UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

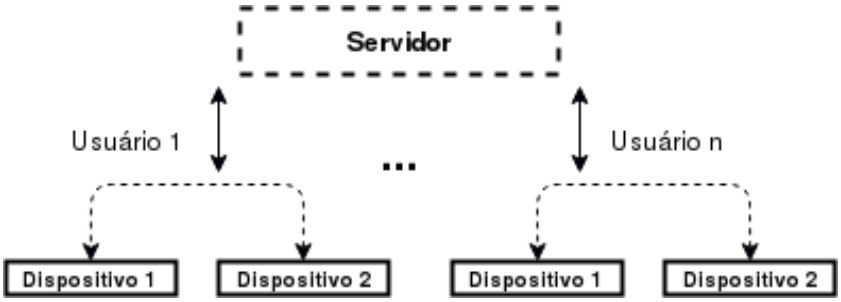
INF01151 – SISTEMAS OPERACIONAIS II N
SEMESTRE 2019/1
TRABALHO PRÁTICO PARTE 1: THREADS, SINCRONIZAÇÃO E COMUNICAÇÃO

ESPECIFICAÇÃO DO TRABALHO

Este projeto consiste na implementação de um serviço semelhante ao Dropbox, para permitir o compartilhamento e a sincronização automática de arquivos entre diferentes dispositivos de um mesmo usuário. O trabalho está dividido em duas partes. A primeira parte compreende tópicos como: threads, processos, comunicação e sincronização. Posteriormente, novas funcionalidades serão adicionadas ao projeto. A aplicação deverá executar em **ambientes Unix (Linux)**, mesmo que tenha sido desenvolvida em outras plataformas. O programa deverá ser implementado utilizando a API **Transmission Control Protocol (TCP) sockets** do Unix e utilizando **C/C++**.

FUNCIONALIDADES BÁSICAS

Sua aplicação deve possuir um servidor e um cliente. O servidor deve ser capaz de gerenciar arquivos de diversos usuários remotos. Já o cliente corresponde à parte da aplicação presente na máquina dos usuários, que permite ao usuário acessar remotamente seus arquivos mantidos pelo servidor.



Sua aplicação deve fornecer suporte às seguintes funcionalidades básicas:

- **Múltiplos usuários:** O servidor deve ser capaz de tratar requisições simultâneas de vários usuários.
- **Múltiplas sessões:** Um usuário deve poder utilizar o serviço através de até **dois** dispositivos distintos simultaneamente.[1]
- **Consistência nas estruturas de armazenamento:** As estruturas de armazenamento de dados no servidor devem ser mantidas em um estado consistente.

[1] *Para simplificar, assuma que mesmo que um usuário esteja com dois dispositivos/terminais abertos simultaneamente, ele **NÃO** irá editar o mesmo arquivo simultaneamente.*

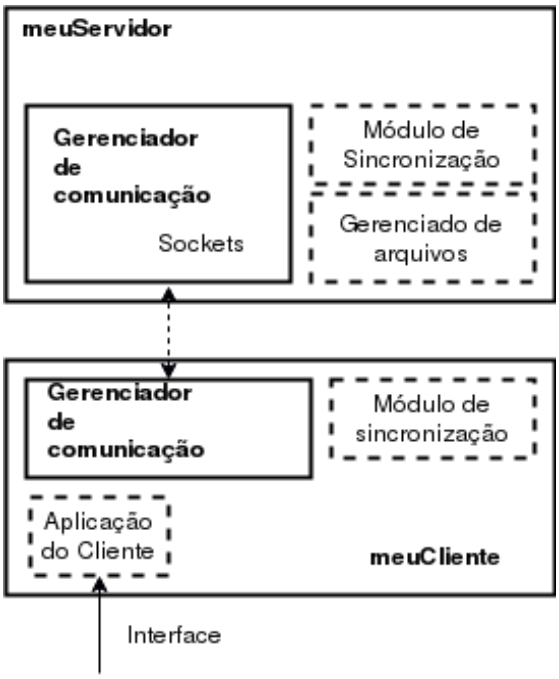
- **Sincronização**: Cada vez que um usuário modificar um arquivo contido no diretório *'sync_dir'* em seu dispositivo, o arquivo deverá ser atualizado no servidor e no diretório *'sync_dir'* dos demais dispositivos daquele usuário.

- **Persistência de dados no servidor**: Diretórios e arquivos de usuários devem ser restabelecidos quando o servidor for reiniciado.

O SISTEMA

Este trabalho está dividido em duas partes, sendo que a segunda parte irá adicionar funcionalidades extras ao resultado desta. Portanto, considere uma implementação modular e com possibilidade de extensão, e o encapsulamento das funções de comunicação do cliente e do servidor em módulos isolados.

A figura abaixo apresenta uma sugestão de como você pode implementar os módulos do sistema. Os módulos de comunicação são responsáveis por operações de envio e recebimento de arquivos[2]. O módulo de gerenciamento de arquivos é responsável por gerenciar os diretórios de cada usuário, os dados e metadados dos arquivos armazenados. Para isso, deve ser mantido um diretório para cada cliente, que pode ser identificado pelo próprio identificador do usuário.



A sincronização está vinculada ao diretório '*sync_dir'* no cliente (de forma similar ao que ocorre com o Dropbox, onde apenas arquivos dentro da pasta da sua aplicação são sincronizados com o servidor). O módulo de sincronização no cliente deve verificar o estado dos arquivos periodicamente, mantendo o diretório no servidor e nos outros dispositivos sempre atualizados[3], de acordo com a última modificação do usuário. Por exemplo, se um arquivo for removido do '*sync_dir'* em um dispositivo, essa mudança deve ser percebida pelo servidor e aplicada aos outros dispositivos ativos daquele mesmo usuário.

---

[2] *Utilize múltiplas threads/processos e sockets para que o módulo de comunicação do servidor possa suportar usuários simultâneos.*

[3] *É possível verificar se um arquivo foi modificado utilizando as seguintes APIs: inotify (Unix), stat (Unix) e dirent (Posix C, que pode ser utilizada no MacOS X). Por exemplo, no inotify é necessário verificar os eventos IN_NOTIFY e IN_CLOSE_WRITE.*

INTERFACE DO USUÁRIO

Um cliente deve poder estabelecer uma sessão com o servidor via linha de comando utilizando:

># ./myClient <username> <server_ip_address> <port>

onde <username> representa o identificador do usuário, e <server_ip_address> e <port> representam o endereço IP do servidor e a porta, respectivamente.

Após iniciar uma sessão, o usuário deve ser capaz de arrastar arquivos para o diretório *'sync_dir'* utilizando o gerenciador de arquivos do sistema operacional, e ter esses arquivos sincronizados automaticamente com o servidor e com os demais dispositivos daquele usuário. Da mesma forma, o usuário deve ser capaz de editar ou deletar os arquivos, e ter essas modificações refletidas automaticamente no servidor e nos demais dispositivos daquele usuário.

Além disso, uma interface deve ser acessível via linha de comando, permitindo que o usuário realize as operações básicas do sistema, detalhadas na tabela abaixo.

Comando	**Descrição**
# upload <path/filename.ext>	Envia o arquivo *filename.ext* para o servidor, colocando-o no "*sync_dir"* do servidor e propagando-o para todos os dispositivos daquele usuário. *e.g. upload /home/user/MyFolder/filename.ext*
# download <filename.ext>	Faz uma cópia não sincronizada do arquivo *filename.ext* do servidor para o diretório local (de onde o servidor foi chamado). *e.g. download mySpreadsheet.xlsx*
# delete <filename.ext>	Exclui o arquivo <*filename.ext*> de "*sync_dir*".
# list_server	Lista os arquivos salvos no servidor associados ao usuário.
# list_client	Lista os arquivos salvos no diretório "*sync_dir*"
# get_sync_dir	Cria o diretório "*sync_dir*" e inicia as atividades de sincronização
# exit	Fecha a sessão com o servidor.

- Em relação aos comandos *list_server* e *list_client*, é importante que esteja disponível a visualização de, pelo menos, os MAC times: *modification time* (mtime), *access time (atime)* e *change or creation time (ctime)* – plataformas Unix e Windows os interpretam diferentemente – dos arquivos exibidos no terminal.

- O comando *get_sync_dir* deve ser executado automaticamente logo após o estabelecimento de uma sessão entre cliente e servidor. Quando o comando *get_sync_dir* for executado, o servidor verificará se o diretório "sync_dir_<username>" existe no dispositivo do cliente, e criá-lo se necessário. Toda vez que alguma mudança ocorrer dentro desse diretório, por exemplo, um arquivo for alterado, renomeado ou deletado, essa mudança deverá ser espelhada no servidor e em todos os dispositivos daquele cliente.

- Ao utilizar o comando *upload*, o usuário deve ser capaz de carregar no servidor um arquivo não sincronizado que esteja em qualquer diretório no dispositivo local. O servidor, ao processar o comando de upload, deve então propagar o arquivo a todos os dispositivos do cliente (seria equivalente à utilizar a interface web do Dropbox para carregar um arquivo no servidor, que será propagado a todos os dispositivos daquele cliente).

- Ao utilizar o comando de *download*, uma cópia do arquivo existente no servidor deve ser baixada para um diretório local não sincronizado do dispositivo do cliente. Essa cópia local, fora do diretório *'sync_dir'*, não deverá sofrer sincronizações posteriores (seria equivalente à utilizar a interface web do Dropbox para baixar um arquivo do servidor para um diretório local na máquina do usuário).

FORMATO DE ESTRUTURAS

Você tem liberdade para definir o tamanho e formato das mensagens que serão usadas para transferir comandos e blocos de arquivos. Sugere-se a especificação de uma estrutura para definir as mensagens trocadas entre cliente/servidor. Abaixo é apresentada uma sugestão de como implementar a estrutura das mensagens.

```
typedef struct packet{
    uint16_t type;              //Tipo do pacote (p.ex. DATA | CMD)
    uint16_t seqn;              //Número de sequência
    uint32_t total_size;        //Número total de fragmentos
    uint16_t length;            //Comprimento do payload
    const char* _payload;       //Dados do pacote
} packet;
```

DESCRIÇÃO DO RELATÓRIO

Deverá ser produzido um relatório fornecendo os seguintes dados:

- Descrição do ambiente de testes: versão do sistema operacional e distribuição, configuração da máquina (processador(es) e memória) e compiladores utilizados (versões).

- Explique suas respectivas justificativas a respeito de:
    - (A) Como foi implementada a concorrência no servidor para atender múltiplos clientes;
    - (B) Em quais áreas do código foi necessário garantir sincronização no acesso a dados;
    - (C) Descrição das principais estruturas e funções que você implementou;
    - (D) Explicar o uso das diferentes primitivas de comunicação;

- Também inclua no relatório uma descrição dos problemas que você encontrou durante a implementação e como estes foram resolvidos (ou não).

A **nota será atribuída baseando-se nos seguintes critérios**: (1) qualidade do relatório produzido conforme os itens acima, (2) correta implementação das funcionalidades requisitadas e (3) qualidade do programa em si (incluindo uma interface limpa e amigável, documentação do código, funcionalidades adicionais implementadas, etc).

DATAS E MÉTODO DE AVALIAÇÃO

O trabalho deve ser feito em grupos de **3 OU 4 INTEGRANTES**. Não esquecer de identificar claramente os componentes do grupo no relatório.

Faz parte do pacote de entrega os arquivos fonte e o relatório em um arquivo ZIP. O trabalho deverá ser entregue até às **08:30 do dia 16 de maio (turma A)** ou até às **08:30 do dia 15 de maio (turma B)**. A entrega deverá ser via moodle (link para submissão na Aula 08). As demonstrações ocorrerão no mesmo dia, no horário da aula.

Após a data de entrega, o trabalho deverá ser entregue via e-mail para alberto@inf.ufrgs.br (subject do e-mail deve ser “INF01151: Trabalho Parte 1”). Neste caso, será descontado 02 (dois) pontos por semana de atraso. O atraso máximo permitido é de duas semanas após a data prevista para entrega. Isto é, nenhum trabalho será aceito após o dia 30 de maio (turma A) ou 29 de maio (turma B).