Dropbox - Parte I

Augusto Bennemann, Fabrício Martins Mazzola e Txai Wieser

1 Ambiente de desenvolvimento e teste

O projeto foi desenvolvido utilizando um ambiente com as seguintes caracteristicas:

- Configuração da máquina:
  - Processador(es): Intel Core i5-2520M
  - Memória: 2x Kingston 4GB 1333Mhz DDR3
- Sistema operacional:
  - Distribuição: Debian GNU/Linux
  - Versão: Stretch (testing) Kernel 4.9.0-3-amd64
- Software suporte:
  - Compiladores (versões): gcc 6.3.0 20170516

2 Descrições e justificativas

2.1 Funcionamento

A aplicação do cliente é composta por três threads. A primeira thread representa a interface entre os comandos disponíveis para o usuário e a comunicação entre cliente e servidor (shell). Ela é responsável por esperar comandos inseridos pelo usuário através da CLI e invocar as primitivas correspondentes na aplicação cliente.

A segunda thread é um *daemon* responsável por aguardar notificações do sistema de arquivos, através da biblioteca *inotify*. O objetivo da thread é de monitorar as modificações realizadas nos arquivos presentes no diretório de sincronização do usuário (*sync_dir*). Quando um evento de criação ou renomeação de arquivo é constatado, essa thread automaticamente faz o upload desse arquivo para o servidor.

A terceira thread representa um servidor local (*local_server*), encarregado de ouvir chamadas de *PUSH* e *DELETE* vindas do servidor. Essas chamadas são utilizadas com o objetivo de manter a consistência e a sincronização entre os dispositivos dos clientes. Ao receber uma chamada *PUSH filename*, a aplicação cliente realiza, de forma implícita, o download do arquivo *filename*. De modo similar, o recebimento de uma chamada *DELETE filename* fará a aplicação cliente remover o arquivo *filename* da pasta de sincronização do usuário.

Ademais, foi criada uma lista de arquivos ignorados, utilizada para evitar que arquivos recém baixados ou removidos por ordem do servidor sejam capturados pelo *daemon*. Quando a aplicação cliente recebe um *PUSH* ou um *DELETE* do servidor e, respectivamente, baixa ou remove o arquivo solicitado, o arquivo é inserido nessa lista. Assim, quando o *inofity* percebe que algum desses arquivos é alterado, o encontra na lista de arquivos ignorados e então não propaga essas alterações ao servidor. Desse modo, evita-se situações em que diferentes dispositivos do mesmo usuário ficam enviando o mesmo arquivo um para o outro infinitamente. Por fim, o arquivo é excluído dessa lista, para que novas modificações nele possam ser capturadas pelo *inotify*.

A aplicação servidor é composta por uma thread principal responsável por lidar com as requisições de conexões dos clientes. A cada tentativa de conexão que é aceita pelo servidor, uma nova thread é criada para o usuário solicitante. Essa thread aguarda requisições do cliente e gerencia o envio e remoção de arquivos na pasta *server_sync_dir* do cliente. Além disso, quando um arquivo é recebido ou removido pelo servidor, a thread é encarregada de propagar essas alterações, através do envio de mensagens *PUSH* e *DELETE*, aos outros dispositivos conectados do usuário.

2.2 Sincronização

A sincronização da aplicação cliente foi feita utilizando-se duas variáveis de mutex: *fileOperationMutex* e *inotifyMutex*. A primeira considera uma operação sobre um arquivo como seção crítica e é usada para garantir que somente uma operação (p. ex. download, upload) seja realizada por vez. Já a segunda considera manipulações na lista de arquivos ignorados e nos *timestamps* dos arquivos como seção crítica, de modo a garantir que o *inotify* não seja ativado antes que um arquivo seja incluído na lista de arquivos ignorados e/ou tenha sua data de modificação original salva.

Na aplicação servidor, utilizou-se somente uma variável de mutex: *clientCreationLock*. Essa variável é armazenada na struct de cada usuário e serve para garantir que a conexão de clientes ao servidor sejam executada em somente um dispositivo por vez.

Além de proteção de seções críticas, duas barreiras - *syncbarrier* e *localserverbarrier* - foram utilizadas para realizar a sincronização entre diferentes tarefas do cliente. A primeira barreira garante que o *shell* somente seja exibido para o usuário depois que a conexão com o servidor for realizada com sucesso. Enquanto a thread principal permanece bloqueada esperando a sincronização ser concluída, uma mensagem de "Syncing..."é exibida ao usuário. Uma vez terminada a sincronização, uma mensagem de "Done"é exibida e então o programa passa a esperar pelos comandos do usuário. A segunda barreira garante que a sincronização inicial de arquivos só será feita após o servidor local (*local_server*) estar disponível, ouvindo requisições do servidor do Dropbox.

2.3 Comunicação

A comunicação entre os clientes e o servidor é realizada por meio de sockets TCP. No momento em que a conexão TCP é estabelecida, um socket único é criado para cada dispositivo de cliente. Os comandos básicos da comunicação são descritos abaixo:

Cliente -> Servidor:

- LIST:
  Cliente pede ao servidor uma lista com os arquivos salvos. O servidor responde confirmando a requisição, seguido pela quantidade de arquivos e pelos nomes dos mesmos.
- DOWNLOAD 'file':
  Cliente pede ao servidor o arquivo definido em file. O servidor responde confirmando a requisição, seguido pelo tamanho do arquivo e os dados do arquivo em si.
- UPLOAD 'file':
  Cliente comunica ao servidor que um arquivo deseja ser enviado seguido pela sua data de modificação. O servidor compara se o arquivo do cliente é mais recente do que o do servidor. Nesse caso, o servidor responde aceitando a transação, e assim que o cliente recebe a confirmação do request, envia o tamanho do arquivo seguido pelos dados do arquivo em si.
- DELETE 'file'
  Cliente avisa ao servidor que um arquivo deve ser deletado.
- SYNC:
  Cliente pede para ao servidor para realizar a sincronia do diretório. Nesse caso o servidor responde o numero de arquivos a serem sincronizados. E para cada arquivo a ser sincronizado, o servidor começa o envio através da comunicação *PUSH 'file'*.

Servidor -> Cliente:

- PUSH 'file':
  Servidor avisa ao cliente que um arquivo deve ser baixado, a partir dessa mensagem o cliente envia um request *DOWNLOAD 'file'*.
- DELETE 'file' [server -> client]:
  Servidor avisa ao cliente que um arquivo deve ser deletado.

Ademais, o mecanismo de sinais também foi utilizado. Quando o processo servidor é encerrado por meio da combinação de teclas *CTRL + c*, o sinal *SIGINT* é captado. O servidor, então, encerra sua execução de modo gracioso, fechando as conexões abertas. Isso permite que os clientes conectados instantaneamente percebam que a conexão foi fechada e, então, também encerrem sua execução com uma mensagem alertando que o servidor desconectou.

2.4 Estruturas e funções adicionais

2.4.1 dropboxUtil

- void makedir_if_not_exists(const char* path)
  Verifica se o diretório especificado no parâmetro *path* existe e caso contrario o cria.

- int file_exists(const char* path)
  Verifica se o arquivo especificado no parâmetro *path* existe.

- int connect_server(char* host, int port)
  Abre uma conexão via socket com o *host*.

2.4.2 dropboxServer

- struct file_info
  Foram feitas duas mudanças na estrutura *client*. A primeira foi a remoção da propriedade *char extension[MAXNAME]*, pois escolhemos armazenar o nome completo do arquivo em *char name[MAXNAME]* e, então, não havia motivos para manter essa informação duplicada. A outra modificação foi o uso de uma estrutura 'time_t' ao invés de um 'char *' para a propriedade 'last_modified'.

- struct client
  Na estrutura *client* foram adicionadas duas propriedades. A primeira é a *int devices_server[MAXDEVICES]* que armazena o socket *servidor* do cliente (que recebe requisições de PUSH e DELETE). Além disso, foi adicionado o *pthread_mutex_t mutex*, que é o mutex utilizado para proteger seções críticas de cada cliente, para evitar que sejam executadas simultaneamente em diferentes dispositivos.

- struct tailq_entry
  Estrutura usada na lista de clientes.

- void delete_file(char *file)
  Remove o arquivo *file* do servidor. 'file' – filename.ext

- void *connection_handler(void *socket_desc)
  Função executada pela thread que trata as conexões de cada cliente.

2.4.3 dropboxClient

- struct tailq_entry
  Estrutura usada na lista de clientes.

- void delete_server_file(char *file)
  Remove o arquivo file para do servidor. 'file' – filename.ext

2.5 Testes

Para testar a aplicação cliente e servidor, utilizamos métodos não automatizados (manuais). Foram realizados testes das funções da especificação através da interface de linha de comando (CLI) da aplicação cliente e os resultados foram sendo verificados nos diretórios do servidor e de clientes.

Um ponto importante e difícil de testar na prática foram as proteções de seções críticas. Para isso, tentou-se simular as condições de corrida de forma a realizar testes antes e depois da adição das variáreis de *mutex* com o objetivo de comprovar a eficácia das mesmas.

3 Dificuldades encontradas

3.1 Data de modificação incorreta

Ao receber arquivos, o servidor às vezes os gravava com a data de modificação incorreta. A origem do problema estava no cliente, que transmitia a data errada. Isso estava ocorrendo pois quando um arquivo no diretório de sincronização do usuário era alterado, o daemon prontamente capturava

a alteração, muitas vezes antes que a data de modificação fosse definida para a data do arquivo original. Assim, a solução encontrada foi utilizar um *sleep* para garantir que o daemon espere alguns milissegundos antes de tratar o evento.

3.2 Leitura de inteiros errados pelo socket

Ao transmitir dados numéricos pelo socket, algumas vezes o valor chegava aparentemente corrompido. A saber, diversas vezes esse número era o tamanho do arquivo que seria lido em seguida, o que levava o receptor a ler menos caracteres que o esperado ou a ficar trancado esperando por caracteres que nunca chegariam. O problema estava na leitura feita anteriormente, que lia mais caracteres do que de fato a string enviada possuía. A solução encontrada, portanto, foi definir um tamanho fixo para essas mensagens, de modo que a leitura não recebesse bytes do dado numérico a ser lido em seguida.

3.3 Utilização de exclusões mútuas

A princípio, definir exclusões mútuas parecia simples, mas logo descobrimos que essa tarefa pode ser bastante complexa, pois exige pensar a fundo o comportamento de todas as threads simultaneamente, em suas diferentes possibilidades de intercalação. Foram várias as situações em que o programa chegava a uma situação de *deadlock*.

3.4 Sincronização de alterações *offline*

Seria ótimo que as alterações feitas durante o período em que o usuário esteve *offline* pudessem ser refletidas no servidor assim que ele se conectasse. Na tentativa de implementar essa funcionalidade, fizemos com que o cliente automaticamente salve no seu diretório um arquivo *.dropboxfiles* contendo nome e data de modificação de cada um de seus arquivos.

Assim, na próxima vez que ele se conectasse seria possível descobrir, comparando esse arquivo e os arquivos no diretório, quais alterações foram feitas no período de tempo que o usuário esteve desconectado. No entanto, apesar de termos começado a implementação, não a concluímos por falta de tempo.

4 Conclusões

Ao terminar a implementação do trabalho, percebemos o quanto aprendemos com a construção do mesmo. Ao analisar a especificação vimos que teríamos que aprender diversas técnicas para construir a aplicação, como notificações do sistema de arquivos, comunicação utilizando sockets, proteções de seções críticas, barreiras, threads, e tudo isso utilizando uma linguagem de mais baixo nível. Mas construindo aos poucos, fomos estruturando nosso conhecimento e cada vez entendendo mais sobre o escopo envolvido.

A oportunidade de poder colocar em prática os conhecimentos adquiridos na disciplina (e no restante do curso) em um trabalho com conceitos e funções bem atuais foi fantástica. Implementar em um trabalho funções similares a serviços/produtos populares (como o Dropbox) nos faz relacionar e visualizar os conceitos aprendidos com o que realmente utilizamos no nosso dia-a-dia.