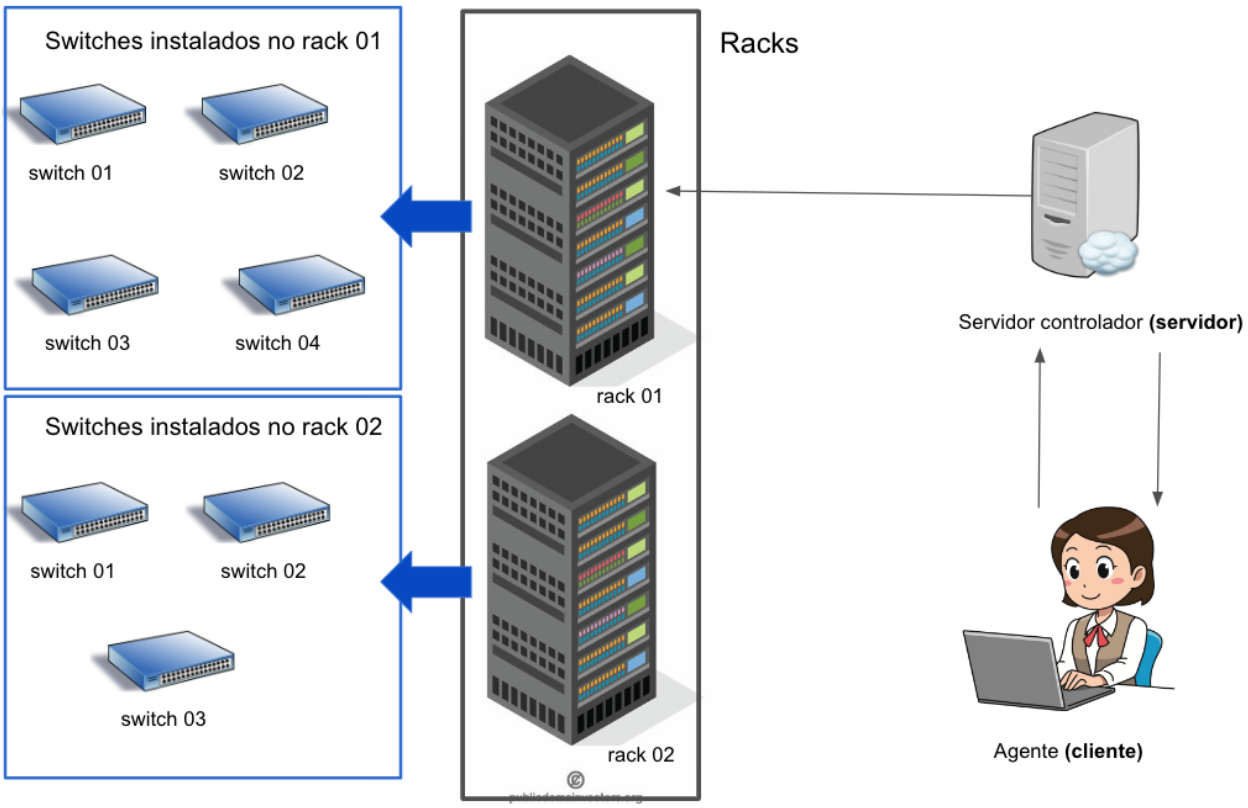

Figura 01 - Exemplo de Centro de Dados

Dado este cenário, seu trabalho será implementar um novo protocolo chamado “Flamingo” que executa em um sistema cliente-servidor, simulando a interação entre o **agente (cliente)** e o **controlador (servidor)**. Cada solicitação feita pelo cliente deve ser enviada ao controlador através de uma mensagem, que será interpretada pelo servidor.

O protocolo Flamingo deve suportar **04** tipos diferentes de mensagens, cada uma correspondendo a uma operação que o cliente pode solicitar ao servidor:

1. Instalar Switch
2. Desinstalar Switch
3. Ler dados do Switch
4. Listar equipamentos de um Rack

Os detalhes do formato de cada uma dessas mensagens está descrito na próxima seção.

Protocolo

O protocolo utilizado para troca de mensagens entre o cliente e o servidor controlador deve ser construído sobre o protocolo TCP. Cada mensagem deve comportar um texto contendo somente letras, números e espaços.

O agente (cliente) deve ser capaz de solicitar ao controlador (servidor) as seguintes operações:

- **Instalar novo switch:** Através da mensagem "**`add sw <switch_id> in <rack_id>`**", onde "`switch_id`" é um valor inteiro numérico representando o tipo de switch a ser instalado (vide tabela 1), e "`rack_id`" é também um valor inteiro numérico que representa o rack físico no centro de dados. Em caso de sucesso o servidor deve responder "**`switch <switch_id> installed`**".

  > Cada rack comporta uma quantidade máxima de 03 *switches* instalados. Caso o controlador envie uma mensagem "`add sw`" em um rack que não comporte mais a instalação de um equipamento, a mensagem de erro "**`error rack limit exceeded`**" deve ser retornada pelo servidor.

  > Cada switch instalado deve corresponder a um tipo existente (01, 02, 03 ou 04), conforme descrito na Tabela 1. Caso o cliente solicite a instalação de um switch com um `<switch_id>` desconhecido, a mensagem de erro "**`error switch type unknown`**" deve ser retornada pelo servidor.

  > Deve ser possível instalar mais de um switch na mesma mensagem do tipo "`add sw`". Os ids dos switches a serem instalados devem ser separados por um espaço, e você deve suportar até 03 switches sendo criados na mesma mensagem. Exemplo: "**`add sw 01 02 04 in 01`**". Neste caso, a mensagem de sucesso deve conter na mesma linha os ids dos switches instalados: **`switch 01 02 04 installed`**".

  > Caso o cliente solicite a instalação de um switch que já foi instalado no rack, a mensagem de erro "**`error switch <switch_id> already installed in <rack_id>`**".

- **Desinstalar switch:** Quando recebida a mensagem "**`rm sw <switch_id> in <rack_id>`**", em caso de sucesso o servidor deve responder "**`switch <switch_id> removed from <rack_id>`**".

> Caso o switch não tenha sido préviamente instalado no rack, a mensagem de erro a seguir deve ser retornada "**`error switch doesn't exist`**".

- **Ler dados de um switch:** Quando recebida a mensagem "**`get <switch_id> in <rack_id>`**", o servior deve retornar a informação sobre o tráfego de entrada no equipamento no último minuto com a mensagem "**`4300 Kbs`**" por exemplo.

  > Caso o switch não tenha sido préviamente instalado no rack, a mensagem de erro a seguir deve ser retornada "**`error switch doesn't exist`**".

  > O agente (cliente) deve ser capaz de solicitar a leitura de mais de um equipamento na mesma mensagem (limite máximo de 2 switches por mensagem), por exemplo "**`get <switch_id_1> <switch_id_2> in <rack_id>`**". Assim o servidor controlador deve responder com a mensagem: "**`4300 Kbs 134 Kbs`**".

- **Listar equipamentos de um rack:** O cliente deve ser capaz de solicitar informações sobre quais são os switches atualmente instalados em um rack. Assim, quando enviada a mensagem "**`ls <rack_id>`**", o servidor deve responder uma mensagem com os dados de quais switches estão atualmente instalados no rack requisitado: "**`<switch_id_1>, <switch_id_2>, <switch_id_3>, …`**".

  > Caso o rack em questão esteja vazio, ou seja, nenhum switch tenha ainda sido instalado nele, a mensagem "**`empty rack`**" deve ser retornada pelo servidor controlador.

Exemplo de execução

cliente > add sw 01 02 in rack 01
servidor > switch 01 02 installed
cliente > add sw 01 in rack 01
servidor > error switch 01 already installed in 01
cliente > ls 01
servidor > 01 02
cliente > rm sw 01 in 01
servidor > switch 01 removed from 01
cliente > ls 01
servidor > 02
cliente > get 01 in rack 01

servidor > 4332 Kbs

Restrições de implementação

Tanto o agente (cliente) quando controlador (servidor) devem ser implementados com POSIX sockets na **linguagem C**, utilizando apenas a **biblioteca padrão de sockets** TCP. Seu programa deve ser executável no sistema operacional **Linux**, não precisa funcionar em Windows.

O cliente deve ler mensagens via input do teclado, e cada mensagem recebida pelo servidor deve ser exibida no terminal. O servidor deve também exibir no terminal cada mensagem recebida do cliente.

O servidor deve suportar conexões do tipo **IPV4** e **IPV6**: use no código as funções `socket(AF_INET, SOCK_STREAM)` e `socket(AF_INET6, SOCK_STREAM)`.

O servidor deve aceitar conexão de **apenas um cliente**. Multiplas conexões de clientes não precisam ser suportadas pelo servidor.

O servidor deve receber, **necessáriamente nesta ordem**, o tipo de endereço a ser utilizado (v4 para IPv4 e v6 para IPv6) e o número de porta na linha de comando especificando em qual porta ele receberá conexões. O cliente deve receber, **necessáriamente nesta ordem**, o endereço IP e a porta de onde estabelecerá conexão. Exemplo de dois programas executando em terminais diferentes:

IPv4
servidor terminal 1 > ./server **v4** 51511
cliente terminal 2 > ./client **127.0.0.1** 51511

IPv6
servidor terminal 1 > ./server **v6** 51511
cliente terminal 2 > ./client **::1** 51511

Todas as mensagens devem ser terminadas com o indicador de quebra de linha "`\n`".

Comandos com mensagens desconhecidas não devem ser suportados. O servidor deve desconectar do cliente caso receba uma mensagem desconhecida, por exemplo "`list <rack_id>`" ao invés de "`ls <rack_id>`", ou "`ad <switch_id>`" ao invés de "`add <switch_id>`".

A qualquer momento, o servidor deve encerrar todas as conexões caso receba do cliente a mensagem “`exit`”.

O datacenter deve comportar um limite **máximo de 04 racks**. Caso o cliente tente instalar, desinstalar, ler dados de switch ou listar equipamentos de um rack que não está dentre os 04 racks existentes, a mensagem de erro "**`error rack doesn't exist`**" deve ser retornada pelo servidor.

Dicas e Material para consulta

Pensar na ordem de leitura e envio das mensagens previamente, antes de escrever o programa pode te ajudar a visualizar o design do protocolo. Pode ser uma boa ideia pensar na ordem de execução das `send()` e `recv()` préviamente.

A playlist de Introdução a Programação em Redes mostra o passo a passo do desenvolvimento de algumas aplicações, com exemplos de troca de mensagens com socket TCP.

O guia do Beej para Programação em Redes é um dos mais completos e didáticos disponíveis na internet. Apresenta em detalhes como funcionam e quando podem ser utilizadas as funções da biblioteca de sockets da linguagem C. Contém muitos exemplos.

Os capítulos 2 e 3 do livro “TCP IP Sockets in C, Second Edition Practical Guide for Programmers” disponível do moodle explica algumas funções da biblioteca padrão de sockets em C, e também apresenta exemplos.

Entrega

Cada aluno deve entregar documentação em formado PDF contendo no **máximo 03 páginas,** sem capa e fonte tamanho 12. Use a documentação para apresentar as dificuldades encontradas durante o desenvolvimento e as soluções adotadas para superar estes desafios. A documentação corresponde a 20% da pontuação do trabalho (5 pontos), portanto, dedique tempo à desenvolvê-la bem.

Cada aluno deve entregar também o código fonte em C, e um Makefile para compilar o programa.

- O Makefile deve compilar o cliente em um binário chamado “client” e o servidor em um binário chamado “server”

- Seu programa deve ser compilado ao se executar apenas o comando “make”, ou seja, sem a necessidade de parâmetros adicionais.

- Os nomes dos arquivos devem ser padronizados: `server.c`, `client.c` e `common.c` ou `common.h` (se necessário utilizar arquivo com funções comuns).

- A entrega deve ser feita no formato ZIP, com o nome seguindo o padrão: TP1_MATRICULA.zip

Importante: Teste seu código **antes** de submeter a tarefa no moodle. Corrija eventuais erros de compilação e tenha certeza de que tudo funcione corretamente.

Avaliação

Este trabalho deve ser feito individualmente, deve ser utilizada linguagem de programação C, e seguir as restrições descritas em "Restrições de implementação".

Para correção, as seguintes funcionalidades serão avaliadas (IPv4 e IPv6):

- Instalar switch (**4 pontos**)
- Desinstalar switch (**4 pontos**)
- Ler dados de um switch (**4 pontos**)
- Listar equipamentos de um rack (**4 pontos**)
- Mensagens inválidas (validações) (**2 pontos**)
- Cliente envia mensagem "exit" para o servidor encerrar a conexão (**2 pontos**)
- Documentação (**5 pontos**)

Importante: Caso as funcionalidades funcionem em apenas um tipo de endereço (IPv4 ou IPv6), a pontuação da respectiva funcionalidade será penalizada em 50% (reduzida pela metade).

Importante 2: Tanto a implementação do cliente quanto do servidor serão avaliadas.

Será utilizado um sistema para detecção de código repetido, portanto não é admitido cópia de trabalhos.

Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.

Desconto de Nota por Atraso

Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:01 do dia seguinte à entrega no relógio do Moodle, os trabalhos já estarão sujeitos a penalidades.

A fórmula para desconto por atraso na entrega do trabalho prático é:

$$desconto = 2d - 1$$

onde *d* é o atraso em dias úteis. Note que após **3 dias**, o trabalho não pode ser mais entregue.