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

RELATÓRIO

ITALO FRANCO, PIETRA FREITAS, VITOR VANACOR E VITÓRIA ROSA

---

1. Ambiente de testes

Descrição do ambiente de testes: versão do sistema operacional e distribuição, configuração da máquina (processador(es) e memória) e compiladores utilizados (versões).

Sistema operacional: Ubuntu 16.04 LTS | 64-bit
Configuração da máquina:
Memória: 7,7 GiB
Processador: Intel® Core™ i5-4210U CPU @ 1.70GHz × 4
Compiladores: GCC com flags específicas para c++

2. Explique suas respectivas justificativas a respeito de:

o (A) Como foi implementada a concorrência no servidor para atender múltiplos clientes;

Servidor: Na thread principal, um socket espera qualquer tipo/origem de mensagem. Quando ela chega, ele verifica se o número de sessão já está na lista e

se é um pedido de conexão (SYN). Caso seja, é criada uma nova thread passando o endereço de origem da mensagem. A nova thread, então, confirma para o cliente que a conexão foi aceita (SYN/ACK).

Cliente: Após receber a confirmação, o cliente também envia um ACK, e a partir de então, com o 3-way handshake estabelecido, o cliente se comunica apenas com esta nova thread do servidor.

o (B) Em quais áreas do código foi necessário garantir sincronização no acesso a dados;

Sempre que um arquivo irá ser transferido, seja por comando do cliente ou pela thread de sincronização, verifica-se se o arquivo já não está sendo transferido por um dos casos citados anteriormente. Caso esteja, a operação não é realizada (é exibida uma mensagem pro cliente esperar ou o arquivo é ignorado na iteração da thread de sync e será feito uma nova tentativa na próxima iteração). Isso evita, por exemplo, que um arquivo seja baixado enquanto a thread de sync já está baixando um arquivo de mesmo nome, o que geraria inconsistência.
Este controle é realizado através de uma lista que registra o nome dos arquivos sendo transferidos, tanto no cliente quanto no servidor. Entretanto, como há duas threads que não devem acessar a lista ao mesmo tempo, ela é uma seção crítica, sendo então necessário pegar o lock de um mutex para adicionar ou remover arquivos desta lista, garantindo que a busca e alterações na mesma sejam feitas de maneira atômica.

o (C) Estruturas e funções modificadas e/ou adicionais que você implementou;

Seguimos uma abordagem mais orientada a objetos, com melhor separação de responsabilidades e seguindo boas práticas de programação, criando módulos separados para cada funcionalidade e buscando um código auto-documentado.
Segue uma breve explicação sobre cada classe implementada:

- **Socket:** Encapsula as funções do socket.h gerando uma interface mais abstrata, para não ser necessário se preocupar com os detalhes de mais baixo-nível da biblioteca em C.
- **Connection:** Representa uma conexão entre um cliente e o servidor. A classe verifica que as mensagens foram mesmo entregues (sendACK e receiveACK), na ordem correta e sem duplicatas. Foi implementado também um timeout de retransmissão, ou seja, se não for recebido uma resposta após um tempo especificado, as mensagens não confirmada são reenviadas.
  Connection também é responsável em enviar e receber arquivos através de pacotes.

- **Message:** Estrutura as mensagens que serão trocadas. Possui os campos session e sequence para controle da conexão, type para informar qual o propósito da mensagem e indicar como o content - a quarta parte da mensagem - que contém os dados em si, deve ser interpretado.
  Tem método para ser transformado em string a fim de ser transmitido, e outro de parse para ser interpretado no recebimento.
- **File:** Fornece funções para manipulação do sistema de arquivos.
- **ServerSync:** Implementa a manutenção da consistência dos arquivos do lado do servidor. Após a conexão com o cliente, é feito o envio das informações sobre os arquivos, como seu nome e o momento em que foi feita sua última modificação, para comparar com os dados salvos no servidor (mensagem T_STAT). O servidor então responde informando ao cliente a ação a ser tomada, isto é, se é necessário que o cliente faça upload (T_UPLOAD), download (T_DOWNLOAD) ou se o arquivo é idêntico ao que armazenado no servidor (T_EQUAL). O cliente envia uma mensagem de término quando todos seus arquivos foram sincronizados (T_DONE). Por sua vez, o servidor então verifica se não há nenhum arquivo que o cliente não requisitou sincronização, significando então que o cliente não o contém e precisa realizar seu download (T_DOWNLOAD). Caso não haja nenhum arquivo faltante, o servidor então envia uma mensagem de fim da sincronização (T_DONE).
- **ClientSync:** Responsável pela verificação da sincronização de arquivos do lado do cliente. Implementa funções que esperam informações do servidor, avisando que o cliente precisa enviar seus dados para o servidor (T_UPLOAD), que o cliente não possui a versão mais atualizada e precisa baixar dados (T_DOWNLOAD), ou se ambos estão sincronizados (T_EQUAL). Registram-se os logs das transações necessárias e de erros, caso estes ocorram durante a sincronização dos arquivos.
- **ServerThread:** É uma thread do servidor. Trata as requisições enviadas pelo cliente através da linha de comando. Roda concorrentemente à ServerSync.
- **sydClient:** Aplicação do cliente. É o terminal, onde podemos fazer requisições de upload, download ou listagem dos arquivos das pastas sincronizadas. Roda concorrentemente à ClientSync.
- **sydServer:** Aplicação do servidor, espera por conexões e cria uma nova ServerThread para cada cliente conectado.
- **sydUtil:** Funções de debug e de uso comuns.

o (D) Explicar o uso das diferentes primitivas de comunicação;

Utilizamos SOCKETS UDP com receive bloqueante e send não bloqueante. Por ser UDP, foram utilizadas as primitivas: rcvfrom que armazena o endereço da

origem para saber para quem deve ser enviada a resposta; e a sendto, que recebe o endereço de destino para onde deseja-se enviar a mensagem.

• Problemas encontrados durante a implementação

Um dos grandes problemas é o de desenvolver um tratamento adequado dos erros. Cada operação que envolve rede e cada operação no sistema de arquivos está sujeito a dar problemas, então um sistema de transferência de arquivos através da rede é por natureza sujeito a uma quantidade imensa de erros. Tratar todos os casos excepcionais de uma maneira robusta leva muitas vezes mais tempo do que de fato implementar o código, considerando só o caso em que tudo dá certo.
Tratamos vários erros, mas sempre há mais para serem considerados.