Segundo Trabalho de INF1019 – 2016.2

Um Simple File Server (SFS)

Entrega: 14 de dezembro

Introdução

Como voce viu – ou verá - na disciplina, muitos sistemas de arquivos modernos são distribuidos, tais como o NFS ou o Samba, e implementam um protocolo entre um cliente em uma máquina e um Servidor de Arquivos, executando em outra máquina. Na realidade, esses protocolos são bastante complexos e os Servidores de arquivos atuais são bastante sofisticados. Mas para voce obter um "gostinho" dessa complexidade, nesse trabalho voce deverá implementar um protocolo de comunicação (Simple File Management Protocol – SFMP) e um Simple File Server (SFS) correspondente com as seguintes funcionalidades e catacterísticas. Para tal, voce deverá usar como base o udpClient.c e udpServer.c disponibilizados na página da disciplina. Para simplificar, considere que todos os arquivos só contém caracteres ASCII.

Serviço Stateless

O seu SFS deve ser *stateless*, o que significa que o servidor não deve guardar quaqluer informação sobre o estado de acesso dos arquivos por um cliente. Por exemplo, o SFS não deve manter a informação sobre os arquivos abertos por um cliente, ou qual é a posição corrente em um arquivo de um cliente realizando uma leitura ou escrita. Essa característica de ser stateless é importante porque o servidor não tem como saber se um cliente simplesmente está demorando para fazer novas operações. ou se houve uma partição da rede ou se o processo cliente falhou, etc. Consequentemente, todas as operações de leitura e escrita devem conter um *offset*, ou seja, uma distância para o começo do arquivo onde a operação de leitura/escrita deverá ser realizada. Por exemplo: **`read(path, &buffer,100,10)`** indicaria que o processo quer ler 100 bytes a partir da posição 10 do arquivo especificado por path. O mesmo se aplica para a localização do arquivo alvo no sistema diretórios (hierarquico) no servidor: o cliente precisará sempre informar o caminho completo do arquivo (*path*), já que o servidor não armazenará o *diretório corrente* de um cliente.

As funcionalidades (e mensagens do SFMP)

O seu prcesso servidor deve rodar em background e deve ser iniciado de um diretório, **SFS-root-di,** especialmente criado para abrigar os diretórios e arquivos de clientes remotos, Esse diretório será o root de todos os arquivos dos clientes do SFS. Então, o path (de um arquivo) deverá sempre começar com '/'. Exemplo: read (/teste/markus.txt,…) causará a leitura do arquivo **SFS-root-di/teste/markus.txt.**

O seu cliente e servidor se comunicarão através do seguinte conjunto de mensagens e parâmetros.

Mensagens para manipulação de arquivos

RD-REQ, *path*(string), *strlen*(int), *payload* (string vazio), *nrbytes*(int), *offset*(int) // lê nrbytes do arquivo path a partir da posição offset

RD-REP, *path*(string), *strlen*(int), *payload* (string), *nrbytes* lidos(int), *offset* igual ao do R-REQ(int) // retorna em payload os nrbytes lidos

WR-REQ, *path*(string), *strlen*(int), *payload*(string), *nrbytes*(int), *offset*(int) // escreve nrbytes do conteúdo de payload no arquivo path a partir da posição offset

WR-REP, *path* (string), *strlen*(int), *payload*(string vazio), *nrbytes* escritos (int), *offset* igual ao do W-REQ(int) // retorna a confirmação da operação de escrita, se nrbytes for diferente de zero; serve também para criar novo arquivo ou remover o arquivo *path* (vide abaixo)

Obs: note que o W-REQ sobre-escreve o arquivo *path* a partir da posição *offset*, independente do seu conteúdo anterior. E se offset for maior do que o tamanho do arquivo, então offset é considerado como o ultimo endereço do arquivo. Note que se o arquivo não existir, mas o subdiretório indicado em *path* estiver correto, então o W-REQ pode ser usado para criar (e já escrever os primeiros bytes no) arquivo recém criado. Analogamente, pode-se remover um arquivo usando-se W-REQ com nrbytes=0.

Além das leituras e escritas, também haverá uma consulta aos atributos de arquvos (FI de File Information):

FI-REQ, *path* (string), *strlen*(int) // requisita informações sobre o arquivo *path*

FI-REP, *path* (string), *strlen*(int), *owner*(int), *permissions* (2char), *filelength* (int) // retorna o cliente dono do arquivo e o seu temanho, em bytes

Mensagens para manipulação de diretórios

Além disso, o SFS deverá implementar funções para criar, remover e listar o conteúdo de um diretório:

DC-REQ, *path*(string), strlen(int), *dirname*(string), *strlen*(int) // cria um novo subdiretório *dirname* em *path*

DC-REP, *path*(string), *strlen*(int) // retorna o novo path já contend dirname, ou strlen=0 se operação não pôde ser realizada

DR-REQ, *path* (string), *strlen*(int), *dirname*(string), *strlen*(int) // remove o subdiretório *dirname* de *path*

DR-REP, *path*(string), *strlen*(int) // retorna o novo path já sem dirname, ou strlen=0 se operação não pôde ser realizada

DL-REQ, *path*(string), *strlen*(int) // mostra o nome dos arquivos no diretório *path*

DL-REP, *allfilenames* (string), *fstlstpositions* (array[40] of struct {int, int}), *nrnames* (int) // retorna os nomes de todos os arquivos/subdiretórios em path

em um unico char array, sendo que fstlstpositions[i] indica a posição inicial e final do i-ésimo nome em allfilenames

Incluindo permissões

Como voce sabe, um elemento importante de um sistema de arquivos é o controle de acesso, em UNIX implementado através das permissões **R**ead, **W**rite e e**X**ecute para o dono, o grupo e outros. Um SFS implementando apenas as funcionalidades acima permitirá que qualquer cliente manipule os arquivos e diretórios sem qualquer restrição. Para simplificar, vamos dizer que para cada objeto só existam o "dono" e os "outros", e que as permissões sejam somente R e W.

Para implementar um controle de acesso o seu SFS terá que armazenar internamente, qual é o dono (i.e. o cliente) de cada arquivo e/ou subdiretório. Para fazer isso, voce precisará incluir em todas as mensagens uma identificação do Cliente ,CIientI, (inteiro), e em cada uma das operações acima seu SFS deve verificar se a operação pode ou não ser realizada. Para isso, nas mensagens DC-REQ e WR-REQ voce deverá incluir mais dois parâmetros, ownerPerm, e otherPerm (byte) que indicarão quais permissões de acesso o dono terá sobre o arquivo/diretório e quais ele está concedendo aos demais clientes, pondendo ser R ou W. Para simplificar (e economizar alguns bits nas mensagens ☺) vamos assumir que a permissão de escrita W, subsume a de leitura, R.

Observações finais

Como o *payload* de um datagrama UDP tem um tamanho máximo de 65507 bytes, voce pode assumir que as leituras e escritas nos arquivos sempre serão tais que todos os parâmetros (e os dados lidos/escritos) contidos nas mensagens REQ e REP sempre caibam em um datagrama UDP.

Atenção: Esse trabalho irá demander um pouco de trabalho. Por isso recomendo que voce começe assim que possível. Além disso, começe implementando as operações de manipulação de arquivos e diretórios do processo servidor, depois inclua o registro do dono (owner) e das permissões em cada arquivo/diretório, para só depois implementar o protocol de comunicação SFMP. Finalmente, implemente um cliente bacaninho que permita mostrar que as consultas e modificacões no sistema de arquivos remote estão ocorrendo corretamente.

Boa sorte e happy coding!