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

INF01151 – SISTEMAS OPERACIONAIS II N
SEMESTRE 2018/1
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 aprendidos durante a primeira metade da disciplina, tais como: threads, processos, comunicação e sincronização. Posteriormente, novas funcionalidades serão adicionadas ao projeto.

A implementação deverá ser executada, obrigatoriamente, em **ambientes Unix (Linux)**, mesmo que o trabalho tenha sido desenvolvido sobre outras plataformas. O programa deverá ser implementado utilizando a API **User Datagram Protocol (UDP) sockets** do Unix.

FUNCIONALIDADES BÁSICAS

Você deverá criar um servidor (Dropbox) responsável por gerenciar arquivos de diversos usuários (clientes). Deste modo, a estrutura mínima que você deverá elaborar corresponde aos seguintes arquivos fonte implementados, obrigatoriamente, em linguagem C/C++:

dropboxServer.h, dropboxServer.c: implementação das funções relacionados ao servidor
dropboxClient.h, dropboxClient.c: implementação das funções relacionadas ao cliente
dropboxUtil.h, dropboxUtil.c: constantes, métodos auxiliares etc.

Um exemplo de como pode ser estabelecida uma sessão com o servidor via linha de comando é mostrado abaixo, onde *user* representa o *userid [MAXNAME]* do cliente e *endereço* e *porta* representam o servidor:

./dropboxClient user endereço porta

Para cada usuário, o servidor deverá garantir:

- **Múltiplas sessões:** um mesmo usuário deve poder utilizar o servidor através de dispositivos distintos simultaneamente, limitado a 2 sessões simultâneas.

- **Consistência nas estruturas de armazenamento:** as estruturas de armazenamento de dados no servidor devem ser mantidas em estados consistentes ao realizarem alguma atividade.

- **Sincronização no acesso de arquivos**: cada vez que um usuário modificar um arquivo em seu dispositivo, o arquivo deverá ser atualizado no servidor e nos demais dispositivos daquele usuário.

Obs.: para simplificar, assumiremos na parte 1 do trabalho que mesmo que um usuário esteja com dois dispositivos/terminais abertos simultaneamente, ele NÃO irá editar o mesmo arquivo simultaneamente. O objetivo é apenas garantir que a alteração feita no arquivo em um dispositivo possa ser observada no outro dispositivo. Lidar com alterações simultâneas está além do escopo da parte 1 do trabalho.

PROJETO DO SISTEMA E ESTRUTURAS DE DADOS

Para cada cliente conectado ao servidor, uma interface com o usuário deverá ser acessível via linha de comando, de modo a permitir as seguintes requisições:

Comando	Descrição
upload <path/filename.ext>	Envia o arquivo *filename.ext* para o servidor. *e.g. upload /home/fulano/MyFolder/teste.txt*
download <filename.ext>	Faz download do arquivo *filename.ext* do servidor para o diretório local (de onde o servidor foi chamado). *e.g. download mySpreadsheet.xlsx*
list_server	Lista os arquivos salvos no servidor associados ao usuário.
list_client	Lista os arquivos salvos no diretório "*sync_dir_<nomeusuário>*"
get_sync_dir	Cria o diretório "*sync_dir_<nomeusuário>*" no /home do usuário.
exit	Fecha a sessão com o servidor.

Para implementar o cliente/servidor Dropbox, você deverá seguir as diretrizes abaixo:

- 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_<nomeusuário>" existe no dispositivo do cliente, e criá-lo se necessário. A sincronização dos arquivos deverá ser efetuada. Toda vez que alguma mudança ocorrer dentro desse diretório, por exemplo, um arquivo for renomeado ou deletado, essa mudança deverá ser espelhada no servidor e em todos os dispositivos daquele cliente.

- A sincronização está vinculada apenas ao diretório "sync_dir_<nomeusuário>" (de forma similar ao que ocorre com o Dropbox, onde apenas arquivos dentro da pasta Dropbox são sincronizados com o servidor). Caso o usuário possua múltiplos arquivos de mesmo nome e extensão em diferentes diretórios, apenas o arquivo contido no diretório "sync_dir_<nomeusuário>" deverá ser sincronizado. Caso o usuário realize o download de um arquivo para outro diretório que não seja "sync_dir_<nomeusuário>", este arquivo não sofrerá sincronizações posteriores.

- Deverá haver uma thread de sincronização/daemon no cliente, que, por exemplo, a cada 10 segundos, irá verificar todos os arquivos no diretório *"sync_dir_<nomeusuário>"*. Caso houver modificação em algum arquivo, este será enviado ao servidor. **Obs.:** 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). Todas as APIs citadas demandam leitura do manual das mesmas para melhor entendimento. Por exemplo, no `inotify` é necessário verificar os eventos IN_NOTIFY e IN_CLOSE_WRITE.

- Os arquivos de um usuário estarão na raiz que o servidor designar para aquele usuário. Logo, o servidor terá um diretório apenas para cada cliente. O servidor usará como nome do diretório do cliente o campo *userid [MAXNAME]* passado na linha de comando. Assuma que o *userid [MAXNAME]* será único.

- 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 o interpretam diferentemente – dos arquivos exibidos no terminal.

- O servidor deverá ser persistente, ou seja, deverá manter o último estado válido após o desligamento do servidor. Logo, se o servidor continha diretórios e arquivos de usuários, estes deverão ser restabelecidos quando o servidor ficar *online* novamente.

O programa deverá ser implementado utilizando a **API de sockets Unix**. O tipo de socket a ser utilizado deve ser **User Datagram Protocol (UDP)**, e o servidor deve ser capaz de tratar simultaneamente requisições de vários clientes (possivelmente criando descritores de sockets auxiliares para tratar diferentes sessões). Você terá liberdade para definir o tamanho e formato dos datagramas que serão

usados para transferir comandos e blocos de arquivos entre clientes e servidor. No entanto, como sockets UDP não garantem entrega de mensagens, você deverá **implementar o seu próprio mecanismo de confirmações (ack) e reenvio de mensagens** para garantir o correto funcionamento do programa.

Os protótipos das funções para a implementação do cliente deverão seguir o formato abaixo:

Interface	Descrição
`int login_server(char *host, int port);`	Estabelece uma sessão entre o cliente com o servidor. *host – endereço do servidor* *port – porta do servidor*
`void sync_client();`	Sincroniza o diretório "*sync_dir_<nomeusuário>*" com o servidor.
`void send_file(char *file);`	Envia um arquivo *file* para o servidor. Deverá ser executada quando for realizar upload de um arquivo. *file –filename.ext do arquivo a ser enviado*
`void get_file(char *file);`	Obtém um arquivo *file* do servidor. Deverá ser executada quando for realizar download de um arquivo. *file – filename.ext*
`void delete_file(char *file);`	Exclui um arquivo *file* de "*sync_dir_<nomeusuário>*". *file –filename.ext*
`void close_session ();`	Fecha a sessão com o servidor.

Os protótipos das funções para a implementação do servidor deverão seguir o formato abaixo:

Interface	Descrição
`void sync_server();`	Sincroniza o servidor com o diretório "*sync_dir_<nomeusuário>*" do cliente.
`void receive_file(char *file);`	Recebe um arquivo *file* do cliente. Deverá ser executada quando for realizar upload de um arquivo. *file – path/filename.ext do arquivo a ser recebido*
`void send_file(char *file);`	Envia o arquivo *file* para o usuário. Deverá ser executada quando for realizar download de um arquivo. *file – filename.ext*

As funções acima servem como exemplos para o comportamento mínimo do cliente e do servidor. Alterações de parâmetros podem ser feitas, assim como funções adicionais podem ser utilizadas, mas tais modificações deverão estar documentadas no relatório.

As estruturas de dados contendo os campos mínimos necessários para o projeto, representando o cliente e os arquivos salvos por este no servidor, estão representadas no seguinte formato:

Interface	Descrição
`struct client {` `  int devices[2];` `  char userid[MAXNAME];` `  struct file_info[MAXFILES];` `  int logged_in;` `}`	*int devices[2] – associado aos dispositivos do usuário* *userid[MAXNAME] – id do usuário no servidor, que deverá ser único. Informado pela linha de comando.* *file_info[MAXFILES] – metadados de cada arquivo que o cliente possui no servidor.* *logged_in – cliente está logado ou não.*
`struct file_info {` `  char name[MAXNAME];` `  char extension[MAXNAME];` `  char last_modified[MAXNAME];` `  int size;` `}`	*name[MAXNAME]* refere-se ao *nome do arquivo.* *extension[MAXNAME]* refere-se ao tipo de *extensão do arquivo*. *last_modified [MAXNAME]* refere-se a data da *última mofidicação* no arquivo. *size* indica o *tamanho do arquivo, em bytes.*

Tais estruturas estarão dispostas no servidor na forma de listas encadeadas. Justifique a inserção de quaisquer estruturas de dados adicionais para manter os dados dos clientes conectados ao servidor.

IMPORTANTE: 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.

DESCRIÇÃO DO RELATÓRIO A SER ENTREGUE

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) Estruturas e funções modificadas e/ou adicionais que você implementou;
  - (D) Explicar o uso das diferentes primitivas de comunicação;

- Também inclua no relatório 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 15 de maio (turma A)** ou às **08:30 do dia 14 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 29 de maio (turma A) ou dia 28 de maio (turma B).
