Universidade Federal do Rio Grande do Sul

Instituto de Informática - Departamento de Informática Aplicada

Sistemas Operacionais II N - Turma A
Professor Weverton Cordeiro

Trabalho Prático 2

Um chat cliente servidor utilizando sockets com Replicação Passiva e Tolerância a falhas

Douglas Gehring - 243682
Henrique Valcanaia - 240501
Leonardo Felipe Mendes - 250383
Rodrigo Cardoso Buske - 206526

Porto Alegre, 25 de Novembro de 2020.

Especificação do Trabalho

A proposta desta etapa do trabalho é implementar **replicação passiva** e um **algoritmo de eleição de líder** a partir do projeto de troca de mensagens criado na etapa 1 do trabalho.

Visão geral

Foi criado uma nova aplicação chamada **`front`** no projeto. Esta nova aplicação tem apenas 1 responsabilidade, que é receber e reencaminhar mensagens de clients e do servidor primário, mantendo basicamente nenhum estado nele, apenas as threads abertas com o conhecimento do socket de qual estão lendo.

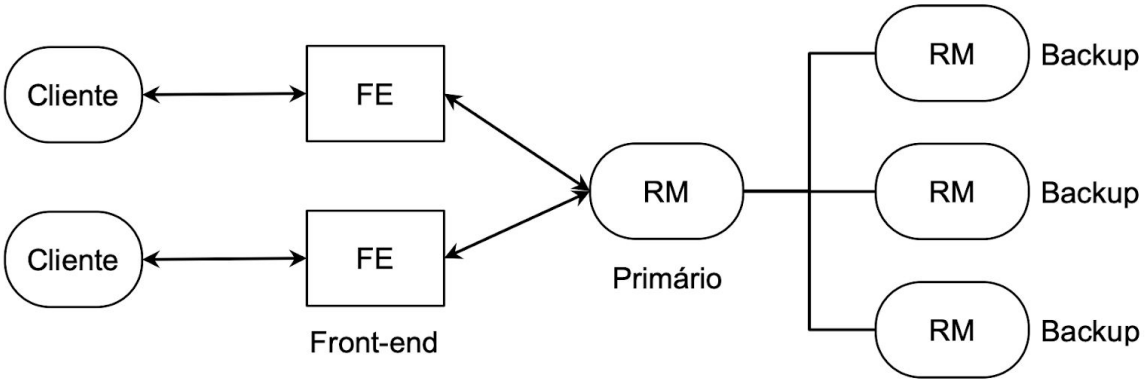
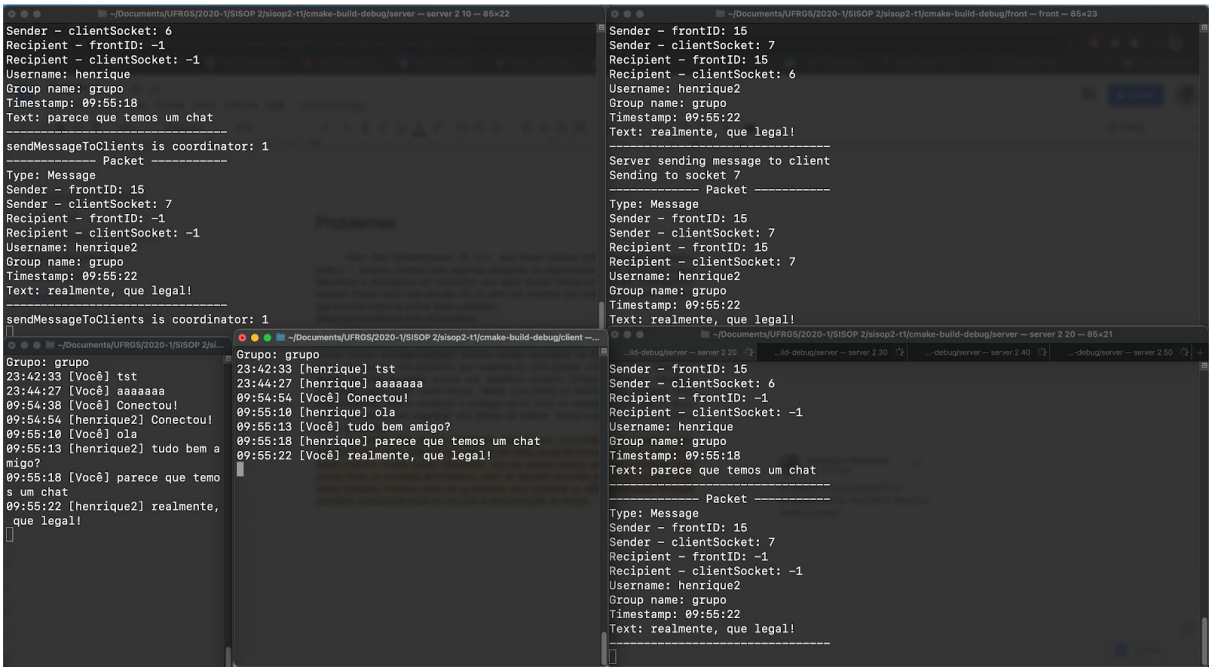


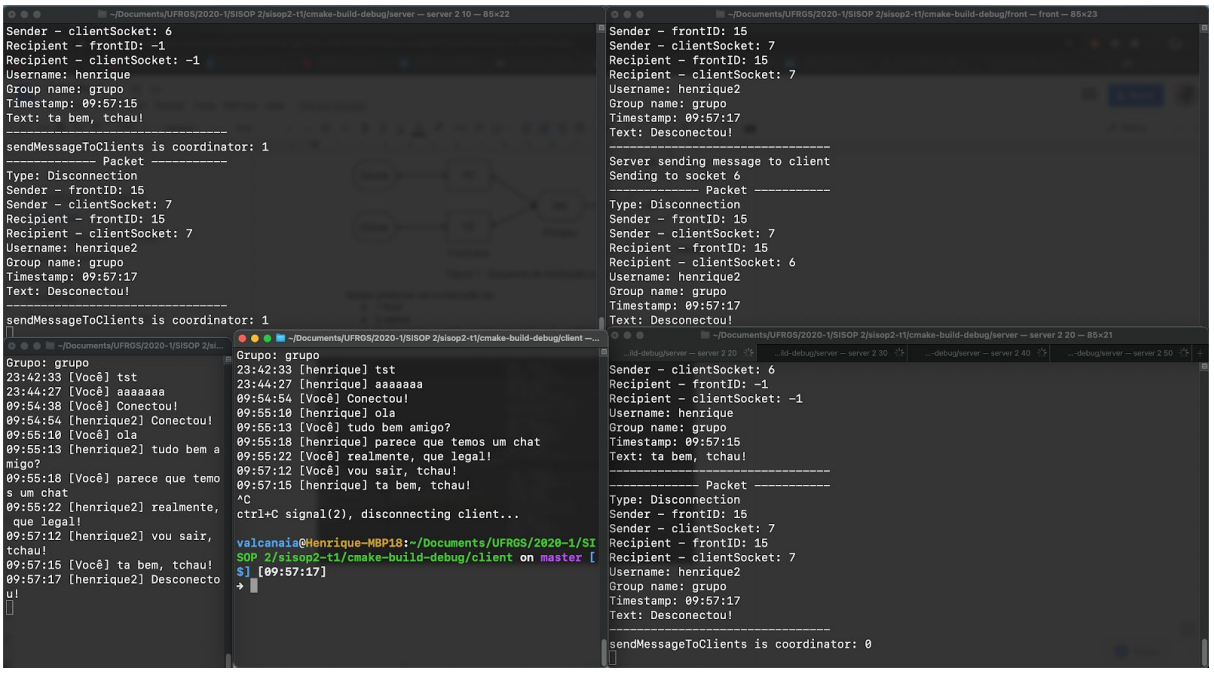
Figura 1 - Esquema de replicação passiva

Abaixo podemos ver a execução de:

- 1 front
- 2 clients
- 5 servers (1 no canto superior esquerdo e 4 no inferior direito)

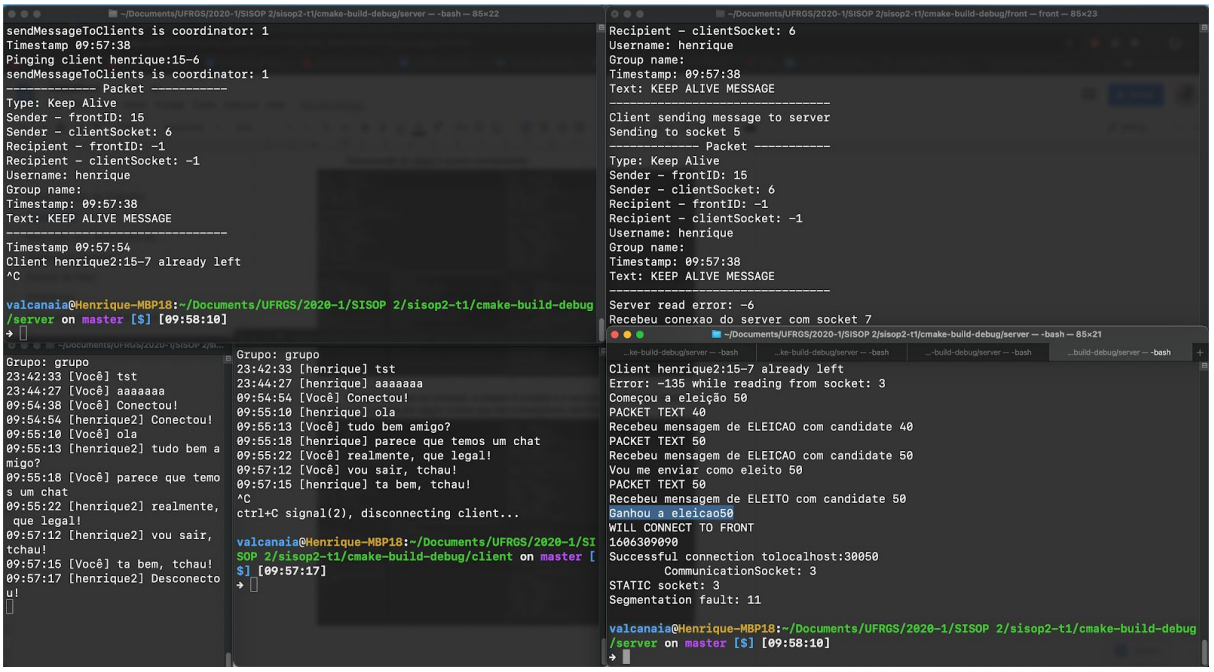


Desconexão do client 2 ocorre normalmente:



Ao derrubar o server principal, a eleição é iniciada e o servidor correto com o maior id(50) é eleito, porém, logo após a eleição e conexão do servidor principal com o front e dos backups com o principal, ocorre segmentation fault por algum motivo que não conseguimos identificar a tempo da entrega.

Importante ressaltar que a eleição ocorre normalmente - o servidor correto vence a eleição e as conexões posteriores entre o servidor principal e o front, e dos servidores de backups com o servidor principal, parece ocorrer normalmente.



Stack trace:

```
Thread 0:: Dispatch queue: com.apple.main-thread
0   libsystem_kernel.dylib           0x00007fff203014e6 __accept + 10
1   server                           0x0000000104f7f060
ServerCommunicationManager::setupServerToServerConnection(SocketConnectionInfo) + 96
(ServerCommunicationManager.cpp:410)
2   server                           0x0000000104f7fe08
ServerCommunicationManager::startServer(int, int) + 168 (ServerCommunicationManager.cpp:503)
3   server                           0x0000000104f6b5ca main + 730 (main.cpp:33)
4   libdyld.dylib                    0x00007fff2034a631 start + 1

Thread 1:
0   libsystem_kernel.dylib           0x00007fff202fd7b6 __semwait_signal + 10
1   libsystem_c.dylib                0x00007fff2027ac92 nanosleep + 196
2   libsystem_c.dylib                0x00007fff2027aafa sleep + 41
3   server                           0x0000000104f7cb41
ServerCommunicationManager::newClientConnectionKeepAlive(KeepAliveThreadArguments*) + 145
(ServerCommunicationManager.cpp:309)
4   server                           0x0000000104f7ca95
ServerCommunicationManager::staticNewClientConnectionKeepAliveThread(void*) + 37
(ServerCommunicationManager.cpp:43)
5   libsystem_pthread.dylib          0x00007fff2032f950 _pthread_start + 224
6   libsystem_pthread.dylib          0x00007fff2032b47b thread_start + 15

Thread 2 Crashed:
0   server                           0x0000000104f7c26e
ServerCommunicationManager::staticHandleNewFrontConnectionThread(void*) + 46
(ServerCommunicationManager.cpp:35)
1   libsystem_pthread.dylib          0x00007fff2032f950 _pthread_start + 224
2   libsystem_pthread.dylib          0x00007fff2032b47b thread_start + 15
```

Descrição dos Ambientes de desenvolvimento e testes

No grupo foram utilizados dois sistemas operacionais distintos, a maior parte do tempo foi macOS 11.0.1, mas também foi utilizado o Ubuntu numa VM no Windows. Importante salientar que desenvolvemos e testamos tanto no macOS quanto no Ubuntu, incluindo comunicação entre os sistemas distintos.

Todos os membros da equipe utilizaram a IDE CLion da JetBrains, a qual é uma excelente IDE para projetos que são implementados nas linguagens C/C++ (as instruções para rodar o projeto não necessitam do CLion).

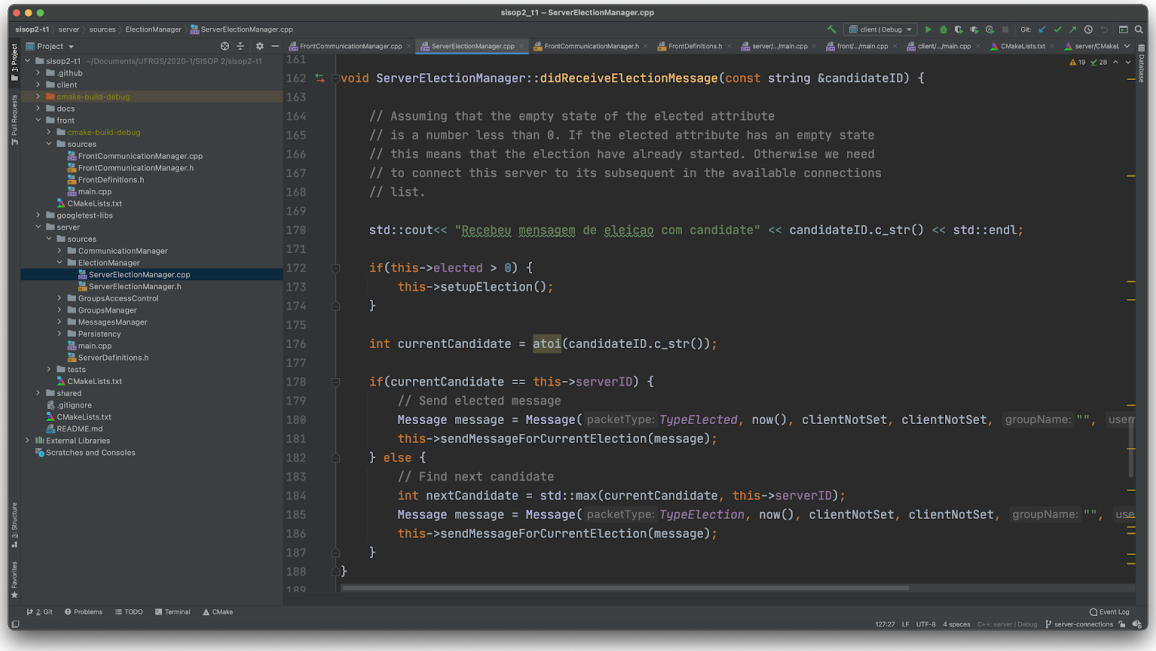


Figura 2: Interface da IDE CLion

Para controle de versões, utilizamos o Git na plataforma GitHub. Também utilizamos o Tower, um cliente de Git principalmente para facilitar a visualização de alterações e commit de linhas individuais.

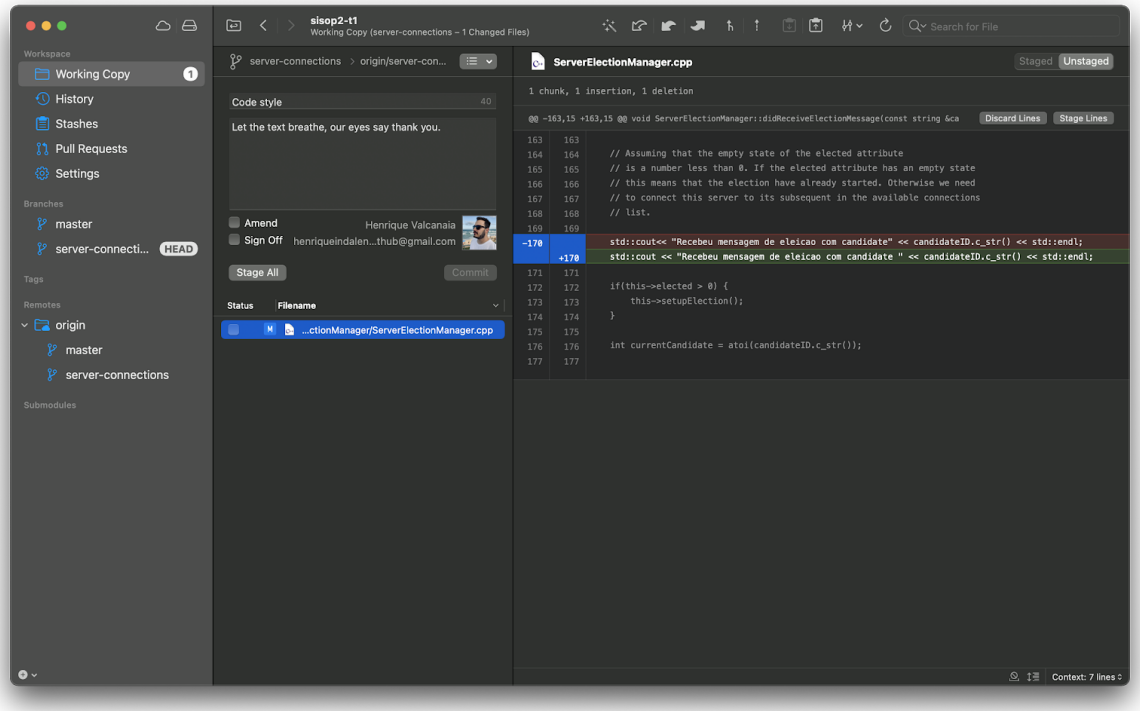


Figura 3: Interface do Tower

O GitHub também foi bastante importante para garantir o funcionamento entre os sistemas operacionais. Utilizamos o Github Actions para compilar o projeto no macOS e Ubuntu a cada "*pull request*", evitando problemas de integração. O projeto pode ser encontrado em https://github.com/Robuske/sisop2-t1.

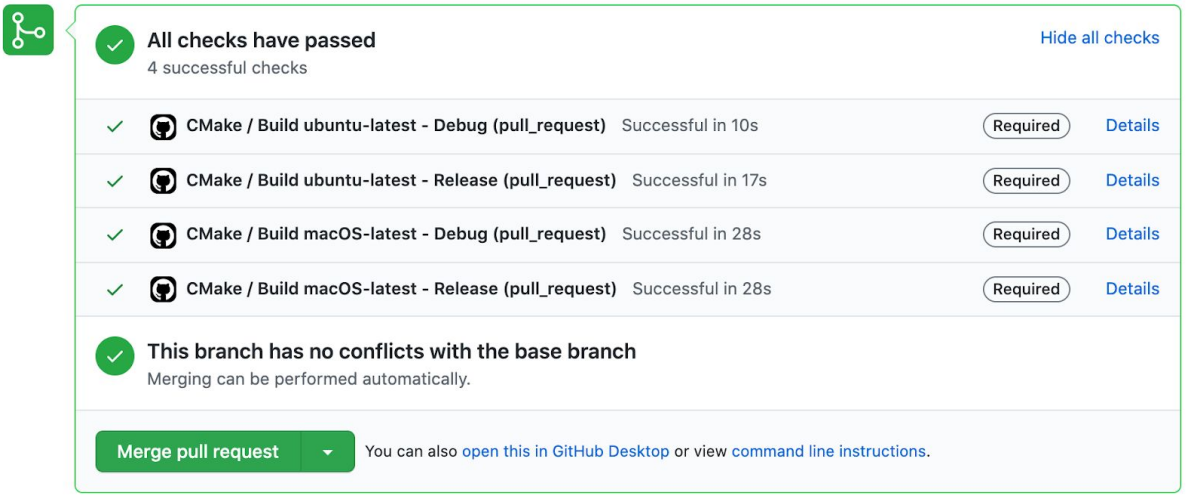


Figura 4: Checklist para as releases tanto em ambientes Linux x macOS

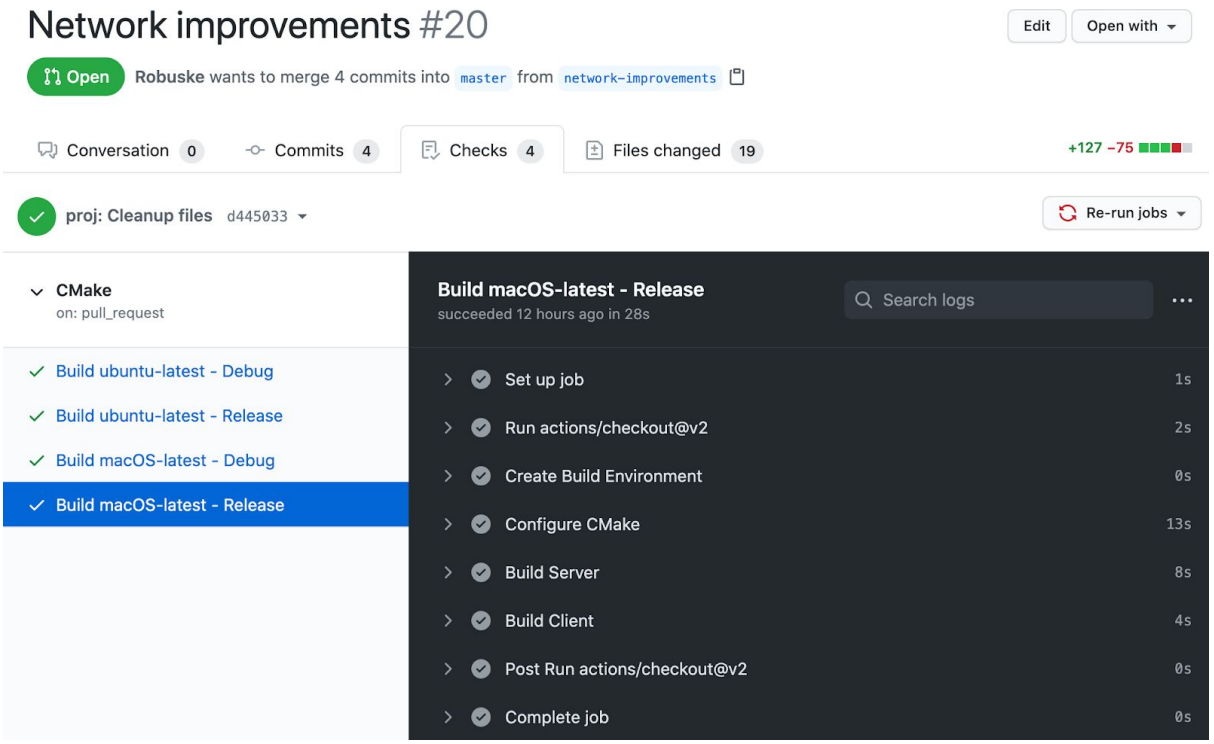


Figura 5: Detalhe do tempo de build para macOS

O ambiente utilizado para testes foi uma máquina com o sistema operacional Ubuntu Linux 20.04 LTS, com 8 GB de Memória RAM, processador Intel Core i5 8th Gen 1.8 Ghz.

Para compilar o projeto, foram necessários alguns passos iniciais listados abaixo. Atenção para o fato de que a versão utilizada do C++ é a 14, e a do CMake é 3.17, o passo 4 deve baixar a versão mais atual do CMake. Não ficou muito claro quando é necessário o passo 5, mas parece ser relacionado a ter ocorrido mudanças na estrutura do projeto (como trocar de *branch*), ao rodar a primeira vez provavelmente não vai precisar:

Pré-requisitos necessários para a *"build"* e *"run"* do projeto (Linux):

1. Abrir o terminal do Ubuntu
2. `sudo apt-get update`
3. `sudo apt-get install build-essential`
4. `(opcional) sudo snap install cmake --classic`
5. `(opcional) cmake --configure .`
6. `cmake --build . --target front`
7. `cmake --build . --target server`
8. `cmake --build . --target client`
9. `Os executáveis estarão disponíveis em front/front, server/server e client/client.`

Observamos durante o *troubleshooting* que o Ubuntu não estava completamente atualizado, e através da execução desses comandos a compilação em C++ e C foi possível, visto que durante o passo 3 é instalado tanto o compilador GCC quanto o Cpp.

Segue abaixo um exemplo do software do projeto em execução:

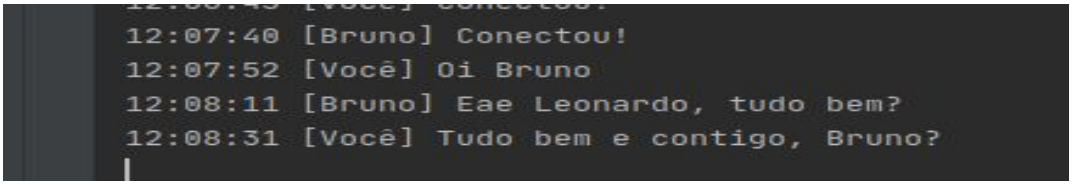


Figura 6: Conversa entre Leonardo (Você) e Bruno.

Replicação passiva

A replicação passiva, conforme definido por único gerenciador de réplica (RM) primário e um ou mais gerenciadores de réplica secundários – backups ou escravos. Na forma pura do modelo, os front-ends (FE) se comunicam somente com o gerenciador de réplica primário para obterem o serviço. O gerenciador de réplica primário executa as operações e envia cópias dos dados atualizados para os backups.

O papel dela neste projeto é garantir que todas as replicações contenham as administrações das mensagens enviadas entre os clientes e servidores. Por exemplo, caso o Front-End venha ter um evento de falha, ele deve ser imediatamente substituído e todo o contexto do usuário no grupo deve ser restaurado no ponto imediatamente em que foi ocorrido a falha. Dessa forma é um mecanismo muito importante para garantir a tolerância a falhas, principalmente no quesito confiabilidade e disponibilidade do Chat que estamos implementando.

Multicast nos parece ser a melhor opção para implementar o envio de mensagens de replicação, pois desta forma enviamos apenas 1 mensagem, porém, como já temos a implementação unicast pronta e esta já inclui uma forma de detectar que o server principal caiu (a conexão ter sido fechada), decidimos por utilizar unicast.

Abaixo um fluxo demonstrando.

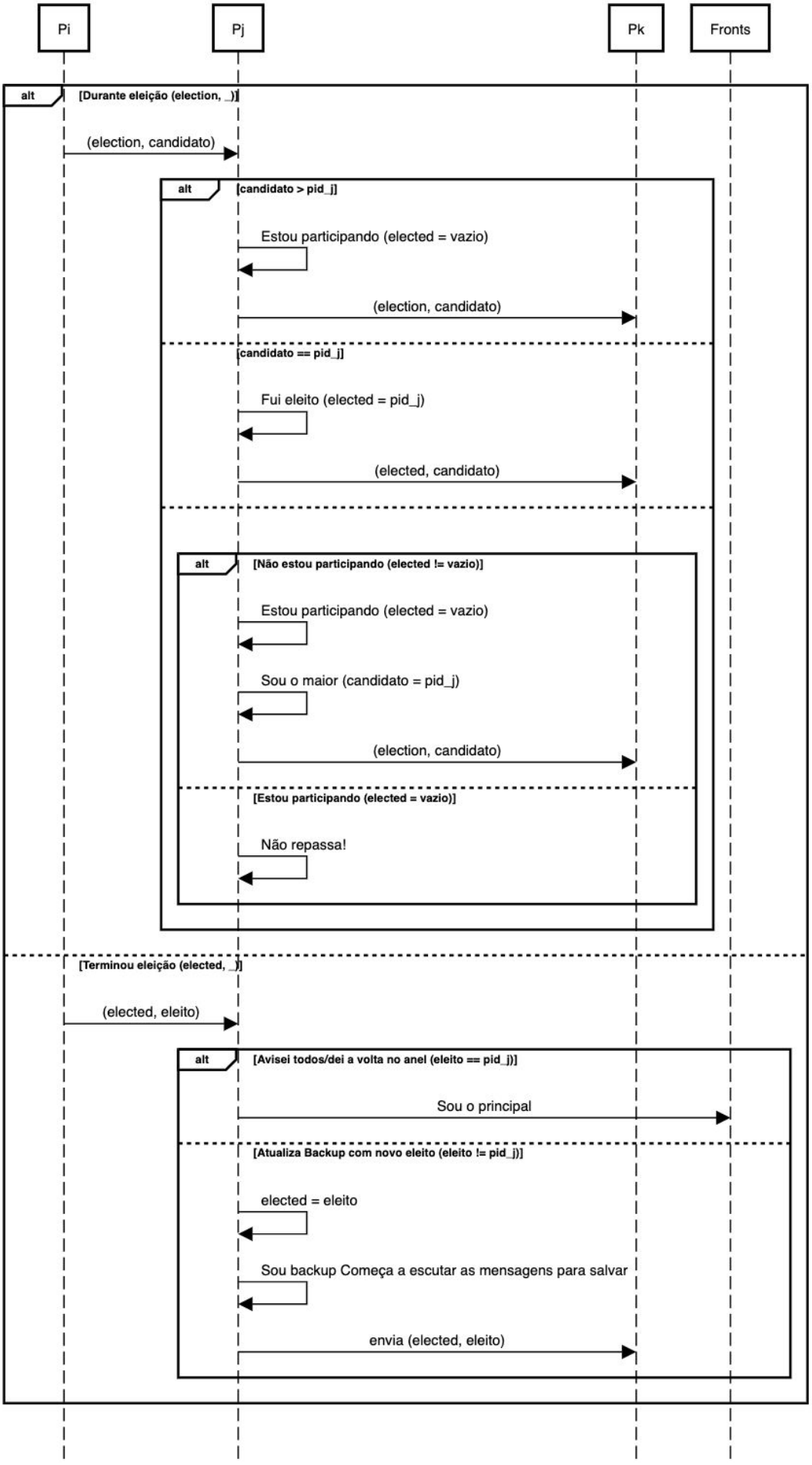
Eleição de líder

A eleição de líder é importante para sabermos quem se tornará o target principal da comunicação após uma falha acontecer e o anterior não estar mais apto a receber mensagens. Dessa forma, além de uma maior robustez, temos também uma maior disponibilidade.

Neste projeto utilizamos o algoritmo do Anel de Chang & Roberts, 1979. Escolhemos este algoritmo por ser o de implementação mais fácil, se comparado a eleição pelo algoritmo do Valentão, e considerando a data de entrega não teríamos tempo hábil para implementar o outro.

Para implementação do algoritmo da eleição, nos baseamos no seguinte diagrama de sequência elaborado por nós:

Algoritmo do anel [Chang & Roberts, 1979]



Problemas

Como citado no Trabalho Prático 1, grande parte dos problemas foram graças a idiossincrasias do C++, dessa vez chegando a nos travar num *segmentation fault* que não conseguimos resolver.

Este foi um semestre excepcional. Quem, além da estatística, diria que passaríamos por uma pandemia em nossas vidas? Como esse foi o primeiro semestre em regime de ensino remoto emergencial (ERE) tivemos muitas novidades na forma como as cadeiras foram ministradas. Um problema que notamos foi uma grande concorrência de entregas. Muitas cadeiras trocaram provas por trabalhos, portanto tínhamos diversos trabalhos extraclasse para dividir nosso tempo. Talvez uma forma de resolver este problema seria disponibilizando todo o conteúdo e entregas já no início do semestre, para que o aluno tenha autonomia para organizar seu tempo da melhor forma possível e sem prejuízo a outras atividades. Também em específico para essa disciplina nos fez falta ter o resultado da avaliação da outra parte do trabalho, ajudaria termos uma avaliação do que poderíamos ter feito melhor para quem sabe já aplicar na parte 2.

A questão das outras disciplinas estarem em concorrência no quesito tempo “extracurricular”. Vale salientar que mesmo em casa, longe da universidade por si só, ainda assim tivemos muitas outras disciplinas, que ao mesmo tempo, estavam marcando suas provas finais ou entregas de trabalhos, além de também haverem provas em conjunto com essas entregas. Portanto, além de se preparar para enfrentar as dificuldades aqui na nossa disciplina, precisamos estar em dia com a administração do tempo. Salientamos, no entanto, que essa administração de tempo deve ter sido um experimento e um grande teste para todos os pontos de vistas, tanto para a visão de aluno quanto para visão de professor, pois todos sem exceção tem a intenção de entregar o melhor resultado/passar conhecimento da melhor maneira possível.