INF01151 Sistemas Operacionais II N
Relatório do Trabalho Prático 1

Augusto Falcão Flach e Nikolas Sousa Weige
06 de fevereiro de 2023

Introdução:

O objetivo deste trabalho foi implementar uma aplicação similar ao Dropbox, utilizando sockets e TCP(Transmission Control Protocol). A aplicação permite ao usuário o compartilhamento e a sincronização automática de arquivos entre diferentes dispositivos de um mesmo usuário. A linguagem de programação utilizada foi C++.

A máquina utilizada para testes foi com Ubuntu 22.04.1 LTS, com 8GB RAM e um I3 6a geração. O compilador foi o G++

Comandos de usuário:

Para iniciar uma sessão com o servidor o usuário deve utilizar o seguinte comando:

./cliente <username> <server_ip_adress> <port> :

Onde *<username>* representa o identificador do usuário, *<server_ip_adress>* o ip do servidor e *<port>* a porta do servidor. Após a sessão ter sido iniciada o usuário tem acesso aos seguintes comandos:

- *upload <path/filename.ext>* : Faz o upload do arquivo *filename.ext* para o servidor, salvando-o no sync_dir e propagando para todos os dispositivos daquele usuário.
- *download <filename.ext>* : Faz o download do arquivo *filename.ext* do servidor, salvando-o no diretório local do usuário.
- *delete <filename.ext>* : Exclui o arquivo *filename.ext* do sync_dir.
- *list server* : Lista os arquivos relacionados ao usuário salvos no servidor.
- *list client* : Lista os arquivos salvos no sync_dir.
- *get sync dir* : Cria o diretório sync_dir e inicia a sincronização.
- *exit* : Fecha sessão com o servidor.

Concorrência:

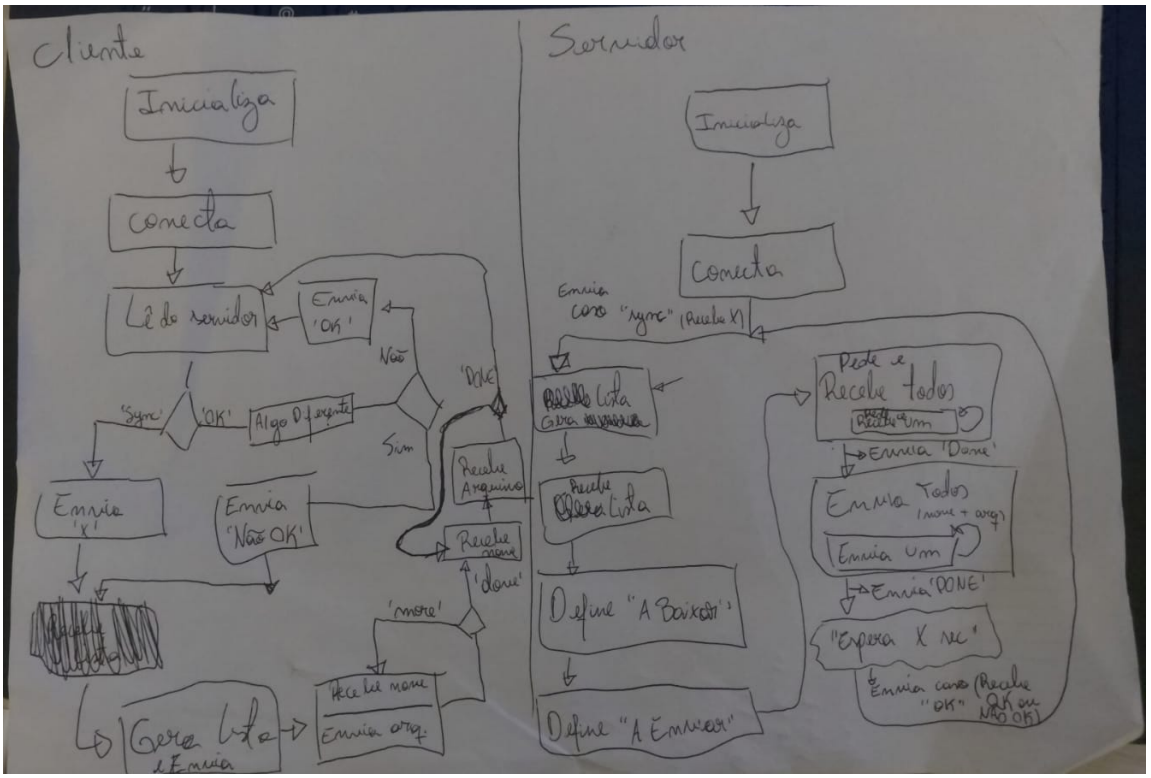
Para isso, foi somente utilizado um sistema de threads. Cada conexão gerada criava uma nova thread, que gerenciava toda a comunicação com o cliente. Essa thread tem um ciclo de leitura/escrita, trocando mensagens

alternadamente com o cliente. Cada um possui uma vez, e isso se alterna. O “detentor da vez” tem o papel de enviar uma “solicitação de procedimento”, que pode ser de sincronização, de upload, de download, de saída, nenhuma operação (0), etc., e do outro lado, o outro tem o papel de escutar a solicitação do procedimento.

Sincronização:

Sobre a concorrência, foi somente utilizado um semáforo somente para as funções de acessos aos arquivos na sincronização (receber algum arquivo e enviar algum arquivo). Não é a melhor solução possível, mas já garantia que duas threads não tentassem acessar o mesmo arquivo ao mesmo tempo.

A principais funções são de sincronização (sync_files_procedure_srv e cli_transaction_loop). O processo de sincronização se dá mais ou menos como na imagem abaixo (modelagem inicial):



Essa versão ainda não contempla a comunicação alternada. O servidor recebe uma lista (do cliente) de todos arquivos, com seus respectivos mtimes, gera uma lista própria, e então gera uma lista de quais arquivos devem ser enviados, e quais devem ser recebidos. Um a um, envia para o cliente uma solicitação de upload ou de download, e então é feita a sincronização com base no papel que cada um recebeu:

```
#define SERVER_SYNC_UPLOAD 1
#define SERVER_SYNC_DOWNLOAD 2

#define CLIENT_SYNC_DOWNLOAD 1
#define CLIENT_SYNC_UPLOAD 2

#define SERVER_SYNC_KEEP_FILE 0
#define CLIENT_SYNC_KEEP_FILE 0
```

Estruturas e funções:

Como citado anteriormente, a ideia desse mecanismo de troca de mensagens é possibilitar que tanto o servidor quanto o cliente possam iniciar qualquer tipo de comunicação, não dependendo de o outro iniciar o processo.

Dividimos o código entre bibliotecas .h para separar os tipos de funções (por não saber trabalhar com classes em c++ especificamente). Também temos uma biblioteca shared.h para compartilhar as funções de leitura, escrita, listagem de arquivos, etc. Isso é para ajudar a garantir a simetria de comunicação (quando um escreve, o outro lê, e vice-versa).

Comunicação

Foi utilizado o mecanismo de sockets no desenvolvimento, como requisitado. Para fazer a transferência de dados, generalizamos uma função de send_data_with_packages, que envia qualquer tipo de dado utilizando a estrutura de packets recomendada no enunciado. Primeiro os dados são quebrados em packets, então é enviado ao cliente/servidor o número de packets a serem repassados, e após isso é feito o envio de cada um. Ao fim, quem recebeu os pacotes refaz os dados concatenando os bytes recebidos.

Dificuldades:

Trabalhar com a criação de arquivos deu semanas de bugs, pois tivemos muitos problemas tentando fazer a criação dos mesmos enviando e recebendo os dados pelos pacotes. O que nos ajudou muito foi generalizar o envio de dados através de pacotes, assim tínhamos como enviar quaisquer estruturas de dados de um lado para o outro sem muita complexidade.