Relatório do trabalho prático Parte 1

Amaury Teixeira Cassola (00287704)
Arthur Bastos Fanck (00208361)
Gustavo Oliva Barnasque (00263056)

29 de Setembro de 2021

Universidade Federal do Rio Grande do Sul
INF01151 - Sistemas Operacionais II - Turma B
For: Alberto Egon Schaeffer Filho

---

1. Funções implementadas

O projeto implementa todos os comandos especificados no enunciado do trabalho. As trocas de mensagens entre servidor e clientes foram feitas utilizando as primitivas *send* e *receive* dos sockets UNIX, após se criar uma conexão TCP entre os dois. Quando uma nova conexão é criada, são utilizados semáforos para garantir que a nova *thread* utilizará o *file descriptor* do *socket* correspondente. Se não fosse feito isso, a nova *thread* poderia acabar referenciando o *socket* da *thread* anterior.

É possível criar um novo usuário ou iniciar a sessão de um usuário já criado iniciando-se o aplicativo cliente, dado que o aplicativo servidor já esteja rodando e que sejam informados o *username*, IP e porta do servidor. É possível que um mesmo usuário tenha até duas sessões simultâneas. Quaisquer tentativas de acesso do mesmo usuário após duas sessões terem sido abertas serão impedidas pelo servidor. Múltiplos clientes diferentes podem estar logados simultaneamente.

Utilizando o comando FOLLOW, o usuário é capaz de se registrar como seguidor de outro usuário, assim recebendo todas as novas mensagens deste usuário. Um usuário não é capaz de seguir a si mesmo ou a perfis não cadastrados.

Utilizando o comando SEND, o usuário é capaz de enviar novas mensagens ao servidor e, consequentemente, aos seus seguidores. Todos os seguidores com sessões ativas receberão as mensagens instantaneamente, enquanto que os seguidores que não estão *online* as receberão assim que iniciarem uma nova sessão.

Terminar o processo do aplicativo cliente resulta na desconexão deste com o servidor, efetivamente terminando sua sessão.

Os perfis dos usuários e as informações de seguidores são mantidas mesmo que o servidor seja desligado, de modo que todas essas informações são recuperadas no momento em que o aplicativo do servidor é reiniciado.

1. Criação de um novo usuário e início da sessão

Ao que um novo acesso é requisitado por um cliente, é criada uma nova *thread* no servidor para atender às requisições deste cliente. Esta *thread* ficará responsável por receber os comandos do cliente e processá-los. Cada cliente terá uma *thread* própria.

Logo que a conexão é firmada entre cliente e servidor, o aplicativo cliente envia um comando CONNECT com o *username* informado pelo usuário na chamada do programa. Se o *username* ainda não estiver cadastrado, é criado um novo perfil (ver sessão Estruturas Utilizadas) e inserido na lista de perfis. Por outro lado, se o *username* já estiver cadastrado, o servidor checa se já existem duas sessões ativas daquele perfil. Se já existirem, a nova é desconectada.

Se tudo der certo, a sessão do usuário é inicializada. A *thread* deste usuário bloqueará esperando um novo comando dele.

No aplicativo do cliente, é criada uma nova *thread*, tendo então dois fluxos de execução. O primeiro vai bloquear esperando input do teclado do usuário e, quando receber, vai enviar os comandos ao servidor. O segundo, por outro lado, vai fazer uma chamada de *receive* e bloquear esperando notificações vindas do servidor. Sendo assim, temos uma *thread* para lidar com o *input* do usuário e outra apenas para lidar com as notificações e as imprimir para o usuário.

2. Comando FOLLOW

Se o *username* informado estiver registrado na lista de perfis, o usuário é adicionado à lista de seguidores deste perfil. O próprio aplicativo do cliente o impede de tentar seguir a si mesmo e o servidor checa se ele já segue o *username* informado, caso no qual nada é executado. Além disso, se o usuário tentar seguir um perfil que não existe ainda, a ação também não terá sucesso. A própria *thread* do usuário que fez a requisição vai modificar a lista de seguidores do outro usuário, o incluindo nela.

3. Comando SEND

Este comando é seguido de uma mensagem de texto de até 128 caracteres (o aplicativo cliente garante esta limitação). Ao receber este comando, o servidor cria uma nova Notificação (vinculada a um ID único), a adiciona na lista de notificações recebidas deste usuário e itera sobre todos os seguidores do usuário, adicionando a nova notificação nas listas de notificações pendentes de recebimento correspondentes. Depois disso, uma *thread* de envio de notificações é sinalizada através de um semáforo. Esta *thread* itera sobre todas as notificações pendentes de todos os perfis e envia a notificação para todos que estiverem online, retirando-a das listas de notificações pendentes. Assim, todos os seguidores online recebem as novas mensagens instantaneamente.

4. Envio de notificações atrasadas

Quando uma nova sessão é inicializada, se o usuário já tiver sido cadastrado anteriormente, o servidor checa sua lista de pendências na *thread* nova deste próprio usuário. Se houverem pendências, a *thread* de envio de notificações será sinalizada, enviará todas elas para o usuário e retirará elas da lista de pendências, assim entregando todas as notificações que foram armazenadas enquanto o usuário estava *offline*.

5. Desconexão do usuário

Quando o aplicativo usuário é terminado (utilizando o comando CTRL+C ou CTRL+D), ele envia ao servidor um comando DISCONNECT. Ao receber este comando, o servidor desfaz a conexão via *socket* com o cliente e finaliza a *thread* deste cliente, terminando sua sessão. Se houver outra sessão deste mesmo cliente ativa, ela não será afetada.

6. Persistência dos dados

Quando o processo servidor é encerrado (através do comando CTRL+C), os perfis, incluindo suas relações de seguidores, são salvos em disco, Além disso, todos os clientes são notificados e desconectados, assim fechando com sucesso todos os *sockets* em uso. Isto se dá através da função *gracefullShutDown*.

Quando o servidor for reiniciado, todos os dados estarão disponíveis em disco e as estruturas serão preenchidas.

2. Estruturas Utilizadas

1. Perfil

É a estrutura que representa um perfil de usuário cadastrado no sistema. Inclui os seguintes campos:

_usuario: uma string contendo o nome do usuário

_seguidores: um vetor de strings listando todos os seguidores deste usuários (novos seguidores são adicionados diretamente neste vetor)

_notificacoesRecebidas: um vetor de Notificacao (definida a seguir) que lista todas as notificações que este usuário criou.

_notificacoesPendentes: um vetor de pares <*string*, *integer*> que lista as notificações que este usuário ainda precisa receber, com a string sendo o *username* de onde veio a notificação e o integer sendo no ID da notificação em si.

_socketDescriptors: Um vetor de no máximo 2 valores que armazena os *file descriptors* referentes aos *sockets* sendo utilizados por sessões deste usuário. Se o usuário não estiver *online*, este vetor estará vazio.

_semaphorePerfil: um semáforo que controla o acesso a este perfil. Isto é importante dado que múltiplas threads poderiam modificar estas estruturas. Isto inclui tanto a possibilidade de termos múltiplas sessões do mesmo cliente (ambas podendo seguir novos usuários, por exemplo) quanto a necessidade de modificar a lista de notificações a serem recebidas quando novas notificações são criadas.

2. Notificação

É a estrutura utilizada para representar uma notificação a ser enviada para os seguidores de algum usuário. Inclui:

_id: um ID numérico único que identifica esta notificação.

_timestamp: o momento em que a notificação foi recebida.

_mensagem: a cadeia de caracteres enviada pelo usuário.

_tamanho: o número de caracteres da mensagem.

_quantidadeSeguidoresAReceber: a quantidade de perfis que devem receber esta notificação.

3. Pacote

Foi criada uma estrutura para abstrair os pacotes a serem enviados através dos *sockets* entre clientes e servidor. Esta inclui:

_*tipo*: um *enum* que identifica esta mensagem como sendo um comando ou uma transferência de dados.

_*timestamp*: informa o momento em que o pacote foi enviado.

_*status*: um *enum* que identifica se este pacote é uma mensagem de erro ou não.

_*comando*: No caso de ser um comando, este *enum* identifica de qual comando se trata

_*usuario*: Quem enviou o pacote

_*payload*: Uma string com o conteúdo de fato do pacote, se houver.

_*tamanhoPayload*: o tamanho do *payload*.

4. Lista de Perfis

Todos os perfis cadastrados são mantidos em um vetor de Perfil na memória do servidor. Este vetor é global e acessível por todas as *threads* do Servidor.

Para manter a persistência dos dados, mesmo após o servidor ser desligado, foi criado um arquivo do tipo .csv, onde ficam armazenados os perfis cadastrados e seus seguidores.

3. Ambientes de testes

Foram utilizados três ambientes de testes principais para testar o projeto ao longo do seu desenvolvimento. Foram eles:

- **Ambiente 1**:

  OS: Mx-Linux 19.2 v. 20200801

  Compilador: g++ version 8.3.0 (Debian 8.3.0-6)

  Thread Model do compilador: posix

- **Ambiente 2**:

  OS: Ubuntu 64-bit emulado através do Oracle Virtualbox

  Compilador: g++ version 8.3.0 (Ubuntu 8.3.0-6ubuntu1)

  Thread Model do compilador: posix

- **Ambiente 3:**

  OS: macOS Big Sur Version 11.5.1

  Compilador: clang++ version 13.0 (clang-1300.0.29.3), com as flags -std=c++11 e -stdlib=libc++

  Thread model do compilador: posix

4. Problemas Encontrados

O controle de concorrência entre as várias threads do servidor se provou um grande problema. Levamos algum tempo para identificar e tratar as várias condições de corrida que aconteciam entre os diferentes usuários tentando modificar valores no servidor ao enviar novas mensagens ou seguir uns aos outros. Por fim, utilizamos principalmente semáforos para remediar o problema e conseguimos implementar a concorrência satisfatoriamente.

Tivemos cuidado com a interface do usuário, para que o uso do aplicativo em si fosse minimamente intuitivo. No projeto dessa interface, consideramos a ideia de dividir ela em duas telas separadas, uma apenas para imprimir as notificações recebidas pelo usuário e outra para agir como caixa de texto, onde o usuário entraria seus comandos. Infelizmente, não tivemos tempo suficiente para implementar isso e a Interface do usuário acabou sendo de uma tela única. O uso da interface é possível e efetivo, embora um pouco mais bagunçado do que gostaríamos.

Por fim, vale citar o problema encontrado ao tentar enviar as notificações antigas a um cliente que anteriormente estava *offline*. Como estamos lidando com cada notificação como um Pacote, simplesmente enviamos cada uma delas com a primitiva *send* através do *socket* correspondente ao usuário destino. Do lado do cliente, utilizamos a primitiva *receive* para receber estas notificações. Porém, ao invés de receber cada uma delas, o cliente recebia apenas a primeira e a última. O problema se deu pelo fato de TCP funcionar como um *stream* de dados e não como envio de datagramas atômicos. Assim sendo, múltiplos *sends* executados com pouco tempo de diferença seriam todos recebidos pelo mesmo *receive* como um único pacote, cabendo ao receptor interpretar os dados corretamente. Para resolver o problema, criamos uma função de bufferização dos dados recebidos, passando a interpretar eles como uma potencial lista de notificações, ao invés de uma única notificação por vez. Assim sendo, o cliente passou a receber todas as notificações pendentes e as interpretar corretamente.