UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANDREI POCHMANN KOENICH

ANDREI ROCHA BERETA

PEDRO COMPANY BECK

TURMA A

RELATÓRIO - TRABALHO PRÁTICO PARTE 2

Disciplina: Sistemas Operacionais II

Professor: Alberto Egon Schaeffer Filho

Porto Alegre, janeiro de 2024.

1 INTRODUÇÃO

O relatório a seguir tem por objetivo fornecer dados a respeito da implementação em C++ da segunda parte do desenvolvimento de uma aplicação cliente-servidor, utilizada para fornecer um serviço de compartilhamento e sincronização automática de arquivos, entre diferentes dispositivos de um mesmo usuário.

Na primeira parte do projeto, foi desenvolvida uma aplicação servidor deve ser capaz de gerenciar arquivos de diversos usuários remotos. A aplicação cliente desenvolvida, por sua vez, corresponde à parte da aplicação presente na máquina dos usuários, permitindo a cada um deles acessar remotamente seus arquivos mantidos pelo servidor. Foi utilizado o mecanismo de *sockets* para o estabelecimento de comunicação entre o cliente e o servidor.

A segunda parte do projeto consiste em incorporar um esquema de Replicação Passiva à aplicação. A fim de manter a disponibilidade do sistema quando da ocorrência de falhas no servidor principal, um novo servidor deverá assumir o seu papel e manter o serviço de gerenciamento de arquivos funcionando. Essa mudança deve ser transparente para os usuários, e seus arquivos devem ser mantidos disponíveis, mesmo após a falha. Para garantir que as modificações de arquivos estarão disponíveis em um novo servidor, o esquema de replicação de informar todas as modificações realizadas, aos servidores secundários.

O esquema de Replicação Passiva implementado possui as seguintes garantias:

(1) todos os clientes sempre utilizarão a mesma cópia primária;

(2) após cada operação, o servidor primário irá propagar o estado dos arquivos aos servidores de *backup*;

(3) somente após os *backups* serem atualizados o primário confirmará a operação ao cliente.

Caso o servidor principal venha a falhar, um Algoritmo de Eleição de líder (Algoritmo do Valentão) será utilizado para determinar o próximo servidor primário. Nesse caso, um dos servidores *backup* deverá assumir essa função, mantendo um estado consistente do sistema.

2 DESCRIÇÃO DO AMBIENTE DE TESTES

Abaixo, segue a descrição do ambiente de testes utilizado por cada um dos integrantes do grupo.

2.1 Ambiente de testes do aluno Andrei Pochmann Koenich

Versão do Sistema Operacional: Linux Ubuntu 22.04.3 LTS.

Configurações da máquina: Processador Intel(R) Core(TM) i7-8700 CPU 4 3.20GHz, com memória RAM de 16GB.

Compilador utilizado: GCC 12.3.0.

2.2 Ambiente de testes do aluno Andrei Rocha Bereta

Versão do Sistema Operacional: Linux Ubuntu 22.04.3 LTS, sobre o WSL2 no Windows 10.

Configurações da máquina: Processador AMD RYZEN 7 5700X, com memória RAM de 16GB.

Compilador utilizado: GCC 11.4.0.

2.3 Ambiente de testes do aluno Pedro Company Beck

Versão do Sistema Operacional: Linux Ubuntu 20.04.3 LTS, sobre o WSL2 no Windows 10.

Configurações da máquina: Processador Intel Core i5 8350U, com memória RAM de 16GB.

Compilador utilizado: GCC 9.4.0.

3 NOVOS PACOTES DEFINIDOS

Abaixo, segue a descrição de todos os novos pacotes definidos, em adição aos pacotes já existentes na parte 1 da implementação. As definições das estruturas dos pacotes estão localizadas em *Common/package.hpp*, com conversão adequada de formatos e a garantia de um *padding* bem definido e simples de calcular, conforme já destacado no relatório anterior.

Tabela 1 – Pacotes Enviados de um Servidor de *Backup* para o Servidor Principal	
Tipo do Pacote	**Descrição**
PackageReplicaManagerIdentification	Identificação de um novo servidor de backup (*Replica Manager*).
PackageReplicaManagerTransferIdentification	Indica ao servidor principal que deverá ser iniciada uma *thread* de transferência de dados (arquivo ou propagação de evento detectado no diretório do cliente) para o servidor de *backup* emissor do pacote.
PackageReplicaManagerPing	Utilizado para enviar constantemente um sinal ao servidor principal (*ping*), a fim de verificar se ele está ativo, por meio de uma resposta enviada pelo servidor. Caso não esteja (ocorrência de *timeout*), será iniciada uma eleição pelo servidor de backup, para definir um novo servidor principal.

Tabela 2 – Pacotes Enviados de um Servidor Principal para um Servidor de *Backup*	
Tipo do Pacote	**Descrição**
PackageReplicaManager IdentificationResponse	Indica se o pedido de conexão de um servidor *backup* foi aceito pelo servidor principal.
PackageReplicaManager TransferIdentification Response	Indica ao servidor de *backup* que a *thread* de transferência de dados (arquivo ou propagação de evento detectado no diretório do cliente) pode será inicializada, para iniciar a transferência.
PackageActiveConnectionsList	Contém uma lista de todos os dispositivos conectados ao servidor principal (dispositivos de clientes conectados e servidores de *backup* conectados).
PackageReplicaManagerPingResponse	Indica a um servidor de *backup* que o servidor principal remetente está em funcionamento.

Tabela 3 – Pacotes Enviados de um Servidor de *Backup* para um Servidor de *Backup*	
Tipo do Pacote	**Descrição**
PackageReplicaManagerElectionElection	Utilizado para iniciar uma eleição, a fim de determinar um novo servidor principal, entre os servidores de *backup*. Enviado por um servidor de *backup* para todos os outros servidores de *backup* com ID superior ao do remetente, para verificar se eles estão ativos.
PackageReplicaManagerElectionAnswer	Enviado por um servidor de *backup* em funcionamento regular para um outro servidor de *backup,* após receber desse último um pacote de *election*, para viabilizar a eleição.
PackageReplicaManagerElectionCoordinator	Enviado por um servidor de *backup* para todos os demais servidores de *backup* existentes, para informar que o remetente é o novo servidor principal, após a execução do Algoritmo de Eleição (ver seção 5).

Tabela 4 – Pacote Enviado de um Servidor Principal para os Clientes	
Tipo do Pacote	**Descrição**
PackageNewServerInfo	Enviado por um novo servidor principal (recém-eleito, após a execução do Algoritmo de Eleição) para todos os dispositivos clientes conectados ao sistema, com as informações necessárias para que eles possam efetuar uma nova conexão com o servidor recém-eleito.

4 PRINCIPAIS NOVAS SEÇÕES CRÍTICAS IMPLEMENTADAS

Assim como na parte 1 do projeto desenvolvido, a garantia de sincronização ocorre com o uso de *mutexes*, com as primitivas `pthread_mutex_lock()` e `pthread_mutex_unlock()`. No cliente, também foram utilizadas as primitivas `pthread_cond_wait()` e `pthread_cond_signal()`. A seguir, estão descritos os principais novos trechos do projeto nos quais houve a necessidade de utilizar a garantia de sincronização com essas primitivas.

Servidor/myServer.cpp:316: seção crítica utilizada para abrir uma conexão por vez com cada possível conector (servidor *backup* ou dispositivo cliente).

Servidor/deviceManager.cpp:406: seção crítica utilizada para adicionar dados de conexão (número de IP e porta) de um dispositivo cliente, na lista de conexões ativas.

Servidor/deviceManager.cpp:539: seção crítica utilizada para remover dados de conexão (número de IP e porta) de um dispositivo cliente, da lista de conexões ativas.

Servidor/pingThread.cpp:85: seção crítica utilizada para garantir que as conexões para transferência de mensagens *ping* e de transferência de arquivos sejam abertas uma de cada vez, evitando conflitos de conexão.

5 FUNCIONAMENTO DO ALGORITMO DE ELEIÇÃO IMPLEMENTADO

O Algoritmo de Eleição implementado consiste no Algoritmo do Valentão (*Bully Algorithm*). A escolha para esse algoritmo é justificada pela facilidade na sua implementação, visto que o modo como a aplicação foi desenvolvida na parte 1 possibilita que novos tipos de pacotes (referentes às mensagens *coordinator*, *answer* e *election*, características do Algoritmo do Valentão) sejam facilmente definidos, com as definições desses pacotes estando presentes em *Common/package.hpp.*

Para viabilizar o funcionamento do Algoritmo do Valentão na aplicação desenvolvida, utiliza-se a estrutura `ActiveConnections_t` definida em *Common/connections.hpp*, por meio da qual é possível a criação de uma lista contendo informações de conexão (endereço IP e número de porta) de todos os servidores de *backup* conectados, para que seja possível a transferência de pacotes referentes às mensagens *election*, *coordinator* e *answer* (ver Tabela 3). Nessa lista, também são armazenadas as informações de conexão de todos os dispositivos dos clientes conectados ao servidor principal. Nos casos em que ocorre uma nova conexão de um dispositivo cliente, os dados de conexão desse dispositivo são armazenados na lista, com essa lista atualizada sendo propagada pelo servidor principal para todos os servidores de *backup*. Da mesma forma, toda vez que um novo servidor de *backup* é conectado ao servidor principal, as informações de conexão do novo servidor de *backup* são inseridas na lista atualizada, que também será propagada pelo servidor principal para todos os demais servidores de *backup* conectados. Em suma, todos os servidores de *backup* possuem informações de conexão de todos os dispositivos clientes conectados, além dos demais servidores de *backup* do sistema.

Com essas implementações, uma vez que um servidor de *backup* é eleito como novo servidor principal, é possível enviar as informações para a realização da nova conexão com esse servidor recém-eleito para todos os dispositivos dos clientes conectados.

5.1 Envio de mensagens de *ping* e início da eleição com envio de mensagens *election*

Em *Servidor/myServer.cpp:170* ocorre a inicialização de uma *thread* a ser executada por um servidor de *backup*, responsável por enviar mensagens de *ping* ao servidor principal. Essa *thread* realiza a chamada da função `pingThread()` (definida em *Servidor/pingThread.cpp:83*). Com a chamada da função `send_ping_to_socket()` (definida em *Common/package_file.cpp:90*), é feito um envio de um pacote específico para a operação de *ping* (ver Tabela 1) do servidor de *backup* para o servidor principal em funcionamento.

A chamada da função `read_package_from_socket()` em *Servidor/pingThread.cpp:100* garante que o servidor de *backup* permanecerá aguardando um retorno à mensagem de *ping* enviada, por parte do servidor principal. Caso isso não ocorra, a eleição será iniciada pelo servidor de *backup*, com a chamada da função `send_election_to_backups_list()` em

Servidor/pingThread.cpp:107, que passará a enviar mensagens de election para todos os servidores de backup conectados. Caso o servidor principal responda normalmente à mensagem de *ping* enviada pelo servidor de *backup*, depois de um segundo será enviada novamente uma mensagem de *ping*. Assim, realiza-se uma verificação contínua a respeito do funcionamento adequado do servidor principal.

5.2 Recebimento de mensagens *coordinator, election* e *answer,* e envio de mensagens *answer*

A iteração definida (após o disparo da *thread* responsável pelo envio das mensagens de *ping* ao servidor principal) em *Servidor/myServer.cpp:174* é responsável por garantir que o servidor de *backup* permaneça aguardando por mensagens do tipo *election* ou do tipo *coordinator*. Caso ocorra o recebimento de uma mensagem do tipo *election*, realiza-se o envio de uma mensagem do tipo *answer*, por meio da chamada da função `send_answer_to_socket()` em *Servidor/myServer.cpp:203.* Caso ocorra o recebimento de uma mensagem do tipo *coordinator*, todas as *threads* de comunicação com o antigo servidor principal são excluídas, e são inicializadas *threads* de conexão com servidor recém-eleito, para transferência de mensagens de *ping* e de dados (arquivo ou propagação de evento detectado no diretório do cliente) com as operações realizadas em *Servidor/myServer.cpp:208.*

Após a chamada da função `send_election_to_backups_list()` em *Servidor/pingThread.cpp:107,* verifica-se se o número de mensagens do tipo *answer* recebidas é igual a zero. Caso seja, a iteração descrita no parágrafo anterior é interrompida, com a chamada da função `break_accept_on_main_thread()` em *Servidor/pingThread.cpp:112.* Em seguida, realiza-se o envio de mensagens do tipo *coordinator* para todos os demais servidores de *backup*, com a iteração presente em *Servidor/myServer.cpp:263.* Por fim, com a iteração presente em *Servidor/myServer.cpp:288,* todos os dispositivos de todos os clientes conectados realizam uma nova conexão com o servidor principal recém-eleito.

6 IMPLEMENTAÇÃO DA REPLICAÇÃO PASSIVA

A *thread* inicializada por um servidor de *backup* em *Servidor/myServer.cpp:165* é responsável por aguardar conexões do servidor principal, que irá realizar o envio de dados (arquivos ou propagação de eventos detectados nos diretórios dos clientes). Essa *thread* realiza a chamada da função `backupThread()`, definida em *Servidor/backupThread.cpp:145*. A iteração presente em *Servidor/backupThread.cpp:159* é responsável por aguardar o envio de pacotes por parte do servidor principal, e invocar os procedimentos adequados, dependendo do tipo do pacote recebido (ver Tabela 2).

No teste presente em *Servidor/backupThread.cpp:170*, *c*aso seja recebido um pacote referente à atualização da lista de conexões de dispositivos clientes e servidores de *backup* que estão ativas (descrita na seção 5), a atualização em si ocorre por meio da chamada da função `handleActiveConnectionsList()` em *Servidor/backupThread.cpp:174*. Em seguida, são criados diretórios para cada um dos clientes referenciados na lista de conexões ativas, com a chamada da função `create_client_dirs()` em *Servidor/backupThread.cpp:178*, para armazenamento dos dados de um cliente a serem recebidos.

No teste presente em *Servidor/backupThread.cpp:200*, caso seja recebido um pacote referente à detecção de um determinado evento em um diretório de um dispositivo cliente (a detecção ocorre por meio da utilização da API *inotify* que é utilizada para monitorar o diretório local de cada dispositivo cliente, assim como na parte 1 do projeto), tal evento é propagado no diretório do dispositivo cliente presente no diretório do servidor *backup* que recebeu esse pacote, com a chamada da função `handleChangeEvent()` em *Servidor/backupThread.cpp:202*. A função `handleChangeEvent()`, definida em *Servidor/backupThread.cpp:40*, é responsável por verificar qual o evento em ocorrência, o qual pode ser criação, modificação, renomeação ou deleção de um determinado arquivo de um dispositivo cliente.

7 PROBLEMAS ENCONTRADOS E SUAS SOLUÇÕES

7.1 Conexões de servidores *backup* a um novo servidor principal recém-eleito

Nos casos em que um novo servidor principal era eleito após a execução do Algoritmo de Eleição (Algoritmo do Valentão), duas ou mais conexões podiam ser abertas simultaneamente no servidor principal pelos servidores de *backup* a serem conectados. Em razão disso, ocorriam conflitos na identificação dos servidores de backup frente ao novo servidor principal. Isso foi resolvido por meio da implementação de uma seção crítica, presente em *Servidor/myServer.cpp:316* (ver seção 4).