DOCUMENTAÇÃO TP1 - REDES DE COMPUTADORES

Bárbara D'Avila Duarte – 2013062707
Guilherme Viegas de Faria – 2013000760

Considerações prévias

O trabalho teve seus testes finais realizado em um computador pessoal com as seguintes configurações:

Processador: Intel®Core™i5-5257U CPU @ 2.70GHz 3.1GHz
Memória Instalada (RAM): 8 GB 1867 MHz DDR3
Tipo de Sistema: Sistema Operacional de 64 bits, processador com base em x64
Sistema Operacional: Apple macOS Sierra 10.12.4

Foi utilizado para a solução do problema a linguagem Python na versão 2.7.6.

Introdução

Este trabalho consiste em realizar os conceitos de comunicação em sockets usando select (para a comunicação múltipla) para os alunos da disciplina de Redes de Computadores, a fim de fazer várias entidades se comunicarem e trocarem dados ao mesmo tempo.

Sendo mais específicos, a ideia principal é fazer com que várias entidades se comuniquem entre si e troquem informações da seguinte maneira: vários emissores podem enviar mensagens para um ou para vários exibidores, e um emissor pode ter um exibidor associado ou não, assim como emissores e exibidores podem trabalhar de maneira independente.

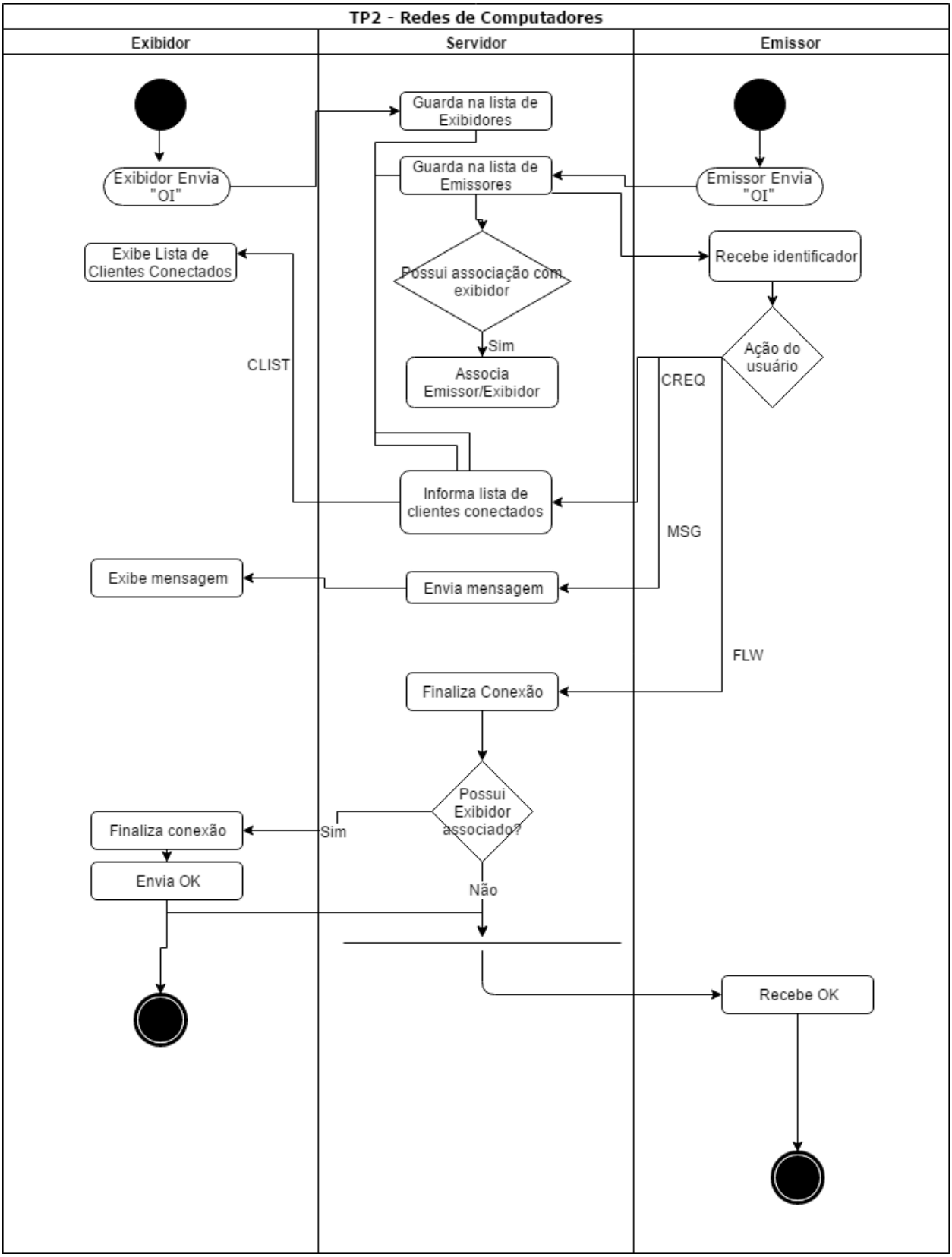
Apenas emissores enviam mensagens e apenas exibidores exibem mensagens, caso a mensagem seja destinada a um emissor, essa mensagem deve ser exibida no exibidor associado a ele ou retornar ERRO caso não possua. A troca de mensagens é feita através de acordo com um protocolo previamente definido, ao receber estes dados a entidade intermediária (servidor) deve validar os dados recebidos, de acordo com este protocolo pré-estabelecido, e redirecionar as mensagens conforme a solicitação do usuário na entidade emissora.

Ao final espera-se que as entidades emissoras e exibidoras se comportem conforme o protocolo pré-estabelecido na divulgação deste trabalho prático.

Modelagem

A modelagem de solução do problema se baseia no conceito de programação em sockets com protocolo TCP/IP, onde há conexão entre as duas entidades. Além das funções de sockets para realizar as múltiplas conexões com os clientes (no caso do servidor).

Para este problema foi proposto a criação de três programas: Um Servidor, responsável por orquestrar as entidades clientes. E dois clientes: Emissor, responsável por enviar as mensagens e por enviar a solicitação de encerramento da conexão, e o Exibidor, que é responsável por exibir as mensagens recebidas.


TP2 - Redes de Computadores
Exibidor
Servidor
Emissor
Guarda na lista de Exibidores
Exibidor Envia "OI"
Guarda na lista de Emissores
Emissor Envia "OI"
Exibe Lista de Clientes Conectados
Possui associação com exibidor
Recebe identificador
Sim
Ação do usuário
CLIST
Associa Emissor/Exibidor
CREQ
Informa lista de clientes conectados
MSG
Exibe mensagem
Envia mensagem
FLW
Finaliza Conexão
Possui Exibidor associado?
Finaliza conexão
Sim
Envia OK
Não
Recebe OK

Solução Proposta

A proposta de solução deste problema é separada em três arquivos/programas principal: “Servidor.py”, “ClienteEmissor.py” e “ClienteExibidor.py”. Para este documento veremos portanto com mais detalhes a solução utilizada para estes programas conforme descrito na modelagem. A solução portanto, pode ser separada em **quatro** tópicos relacionados e detalhados a seguir:

Servidor

1. *Abertura passiva:*

O servidor deve ser o primeiro programa a ser executado. Ao iniciar, o mesmo cria um socket e fica escutando, esperando por conexões através dele. Ao receber uma conexão, um novo socket é atribuído e a comunicação entre o cliente e servidor passará a ser realizada através do mesmo.

2. *Função select:*

Sempre que uma nova conexão é aceita, o socket correspondente é adicionado à lista de *sockets*. Sempre que um desses sockets recebe algum evento, uma mensagem, por exemplo, é enviada do cliente para servidor, o mesmo é adicionado a uma lista de evento *inputready*. Cada evento em *inputready* dispara uma rotina, uma sequência de tarefas, por exemplo, encaminhar mensagem e mandar uma mensagem de confirmação *OK* para o remetente da mensagem.

3. *Solicitação ID:*

Ao receber uma mensagem de OI, o remetente solicita que seja reconhecido pelo servidor, como uma entidade emissora ou exibidora; e no caso de um emissor, ele ainda pode solicitar que seja associado a um exibidor. O servidor atribui um ID de emissor ou exibidor conforme a seguinte regra: caso o remetente da mensagem tenha enviado um 0 no campo *ID_ORIG,* o servidor entende que aquele remetente é um exibidor. Caso o remetente da mensagem envie, no campo *ID_ORIG,* um valor entre 4096 e 8192(não incluso), o servidor entende que o mesmo é um emissor e deseja ser associado com um exibidor cujo ID é igual a *ID_ORIG*. Caso o servidor receba qualquer outro valor fora dessa faixa(4096 á 8191) e diferente de 0, o servidor entende que o mesmo é um emissor e não quer deseja se associar a ninguém.

Mensagens de erro podem ser retornada em 3 situações possíveis: o número de *IDENTIFICADORES_EMISSORES* tenha se esgotado, o número de I*DENTIFICADORES_EXIBIDORES* tenha se esgotado ou o remetente/emissor tente se associar a um exibidor não registrado.

4. *Encaminhamento mensagem:*

Uma mensagem pode ser encaminhada para todos os exibidores, nesse caso o *ID_DEST* da mensagem é igual a 0, a um exibidor, *ID_DEST* entre 4096 e 8192 (não incluso); ou ao exibidor associado a um emissor, *ID_DEST* entre 1 e 4096(não incluso) designando um emissor. O que distingue as duas últimas situações é que na primeira, a mensagem é diretamente encaminhada, caso exista um exibidor com o ID indicado no dicionário de exibidores. Já na segunda situação é realizada uma pesquisa prévia ao dicionário que relaciona emissores e exibidores a fim de resgatar o ID do exibidor associado ao emissor destino da mensagem. Só então a mensagem é encaminhada para o exibidor associado ao emissor destino da mensagem.

Em todas situações, caso seja verificado que não existe o destinatário da mensagem, seja ele exibidor, emissor ou emissor associado, uma mensagem de erro é retornada ao remetente, indicando que não foi encontrado o destinatário da mensagem.

5. Desconexão emissor:

Sempre que um emissor envia um mensagem *FLW* para o servidor, o servidor repassa essa mensagem para o exibidor associado aquele emissor, se houver, e encerra a conexão com o mesmo. O exibidor, ao receber a mensagem *FLW*, trata de encerrar a conexão com o servidor. As referências para as entidades que solicitaram ou para o qual foi encaminhada uma mensagem *FLW* são removidas dos respectivos dicionários.

Da forma como foi implementado o programa exibidor, o mesmo, após sua inicialização, não troca mais mensagens com o servidor. Desta forma o exibidor nunca enviará uma mensagem *FLW* para o servidor e nunca encerrará sua execução por si só. Todavia o servidor foi implementado de forma a contemplar essa hipótese.

6. Envio CLIST:

O mecanismo de envio de CLIST é bastante similar ao mecanismo de envio de mensagem, com a diferença que após o cabeçalho N inteiros, representado os ID dos emissores e exibidores registrado no sistema, são enviados para o emissor ou emissor associado destino da mensagem.

Como exibidor associado entende-se aquele que está associado a um emissor, entidade para o qual um outro emissor enviou uma mensagem ou uma solicitação CREQ. Todavia, emissores não estão aptos a exibir uma mensagem/informação e por isso a mensagem é encaminhada para um exibidor associado ao mesmo.

ClienteEmissor

1. Início da conexão:

Ao iniciar o programa Emissor, o programa envia automaticamente uma mensagem “OI” (3), informando ao servidor que ele está se conectando e solicitando a ele um identificador de origem. É também através desta mensagem que é solicitada a associação deste emissor a um exibidor (que já está em execução) informando no campo IdentificadorDestino (Campos do enquadramento serão mais detalhados abaixo, neste documento) o número do identificador do Exibidor que deverá ser associado.

2. Comunicação com o usuário:

Após a conexão é papel do Emissor fazer a interface de entrada com o usuário. Ou seja, é através dos programas “Emissores” que o usuário poderá enviar mensagens e finalizar as conexões. Para isso a dupla responsável por esta solução optou por uma comunicação simplificada, com apresentação das opções através de mensagens no TERMINAL através da função “print()” e leitura do teclado através da função raw_input().

As opções de ação são apresentadas ao usuário a cada envio de mensagem, identificando se a mensagem deve ou não ser entregue a um único exibidor, se sim, qual exibidor deve ser este. E ao fim do envio da mensagem é questionado se o usuário deseja, ou não, enviar uma nova mensagem.

3. Envio da mensagem:

Após a escolha do usuário entre o envio de um BROADCAST ou UNICAST, e a digitação da mensagem, o programa monta o cabeçalho com as informações inerentes ao protocolo, mais um novo

campo com a quantidade de caracteres presente na mensagem a ser enviada e a mensagem propriamente dita.

Além disso no campo IdentificadorDestino vai o número do identificador do exibir a exibir aquela mensagem, ou número do identificador de um emissor que possui um exibidor associado, para o caso de ser uma mensagem UNICAST. Caso no campo IdentificadorDestino tenha sido informado um identificador de um emissor que não possui um exibidor associado, será recebido um erro, informando que não há um exibidor para aquela mensagem.

4. Encerramento da Conexão:

Quando o usuário informa ao programa que não deseja mais enviar mensagens o emissor envia ao servidor uma mensagem FLW(4). O Programa recebe uma mensagem de OK(1) e pode encerrar a conexão. Além disso com este comando espera-se também que, caso haja um exibidor associado a este emissor, encerre-se também este exibidor associado.

5. Envia CREQ:

Ao enviar uma mensagem desse tipo, o emissor deseja que uma *CLIST*, uma lista de todos os emissores e exibidores conectados ao sistema, seja enviada para um exibidor ou exibidor associado. Uma mensagem de *OK* é retornada caso o *ID_DEST* da mensagem seja encontrado na lista de exibidores ou exibidores associados a um emissor. Caso contrário uma mensagem de *ERRO* é retornada.

ClienteExibidor

1. Início da conexão:

Ao iniciar o programa Exibidor, o programa envia automaticamente uma mensagem “OI” (3), informando ao servidor que ele está se conectando e solicitando a ele um identificador de origem. Exatamente como é feito no programa Emissor exceto pela associação que é feita somente Emissor-Exibidor e não Exibidor-Emissor.

2. Recebimento da mensagem:

Ao receber uma mensagem MSG(5) o programa identifica a origem da mensagem (IdentificadorOrigem, presente no cabeçalho), a quantidade de caracteres que a mensagem possui e a mensagem propriamente dita.

Após isso ela exibe na tela o número do identificador de origem da mensagem e a mensagem como no exemplo abaixo:

“Mensagem de 37: Oi, tudo bem?”

3. Encerramento da Conexão:

Ao receber uma mensagem do tipo FLW(4) o programa envia uma mensagem de OK(1) e encerra a sua conexão. Para exibidores essa mensagem será recebida quando o emissor associado a ele for encerrado.

6. Recebe CLIST::

Um exibidor recebe uma *CLIST* sempre que um emissor envia ao servidor uma mensagem de *CREQ*. Uma *CLIST* é nada mais do que uma lista com os ID de todos emissores e exibidores registrados no sistema.

Tópicos em Comum

1. Comunicação:

A comunicação das entidades é estruturada para que os programas cliente (emissor e exibidor) usem o programa servidor como o centralizador da comunicação. Além disso, para que a comunicação ocorra, de maneira parametrizável, foi definido que os programas cliente deverão escutar no endereço IP fornecido na passagem de parâmetro na chamada do programa e no porto, também passado por parâmetro. O programa Emissor ainda pode receber mais um parâmetro que é o identificador do exibidor ao qual ele deve ser associado. Já o servidor recebe como parâmetro apenas o número do PORTO.

2. Enquadramento:

O enquadramento foi feito da seguinte forma:

- O primeiro campo do cabeçalho da mensagem corresponde ao tipo da mensagem que foi transformado em um inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H). (2bytes, 16bits)
- O segundo campo do cabeçalho da mensagem corresponde ao identificador de origem ou seja, o número (identificador único) da origem da mensagem, neste caso mensagem pode conter ou não dados. Este campo foi transformado em um inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H). (2bytes, 16bits)
- O terceiro campo do cabeçalho da mensagem corresponde ao identificador de destino ou seja, o número (identificador único) do destino da mensagem, neste caso mensagem pode conter ou não dados. Este campo foi transformado em um inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H). (2bytes, 16bits)
- O último campo do cabeçalho da mensagem corresponde ao número sequencial da mensagem propriamente dita, apenas mensagens do tipo 5 podem incrementar este número, que é transformado em um inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H). (2bytes, 16bits)

Para mensagens do tipo 5 há ainda um campo inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H) contendo a quantidade de caracteres a serem lidas na mensagem.

Para mensagens do tipo 7 há ainda um campo inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H) contendo a quantidade (N) de clientes conectados na rede. Além disso, a mensagem do tipo 7 segue com N inteiros hexadecimais enviados pela rede como um *unsigned short* (>H) contendo os identificadores de cada cliente.

3. Execução do código:

Deve-se chamar, por linha de comando, o programa SERVIDOR como descrito abaixo:

python server.py <PORTO>

Deve-se chamar, por linha de comando, o programa EXIBIDOR como descrito abaixo:

python exibidor.py <IP>:<PORTO>

Deve-se chamar, por linha de comando, o programa EMISSOR como descrito abaixo:

python emissor.py <IP>:<PORTO> arg

Onde arg é o número do identificador do exibidor que deve ser associado com aquele emissor, caso não se deseje fazer a associação com um exibidor este campo pode vir em branco.

Sempre que se desejar associar um exibidor a um emissor, o exibidor deverá ser previamente iniciado. A associação entre emissores e exibidores acontece no momento da inicialização do programa emissor.

4. Testes realizados:

Foram realizados testes simulando múltiplos emissores e exibidores registrados no sistema. A seguir será detalhado em tópicos alguns testes que foram realizados:

- Enviar *broadcast:* enviar uma mensagem para todas as entidades exibidoras do sistema (*OK*).
- Enviar mensagem para um exibidor específico (*OK*).
- Enviar mensagem para um emissor que possui um exibidor associado (*OK*).
- Enviar mensagem para um exibidor não registrado no sistema (*ERRO*).
- Enviar mensagem para um emissor não registrado no sistema (*ERRO*)
- Enviar mensagem para um emissor sem um exibidor associado (*ERRO*).
- Tentar inicializar um emissor indicando um exibidor que não esteja registrado no sistema(*ERRO/Ok*). - Neste caso uma primeira mensagem de erro é retornada ao emissor remetente e posteriormente o mesmo reenvia uma mensagem *OI* pedindo para ser registrado no sistema sem estar associado a um exibidor.
- *Broadcast* de *CLIST* (*OK*).
- Enviar *CLIST* para um exibidor específico (*OK*).
- Enviar *CLIST* para um emissor que possui um exibidor associado (*OK*).
- Enviar *CLIST* para um exibidor não registrado no sistema (*ERRO*).
- Enviar *CLIST* para um emissor não registrado no sistema (*ERRO*)
- Enviar *CLIST* para um emissor sem um exibidor associado (*ERRO*).

Como se pode constatar vários testes foram empreendidos a fim de se exaurir ao máximo as possibilidades do sistema. Todavia em situações excepcionais os programas podem não apresentar um funcionamento consistente. Por exemplo no caso de uma quantidade elevada de clientes conectados ao sistema, caso uma um cliente seja desconectado inesperadamente, a remoção do registro do mesmo nos dicionários pode não ser realizado corretamente, gerando, por exemplo, uma inconsistência na *CLIST*.

Possíveis pontos de melhores

Possíveis melhorias podem ser realizados na parte de tratamento de exceções, na usabilidade do sistema, na robustez do servidor. Além disso, melhorias na modularidade do código, principalmente, do servidor podem ser empreendidas.

Conclusão

O objetivo era a implementação, utilizando linguagem Python, de uma entidade multidirecional que enviava uma mensagem para vários clientes no modelo BROADCAST ou para apenas um no modelo UNICAST.

O objetivo foi alcançado após algumas tentativas. As maiores dificuldades encontradas foram a utilização da biblioteca de sockets para orquestrar os vários clientes conectados no servidor sua solução está melhor detalhada no tópico “SOLUÇÃO PROPOSTA” deste documento.

Apesar das dificuldades encontradas, todos os problemas foram corrigidos e o programa está executando conforme esperado.