Universidade do Minho

Departamento de Informática

licenciatura em engenharia informática

Segurança de Sistemas Informáticos - TP2

Grupo 5

Simão Antunes

Henrique Pereira

Luís Caetano

Simão Antunes A100597
Henrique Pereira A100831
Luís Caetano A100893

Maio de 2024

Conteúdo

1 Introdução

Este relatório apresenta o projeto de desenvolvimento de um serviço de conversação para utilizadores locais de um sistema Linux. O serviço tem como objetivo principal facilitar a comunicação entre os utilizadores, permitindo o envio e a leitura de mensagens de forma assíncrona, semelhante ao funcionamento do correio eletrónico.

O projeto proposto tem como referência distribuições Linux que suportam listas estendidas de controlo de acesso e o serviço de conversação deverá suportar a gestão de utilizadores, permitindo a adição e remoção de utilizadores já existentes no sistema operativo. Além disso, o serviço deverá também suportar a criação de grupos privados de conversação, oferecendo mecanismos para a criação e remoção de grupos, bem como a gestão dos seus membros.

A segurança é um aspeto central deste projeto, com especial enfoque na proteção da confidencialidade e integridade das mensagens trocadas entre os utilizadores. A disponibilidade do serviço é também uma preocupação importante, garantindo que os utilizadores possam comunicar-se de forma eficaz e sem interrupções.

Este relatório irá detalhar as várias etapas do desenvolvimento deste serviço de conversação, desde a concepção inicial até à implementação final, com foco na segurança, funcionalidade e usabilidade dados estes serem pontos fulcrais para a boa implementação do que é pretendido. Referindo também todos os pontos que não correram como esperado.

1.1 Arquitetura do sistema

A arquitetura do nosso projeto está exposta no seguinte diagrama que retrata todos os programas e estruturas para o funcionamento da aplicação

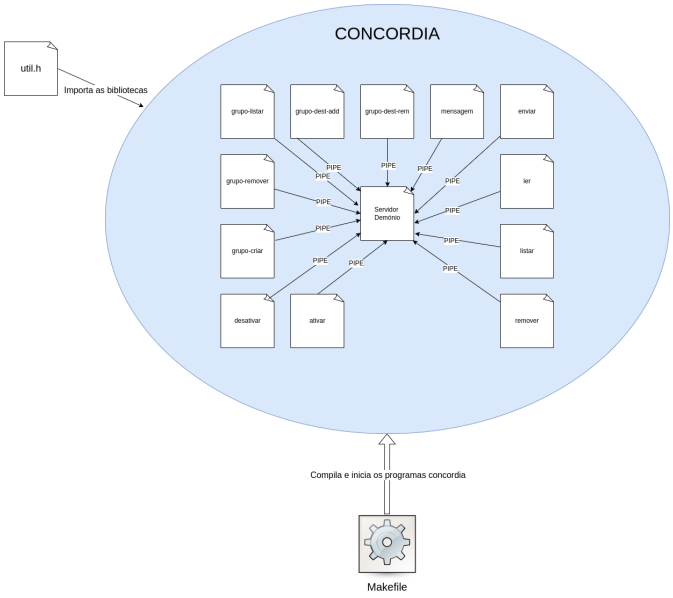

Figura 1: Diagrama que retrata a arquitetura do sistema

Primeiramente o corremos o programa *concordia-demonio* que atua como um servidor local para gerir as mensagens, utilizadores e grupos.

2 Arquitetura funcional

2.1 Programa concordia-demonio.c

Começa por criar um *pipe com nome* caso não exista para posteriormente poder receber mensagens dos diversos utilizadores. De maneira a poder atender os vários utilizadores implementamos processos que executam com as permissões do mesmo utilizador a ser atendido.

Uma vez que os *pipes com nome* são uniderionais, é necessário criar outro de maneira a que o demónio possa responder aos pedidos. Para este efeito, é criado um pipe com o nome do utilizador para que o servidor possa responder e mais tarde é apagado através de um *unlink()*. Para manter a segurança entre a comunicação o apenas o servidor terá permissões de escrita e o utilizador de leitura.

De modo a gerir as mensagens o servidor criará as pastas necessárias e com um *chmod()* atribuir as permissões corretas. Assim, se o utilizador A e B pertencem ambais e dentro da mesma mais duas pastas, a A e a B, onde ficarão armazenadas as mensagens destiandas a cada utilizador, tendo apenas permissão os utilizadores respetivos e o servidor. Conseguimos fazer isso através de comandos como Decidimos também que o servidor teria apenas permissão de escrita e os utlizadores apenas de leitura. Após isso irá processar os comandos que lhe serão enviados através do pipe e para cada um fará o processo necessário à sua interpretação.

Quando recebe o comando concordia-ativar começa por verificar o nome do utilizador, após isso vai criar a pasta com o seu nome e aplicar as devidas permissões. Retorna uma mensagem de debug para saber que o diretório foi criado com sucesso. O estado da variável concordia_ativado, que analisa se o estado de um utilizador é ativo ou não, passa a 1. O comando concordia-desativar permite eliminar a pasta e qualquer mensagem de um utilizador.

Uma vez que se encontra ativo poderá receber outros comandos como relativos à mensagem ou ao grupo. No que toca aos diversos comandos de gestão estes só poderão ser interpretados após o recebimento do comando concordia-mensagem, que ativa o recibimento das mensagens e a indicação de que o utilizador quererá enviar uma mensagem.

Juntamente com o comando concordia-enviar, o servidor recebe o nome do destinatário, ou seja a diretoria de destino e a mensagem, estas são dividias em tokens para serem mais facilmente processadas e então na pasta é criado um ficheiro com o nome msgX.txt, onde X será um inteiro variável dependendo do número da mensgem, com o conteúdo da mensagem.

Com o comando concordia-ler, vem o número da mensagem correspondente à que o utilizador quer ler, são divididos em tokens para uma interpretação mais fácil e então através da função system é corrido o comando 'cat' com o nome do ficheiro correspondente, o que permite ao utilizador visualizar o conteúdo da mensagem.

O comando concordia-remover tem a mesma lógica que o anterior com a difereça que o comando passado como argumento à funão system é 'rm' juntamente com o nome do ficheiro em questão, que corresponde à mensagem.

O comando concordia-listar vai apenas aplicar o comando 'ls -la', juntamente com o nome do diretório do utilizador como argumento único da funcção system. Vai fazer com que sejam listados todos os ficheiros presentes na pasta.

O comando concordia-grupo-criar é recebido juntamente com o nome que o grupo criado irá tomar. Após a divisão por tokens para melhor interpretação e através do comando 'grouppadd' passado como argumento em system, vai ser possível ser feita a criação de um novo grupo. Do mesmo modo estão implemntados outros dois comandos. Num sentido inverso o comando concordia-grupo-remover, elimina esse grupo através da utilização do comando 'groupdel'. O comando concordia-grupo-listar apresenta todos os utilizadores do grupo através do comando 'getent group' e do nome do grupo, passados como argumentos na função system.

Por fim os comandos concordia-grupo-destinatario-adicionar e concordia-grupo-destinatario-remover enviados ao servidor juntamente com o nome do utilizador e do grupo em questão. Após serem divididos em tokens, o nome e o grupo são passados como argumento da funcção system juntamente com os respetivos comandos que permitem adicionar um utilizador a um grupo, 'usermod -a -G' e que remover um utilizador de um grupo, 'deluser';

2.2 Programa concordia-ativar.c

Este ficheiro contém o código que permite enviar o comando "concordia-ativar"através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que então dará início à funcionalidade de adicionar o utilizador ou serviço. Se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.3 Programa concordia-desativar.c

Este ficheiro contém o código que permite enviar o comando "concordia-desativar"através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que pelo processo inverso fará a remoção do utilizador do seu serviço. Da mesma forma se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.4 Programa concordia-mensagem.c

Este ficheiro contém o código que permite enviar o comando "concordia-mens"através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que fará com que sejam aceites a partir de então os comandos necessário à leitura, listamento, leitura ou resposta de mensagens. Da mesma forma se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.5 Programa concordia-enviar.c

Este ficheiro contém o código que permite enviar o comando "concordia-enviar"bem como os argumentos destinatário e conteúdo da mensagem, através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que fará com que seja enviado e escrito na pasta do destinatário um ficheiro que contém a mensagem pretendida pelo remetente. Da mesma forma se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.6 Programa concordia-ler.c

Este ficheiro contém o código que permite enviar o comando "concordia-enviar"bem como os argumentos que simboliza o número da mensagem a ser lida, através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que fará com Da mesma forma se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.7 Programa concordia-remover.c

Este ficheiro contém o código que permite enviar o comando "concordia-remover"através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que fará com que todas as mensagens desse utilizador sejam removidas da sua diretoria. Da mesma forma se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.8 Programa concordia-grupo-criar.c

Este ficheiro contém o código que permite enviar o comando "concordia-grupo-criar"bem como o outro argumentos que simboliza o nome do grupo, através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que fará com que um grupo com o respetivo nome seja criado. Da mesma forma se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.9 Programa concordia-grupo-remover.c

Este ficheiro contém o código que permite enviar o comando "concordia-grupo-remover"bem como o outro argumento que simboliza o nome do grupo, através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que fará com que um grupo com o respetivo nome seja removido do sistema. Da mesma forma se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.10 Programa concordia-grupo-listar.c

Este ficheiro contém o código que permite enviar o comando "concordia-grupo-listar"bem como o outro argumento que simboliza o nome do grupo, através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que fará com que um grupo com o respetivo nome seja removido do sistema. Da mesma forma se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.11 Programa concordia-grupo-destinatario-adicionar.c

Este ficheiro contém o código que permite enviar o comando "concordia-grupo-destinatario-adicionar"bem como os argumentos que simbolizam o nome do grupo e o utilizador em questão, através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que fará com que o utlizador com o respetivo nome seja adicionado ao respetivo grupo. Da mesma forma se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.12 Programa concordia-grupo-destinatario-remover.c

Este ficheiro contém o código que permite enviar o comando "concordia-grupo-destinatario-remover"bem como os argumentos que simbolizam o nome do grupo e o utilizador em questão, através de um pipe. O pipe será analisado através da sua abertura e leitura pelo concordia-demonio que fará com que o utlizador com o respetivo nome seja removido do respetivo grupo. Da mesma forma se a abertura ou escrita do pipe forem mal sucedidas serão devolvidas mensagens de erro que o comprovem.

2.13 util.h

O arquivo util.h, presente na diretoria include, contém todos as chamadas nessecárias às bibliotecas que permitem a utilização das diversas ferramentas da linguagem C.

3 Reflexões

3.1 Segurança do serviço

Relativamente à segurança e à confidencialidade do serviço, não encontra uma implementação direta que faça com que seja assegurada no decorrer de todo do programa. Algo que consideraríamos fazer numa nova implemntação do trabalho seria começar por fazer uma melhor gestão de utilizadores e grupos, tendo em conta aquelas que seriam as permissões de leitura, escrita e acesso dos mesmos. Depois de todos os dados guardados numa estrutura adequada, a verificação seria feita pelo servidor demonio antes de qualquer ação realizada.

A implementação de permissões apenas está a ser realizada aquando da criação da pasta relativa a cada utilizador com os acessos à mesma e as permissões de leitura e escrita tendo o servidor apenas permissão de escrita e os utlizadores apenas de leitura.

Para além disso as permissões são geridas através da utilização do systemctl() após a execução dos respetivos comandos.

3.2 Modularidade e Encapsulamento

Apesar de uma divisão em vários executáveis para o tratamento dos diferentes comandos possíveis do utilizador estes acabam por ser algo redundantes e grande parte do código fulcral para o funcionamento do programa encontra-se no servidor. Acreditamos que a utilização de um módulo cliente que fizesse uma gestão mais organizada dos pedidos ao servidor seria uma melhor forma de implementar de modo a uma melhor organização e até mesmo perceção e leitura do código.

Relativamente ao encapsulamento este não se encontra bem implementado visto haver bastante reutilização de código em diversos módulos. Algo que poderíamos ter feito para melhorar seria criar módulos auxiliares que diminuissem a redundância de certas partes do serviço.

3.3 Reflexões gerais

Podemos destacar a falta de troca de mensagem entre grupos. Como referido em cima uma estrutura de dados que permitisse armazenar as informações necessárias a uma distribuição simples e prática das mensagens entre os grupos permitiria concluir este, que era um dos objetivos iniciais, com sucesso.

Para além disso a implementação completa da troca de mensagens e informação cliente-servidor deveria ter sido um ponto implementado na nossa arquitetura, visto que apenas faz sentido haver um servidor a atender os pedidos se conseguir dar resposta a quem os faz.

Em suma, consideramos que arquitetura funcional está um pouco aquém da que planeamos implementar, contudo conseguimos consolidar os conhecimentos necessários para a realização do projeto.

4 Conclusão

Com a realização deste trabalho foram aprofundados vários conceitos de Segirança de Sistemas Informáticos, tais como: Controlo de Acesso ao Sistema de Ficheiros em Linux, Infra-estrutura de serviço em Linux e outros temas abordados nas aulas da unidade curricular.

O desenvolvimento deste trabalho permitiu não só expandir os conhecimentos acima referidos como fazer nos questionar constantemente acerca da implmentação e da forma como esta era realizada. Sem dúvida que essa foi uma das maiores dificuldades ao longo do trabalho.

Consideramos que poderíamos e conseguíramos obter um projeto mais rico e que correspondesse quase na sua totalidade ao que nos foi proposto, contudo e devido a uma má gestão do tempo de trabalho acabou por não se conseguir realizar. Isso também se reflete na entrega do trabalho prático- A demora na tomar de decisões sobre o que pretendíamos e como pretendíamos fazer para o funcionamento do programa acabou por afetar os resultados finais.

Apesar de tudo demos o nosso melhor para ir de acordo com aquela que eram as diretrizes do trabalho, fomos capazes de superar as dificuldades que nos foram aparecendo, discutimos sobre qual seria a melhoropção a tomar e implementamos bastantes funcionalidades.

Em conclusão, para trabalhos futuros pretendemos melhorar em diversos aspetos mas não achamos que tudo seja dado como perdido e consíderamos que o que desenvolvemos tem o seu valor, esperando um resultado razoável.