

INF01151 – SISTEMAS OPERACIONAIS II N

SEMESTRE 2017/2

TRABALHO PRÁTICO PARTE 2: EXCLUSÃO MÚTUA, REPLICAÇÃO E SEGURANÇA

ESPECIFICAÇÃO DO TRABALHO

Este projeto consiste na implementação de um serviço semelhante ao Dropbox e está dividido em duas partes. Na primeira etapa, foi necessário implementar o funcionamento básico do serviço, com enfoque nos aspectos de programação com múltiplos processos/threads, comunicação e controle de concorrência. Nessa etapa, você deverá estender o serviço com algumas funcionalidades avançadas, onde destacam-se: exclusão mútua distribuída, replicação e segurança.

O programa deverá executar obrigatoriamente em ambientes Unix (Linux) mesmo que o trabalho tenha sido desenvolvido em outra plataforma. Quando apropriado, a especificação abaixo define funções e trechos de código que você deverá implementar. No entanto, é possível estender estas funções caso for julgado necessário, destacando as modificações no relatório final a ser entregue.

FUNCIONALIDADES AVANÇADAS

Na segunda parte deste projeto, a aplicação deverá ser estendida para atender os conceitos estudados na segunda metade da disciplina. Isto inclui: exclusão mútua distribuída, replicação e segurança.

1. Exclusão Mútua Distribuída

Na primeira parte do trabalho, assumiu-se que dois dispositivos do mesmo usuário não iriam realizar atualizações em um mesmo arquivo ao mesmo tempo. Nessa etapa do trabalho, no entanto, o sistema deverá permitir tal condição. Como em um sistema distribuído não existe a noção de um semáforo/mutex global, você deverá implementar um dos algoritmos baseados em troca de mensagens para implementar um mecanismo de exclusão mútua distribuída (aula 23, capítulo 15 do livro do Coulouris).

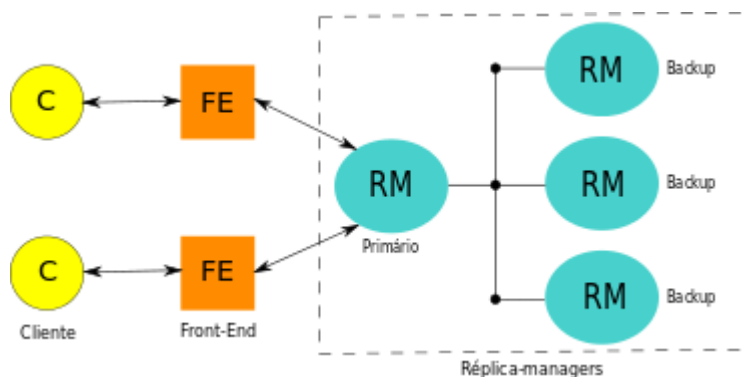
Essencialmente, o algoritmo deverá impedir que um processo realize edições a um arquivo até que ele obtenha o “token” de permissão através do mecanismo de controle de exclusão mútua. Uma das formas de evitar que um processo realize edições no arquivo é através da alteração de atributos do respectivo arquivo no sistema operacional local. Ao receber o “token” do algoritmo, o acesso ao arquivo pode então ser liberado. O algoritmo a ser empregado pode ser qualquer um dos vistos em aula e descritos no capítulo 15 do livro do Coulouris (Algoritmo Centralizado, Algoritmo do Anel, ou Algoritmo de Ricart-Agrawala).

2. Replicação Passiva

O servidor implementado na Parte I, caso por ventura falhar, acarretará em indisponibilidade de serviço para seus clientes. Este é um estado que não é desejável no sistema. Para aumentar a disponibilidade do sistema implementado, você deverá aplicar o conceito de *replicação passiva*, onde o servidor Dropbox será representado por uma instância de replica manager (RM) primária, e uma ou mais instâncias de replica managers secundárias (ou backup).

Podemos entender este modelo adicionando um *front-end* (FE) entre a comunicação do cliente (C) e servidor, agora representado por um conjunto de *replica managers* (RMs). Este *front-end* será responsável por realizar a comunicação entre estas entidades, tornando transparente para o cliente qual é a cópia primária do servidor. Você precisará garantir que:

- (1) todos os clientes sempre utilizarão a mesma cópia primária;
- (2) após cada operação, o RM primário irá propagar o estado dos arquivos aos RMs de backup;
- (3) somente após os backups serem atualizados o primário confirmará a operação ao cliente.



Como referência, o RM primário deverá solicitar a execução da função abaixo para propagar as alterações aos backups:

Interface	Descrição
<code>int updateReplicas(void);</code>	Executado pela réplica primária para realizar o espelhamento da operação executada em todas as réplicas de backup. A operação só é confirmada caso todos os backups recebam a atualização.

3. Autenticação do Cliente/Servidor e Comunicação Segura

Autenticação permite que um participante (cliente) possa verificar e validar a identidade de outro participante (servidor) em um canal de comunicação. Nessa etapa do trabalho, será necessário estender as aplicações cliente e servidor para que suportem a funcionalidade de autenticação. Você deverá utilizar a biblioteca OpenSSL. Por padrão, um servidor SSL sempre envia seu certificado, se disponível, permitindo verificar a autenticidade apenas do servidor. É necessário ter o pacote *libssl-dev* instalado. Para tal, execute:

```
%> sudo apt-get install libssl-dev
```

A solução desenvolvida deverá utilizar criptografia assimétrica (criptografia de chaves pública/privada). Para isso, você deverá gerar a chave privada e o certificado digital utilizando OpenSSL. Utilize o seguinte comando:

```
%> openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout KeyFile.pem -out CertFile.pem
```

Nesta demonstração, *KeyFile* se refere à chave privada e *CertFile* corresponde ao certificado digital gerado. Este certificado será auto-assinado (obs.: a geração de certificados auto-assinados não é uma boa prática fora do contexto deste trabalho).

- **Adicionando conexão segura ao Cliente**

Estenda o cliente de forma a comportar SSL. Para tal, será necessário a adição de, pelo menos, as seguintes funções abaixo. Você também precisará dos includes *openssl/ssl.h* e *openssl/err.h*.

Passos:

1. **Inicialize a engine SSL:** No código do cliente, crie o contexto, a estrutura principal SSL (a qual mantém todas as outras estruturas necessárias), inicialize o contexto fazendo uso da função definida no resumo das funções e crie a conexão.

<pre>SSL_METHOD *method; SSL_CTX *ctx; OpenSSL_add_all_algorithms(); SSL_load_error_strings(); method = SSLv23_client_method(); ctx = SSL_CTX_new(method); if (ctx == NULL){ ERR_print_errors_fp(stderr); abort(); }</pre>	<p>SSL_METHOD *method - inicializa um ponteiro para armazenar a estrutura que descreve as funções internas da biblioteca SSL que implementa o protocolo SSLv23. É necessário para criar o SSL_CTX.</p> <p>SSL_CTX *ctx - ponteiro que armazenará o contexto.</p> <p>OpenSSL_add_all_algorithms(); - Adiciona todos os algoritmos (digests e ciphers).</p>
---	---

2. **Anexe SSL ao socket:** Crie um novo estado de conexão SSL, anexe este ao descritor de socket e realize a conexão.

<pre>ssl = SSL_new(ctx); SSL_set_fd(ssl, sockfd); if (SSL_connect(ssl) == -1) ERR_print_errors_fp(stderr); else{ /* conexão aceita */ }</pre>	Anexando SSL ao descritor de socket já criado na Parte I.
--	---

3. **Leia e escreva de forma segura:** Ao ter a conexão aceita, é possível fazer uso das funções para envio e recebimento, SSL_write e SSL_read, respectivamente.

<pre>int SSL_write(SSL *ssl, const void *buf, int num)</pre>	Escreve <i>num</i> bytes do buffer <i>buf</i> na conexão <i>ssl</i> .
<pre>int SSL_read(SSL *ssl, void *buf, int num)</pre>	Lê <i>num</i> bytes do buffer <i>buf</i> na conexão <i>ssl</i> .

4. **Mostre o certificado:** você poderá mostrar no terminal os dados do certificado digital.

<pre>X509 *cert; char *line; cert = SSL_get_peer_certificate(ssl); if (cert != NULL){ line = X509_NAME_oneline(X509_get_subject_name(cert),0,0); printf("Subject: %s\n", line); free(line); line = X509_NAME_oneline(X509_get_issuer_name(cert),0,0); printf("Issuer: %s\n", line); }</pre>	<p>Mostra o certificado do peer com o seu nome e quem assinou seu certificado.</p> <p><i>X509 *cert;</i> - cria um ponteiro para receber um certificado do padrão X.509;</p> <p><i>cert = SSL_get_peer_certificate(ssl);</i> - Obtém o certificado do servidor.</p>
--	---

- **Adicionado conexão segura ao Servidor**

Estenda o servidor de modo a comportar SSL. Para tal, será necessário a adição de, ao menos, as seguintes funções abaixo. Você precisará dos includes *openssl/ssl.h* e *openssl/err.h*.

Passos:

1. **Inicialize a engine SSL:** No código do servidor, crie o contexto, a estrutura principal SSL (a qual mantém todas as outras estruturas necessárias), inicialize o contexto fazendo uso da função definida no resumo das funções e crie a conexão.

<pre>SSL_METHOD *method; SSL_CTX *ctx; OpenSSL_add_all_algoritms(); SSL_load_error_strings(); method = SSLv23_server_method(); ctx = SSL_CTX_new(method); if (ctx == NULL){ ERR_print_errors_fp(stderr); abort(); }</pre>	<p>SSL_METHOD *method - inicializa um ponteiro para armazenar a estrutura que descreve as funções internas da biblioteca ssl que implementa o protocolo SSLv23. É necessário para criar o SSL_CTX.</p> <p>SSL_CTX *ctx - ponteiro que armazenará o contexto.</p> <p>OpenSSL_add_all_algoritms(); - Adiciona todos os algoritmos (digests e ciphers).</p>
--	--

2. **Carregue o certificado:** Crie o contexto, inicialize o contexto fazendo uso da função definida no resumo das funções, carregue o certificado e abra o listener.

<pre>int SSL_CTX_use_certificate_file(SSL_CTX *ctx, const char *file, int type);</pre>	Carrega o certificado armazenado em <i>file</i> para <i>ctx</i> .
<pre>int SSL_CTX_use_PrivateKey_file(SSL_CTX *ctx, const char *file, int type);</pre>	Adiciona a chave privada armazenada em <i>file</i> para <i>ctx</i> .

3. **Anexe SSL ao socket:** No laço, aceite a conexão do cliente, crie um novo contexto para a conexão SSL e anexe-a ao descritor de socket.

<pre>int client = accept (server, &addr, &len); ssl = SSL_new(ctx); SSL_set_fd(ssl, client);</pre>	Anexando SSL ao descritor de socket já criado na Parte I.
--	---

4. **Aceite a conexão, leia e escreva de forma segura:** Ao ter a conexão aceita, é possível fazer uso das funções para envio e recebimento, SSL_write e SSL_read, respectivamente.

<pre>int SSL_write(SSL *ssl, const void *buf, int num)</pre>	Escreve <i>num</i> bytes do buffer <i>buf</i> na conexão <i>ssl</i> .
<pre>int SSL_read(SSL *ssl, void *buf, int num)</pre>	Lê <i>num</i> bytes do buffer <i>buf</i> na conexão <i>ssl</i> .

5. **Mostre o certificado:** você poderá mostrar no terminal os dados.

<pre>X509 *cert; char *line; cert = SSL_get_peer_certificate(ssl); if (cert != NULL){ line = X509_NAME_oneline(X509_get_subject_name(cert),0,0); printf("Subject: %s\n", line); free(line); line = X509_NAME_oneline(X509_get_issuer_name(cert),0,0); printf("Issuer: %s\n", line); }</pre>	<p>Mostra o certificado do peer com o seu nome e quem assinou seu certificado.</p> <p><i>X509 *cert;</i> - cria um ponteiro para receber um certificado do padrão X.509;</p> <p><i>cert = SSL_get_peer_certificate(ssl);</i> - Obtém o certificado do servidor.</p>
--	---

- **Compilação**

Para compilar tanto o cliente quanto o servidor, utilize os seguintes parâmetros via linha de comando:

```
%> gcc -o server server.c -lssl -lcrypto
%> gcc -o client client.c -lssl -lcrypto
```

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).
- Apresente claramente no relatório uma descrição dos pontos abaixo:
 - o (A) Explique o problema o funcionamento do algoritmo de exclusão mútua distribuído e suas limitações;
 - o (B) Como a replicação passiva foi implementada na sua aplicação e quais foram os desafios encontrados;
 - o (C) Explique os aspectos de segurança fornecidos pela nova versão da aplicação, em relação àquela desenvolvida na Parte I;
 - o (D) Estruturas e funções adicionais que você implementou;
- 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**, conforme a configuração de grupos da Parte I. 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 18 de dezembro.

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 2"). 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 01 de janeiro de 2018.