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

INF01151 – SISTEMAS OPERACIONAIS II N
SEMESTRE 2017/2
TRABALHO PRÁTICO PARTE 1: THREADS, SINCRONIZAÇÃO E COMUNICAÇÃO

ESPECIFICAÇÃO DO TRABALHO

Este projeto consiste na implementação de um serviço semelhante ao Dropbox e 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. O projeto deverá ser executado, **obrigatoriamente, em ambientes Unix (Linux)**, mesmo que o trabalho tenha sido desenvolvido sobre outras plataformas.

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 realizada a conexão ao servidor é mostrado abaixo, onde *user* representa o *userid[MAXNAME]* do cliente e *endereço* e *porta* representam o servidor:

./dropboxClient user endereço porta

Para cada cliente, o servidor deve garantir:

- **Múltiplas sessões:** um mesmo cliente deve poder utilizar o servidor através de dispositivos distintos simultaneamente. Limitado em 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 arquivo for modificado no cliente, ele deverá ser atualizado no servidor. A política de sincronização a ser utiilizada deverá ser “push”, ou seja, o cliente é responsável por sincronizar as mudanças nos arquivos locais.

Para simplificar, assumiremos na parte 1 do trabalho que mesmo que um cliente esteja com dois terminais abertos simultaneamente, ele NÃO irá editar o mesmo arquivo simultaneamente. Assim, 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 ESTRUTURA DE DADOS

Para cada cliente conectado ao servidor, uma interface com o usuário deverá ser acessível via linha de comando, de modo a suportar os 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 conexão com o servidor.

O comando get_sync_dir deve ser executado logo após o estabelecimento de uma conexão entre cliente e servidor. Toda vez que o comando get_sync_dir for executado, o servidor verificará se o diretório "sync_dir_<nomeusuário>" existe no dispositivo do cliente. Caso contrário, o diretório deverá ser criado e a sincronização 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 pelo cliente.

Note, também, que a sincronização está vinculada apenas ao diretório "sync_dir_<nomeusuário>". Caso o usuário possua múltiplos arquivos de mesmo nome e extensão em diferentes diretórios, incluindo "sync_dir_<nomeusuário>", 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. Para que a sincronização seja garantida, haverá uma *thread de sincronização/daemon* no cliente, que, por exemplo, a cada 10 segundos, irá verificar todos os arquivos no diretório *"sync_dir_<nomeusuário>"* e, caso houver modificação em algum arquivo, este será enviado ao servidor. É possível verificar se um arquivo foi modificado utilizando as seguintes APIs: `inotify` *(Unix),* `stat` *(Unix) e* `dirent` *(POSIX C, pode ser utilizada no MacOS)*. 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. Como o envio de arquivo se dá na forma de um fluxo de bytes, poderá ser necessário, junto ao envio dos bytes que compõem o arquivo, enviar um token para delimitar o fim do arquivo, que permita ao servidor reconstruí-lo.

É importante ressaltar, também, que o usuário não poderá criar diretórios no servidor. Todos os seus arquivos estarão na raiz que o servidor designar para determinado cliente. Isto implica em dizer que o servidor terá um diretório para cada cliente. O servidor usará como nome do diretório do cliente o campo *userid[MAXNAME]* passado na linha de comando, armazenado na estrutura do mesmo, descrita abaixo. Assuma que o *userid[MAXNAME]* será único.

Com 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 [plataformas Unix e Windows o interpretam diferentemente] – ctime) dos arquivos exibidos no terminal.

Uma funcionalidade desejável é que o servidor seja persistente, ou seja, que ele mantenha 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.

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

Resumo da interface de implementação mínima para o cliente:

Interface	Descrição
`int connect_server(char *host, int port);`	Conecta o cliente com o servidor. *host – endereço do servidor* *port – porta aguardando conexão*
`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_connection();`	Fecha a conexão com o servidor.

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

Resumo da interface de implementação mínima para o servidor:

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*

Quaisquer funções adicionais utilizadas deverão estar documentadas no relatório.

A estrutura de dados contendo os campos mínimos necessários para o projeto, representando o cliente e os arquivos salvos por este no servidor, está representada no seguinte formato:

Resumo da estrutura de dados mantidas no servidor:

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 uma lista encadeada.

O programa deve, **obrigatoriamente,** ser implementado utilizando a **API de sockets Unix**. O tipo de socket a ser utilizado deve ser **orientado a conexão (TCP)**, e o servidor deve ser um **servidor concorrente** (capaz de tratar simultaneamente requisições de vários clientes). Justifique a **inserção de quaisquer estruturas de dados adicionais** para manter os dados dos clientes conectados ao servidor.

Você deve seguir o princípio do "two-way sincronization" no diretório *"sync_dir_<nomeusuário>"*, ou seja:

- Arquivos criados/removidos no diretório de sincronização do cliente devem ser criados/removidos no servidor, e nos outros dispositivos daquele cliente.
- Se o arquivo não foi modificado no servidor ou cliente, então ele deve manter-se inalterado.
- Se houver algum conflito entre a cópia existente no servidor e a cópia do cliente, então o cliente deve ser atualizado com a versão do servidor.

IMPORTANTE: Não esqueça que este trabalho está dividido em duas partes e que a segunda parte irá adicionar funcionalidades extras, tais como replicação e autenticação, ao resultado deste. 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 teste: 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;
  - (B) Em quais áreas do código foi necessário garantir sincronização no acesso a dados;
  - (C) Estruturas e funções 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 13:30 horas do dia 08/novembro. A entrega deverá ser via moodle. 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 22/novembro.