Universidade do Minho

Departamento de Informática

licenciatura em engenharia informática

Segurança de Sistemas Informáticos

TP2 - Grupo 39

Gonçalo Brandão

Mariana Morais

Maya Gomes

Gonçalo Brandão A95128
Mariana Filipa Morais Gonçalves A100662
Maya GomesA100822

Maio de 2024

Conteúdo

1 Introdução

Este sistema apresenta uma solução para comunicação entre clientes através de mensagens assíncronas num ambiente Unix. O objetivo principal é fornecer um meio eficiente e robusto para a troca de mensagens entre clientes, permitindo que estes possam enviar e receber mensagens de forma assíncrona, ou seja, através de um servidor.

A arquitetura do sistema é baseada no uso de pipes com nome (fifos) para estabelecer canais de comunicação entre os clientes e o servidor. Cada cliente tem um fifo próprio para receber mensagens do servidor, enquanto o servidor tem um fifo central para receber os comandos dos clientes.

Este projeto tem por objetivo a aprendizagem de métodos de segurança alterando permissões de acesso de utilizadores e grupos.

Neste âmbito, as nossas preocupações foram conceder permissões restritas a cada utilizador, garantindo assim a impossibilidade de acesso de terceiros às pastas das mensagens.

Todo o projeto divide-se em dois ficheiros: server.c e client.c. Para além disso, foi criada uma *Makefile* de forma a executar o programa e um ficheiro README.md com algumas instruções.

2 Arquitetura Funcional

2.1 Programa e Processos Envolvidos

O serviço é composto por um servidor principal e múltiplos processos clientes. O servidor principal desempenha o papel de ponto central, recebendo solicitações dos clientes e coordenando as suas respostas. Para garantir eficiência e capacidade de resposta, cada solicitação de cliente é tratada num processo separado pelo servidor, criando assim uma arquitetura de servidor concorrente. Os processos clientes são iniciados pelos clientes para interagir com o servidor. Eles podem enviar solicitações para se registarem e se desativarem; enviar, listar e ler mensagens; criar e remover grupos.

2.2 Comunicação servidor-cliente

A comunicação entre o servidor e os clientes é estabelecida por meio de pipes com nome (fifos), seguindo o modelo cliente-servidor.

O servidor principal mantém um pipe nomeado centralizado na diretoria "fifos"para receber pedidos de todos os clientes.

Cada cliente tem o seu próprio pipe nomeado na diretoria "fifos"para receber respostas do servidor.

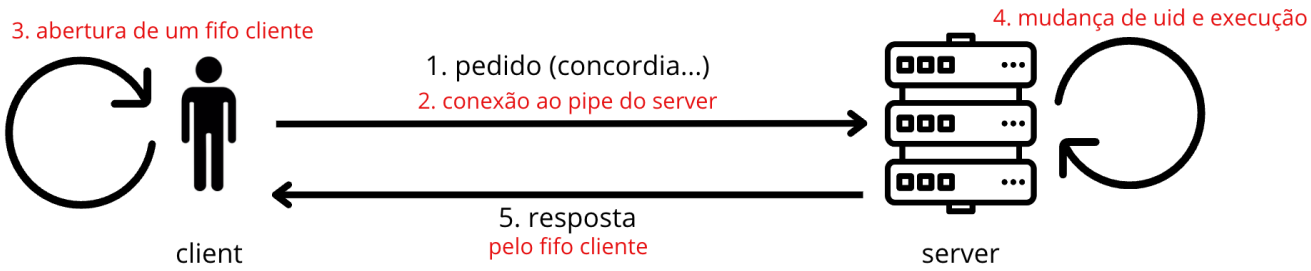

Figura 1: Comunicação servidor-cliente.

Também é importante referir que o servidor altera temporariamente de UID (assumindo o do cliente) de forma a ter as permissões do cliente em questão e poder realizar os pedidos acedendo às pastas.

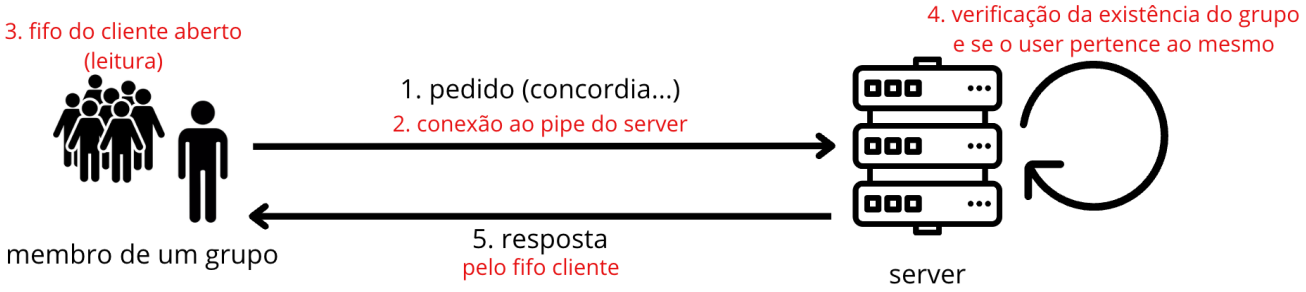

Figura 2: Comunicação servidor-membro de um grupo.

No caso da comunicação com um membro de um grupo, a mudança de UID não é realizada mas sim somente a verificação da existência do grupo e se o user pertence ao grupo. Neste caso, somente conseguimos realizar os pedidos porque o servidor esta em modo *super utilizador* (sudo). É de referir que esta solução não é a mais adequada, pois o suposto seria mudar de UID para o UID do membro do grupo que efetuou o pedido, pois esse tem permissões suficientes para realizar alterações no grupo. No entanto, via código, não conseguimos efetuar as mesmas.

2.3 Componentes de Software desenvolvidas e Dependências

O serviço é implementado como dois programas executáveis, pretendendo permitir uma interação eficiente entre o servidor e os clientes.

O programa utiliza bibliotecas padrão do C, como *stdio.h, stdlib.h, unistd.h*, entre outras, para operações de entrada/saída, gestão de processos. Além disso, o programa depende de funções específicas do sistema operativo, como *mkfifo*, *chmod*, *open*, *read* e *write*, para criar, gerir e interagir com os pipes.

3 Implementação

3.1 Detalhes dos comandos

concordia-enviar dest msg: Este comando permite enviar uma mensagem de texto para um destinatário que pode ser utilizador ou grupo. Para tal, foi criada a função *move_message_to_folder*. Numa primeira fase, o servidor verifica se o destinatário existe, ou seja, se existe uma pasta com o nome do mesmo. De seguida, ele recupera o número de mensagens recebidas e enviadas dos utilizadores (destinatário e remetente) de forma a atualizar os contadores que irão mais tarde servir para a atribuição de identificadores às mensagens. É criado um fork para dividir a escrita nos dois utilizadores. O servidor muda de UID de forma a aceder a pasta *destinatárioEntrada* e *remetenteSaida* e escrever a mensagem trocada nas mesmas.

concordia-listar-g, concordia-listar, concordia-listar -a : Este comando permite listar as novas mensagens de um utilizador ou de um grupo. Mais uma vez, o servidor altera de UID de forma a aceder a pasta de Entrada do utilizador. São recuperadas as mensagens do mesmo sendo que no caso do comando "concordia-listar"listamos as 5 mensagens mais recentes e no caso do "concordia-listar -a"listamos todas as mensagens. Para tal, foi criada a função *list_messages_in_folder*. Esta abre a pasta e ordena os ficheiros (mensagens) que estão guardados ("1.txt", "2.txt", ...) de forma decrescente, sendo os mais recentes os que possuem id e nome maior e devolve-as nesta ordem. No caso da flag **-a** estar ativa e sendo que o tamanho máximo das mensagens é 512, as mensagens são enviadas uma a uma sendo que o cliente fica a espera de uma mensagem que contém apenas "FIM"indicando que todas foram devolvidas. É de notar que no caso do grupo, só são listadas as mesagens se o utilizador pertence ao grupo em questão.

concordia-ler mid, concordia-ler-g grupo mid: Este comando permite ler o conteúdo de uma mensagem identificada pelo seu índice numérico. Mais uma vez, o servidor acede a pasta de entrada do utilizador com uma implementação semelhante ao comando anterior. Desta vez, também é devolvido o parâmetro "conteúdo"da mensagem que anteriormente era omitido.

concordia-responder mid msg: **Este comando não foi implementado** por falta de tempo e por ser mais complexo na nossa perspetiva dando-se prioridade aos restantes. Seguem-se algumas ideias que poderíamos ter aplicado. A nossa solução seria a implementação de uma árvore de forma a permitir resposta de respostas de mensagens e várias respostas para a mesma mensagem. Cada mensagem teria uma pasta própria, um ficheiro com o conteúdo da mensagem e as restantes subpastas que seriam as respostas. Na base do utilizador, teríamos um ficheiro que revelaria a localização de cada mensagem.

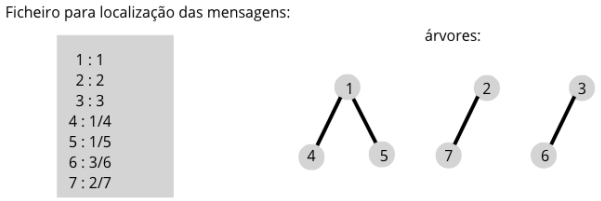

Figura 3: Esquema de uma pseudo solução.

concordia-remover mid: Este comando permite apagar uma mensagem recebida indicada pelo seu índice numérico. Para tal, acedemos a pasta Entrada do utilizador e procura se o ficheiro "mid.txt"que corresponde ao ficheiro que contém a mensagem. Uma vez encontrado, este é removido da pasta.

concordia-grupo-criar nome: Este comando permite criar um grupo de utilizadores do serviço. Para tal, utilizamos: "dseditgroup -o create -r "%s" -i %d %s", group_name, gid, group_name ". O utilizador que efetuou o comando é adicionado ao grupo com "dseditgroup -o edit -a %s -t user %s", username, group_name ". De seguida, são criadas as pastas do grupo seguindo a mesma formatação que nos utilizadores: grupo/ ; grupo/grupoEntrada; grupo/grupoSaida. De seguida, com o **-chown** colocamos o grupo como grupo proprietário das pastas e o utilizador que criou o grupo como administrador do mesmo. Finalmente, com o **-chmod** alteramos as permissões do administrador do grupo e do grupo permitindo apenas a estes a leitura, escrita e execução.

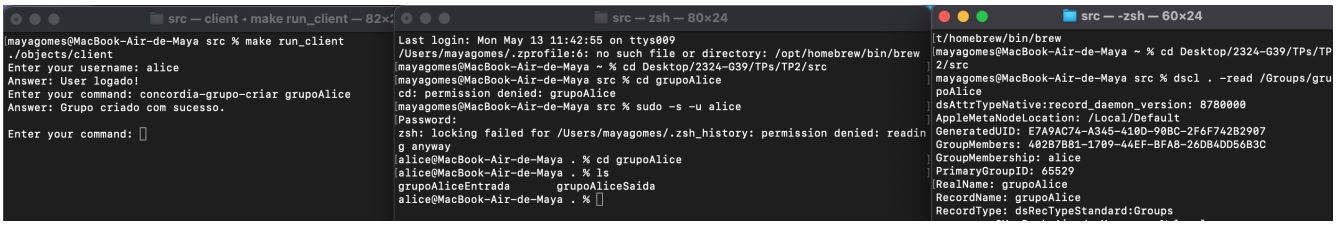

Figura 4: Criação de grupo e permissões.

concordia-grupo-remover nome: Este comando permite a remoção de um grupo. Esta remoção somente pode ser efetuada por membros do grupo, sendo esta verificação efetuada. Para tal, são removidas as pastas do grupo e é utlizado o comando "dseditgroup -o delete %s", group_name"para remover o grupo do sistema.

concordia-grupo-listar nome: Este comando permite listar os utilizadores de um grupo. Para tal, utilizamos a função predefinida *getgrnam(group_name)* de forma a recuperar os membros do grupo.

concordia-grupo-destinario-adicionar nome_grupo user: Este comando permite adicionar um utilizador a um grupo. Para tal, fazemos o mesmo que no comando "concordia-grupo-criar"quando adicionamos o utilizador que cria o grupo ao grupo, ou seja, utilizamos o "dseditgroup -o edit -a %s -t user %s", username, group_name".

concordia-grupo-destinario-remover nome_grupo user: Este comando permite remover um utilizador a um grupo. Para tal, utilizamos o comando "dseditgroup -o edit -d %s -t user %s", username, group_name".

concordia-ativar: Este comando permite a criação de utilizadores. Para tal são efetuados uma série de comandos nomeadamente:

```
sprintf(comandos[0], "%s%s", "dscl . -create /Users/", user_name);
sprintf(comandos[1], "%s%s%s%d", "dscl . -create /Users/", user_name, " UniqueID ", uid);
sprintf(comandos[2], "%s%s%s", "dscl . -create /Users/", user_name, " UserShell /bin/bash");
sprintf(comandos[3], "%s%s%s%d", "dscl . -create /Users/", user_name, " PrimaryGroupID ", uid);
sprintf(comandos[4], "%s%s%s%s", "dscl . -create /Users/", user_name, " NFSHomeDirectory ./", user_name);
sprintf(comandos[5], "%s%s%s", "dscl . passwd /Users/", user_name, " 1234");
```

Figura 5: Comandos para a criação de um utilizador.

De seguida, são criadas as respetivas pastas do novo utilizador (pasta principal, de entrada e de saída). Através do **-chown** alteramos as permissões definindo o novo utilizador como proprietário das pastas criadas. Finalmente, com o **-chmod** damos as permissões de escrever, ler e executar ao utilizador.

Desta forma, um utilizador possui uma pasta com o seu nome e dentro da mesma uma pasta de Entrada e uma pasta de Saída de forma a serem guardadas as mensagens recebidas e enviadas respetivamente. O utilizador é proprietário destas três pastas sendo o único com permissões para alterá-las. Daí a necessidade do servidor mudar de UID quando pretende escrever ou ler as mesmas, como referido anteriormente.

concordia-desativar: Este comando permite a remoção de utilizadores. Para tal é utilizado o seguinte comando: "dscl . -delete /Users/%s", user_name". Também são removidas as pastas do mesmo e por consequente as suas mensagens.

4 Decisões no domínio da segurança do serviço

Nesta secção, as decisões de segurança abrangem principalmente as permissões e proprietários definidos para cada objeto do sistema de arquivos, bem como as medidas adotadas para garantir a integridade e a segurança dos processos necessários à execução do serviço de mensagens.
As principais decisões incluem:

- **Permissões dos Objetos do Sistema de Arquivos:** Foram definidas permissões específicas para os diretórios e arquivos utilizados pelo serviço de mensagens, como as pastas para cada utilizador. Isso inclui permissões de leitura, escrita e execução, de forma a restringir o acesso não autorizado e proteger os dados armazenados.
- **Proprietários dos Objetos do Sistema de Arquivos:** Cada objeto do sistema de arquivos, como as diretorias de cada utilizador, possuem um proprietário designado. Essa atribuição de proprietários ajuda a controlar quem tem autoridade para aceder e modificar os objetos, contribuindo para a segurança do sistema.
- **Medidas de Controlo de Acesso:** Além das permissões e proprietários, foram implementadas medidas de controlo de acesso para restringir o acesso não autorizado aos recursos do utilizador. Isso envolve a validação das credenciais dos clientes (apenas verificação da existência do utilizador), a atribuição de permissões com base nos privilégios concedidos e a execução dos processos com os privilégios mínimos necessários.
- **Isolamento de Processos:** Os processos envolvidos na operação do serviço de mensagens são isolados uns dos outros para evitar interferências indesejadas. Cada processo executa apenas um pedido de um cliente e também assume as permissões de apenas um utilizador.
- **Integridade das mensagens:**

  De forma a garantir a integridade das mensagens decidimos escrever nas pastas em binário, usando o *fwrite*. De seguida, ao devolver as mensagens efetuamos novamente a conversão.

Estas decisões foram tomadas com o objetivo de fortalecer a segurança do serviço de mensagens, protegendo os dados dos clientes e garantindo o funcionamento confiável do sistema.

5 Reflexão sobre o projeto

5.1 Modularidade e Encapsulamento

Neste trabalho não tivemos muito em conta a modularidade e encapsulamento do nosso código, pelo que, poderíamos adaptá-lo. Nomeadamente, definir funções especificas para cada tipo e pedido (concordia) e divisão destas por vários ficheiros, como por exemplo, funções relacionadas com os grupos ficavam num ficheiro grupo.c; funções relacionadas com utilizadores ficavam num ficheiro utilizador.c e funções relacionadas com acesso a ficheiros ficavam num ficheiro files.c.

5.2 Testes

De modo a testar as várias funcionalidades do programa optamos primeiramente por executar as funções fora do programa (ficheiros *scratch*) para ser mais fácil a deteção de erros. Posteriormente, analisávamos a funcionalidade dentro do programa.

5.3 Aspetos de valorização (bonificação)

Sobre estas, não conseguimos a implementação de nenhuma funcionalidade de valorização. Uma ideia para a implementação do código de autenticação OTP (*oauthtool3*) seria alterar a lógica dos utilizadores adicionando um ficheiro na pasta do utilizador que continha o email de modo a que, na altura do início de sessão, o servidor pudesse criar um código, enviá-lo para o email e, assim, o utilizador conseguiria iniciar sessão.

5.4 Funcionalidades implementadas

Sobre as funcionalidades implementadas, temos a noção que estão maioritariamente realizadas com sucesso tendo em conta que as permissões para cada utilizador estão a ser respeitadas. Contudo, as mesmas poderiam ser aperfeiçoadas. Para além da questão da modularidade e encapsulamento referida anteriormente, poderíamos possuir mais verificações de forma a garantir quem efetuou *login*.

Quanto as funcionalidades de grupo, apesar de conseguirmos o acesso as pastas alterando o UID via terminal, não conseguimos implementar isto em código, pois obtínhamos sempre o erro de "*Permision denied*" que não conseguimos resolver por falta de tempo. Foi então escolhido usar **-sudo** e somente verificar a existência do grupo. Este aspeto teria de ser melhorado. Apesar disso, de modo a minimizar o risco verificamos se o utilizador esta inserido no grupo em questão para qualquer comando associado.

É também importante referir que o servidor é executado com **-sudo**, pois necessitamos desta permissão para alterar de UID, isto é, se o servidor muda de o UID para um cliente (cliente1, por exemplo) e de seguida seja necessário mudar de UID para o do cliente2, necessitamos de permissões que o cliente1 não tem ao contrario das que tinha o servidor.

6 Conclusão

Neste projeto, desenvolvemos um sistema de comunicação cliente-servidor utilizando pipes com nome (fifos) em C. O servidor é responsável por receber e processar as solicitações dos clientes, enquanto os clientes enviam solicitações e recebem respostas do servidor.

Decidimos adotar a abordagem de utilizar fifos para a comunicação entre os clientes e o servidor devido à sua simplicidade de implementação e facilidade de uso em ambientes Unix. Isto permitiu-nos criar um sistema eficiente de troca de mensagens entre os processos.

Ao longo do desenvolvimento, enfrentamos alguns desafios, como garantir a correta manipulação dos fifos, lidar com múltiplos clientes e implementar funcionalidades específicas, como envio, receção e listagem de mensagens.

Implementamos funcionalidades robustas para lidar com diferentes tipos de solicitações como por exemplo a verificação do membro no grupo e a mudança de utilizador para ter acesso às pastas.

Uma limitação potencial do sistema é a falta de autenticação segura dos clientes, uma vez que o *login* é baseado apenas no nome do cliente e na existência de uma diretoria correspondente. Para superar isso, poderíamos implementar um sistema de autenticação mais robusto, como o uso de senhas ou chaves de acesso.

Um ponto positivo a considerar é a escalabilidade do sistema para lidar com um grande número de clientes simultâneos. Para isso, exploramos a técnica de multiprocessamento (forks) para aumentar a capacidade de processamento do servidor e atender a mais clientes de forma eficiente.

Em resumo, embora haja espaço para melhorias futuras para tornar o projeto mais robusto e seguro, alcançámos o nosso objetivo de criar um sistema funcional tendo cumprido com grande parte do enunciado.