Relatório de Sistemas Operacionais II N

Nome: *Bruno Ferreira Aires* ***Cartão:*** *******
Nome: *Bruno Grohs Vergara* ***Cartão:*** *324256*
Nome: *Erick Larratéa Knoblich* ***Cartão:*** *324422*

Trabalho Prático: Parte 1

Prof. Alberto Egon Schaeffer Filho

Universidade Federal do Rio Grande do Sul
Instituto de Informática

Introdução

Neste relatório, são mostradas informações sobre o desenvolvimento da Etapa 1 do trabalho prático da cadeira INF01151 - Sistemas Operacionais II N, cujo objetivo foi a implementação de um serviço semelhante ao Dropbox, permitindo o compartilhamento de arquivos entre diferentes dispositivos por meio da rede. É feita uma descrição dos ambientes de desenvolvimento e teste do trabalho prático, detalhes da implementação de concorrências, que trechos foi necessária a implementação de formas para garantir sincronização, além de uma breve explicação das funções e métodos desenvolvidos e problemas encontrados durante a realização do trabalho.

Ambientes de Desenvolvimento

Para o desenvolvimento do trabalho, os testes foram realizados com as seguintes máquinas, todas utilizando gcc com versão C++20 ou superior:

- **PC Bruno**
- Sistema operacional macOS Sonoma v14.6.1, Processador Apple M2 e 8GB de memória. Desenvolvimento e testes realizados utilizando Docker integrado ao Visual Studio Code, executando Ubuntu 22.04 de fundo.
- Sistema operacional Windows 11 Pro, Processador Intel Core i7-12700 e 16GB de memória. Desenvolvimento e testes realizados utilizando a extensão WSL do Visual Studio Code, que executa Ubuntu 22.04 de fundo.

Estrutura do Pacote

Os pacotes são parte fundamental do funcionamento correto do sistema. A estrutura criada foi muito baseada na recomendada nas especificações do trabalho, possuindo ID, total de pacotes sequenciais, tipo e status, além dos bytes sendo carregados pelo pacote e quantidade deles. Na transmissão de arquivos de um fim ao outro, a sequência de pacotes recebida é controlada pela variável que indica o total de pacotes em sequência, reconstruindo no local correto.

Os tipos de pacotes disponíveis são de: conexão, desconexão, dados, sincronização, delete, fetch, download e informação. Os primeiros realizam os procedimentos para adicionar e remover clientes, sendo informativos ao servidor. Os pacotes de dados carregam os bytes de um arquivo. Pacotes de delete são enviados para o servidor para fazer a remoção de um arquivo de *sync_dir*. Pacotes de fetch são usados para realizar o upload de um arquivo de diretório local do usuário. Pacotes de sincronização são utilizados quando é necessário equiparar os diretórios *sync_dir* do cliente e do servidor. Pacotes de download são usados para realizar o download de um arquivo de *sync_dir* para o diretório local do usuário. Pacotes de informação carregam as informações de um arquivo, como tamanho, data e hora de modificação e de acesso.

Servidor

Para o desenvolvimento do servidor e o correto tratamento de múltiplos clientes simultaneamente, o servidor inicializa criando o diretório *sync_dir* caso ainda não exista na máquina e criando um socket para ouvir requests de clientes. A cada cliente novo que tenta conexão com o servidor, é checado o número de conexões estabelecidas com o mesmo username a se conectar utilizando dicionários concorrentes. Caso a conexão seja aceita, é criada uma nova thread para lidar com o novo cliente, permitindo o tratamento de múltiplos usuários ao mesmo tempo.

Cliente

O cliente, ao iniciar uma conexão com o servidor, deve informar um username, importante para a correta sincronização de seus arquivos, endereço IP do servidor e porta (que ficou fixa como 5000). Após o comando, é criado um socket para o cliente e enviado um pacote de conexão e se cria o diretório *sync_dir* caso ainda não exista. O cliente executa três threads: uma que executa a sincronização dos arquivos dos diretórios *sync_dir* local e remoto, outra thread que inicializa a classe de notificação para sincronização e uma última que lida com os comandos inseridos pelo usuário.

Sincronização

A sincronização dos diretórios entre servidor e clientes é feita utilizando a API *inotify*, de Unix. Ela detecta mudanças em arquivos no diretório *sync_dir* e faz atualizações no outro fim. Para garantir que não haja alteração e atualização ao mesmo tempo, ou até mesmo duas alterações ao mesmo tempo, é usado mutex, com lock, para que não seja possível que um mesmo arquivo seja alterado ao mesmo tempo e haja problemas de sincronização e conflito.

Dificuldades e Problemas

A manipulação de arquivos foi uma parte difícil do trabalho, já que a necessidade de dividi-lo em pacotes e reconstrui-lo corretamente no outro lado da conexão criava um desafio a mais. Foram necessárias mudanças em diversas partes de código para que todos os tipos de arquivo fossem transmitidos corretamente, adequando tipos e métodos. A sincronização dos diretórios do servidor e do cliente também foi uma etapa complicada, já que exigiu tratamento especial para que tudo corresse de maneira correta, com tipos de pacotes expecíficos para a situação.

A adaptação à linguagem C++ e as APIs usadas também se mostrou um desafio, causando uma necessidade de se habituar às nuances da linguagem e causando um começo de desenvolvimento mais lento.