# Trabalho Prático 1: Controle de Envio de Arquivos

## Redes de Computadores

Thiago Cleto Miarelli Piedade

Matrícula: 2020007058

## 1. Introdução

Este relatório visa documentar as decisões de design e análisar os resultados do primeiro trabalho prático da disciplina de redes de computadores. Nela, tivemos que implementar um sistema de transferência de arquivos usando a API POSIX, o sistema de transferência de arquivos no Linux.

## 2. Configuração da conexão

A conexão foi feita utilizando a estrutura de Sockets da biblioteca *sockets.h*. Para isso, utilizei as informações de protocolo (IPv6 ou IPv4) e porta que a comunicação se daria. A configuração do socket diferiu entre o servidor e o cliente.

No cliente, encapsulei a configuração do socket na função `setup_client()`. Ela é responsável por chamar o parser do input, converter o endereço de string para `sockaddr_storage` e chamar a conexão para esse endereço.

Já no servidor, a lógica é parecida, com o setup encapsulado na função `setup_server()`, mas com a única diferença de, ao invés de pedir para conectar, ele faz o bind da porta e fica escutando por pedidos de conexão.

O envio de informação é todo feito por strings e é manipulado pelas funções `sendMessage()` e `receiveMessage()`. Elas recebem o socket e realizam validações do sucesso do envio ou recebimento.

## 3. Envio de Arquivos

O programa lida com 6 tipos primários de arquivos: .txt, .cpp, .c, .py, .java e .tex. Ao usar o comando select file com algum deles, é feita a seguinte verificação:

1. O arquivo inserido existe?
2. A extensão é válida?

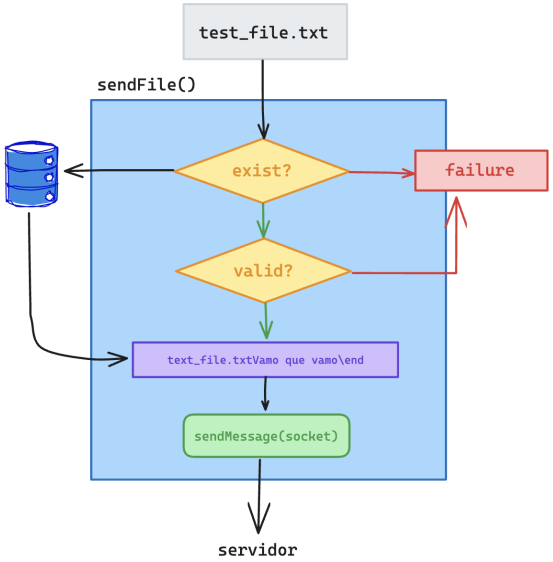
Note que, caso o arquivo não exista e a extensão não seja válida, o erro gerado será apenas ***filename does not exist.*** Para validar a extensão, buscamos o último ponto da string e comparamos as extensões válidas com a string a partir daquele ponto.

Após a verificação, copiamos o filename_candidate para a string filename. Essa decisão de separação das duas variáveis foi necessária pois, caso usássemos a mesma, teríamos o seguinte edge-case:

1. Tenta selecionar o arquivo `text.txt`. O input handler salva o valor do terceiro parâmetro como o nome do arquivo em filename.

2. O file handler encontra o arquivo e valida a extensão, deixando o arquivo pronto para ser enviado.
3. O arquivo é enviado e recebido com sucesso.
4. Agora, tentamos selecionar o arquivo `invalid.exe`. O input handler vai salvar na variável filename.
5. O file handler vai reprovar o arquivo e não salvar na variável content.
6. Ao clicar em send file, o esperado é que o arquivo enviado seja o último arquivo que deu certo, ou seja, `text.txt`. Contudo, como o input handler salvou `invalid.txt` na variável filename, enviaremos a concatenção de `invalid.exe` com o conteúdo de `text.txt`, gerando um erro no servidor.

A função `sendFile()` usa as validações descritas acima e concatena o filename com o conteúdo do arquivo. Por fim, ela envia a string resultante para o servidor usando a função `sendMessage()`:



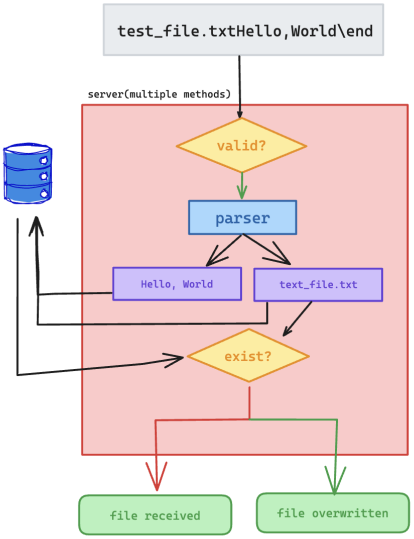

## 4. Recebimento de Arquivos

Do lado do servidor, ficamos esperando pela mensagem. Ao recebê-la, faremos algumas checagens. A primeira delas verifica se a mensagem segue um de dois formatos: o primeiro é se a mensagem contem uma extensão em algum local de sua extensão e termina com `\end`. Para isso, usamos uma biblioteca de regex.


A segunda possibilidade aceita é se a mensagem é exatamente “exit”.

Após validar a composição da mensagem, passamos a mensagem por um parser. Ele vai procurar a primeira extensão válida no texto e retornar a posição do seu último índice. O que faremos é salvar o conteúdo em um buffer e o filename em outro.

Por fim, o conteúdo é salvo na pasta server_files com o nome descrito e retorna se o arquivo foi salvo ou sobrescrito:




## 5. Comando Exit

Ao receber o comando exit, o cliente envia uma mensagem com exit para o servidor. Ao detectar esse cenário, o servidor envia uma mensagem com confirmação que a conexão foi encerrada e fecha o socket do seu lado. O cliente, ao receber a confirmação, também fecha o socket desse lado e sai com sucesso.

Aqui, considerei que apenas o client deve dar exit da aplicação ao se desconectar. **Nesse cenário, o servidor apenas desconecta o cliente e fica disponível para conexão por outro cliente.**

## 6. Como executar

Para executar o sistema, use os seguintes comandos em um computador com Linux:

1. Execute make clean para remover executáveis prévios
2. Execute make para compilar os dados do sistema
3. Inicialize o servidor com `./server <v4/v6> <port>`, sendo `<v4/v6>` a escolha do protocolo e `<port>` a porta do sistema que usaremos para nos conectar.
4. Do lado do cliente, inicialize-o com `./server <address> <port>`, sendo `<address>` o endereço IP local e `<port>` a porta usada no servidor. Via de regra temos os seguintes endereços de IP:

   **IPv4:** `127.0.0.1` **IPv6:** `::1`

5. Use os comandos para manipular dados no cliente:
   a. `select file <filename>`: seleciona um arquivo para envio da memória (quando existe e extensão é válida). **O arquivo deve estar na raíz do diretório.**

b. send file: Envia o arquivo para o servidor conectado
c. exit: desconecta o cliente

## 7. Conclusão

O projeto consistiu na implementação de um sistema de transferência de arquivos utilizando a API POSIX e a biblioteca sockets.h no ambiente Linux. Foram desenvolvidas funcionalidades tanto para o cliente quanto para o servidor. O cliente era capaz de configurar a conexão com o servidor, selecionar arquivos para envio e encerrar a conexão, enquanto o servidor aguardava por pedidos de conexão, recebia arquivos enviados pelo cliente e encerrava as conexões conforme necessário.

No trabalho prático, foram implementadas as principais funcionalidades do sistema de transferência de arquivos, incluindo a configuração da conexão entre cliente e servidor por meio de sockets, o envio e recebimento de arquivos, a validação de extensões de arquivos e a verificação de existência dos mesmos. Além disso, o sistema permitia que o cliente encerrasse a conexão de forma adequada, sendo que o servidor ficava disponível para novas conexões. No geral, o projeto foi bem-sucedido na criação de um sistema funcional de transferência de arquivos utilizando a API POSIX e sockets no ambiente Linux.