Trabalho Prático 3 – Redes de Computadores

Vitor Rodarte Ricoy - 2019007112

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

1. Introdução

O trabalho proposto tem como objetivo 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 implementação consiste em três elementos: um servidor, um emissor e um exibidor. O servidor recebe conexões de emissores e exibidores e trata as mensagens que eles enviam para ele, além de associar cada cliente com um planeta e emissores a um exibidor. Os emissores recebem comandos do teclado e os enviam ao servidor. Por fim, os exibidores recebem mensagens do servidor e as exibe na tela.

2. Arquitetura

A arquitetura adotada para o servidor envolve multithreading, pois já possuo alguma familiaridade com programação paralela e me pareceu mais direto desenvolver dessa forma do que com o comando select. O servidor inicia e começa a esperar por conexões. Quando uma conexão é recebida, uma thread é criada para receber mensagens do cliente que se conectou e ela é encerrada em duas situações: quando o cliente é um exibidor e ele já identificou o seu planeta ou quando o cliente é um emissor e ele enviou o comando kill. O servidor também registra diversos dados dos clientes. Quando um cliente se conecta e envia o comando “hi” o servidor atribui um id a ele e salva em um mapa o seu socket, associando o seu id com o socket. Se o cliente for um emissor, o servidor também salva o id do exibidor associado a ele em um mapa. Também é mantido pelo servidor dois conjuntos de valores: os ids de emissores e os ids de exibidores.

A arquitetura adotada para o exibidor é simples: ele se conecta no servidor, envia automaticamente os comandos “hi” e “origin”, lendo apenas o nome do planeta da entrada padrão, e espera por mensagens do servidor indicando comandos a serem exibidos. A arquitetura do emissor também é simples, sendo que ele também se conecta no servidor e envia os comandos “hi” e “origin” automaticamente, lendo o nome do planeta da entrada, e espera por comandos do teclado com mensagens a serem enviadas ao servidor.

Também foram definidos dois pontos na arquitetura do código. Primeiro, foi criada uma biblioteca separada para centralizar códigos em comum aos clientes e ao servidor, chamada “common”. Essa biblioteca possui a definição da estrutura “Mensagem” que representa uma mensagem enviada no sistema, tanto de um cliente para o servidor quanto do servidor para um cliente; uma função “sairComMensagem” que encerra a execução do programa exibindo uma mensagem de erro; uma função “enviarMensagem”, que envia no socket indicado a mensagem recebida por parâmetro; e uma função “receberMensagem” que recebe uma mensagem no socket indicado e a retorna.

A estrutura “Mensagem” definida possui os seguintes campos:

- tipo: esse campo da estrutura é do tipo unsigned short e define o campo do cabeçalho que indica o tipo da mensagem.

- idOrigem: esse campo da estrutura é do tipo unsigned short e define o campo do cabeçalho que indica o id do cliente remetente da mensagem.

- idDestino: esse campo da estrutura é do tipo unsigned short e define o campo do cabeçalho que indica o id do cliente destinatário da mensagem.

- numSeq: esse campo da estrutura é do tipo unsigned short e define o campo do cabeçalho que indica o número de sequência da mensagem.

- texto: esse campo representa o corpo da mensagem e é do tipo std::string do C++.

- valida: esse campo indica se a mensagem foi lida corretamente, ou seja, se ela possui uma estrutura válida e foi recebida por completo.

A outra biblioteca criada é chamada “estruturas” e ela segue um formato de monitor, centralizando todas as estruturas de dados usadas no servidor e garantindo que o acesso a elas é sincronizado para as threads dos clientes. No caso essa biblioteca declara uma classe que possui diversos métodos que são responsáveis por alterar as estruturas de dados de acordo com a sua semântica. Esses métodos são: “obterProximoIdEmissor”, “obterProximoIdExibidor”,

“insereNovoExibidor”, “insereNovoEmissor”, “inserePlaneta”, “obterPlaneta”, “obterTodosPlanetas”, “obterExibidores”, “obterEmissores”, “removeExibidor”, “removeEmissor”, “isEmissor”, “isExibidor”, “temExibidorAssociado”, “obterExibidorAssociado” e “obterSocket”. Acredito que o nome das funções já são autoexplicativos e, caso não estejam claro o suficiente, o código da biblioteca possui comentários descritivos.

Finalmente, o envio e recebimento de mensagens, implementado na biblioteca “common” possui a seguinte lógica. Para o envio, os campos do cabeçalho (incluindo o tamanho da mensagem do corpo) são convertidos para network byte order e em seguida são convertidos para dois unsigned chars, já que os cabeçalhos eram unsigned short (2 bytes). Essa conversão é feita usando bit shifting e a operação lógica and. Os valores unsigned char do cabeçalho são adicionados em um array de caracteres sem sinal, seguidos pelo corpo da mensagem. Por fim, é adicionado um \n na última posição do vetor para indicar quando a mensagem termina. Vale notar, que o vetor de caracteres possui um tamanho igual ao número de caracteres do corpo da mensagem somado de 10. Após a construção desse vetor de caracteres, ele sofre um cast para o tipo signed char, e é enviado para o destino. Por fim, é verificado se todos os bytes do vetor foram de fatos enviados. Podemos concluir assim, que uma mensagem enviada no sistema segue o formato: “<tipo><id_origem><id_destino><num_seq><tamanho><corpo>\n”, sendo que os cinco primeiros valores são unsigned short, ou seja, inteiros sem sinal de 2 bytes, em network byte order.

Já para o recebimento de mensagens, são recebidos bytes até que o último byte recebido seja um \n. Se o recv falhar em algum momento, lendo nenhum byte, é retornada uma mensagem com a flag “valida” igual a falso. Após a leitura de toda a mensagem, os 10 primeiros bytes do vetor de caracteres recebido são convertidos de volta para os cinco valores do cabeçalho. Isso é feito por meio de um casting do ponteiro para cada posição par do vetor de caracteres para um ponteiro de unsigned short, sendo que cada duas posições do vetor vira um valor. Após essa conversão, os valores do cabeçalho são convertidos de volta do network byte order e o corpo da mensagem é lido. Por fim, uma instância da estrutura Mensagem é retornada com os valores do cabeçalho e com o corpo da mensagem.

3. Servidor

O funcionamento geral do servidor é, como descrito anteriormente, escutar por conexões de clientes, receber uma conexão de um cliente, criar uma thread para tratar as mensagens desse cliente e voltar a esperar por novas conexões. Dessa forma, o servidor é capaz de lidar com múltiplos clientes ao mesmo tempo, já que cada cliente é atendido por uma thread própria. Quanto à saída do terminal, o servidor imprime apenas as mensagens recebidas seguidas do id de quem as enviou. Por exemplo, ao receber uma mensagem “kill” do cliente de id 5, o servidor imprime “Recebido kill de 5”.

Já as threads têm um comportamento diferente de acordo com o tipo do cliente. Quando uma thread recebe o comando “hi” do cliente ela identifica se o cliente é um exibidor ou emissor e registra no servidor tanto o id do cliente quanto o exibidor associado, caso o cliente seja um emissor. Após o comando “hi” o fluxo esperado é receber um comando “origin”, identificando de qual planeta o cliente é. Se o cliente for um exibidor a sua thread salva o planeta recebido associado ao id do cliente e é encerrada em seguida. Se o cliente é um emissor, a thread também salva o planeta recebido associado ao id do cliente, e segue esperando por novas mensagens.

Vale notar que as threads também verificam, a cada mensagem recebida, com exceção da mensagem “hi”, se o socket em que a mensagem foi recebida é de fato do cliente identificado pelo cabeçalho de origem da mensagem. Caso não seja uma mensagem de erro é enviada, já que o cliente está se passando por outro. A thread também verifica se a mensagem recebida está em um formato válido, encerrando sua execução caso não esteja.

O funcionamento do servidor, de acordo com o tipo das mensagens é o seguinte:

- Quando uma mensagem “ok” ou “error” é recebida, a thread apenas imprime que recebeu tal mensagem e volta a esperar por novas mensagens.

- Quando uma mensagem “hi” é recebida, a thread identifica se o cliente que a enviou é um exibidor ou emissor. Se for um exibidor, um id é obtido para ele, ele é inserido no conjunto de estruturas de dados do servidor e uma mensagem “ok” é enviada de resposta. Se for um emissor, um id é obtido para ele, ele é inserido nas estruturas de dados do servidor e um “ok” é enviado de resposta. Vale notar que a inserção do emissor nas estruturas de dados envolve salvar o seu exibidor associado, caso ele tenha.

- Quando uma mensagem “kill” é recebida, é verificado se o cliente que a enviou é um emissor, já que apenas emissores devem enviar esse comando. Caso seja um emissor, é verificado se ele possui um exibidor associado e, caso ele possua um exibidor associado, é encaminhada a mensagem “kill” ao exibidor e ele é removido das estruturas de dados do servidor. Finalmente, o emissor também é removido das estruturas de dados do servidor e uma mensagem “ok” é

encaminhada a ele. Após o tratamento da mensagem a thread do emissor é encerrada.

- Quando uma mensagem “msg” é recebida, é verificado se o remetente dela é um emissor, caso não seja um erro é enviado. Caso contrário, é verificado se o destinatário da mensagem é um emissor ou exibidor. Caso seja um emissor, a mensagem é encaminhada para o exibidor associado a ele. Caso seja um exibidor a mensagem é encaminhada para o próprio exibidor. Caso o destinatário da mensagem seja o id de broadcast, a mensagem é encaminhada para todos os exibidores registrados no servidor. Por fim, se o destinatário não for um valor válido uma mensagem “erro” é retornada. Caso contrário, uma mensagem de “ok” é retornada.

- Quando uma mensagem “creq” é recebida, é verificado se o remetente dela é um emissor, caso não seja uma mensagem “erro” é enviada. Caso contrário, é construída uma mensagem “clist” contendo a lista de ids dos emissores e exibidores, nessa ordem, separados por espaço, em formato de string. Essa mensagem construída é enviada, de acordo com o id de destino, para o exibidor indicado, para o exibidor associado ao emissor indicado ou para todos os exibidores (broadcast). Após o envio de “clist” uma mensagem de “ok” é retornada para o remetente da mensagem “creq”.

- Quando uma mensagem “origin” é recebida, é verificado se o cliente que a enviou já foi registrado, caso contrário, um erro é retornado. Caso ele tenha sido registrado, o planeta indicado por ele é salvo nas estruturas de dados do servidor e uma mensagem de “ok” é retornada ao remetente. Se a thread for de um exibidor, ela é encerrada assim que a mensagem “origin” é respondida.

- Quando uma mensagem “planet” é recebida, é verificado se o cliente do qual queremos descobrir o planeta já foi registrado, caso não tenha sido um erro é retornado. Depois disso é verificado se o cliente destino é um emissor ou exibidor. Caso seja um emissor, a mensagem contendo o planeta dele próprio é enviada para o seu exibidor associado. Caso seja um exibidor, a mensagem contendo o seu planeta é enviada para o próprio exibidor. Em ambos os casos uma mensagem de “ok” é retornada para o remetente da mensagem “planet”.

- Quando uma mensagem “planetlist” é recebida, é verificado se o emissor possui um exibidor associado. Caso ele possua a lista de planetas é enviada ao seu exibidor associado e uma mensagem de “ok” é retornada. Caso contrário, apenas a mensagem de “ok” é retornada.

- Se qualquer outra mensagem for recebida uma mensagem de “erro” é enviada ao remetente.

4. Emissor

O funcionamento geral do emissor é conectar ao servidor, enviar automaticamente uma mensagem de “hi”, imprimir o id recebido, enviar uma mensagem “origin”, lendo o nome do planeta da entrada, e esperar por novos comandos vindos da entrada. O exibidor associado ao emissor é definido pelo segundo parâmetro do programa, que deve indicar o id de um exibidor. Caso seja um id válido de um exibidor, é verificado se esse exibidor existe, caso ela exista o emissor é associado a ele, caso ele não exista um erro é retornado pelo servidor como resposta ao comando “hi”. Ou seja, ao enviar um id que esteja fora do intervalo de 4096 a 8191 o emissor fica sem um exibidor associado. Ao enviar um id nesse intervalo, ele deve corresponder a um exibidor válido, caso contrário o programa do emissor é encerrado com erro.

O emissor imprime no terminal, ao receber uma resposta para sua mensagem, “< ok” ou “< error”, de acordo com a mensagem de resposta. Vale notar que caso a mensagem recebida esteja em um formato inválido, é impresso “< error”. Também, ao receber uma mensagem pelo teclado, ele sempre imprime “> “ antes da mensagem.

Finalmente, as mensagens válidas que podem ser recebidas pelo emissor e seus formatos são:

- “msg <id_destino> <mensagem>”, em que o conteúdo de “<mensagem>” é enviado para o servidor tendo como id de destino no cabeçalho o id “<id_destino>”. Vale notar que “<mensagem>” pode conter espaços e que caso “<id_destino>” seja zero a mensagem é de broadcast.

- “creq <id_destino>”, em que o id de destino no cabeçalho é enviado com o id “<id_destino>”.

Vale notar que caso o id de destino seja zero, a lista de clientes é enviada em broadcast.

- “planet <id_destino>”, em que o id de destino no cabeçalho é enviado com o id “<id_destino>”.
- “kill”, em que é enviado o comando “kill” do emissor para o servidor.
- “planetlist”, em que é enviado ao servidor o comando “planetlist”.
- Qualquer outro comando digitado gera uma saída de “comando inválido”.

Vale notar que é responsabilidade do usuário digitar as mensagens corretamente, sendo que o uso incorreto das mensagens, tendo um comando correto, pode levar a comportamentos inesperados.

5. Exibidor

O funcionamento geral do exibidor é conectar ao servidor, enviar automaticamente uma mensagem de “hi”, imprimir o id recebido, enviar uma mensagem “origin”, lendo o nome do planeta da entrada, e esperar por novas mensagens vindas do servidor, para exibi-las. O exibidor envia o destino do comando “hi” como 0, assim como determinado na especificação.

O exibidor imprime no terminal, as mensagens enviadas ao servidor, no caso de “hi” e “origin”, precedidas de “> “. Ao receber qualquer mensagem o exibidor imprime o seu conteúdo precedido de “< “. Sempre que uma mensagem é recebida, o exibidor verifica o formato da mensagem e retorna uma mensagem de “ok” caso ele esteja correto e de “erro” caso ele esteja incorreto.

O comportamento do exibidor de acordo com a mensagem recebida é o seguinte:

- Mensagem “ok”: é impresso que um ok foi recebido seguido do id do remetente. Esse fluxo não faz muito sentido, mas foi adicionado para tratamento de bugs.
- Mensagem “erro”: é impresso que um erro foi recebido seguido do id do remetente. Esse fluxo não faz muito sentido, mas foi adicionado para tratamento de bugs.
- Mensagem “kill”: é impresso que a mensagem “kill” foi recebida do remetente indicado pelo cabeçalho e o exibidor responde “ok” ao servidor e encerra sua execução em seguida.
- Mensagem “msg”: primeiramente é verificado se a mensagem é de broadcast, caso seja é impresso que uma mensagem de broadcast foi recebida do remetente, segundo o cabeçalho, e a mensagem em si é impressa em seguida entre aspas. Caso não seja de broadcast é impresso que foi recebida uma mensagem do remetente indicado no cabeçalho, seguido da mensagem em si entre aspas. Em ambos os casos o exibidor também envia a mensagem de “ok” ao servidor.
- Mensagem “clist”: ao receber a mensagem "clist", o exibidor imprime que foi recebido um “clist” do remetente indicado no cabeçalho seguido da lista, entre aspas, dos identificadores dos clientes, separados por espaços.
- Mensagem “planet”: ao receber a mensagem “planet” o exibidor imprime que foi recebida uma mensagem “planet” seguido do id de destino da mensagem (o cliente cujo o nome do planeta é recebido) e do nome do planeta entre aspas. Após isso uma mensagem de “ok” é retornada ao servidor.
- Mensagem “planetlist”: ao receber a mensagem “planetlist”, o exibidor imprime que essa mensagem foi recebida do remetente apontado pelo cabeçalho seguido da lista de nomes dos planetas registrados no servidor separados por espaços entre aspas. Após isso, a mensagem de “ok” é retornada ao servidor.
- Qualquer outra mensagem recebida é respondida com uma mensagem de “erro”.

6. Decisões de Implementação

Diversas decisões de implementação já foram explicitadas acima. Outras decisões também serão omitidas por estarem claramente descritas na especificação do trabalho. Assim, as decisões tomadas que são relevantes o suficiente para serem citadas são:

- O servidor possui um parâmetro opcional especificando se ele deve funcionar com IPv4 ou

IPv6. Por padrão o servidor funciona com IPv6.

- Todas as mensagens enviadas na rede foram padronizadas, como discutido no fórum de dúvidas. Todas elas possuem os cabeçalhos pedidos como unsigned short em network byte order, sempre é enviado o tamanho do corpo da mensagem, também como unsigned short em network byte order, mesmo que ele esteja vazia, e o corpo da mensagem é sempre uma string. Vale notar que, pela especificação, a mensagem “clist” deveria enviar uma lista de inteiros, mas como foi discutido no fórum, optei por enviar esse conteúdo como uma string para padronizar o funcionamento das mensagens e a estrutura delas.

- Emissores criados com o campo de destino do cabeçalho com um valor fora do intervalo de 2^{12} a $2^{13} - 1$, são criados sem um exibidor associado. Já se um emissor especifica um id válido de exibidor que não corresponde a um exibidor em execução uma mensagem do tipo “erro” é retornada pelo servidor.

- Toda mensagem diferente de “hi” e “origin” verifica se o remetente é um exibidor, caso não seja uma mensagem do tipo “erro” é retornada.

- Se um emissor for criado sem um exibidor associado recebido por parâmetro, o id enviado pelo “hi” é 16384, logo o emissor é criado com sucesso sem um exibidor associado, por padrão.

- A thread do exibidor no servidor é encerrada assim que ele registra o seu planeta.

- Tanto a lista de “planetlist” quanto a de “clist” vêm ordenadas.

- As estruturas de dados estão encapsuladas dentro de uma classe com o comportamento de monitor, ou seja, ela sincroniza o acesso aos seus métodos por meio de um mutex.

- Todos os ids podem ser reaproveitados, sendo que ao enviar um “hi” o cliente recebe o menor id ainda não alocado do seu tipo (emissor ou exibidor). Ou seja, ao dar um kill em um emissor, por exemplo, o seu id é liberado para ser associado a um possível novo emissor futuramente.

- Se uma mensagem do tipo mensagem, clist ou planetlist têm como destino um emissor sem um exibidor associado, nada é enviado pelo servidor e ele retorna uma resposta do tipo “ok” ao emissor que enviou a requisição.

- Se o destino de uma mensagem do tipo mensagem ou clist não for um exibidor válido, um emissor válido ou broadcast, a resposta do servidor é do tipo “erro”.

- Se o destino de uma mensagem do tipo planet não for um exibidor ou emissor válido, a resposta do servidor é do tipo “erro”.

7. Instruções para Execução

Os códigos foram feitos em C++, para serem executadas em um sistema operacional Linux. Todas as bibliotecas usadas, com exceção da biblioteca pthreads, são bibliotecas padrão do C++. Primeiramente, deve ser executado o comando “make” na raíz do projeto.

Para rodar o servidor, deve-se usar o comando “./server <porta>” ou “./server <porta> <protocolo>”. O parâmetro da porta indica a porta que o servidor será executado, já o parâmetro do protocolo indica se o servidor usará IPv4 ou IPv6, sendo que “v4” indica o primeiro e “v6” o segundo. **Por padrão o servidor utiliza IPv6.**

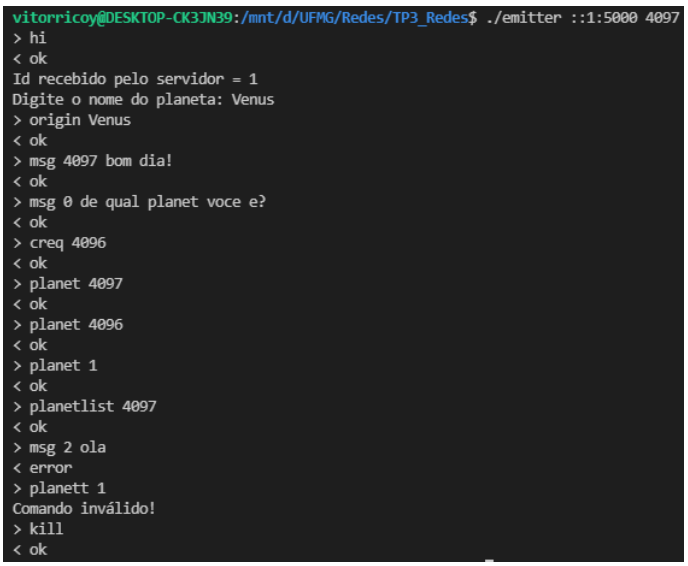
Para rodar o emissor, deve-se usar o comando “./emitter <endereco_servidor>” ou “./emitter <endereco_servidor> <id_exibidor>”. O parâmetro de endereço do servidor deve seguir o formato “IP:porta”, lembrando que o IP fornecido deve seguir o protocolo escolhido para o servidor, IPv4 ou IPv6. Já o parâmetro de id do exibidor indica qual será o exibidor associado ao emissor.

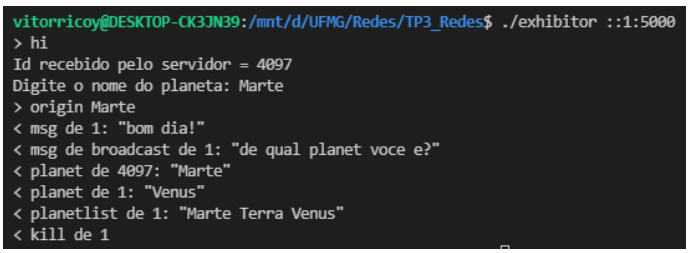
Por fim, para rodar o exibidor, deve-se usar o comando “./exhibitor <endereco_servidor>”, sendo que o parâmetro de endereço do servidor deve seguir o formato “IP:porta”, lembrando que o IP fornecido deve seguir o protocolo escolhido para o servidor, IPv4 ou IPv6.

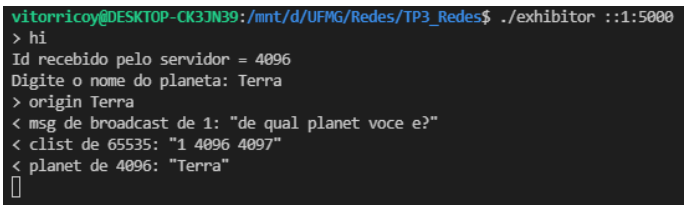
8. Testes

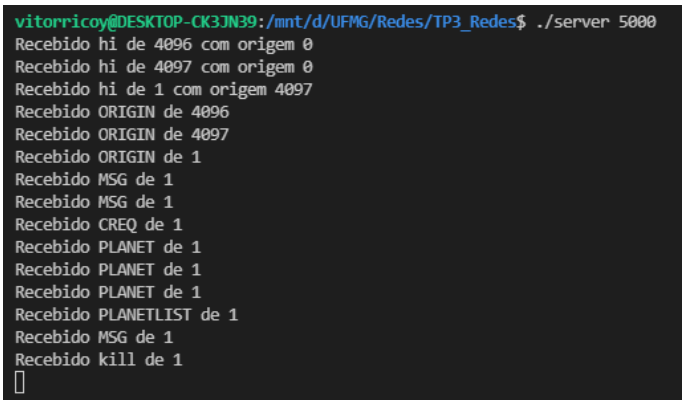
Para testar o meu código executei o cenário de exemplo da documentação, com algumas pequenas mudanças. Seguem os prints da execução, sendo que os dois primeiros programas foram

encerrados e os dois últimos não:









9. Conclusão

O trabalho foi dividido na implementação do emissor, exibidor e do servidor. Achei que a implementação desses programas, com destaque ao servidor devido ao seu uso de multithreading, ajudaram consideravelmente a entender como funciona um sistema de compartilhamento de mensagens orientado a eventos que suporta a comunicação simultânea de diversos clientes.

Os desafios que enfrentei para realizar o trabalho foram razoáveis e puderam ser superados sem maiores dificuldades. Esse trabalho prático foi o trabalho com menos impedimentos e problemas até o momento na disciplina, tanto pelo uso de C++, quanto pela minha familiaridade com o código de sockets ter aumentado bastante com os dois últimos trabalhos. Já as soluções adotadas por mim foram o ponto central do trabalho, pois representam o entendimento que tive do funcionamento esperado desse tipo de sistema e é a parte que consumiu o maior tempo de desenvolvimento.

Finalmente, achei o trabalho muito bom para o aprendizado prático de sockets em um contexto de orientação a eventos e multithreading. Também achei produtivo ter que pensar como fazer o servidor e os clientes se comunicarem de acordo com a especificação.