Servidor de Mensagens Orientado a Eventos

Execução: Individual
Data de entrega: 14 de fevereiro de 2022

INTRODUÇÃO

Para concluir a sequência de trabalhos feitos na matéria de Redes de Computadores, vamos implementar um sistema de compartilhamento de mensagens que se utilize da orientação a eventos para comportar a conexão e a troca de mensagens de diversos clientes simultaneamente.

A NASA está requisitando seu trabalho para auxiliar em uma nova descoberta: **contato com os extraterrestres.** Para isso, você deve criar um canal por onde consiga trocar mensagens com um ou diversos seres alienígenas de diversos planetas diferentes.

Os objetivos gerais deste trabalho são:

1. Implementar um servidor de mensagens orientado a eventos (centrada ao redor da função *select* ou de múltiplas *threads*) utilizando a interface de *sockets* em linguagem C, C++ ou Python*;*
2. Definir o formato das mensagens de controle e do protocolo;
3. Escrever o relatório;

PROBLEMA

Você desenvolverá 3 programas para um sistema simples de troca de mensagens de texto utilizando apenas as funcionalidades da biblioteca de *sockets* POSIX e a comunicação via protocolo TCP. O código do servidor deverá ser organizado ao redor da função *select* ou deve utilizar múltiplas *threads*, usando uma técnica conhecida como *orientação a eventos*.

Três programas devem ser desenvolvidos: um programa **servidor**, que será responsável pelo controle da troca de mensagens, um programa **exibidor** para a exibição das mensagens recebidas e um programa **emissor** (seu amigo extraterrestre) para o envio de mensagens para o servidor. Os programas exibidores e emissores, também chamados de clientes, se comunicam por intermédio do programa servidor. Cada programa cliente se identifica com um valor inteiro único no sistema, alocado pelo servidor. Programas emissores podem enviar mensagens de texto para todos os programas exibidores (*broadcast*) ou apenas para um programa exibidor (*unicast*).

PROTOCOLO

O protocolo de aplicação deverá funcionar sobre TCP. Isso implica que as mensagens serão entregues sobre um canal de bytes com garantias de entrega em ordem, mas é sua responsabilidade determinar onde começa e termina cada mensagem.

Campos Comuns a Todas as Mensagens

- **Tipo da mensagem (2 bytes):** um dos 7 tipos de mensagens definidos a seguir.
- **Identificador de origem (2 bytes):** Cada mensagem carrega o identificador da origem ou um valor pré-definido.

- **Identificador de destino (2 bytes):** Cada mensagem carrega também o identificador do destino.
- **Número de sequência (2 bytes):** Cada mensagem enviada por um programa deve receber um número de sequência, local ao programa. A primeira mensagem deve ter o número de sequência zero e as mensagens seguintes devem ter o número de sequência igual ao da mensagem anterior + 1. O número de sequência deve ser por cliente. Mensagens como MSG, PLANET e PLANETLIST repassam seu número de sequência para os exibidores em questão, que devem adotá-los como novo número de sequência e realizar posteriores incrementações sobre ele.

Tipos de Mensagens

O protocolo possui os seguintes tipos de mensagem. Todos os tipos de mensagem carregam um cabeçalho com todos os campos definidos anteriormente. O número após cada tipo é o valor do campo “Tipo da mensagem" do item anterior.

- **OK (1)**: Todas as mensagens do protocolo devem ser respondidas com uma mensagem de OK ou ERROR. Essa mensagem funcionará como uma confirmação. As mensagens de OK devem carregar o número de sequência da mensagem que está sendo confirmada. **O envio de uma mensagem de OK não incrementa o número de sequência das mensagens do cliente** (mensagens de OK não têm número de sequência próprio).
- **ERROR (2)**: Idêntica à mensagem OK, mas enviada em situações onde uma mensagem, por qualquer motivo que seja, não pode ser aceita ou tenha gerado algum erro. Essa mensagem também é uma espécie de confirmação, porém utilizada para indicar que alguma coisa deu errado. **O envio de uma mensagem de ERROR não incrementa o número de sequência das mensagens do cliente**
- **HI (3)**: Primeira mensagem de um programa cliente (emissor/exibidor) para se identificar para o servidor. Apenas clientes enviam essa mensagem. **O destinatário é sempre o servidor**. Como o cliente envia essa mensagem antes de saber qual é o seu identificador, ele deve preencher **o identificador de origem com o valor zero se ele é um exibidor; um valor diferente de zero identifica um emissor. Se esse valor colocado na origem estiver entre** 2^{12} **e** $(2^{13} - 1)$**, ele identifica um exibidor (que já deve estar em execução) e que deve ser associado àquele emissor.** Se o servidor aceitar a mensagem do cliente, ele envia uma mensagem **OK contendo no campo de destino o identificador que deverá ser usado a partir daí pelo cliente**. Todo cliente deve exibir uma mensagem informando o identificador recebido do servidor.
- **KILL (4)**: Última mensagem de/para um cliente para registrar sua desconexão e saída do sistema. A partir dessa mensagem o servidor/cliente que recebe a mensagem envia um OK de volta e fecha a conexão. **Se o servidor recebe a mensagem de um emissor, ele deve verificar se há um exibidor associado (identificado por ele na mensagem HI). Se houver, ele (servidor) envia uma mensagem KILL para aquele exibidor. Se um exibidor recebe a mensagem, ele deve terminar sua execução depois de responder.**

- **MSG (5)**: Uma mensagem é originada por um emissor, enviada ao servidor, e repassada pelo servidor para um ou mais exibidores. **O emissor deve colocar no campo destino o identificador do exibidor para o qual a mensagem deve ser repassada, ou de um emissor que esteja associado a um exibidor. Se o campo destino possuir o valor zero, o servidor irá enviar a mensagem para todos os exibidores (broadcast). Se o campo destino possuir o identificador de um emissor, a mensagem deve ser enviada para o exibidor associado a ele (se existir) ou um ERROR deve ser retornado. Ao repassar uma MSG, todos os campos, inclusive os campos origem e destino, devem permanecer inalterados.** Uma MSG começa com um **inteiro (2 bytes, *network byte order*) logo após o cabeçalho, indicando o número de caracteres sendo transmitidos, C**. Depois do inteiro, seguem C bytes contendo os caracteres da mensagem em ASCII. Note que o valor C determina quantos bytes a mais devem ser lidos.
- **CREQ (6)**: Enviada pelo emissor para o servidor. **Indica no campo de destino um receptor para o qual deve ser enviada a lista de clientes (emissores e exibidores) que estão conectados ao sistema.** O servidor deve responder com um OK para o emissor da mensagem CREQ. **O servidor deve também enviar uma mensagem do tipo CLIST para o destinatário indicado na mensagem CREQ.** (Opcional: seu servidor pode suportar broadcast de CLIST para todos os exibidores caso o campo destino do CREQ tenha o valor zero).
- **CLIST (7)**: **Essa mensagem possui um inteiro (2 bytes, *network byte order*) indicando número de clientes conectados, N. A mensagem CLIST possui também uma lista de N inteiros (2 bytes cada, todos em *network byte order*) que armazena os identificadores de cada cliente (exibidor e emissor) conectados ao sistema.** Note que o valor N determina quantos valores a mais devem ser lidos. **O remetente dessa mensagem é sempre o servidor e o destinatário é um cliente informado como destinatário da mensagem CREQ.** O cliente deve responder uma mensagem CLIST com uma mensagem OK. **O CLIST incrementa o número de sequência no exibidor.**
- **ORIGIN (8):** Imediatamente após se conectar ao servidor com o comando HI, os programas clientes devem enviar uma mensagem ORIGIN que deve conter apenas a informação de qual planeta ele está fazendo contato. Dessa forma, todos os emissores e exibidores devem ter um planeta associado a eles. **A mensagem deve ser enviada com identificador de origem do emissor que a enviou e destino como sendo o servidor. Após o cabeçalho, deve conter um inteiro P que corresponde ao número de caracteres do nome do planeta que será enviado.** O servidor deve responder essa mensagem com um OK.
- **PLANET (9):** Um emissor pode enviar uma mensagem para descobrir de qual planeta um cliente é. **No campo de destino deve identificar o número do cliente que ele busca descobrir o planeta. O servidor deve responder com um OK para o emissor da mensagem PLANET e enviar para o exibidor associado do destino escolhido o nome do planeta ao qual ele pertence.** Em caso de falha, deve responder com a mensagem ERROR. **A mensagem deve ser integralmente repassada pelo servidor, tal como no caso do tipo MSG (sem alterar origem, destino e sequência).** Broadcast não faz sentido nesse caso. **O PLANET não incrementa o número de sequência no exibidor.**

- **PLANETLIST (10):** O emissor também pode solicitar ao servidor uma lista de TODOS os planetas conectados ao bate-papo (sem repetição) usando o comando PLANETLIST. **Na origem, colocar o cliente que fez a solicitação e no destino o valor do servidor.** O servidor deve responder a mensagem com um OK para o cliente que enviou a mensagem e **uma lista de planetas armazenados para o exibidor associado ao emissor que fez a solicitação. O PLANETLIST não incrementa o número de sequência no exibidor.**

Alocação de Identificadores

O servidor aloca identificadores entre 1 e $(2^{12} - 1)$ para emissores e identificadores entre 2^{12} e $(2^{13} - 1)$ para exibidores. Essa distinção entre identificadores de emissores e exibidores tem o objetivo de facilitar o encaminhamento das mensagens pelo servidor. O servidor tem identificador $(2^{16} - 1)$.

Associação entre Emissores e Exibidores

Não há obrigatoriedade de que um emissor esteja sempre associado a um emissor. Podem haver emissores sem um exibidor, bem como exibidores sem um emissor. A associação acontece no momento da conexão do emissor, se o identificador fornecido for um exibidor presente no sistema. É responsabilidade do servidor manter o controle sobre quem está associado a quem, quando for o caso. Isso pressupõe que a informação sobre a identificação dos exibidores é manipulada pelo usuário: primeiro ele dispara um exibidor e observa o identificador que ele recebe do servidor e exibe na sua saída; em seguida o usuário deve disparar o emissor, passando como parâmetro para ele o identificador que foi informado pelo exibidor.

IMPLEMENTAÇÃO

Pequenos detalhes devem ser observados no desenvolvimento de cada programa que fará parte do sistema. É importante observar que o protocolo é simples e único (o cliente sempre tem que enviar a mensagem codificada para o servidor e vice-versa, de modo que o correto entendimento da mensagem deve ser feito por todos os programas).

Uso de Sockets TCP

Como mencionado anteriormente, a implementação do protocolo da aplicação utilizará TCP. Haverá apenas um socket em cada cliente, independente de quantos outros programas se comunicarem com aquele processo. O programador deve usar as funções *send* e *recv* para enviar e receber mensagens. No caso do servidor, ele deve manter um socket para receber novas conexões (sobre o qual ele executará *accept*) e um socket para cada cliente conectado.

Parâmetros de Execução

O servidor deve ser iniciado recebendo como parâmetro apenas o número do porto onde ele deve ouvir por conexão dos clientes. Um exibidor deve ser disparado com um parâmetro obrigatório que identifica a localização do servidor como um par de endereço “IP:porto”. Um emissor deve ser disparado com um parâmetro obrigatório que identifica a localização do servidor como um par de endereço “IP:porto”, que pode ser seguido por um segundo parâmetro, opcional, com o identificador de um exibidor, caso uma associação deva ser feita.

Emissores e Exibidores

Emissores e exibidores desempenham papéis diferentes, sendo que juntos compõem as duas partes do que seria uma interface completa de um usuário no sistema. Emissores recebem mensagens do teclado e as enviam para o servidor, enquanto exibidores recebem mensagens do servidor e as exibe na tela.

Depois de sua inicialização, o emissor executa um *loop* simples, lendo linhas do teclado, formatando-as como mensagens e enviando-as para o servidor. Sua interface deve oferecer uma forma do usuário identificar o destino (um exibidor específico ou todos os exibidores no sistema), bem como permitir que o usuário indique que deseja montar uma mensagem CREQ, identificando o exibidor que deve recebê-la. Deve também haver uma forma de indicar ao programa que ele deve terminar (quando uma mensagem KILL deve ser enviada para o servidor).

Depois de sua inicialização (mensagens do tipo HI e ORIGIN - note que são os únicos momentos em que o exibidor enviará mensagens), o exibidor também executa um *loop* simples, esperando por mensagens do servidor e exibindo-as na tela. Ele também deve ser capaz de interpretar as mensagens CLIST para apresentar a informação para o usuário. O programa exibidor deve exibir a informação de identificação de quem enviou a mensagem (algo como “Mensagem de 42: Oi, de qual planeta você é?”).

Essas são todas as considerações pré-definidas sobre a parte dos clientes. Sinta-se livre para decidir (e documentar!) qualquer decisão extra. Em particular, é sua tarefa definir a interface de interação do usuário com cada programa.

Servidor

Um sistema simples de mensagens de texto apresenta apenas um processo servidor ou repetidor e sua função é distribuir as mensagens de texto dos emissores para os exibidores a ele conectados. O servidor deve repassar mensagens entre os programas que se identificam por mensagens HI, desconectando os que enviem mensagens KILL.

O código do servidor deverá ser organizado ao redor da função *select* (ou de múltiplas *threads*), que permite a observação de diversos *sockets* em paralelo. O programa deve montar um conjunto de descritores indicando todos os sockets dos quais espera uma mensagem (inclusive o socket usado para fazer o *accept*, que deve ser um só). A função permite indicar sockets também onde se deseja escrever ou onde se procura por alguma exceção, mas esses dois conjuntos não interessam neste trabalho. Também não é necessário usar o temporizador que pode ser associado ao *select*. Uma vez chamada, a função *select* bloqueia até que algo ocorra nos *sockets* que foram indicados na chamada. Ao retornar, o conjunto indica quais sockets tem operações pendentes.

Ao receber qualquer mensagem, o servidor deve primeiro confirmar que o identificador de origem corresponde ao do cliente que a enviou, verificando se o cliente indicado na origem e o cliente que está conectado no *socket* onde a mensagem foi recebida. Esse teste evita que um cliente se passe por outro.

Depois disso, o servidor deve observar o identificador de destino contido na mensagem. Se o identificador de destino indicado na MSG for zero, a mensagem deve ser enviada a todos os exibidores conectados ao sistema. Caso o identificador seja diferente de zero, ele deve verificar se o valor indica um emissor ou um exibidor. Se for um exibidor, o servidor deve procurar pelo registro de um exibidor com o valor indicado e repassar a MSG apenas para aquele cliente. Se for um emissor, o servidor deve verificar se há um exibidor associado a ele e enviar a mensagem para aquele cliente. Caso um exibidor não seja encontrado, o servidor deve responder ao emissor com uma mensagem ERROR, caso contrário uma mensagem OK. Ao enviar uma mensagem MSG a qualquer exibidor, o servidor deve esperar pela mensagem OK em resposta apenas para confirmar que o cliente estava ativo e exibiu corretamente a mensagem. O servidor deve ser capaz de lidar com até 255 clientes, ou seja, deve tratar conexões simultâneas.

Ao receber uma mensagem CREQ de um emissor, o servidor deve consultar seu estado e determinar quais clientes (emissores e exibidores) estão ativos e criar uma mensagem do tipo CLIST com o contador e lista de identificadores de clientes. Como mencionado, CLIST será enviada para um exibidor identificado pelo emissor.

Controle de Recebimento de Mensagens

Como mencionado anteriormente, toda mensagem deve ser respondida com uma mensagem OK ou ERROR. Isso tem um efeito de confirmar se a mensagem foi processada corretamente pelo servidor. Note que ela não é necessária para confirmação da entrega, já que TCP é usado. Ela é necessária apenas para a confirmação no nível da aplicação de que a mensagem foi aceita e processada (ou não).

AVALIAÇÃO

O trabalho deve ser realizado individualmente e **deve ser implementado com a linguagem de programação C, C++ ou Python** utilizando somente a biblioteca padrão (interface POSIX de

sockets de redes). Deve ser possível executar seu programa no sistema operacional **Linux** e **não deve utilizar bibliotecas Windows, como o winsock**. Seu programa deve interoperar com qualquer outro programa implementando o mesmo protocolo (você pode testar com as implementações dos seus colegas). Procure escrever seu código de maneira clara, com comentários pontuais e bem indentados. Isto facilita a correção dos monitores e tem impacto positivo na avaliação.

Entrega

Cada aluno deve entregar documentação em PDF de até 6 páginas, sem capa, utilizando fonte tamanho 10, e figuras de tamanho adequado ao tamanho da fonte. **Ele deve conter uma descrição da arquitetura adotada para o servidor, os refinamentos das ações identificadas no mesmo, as estruturas de dados utilizadas, as decisões de implementação não documentadas nesta especificação e os comandos necessários para executar os programas**. Como sugestão, considere incluir as seguintes seções no relatório: introdução, arquitetura, emissor, exibidor, servidor, discussão e conclusão. O relatório deve ser entregue em formato PDF. A forma de modularização do código fica a seu critério, mas é importante descrever no relatório como compilar, executar e utilizar seus programas. A documentação corresponde a 40% dos pontos do trabalho, mas só será considerada para as funcionalidades implementadas corretamente.

Será utilizado um sistema para detecção de código repetido, portanto não é admitido cola de trabalhos.
Será adotada a 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.
Não será permitida a entrega do trabalho após a data de entrega especificada.

Cada aluno deve entregar, além da documentação, o **código fonte em C** e um **Makefile** para compilação do programa. Instruções para submissão e compatibilidade com o sistema de correção semi-automática:

- O Makefile deve compilar o “emitter”, o “exhibitor” e o “server”.
- Seu programa deve ser compilado ao se executar apenas o comando “make”, ou seja, sem a necessidade de parâmetros adicionais.
- A entrega deve ser feita no formato ZIP, com o nome seguindo o seguinte padrão: TP3_MATRICULA.zip
- O nome dos arquivos deve ser padronizado:
    - server.c
    - emitter.c
    - exhibitor.c
    - common.c, common.h (se houver)

Dicas e Cuidados

- O guia de programação em rede do Beej (http://beej.us/guide/bgnet/) e o Python Module of the Week (https://pymotw.com/2/select/) tem bons exemplos de como organizar um servidor com select.
- Procure escrever seu código de maneira clara, com comentários pontuais e bem identado.
- Consulte o monitor antes de usar qualquer módulo ou estrutura diferente dos indicados.
- Não se esqueça de conferir se seu código não possui erros de compilação ou de execução.
- Implemente o trabalho por partes. Por exemplo, crie o formato da mensagem e tenha certeza que o envio e recebimento da mesma está correto antes de se envolver com o sistema de mensagens.

Prazo de entrega

Os trabalhos poderão ser entregues até às 23:59 (vinte e três e cinquenta e nove) do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:00 do dia seguinte à entrega no relógio do Moodle, os trabalhos **não poderão ser entregues**.

EXEMPLOS

Terminal 1 - Emissor (ID = 1)	**Formatação da Mensagem**
> hi	| 3 | 4097 | 65535 | 0 |
< “ok 1”	| 1 | 65535 | 1 | 0 |
> origin 5 marte	| 8 | 1 | 65535 | 1 | 5 |
< “ok”	| 1 | 65535 | 1 | 1 |
> msg 4097 8 bom dia!	| 5 | 1 | 4097 | 2 | 8 |
> msg 0 23 de qual planeta você é?	| 5 | 1 | 0 | 3 | 23 |
> creq 4096	| 6 | 1 | 4096 | 4 |
< “ok”	| 1 | 65535 | 1 | 4 |
> planet 4097	| 9 | 1 | 4097 | 5 |
< “ok”	| 1 | 65535 | 1 | 5 |
> planet 4096	| 9 | 1 | 4096 | 6 |
< “ok”	| 1 | 65535 | 1 | 6 |
> planet 1	| 9 | 1 | 1 | 7 |
< “ok”	| 1 | 65535 | 1 | 7 |
> planetlist	| 10 | 1 | 65535 | 8 |
< “ok”	| 1 | 65535 | 1 | 8 |
> msg 2 3 olá	| 5 | 1 | 2 | 9 | 3 |
< “error”	| 2 | 65535 | 1 | 9 |
> kill	| 4 | 1 | 65535 | 10 |

< “ok” | 1 | 65535 | 1 | 10 |
========== Finalizar a execução==========

Terminal 2 - Exibidor (ID = 4097)

> hi | 3 | 0 | 65535 | 0 |
< “ok 4097” | 1 | 65535 | 4097 | 0 |
> origin 6 netuno | 8 | 4097 | 65535 | 1 | 6 |
< “ok” | 1 | 65535 | 4097 | 1 |
< msg from 1: “bom dia!” | 5 | 1 | 4097 | 2 | 8 |
< msg from 1: “de qual planeta você é?” | 5 | 1 | 0 | 3 | 23 |
< planet of 4097: “netuno” | 9 | 1 | 4097 | 5 |
< planet of 1: “marte” | 9 | 1 | 1 | 7 |
< planetlist: “netuno marte júpiter” | 10 | 1 | 4097 | 8 |
< “kill” | 4 | 65535 | 4097 | 9 |
(Retorna ok e finaliza)
========== Finalizar a execução==========

Terminal 3 - Exibidor (ID = 4096)

> hi | 3 | 0 | 65535 | 0 |
< “ok 4096” | 1 | 65535 | 4096 | 0 |
> origin 7 júpiter | 8 | 4096 | 65535 | 1 | 7 |
< “ok” | 1 | 65535 | 4096 | 1 |
< msg from 1: “de qual planeta você é?” | 5 | 1 | 0 | 3 | 23 |
< clist: “3 1 4096 4097” | 7 | 65535 | 4096 | 4 |
< planet of 4096: “júpiter” | 9 | 1 | 4096 | 6 |

Terminal 4 - Servidor (ID = 65535)

< initialized server 65535
< received hi
< received netuno from 4097
< received hi
< received marte from 1
< received hi
< received júpiter from 4096
< sent message from 1 to 4097
< ok from 4097
< sent message from 1 to 0
< ok from 4096 4097
< received creq from 1 to 4096
< ok from 4096
< received planet from 1 to 4097
< received planet from 1 to 4096
< received planet from 1 to 1
< received planetlist from 1 to 4097
< sent message from 1 to 2
< error from 2 (Emissor não existe)

< received kill from 1 (Estende o kill para o exibidor associado 4097)
< ok from 4097