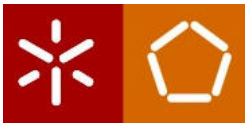


Universidade do Minho
Departamento de Informática

Segurança de Sistemas Informáticos

Trabalho Prático 2
Concórdia

Grupo 3

Gonçalo Costa - A100824

Marta Rodrigues - A100743

José Correia - A100610

Índice

Índice de imagens

1. Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Sistemas de Segurança Informática, onde foi proposto o desenvolvimento de um serviço de conversação entre utilizadores locais de um sistema Linux.

A implementação teve em conta aspetos funcionais, para as quais usamos diretorias de ficheiros para o armazenamento local de mensagens diretas ou provenientes de grupos privados, juntamente com a exploração estruturas algorítmicas para atingir vários desafios propostos. Assim como aspetos de segurança, onde aplicamos conceitos como permissões, noções de grupo e utilizadores, bem como processos e controlo de acesso a quem e não pertence ao serviço. Procuramos também a exploração de gestão de serviços do sistema com a implementação de um servidor em "demónio". Este serviço "Concórdia" procura proteger ao máximo os seus utilizadores e as suas preciosas mensagens.

Ao longo deste relatório iremos detalhar todas as decisões e implementações feitas durante todo o desenvolvimento. Será feita a explicação sobre a arquitetura funcional, medidas para a segurança do sistema e finalmente uma reflexão crítica sobre algumas medidas aplicadas.

2. Arquitetura Funcional

Fundamentalmente, o serviço "Concórdia" deverá suportar o envio de mensagens, com tamanho máximo de 512 caracteres, entre os utilizadores do sistema e a leitura dessas mesmas mensagens pelos respetivos destinatários. Necessita também de gerir os utilizadores ativados dentro do sistema e permitir que estes comuniquem entre si de forma assíncrona, pelo qual compreendemos a possibilidade de enviar mensagens a utilizadores apenas ativados mas que não tivessem necessariamente "ligados" no serviço. Adicionalmente, deve suportar a noção de grupos de conversação.

2.1. Programas e processos desenvolvidos

Os programas realizados podem ser separados em dois grupos diferentes:

- Comandos para o servidor, que representam o conjunto de programas em linha de comando com pedidos para o serviço;
- Serviço Daemon, responsável por tratar de alguns pedidos feitos.

2.1.1. Comandos para o servidor

1. **`concordia-ativar`** - Adiciona o utilizador ao serviço e cria as diretorias necessárias para a receção de mensagens atribuindo permissões;

2. **`concordia-desativar`** - Ao contrário do "ativar", remove o utilizador ao serviço e remove igualmente as suas mensagens e diretorias criadas;

3. **`concordia-enviar <dest> <msg>`** - Garante que as mensagens não ultrapassam os 512 caracteres, garante que o utilizador ativou-se no sistema e coloca a mensagem numa *queue* de mensagens que posteriormente será enviada para a caixa de entrada do utilizador destino;

4. **`concordia-ler <ler|listar> <mid|[-a]>`** - Este comando agrupa dois tipos de leitura diferentes mas com funcionalidades semelhantes. O argumento *"listar"* indica que devem ser listadas todas informações sobre datas de receção, remetente e tamanho das mensagens que o utilizador ainda não leu. Adicionalmente ao *listar* pode ser invocado o argumento *-a* que fará com que todas as mensagens lidas e não lidas sejam listadas. O argumento *ler* deverá imprimir a mensagem com identificador mid, apresentando o remetente, o tamanho e o conteúdo da mensagem.

5. **`concordia-responder <mid>`** - Responde ao remetente de uma mensagem recebida previamente. Utiliza um processo semelhante ao comando *enviar*

6. **`concordia-remover <remover mid | tudo>`** - A utilização do argumento "remover <mid>" permite remover uma única mensagem consoante o seu id, já o argumento "tudo" garante a

remoção de todas as mensagens. Para poder ser removida uma mensagem necessita ser lida primeiro.

7. **`concordia-grupo-criar <nome>`** - Cria um grupo de utilizadores do serviço.
8. **`concordia-grupo-remover <nome>`** - Remove um grupo de utilizadores do serviço.
9. **`concordia-grupo-listar <nome>`** - Lista os utilizadores membros de um grupo.
10. **`concordia-grupo-adicionar-membro <nome> <uid>`** - Adiciona um utilizador a um grupo.
11. **`concordia-grupo-remover-membro <nome> <uid>`** - Remove um utilizador de um grupo.

Os últimos 3 comandos apenas são corretamente executados quando é o utilizador criador do grupo a invoca-los.

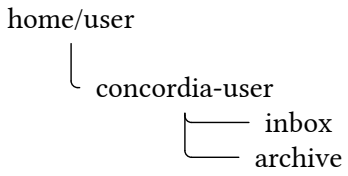
2.1.2. Serviço Daemon

Executa como um servidor e está encarregue de responder os pedidos executados pelo utilizador local que necessitam de chegar aos restantes utilizadores, ou seja, comandos como o ler e remover mensagens não passam pelo o "daemon", esses são feitos "diretamente" pelo utilizador. Também garante, sempre que necessitar se o utilizador está ativado no sistema no inicio de um comando. As estruturas de dados e mecanismos que requer serão explicados de seguida.

Desenvolvemos adicionalmente um ficheiro de instalação, o *install.sh*, que funciona de forma similar a uma Makefile (sendo que esta mesma também foi criada) e trata de colocar os executáveis criados disponíveis para todos os utilizadores visto que estão na pasta */usr/bin*. Apenas o primeiro utilizador a "ativar" o Concórdia necessita de compilar este e deixa de precisar correr o programa `concordia-ativar`, já os restantes a seguir podem simplesmente fazer a ativação sem precisar instalar o ficheiro.

2.2. Estruturas de Dados e Formatos

O **armazenamento de mensagens** trocadas entre os utilizadores foi feito numa diretoria chamada "concordia-<nome do utilizador>", esta possui 2 diretorias separadas, uma "inbox", que guarda todas as mensagens por ler, e a "archive" encarregue de armazenar todas aquelas que já foram lidas. Ao abrir uma mensagem essa passará da pasta "inbox" para a "archive". Devido aos perigos que poderia ter a permissão da manipulação das mensagens por parte de outros se fossemos altera-la internamente para registar como lida, consideramos esta abordagem a mais simples e eficaz.



As **mensagens** são sempre criadas pelo "Daemon", que as instância como uma *struct* que possui um *mid*, obtido através de uma variável global, o que pode de facto ser perigoso se múltiplos utilizadores criarem mensagens ao mesmo tempo. São constituídas, igualmente por o username do destinatário e remetente, o tamanho da mensagem em número de caracteres e o conteúdo da mensagem. No instante que forem enviadas para o seu destino, estas encontrarão-se nas diretorias com o nome <mid>.txt e o conteúdo desses ficheiros terá o nome do remetente, o tamanho da mensagem e o conteúdo. A data de receção é determinada consoante a data de modificação do ficheiro.

Para que o "daemon" saiba que utilizadores estão ativos, foi criada uma ***hashtable*** de utilizadores. As chaves para os seus elementos são o username do utilizador que ativou, e o seu conteúdo é apenas o username e o número de mensagens que possui. Apesar de termos escolhido esta arquitetura, não consideramos que o número de mensagens nela seja extremamente útil. Foram escolhidas a utilização das *hashtables*, mais especificamente de "closed addressing", visto que são mais eficientes na pesquisa e evitam os casos de colisões.

Ainda dentro do "daemon", foi importante a implementação de uma ***queue*** de mensagens, que está encarregue de as armazenar temporariamente. O porquê da sua implementação será aprofundado no capitulo da segurança.

O serviço devia também acomodar a noção de **grupos privados** de conversação, permitindo, que um utilizador pudesse enviar a mesma mensagem para múltiplos outros. Após várias tentativas de alcançar este resultado com diversos métodos, nomeadamente permissões ou criação de grupos no sistema Linux, decidimos criar uma pasta. Esta pasta deveria ser apenas acedida pelo "daemon" e é constituída por ficheiros cujo nome seria o nome do grupo. Esses ficheiros por sua vez deveriam conter o nome do utilizador criador do grupo, o número de membros dentro do grupo e os seus respetivos nomes. Tornou-se importante armazenar o nome do criador do grupo pois só este pode adicionar e remover membros aos seus grupos e, caso entendam, remover os mesmos do serviço. Essas manipulações seriam feitas no ficheiro desse grupo.

2.3. Mecanismos de Comunicação

A **comunicação** entre os diversos programas e serviço é feita através de *FIFOs*, pipes com nome, cujo nome é o *concordia*, pois permite concorrência nos instantes em que múltiplos utilizadores pretendem usar o serviço, garantindo assim a independência de processos. Estes *pipes* permitem enviar mensagens de volta para o cliente e visto que pretendíamos implementar o "daemon" realmente a nível de "demónio", seria beneficiário para o utilizador estar a par do que se passa no servidor, por exemplo, erros, sucessos, entre outros. As mensagens transportadas dentro dos *FIFOs* é constituída pelo comando inserido, o username do utilizador que chamou um programa e os argumentos que necessita.

2.4. Arquitetura e Funcionamento

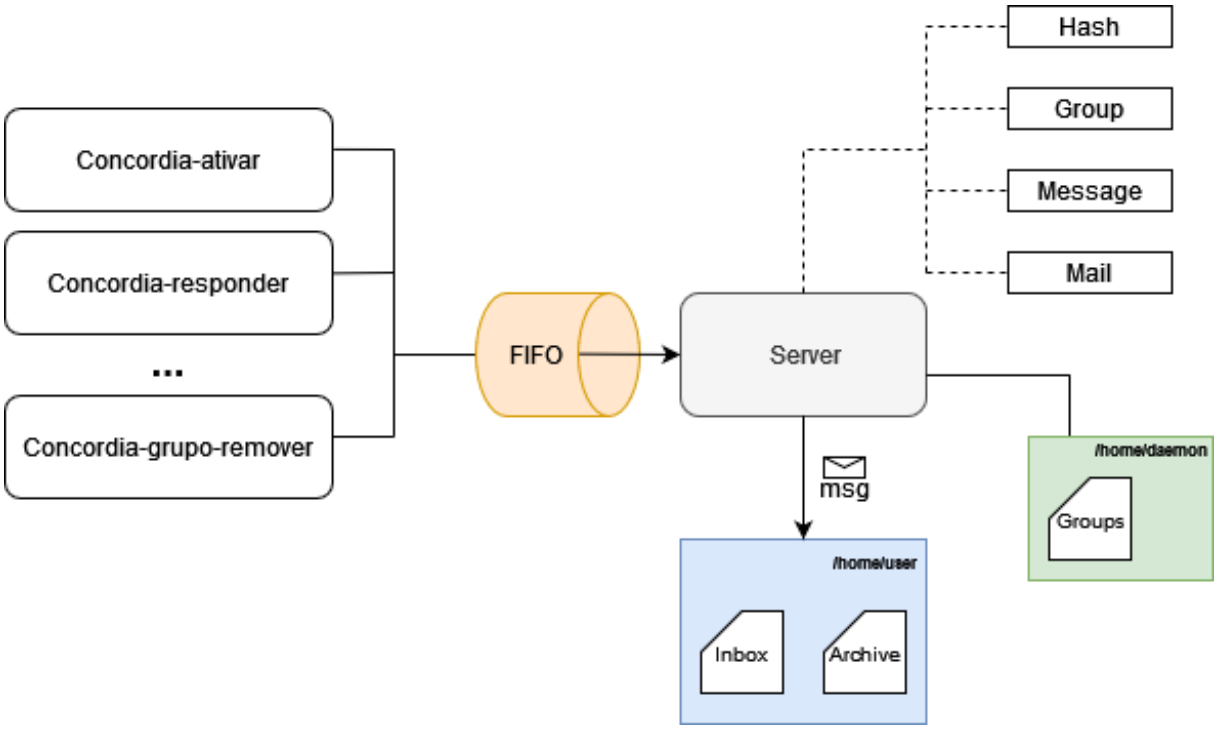


Figura 1: Desenho da arquitura do serviço

3. Arquitetura de Segurança

O nosso sistema tem como preocupação principal a proteção da confidencialidade e integridade das mensagens trocadas entre os usuários do sistema.

3.1. Detalhamento das decisões de segurança

Precedentemente à implementação da aplicação foram discutidas as várias medidas de segurança que o sistema precisaria de suportar. Tendo como objetivo criar um sistema descentralizado, isto é, utilizar o mínimo de comandos *super-user* quanto possível.

Em primeiro lugar, precisamos de distinguir os usuários que pertencem a este serviço dos que não pertencem, perante essa medida criamos o grupo **concordia**.

Decidimos também que só as pessoas que pertencem ao grupo **concordia** é que podem utilizar o *FIFO* para leitura e escrita, assim pessoas fora do sistema não conseguem ler o conteúdo do *FIFO*. Idealmente, deveria existir um *FIFO* para cada usuário, mas devido a razões temporais acabamos por não implementar isso.

No que toca à confidencialidade e integridade das mensagens, decidimos que os utilizadores do sistema apenas poderiam ver as suas mensagem e o servidor daemon, sim, teria permissões de escrita.

Como referido anteriormente, os grupos do sistema são controlados exclusivamente pelo servidor, este possui permissões `rwx` sobre a pasta grupos.

3.2. Permissões de processos

Antes de tudo, precisamos de instalar o nosso serviço numa máquina local. Começamos por executar o nosso *script* de instalação com permissões **sudo**. Este vai ser o único local onde o proprietário da máquina local precisará de permissões **root.** Este *script*, começa por verificar que o usuário que o quer executar é de facto a root, para que de seguida possa proceder à criação e armazenamento dos executáveis na pasta `/usr/bin`. De seguida executa os seguintes comandos:

```
# Criar grupo concordia, caso este ainda não exista
  groupadd -f concordia
# Atribuir setuid para os executáveis concordia-ativar e concordia-desativar
  chmod u+s $OUT_DIR/concordia-ativar
  chmod u+s $OUT_DIR/concordia-desativar
# Criar FIFO e atribuir as permissões de rw aos elementos do grupo concordia
  if [ ! -p "/tmp/concordia" ]; then
    mkfifo /tmp/concordia
  fi
  chmod 660 /tmp/concordia
  chown :concordia /tmp/concordia
```

Após dadas estas permissões colocamos o nosso servidor daemon a correr em background, isto pode ser feito com os comandos:

```
sudo systemcl start daemon.service
sudo systemcl status daemon.service
```

Ou simplesmente correr o executável daemon.

Com o servidor à escuta, aguardamos que um utilizador local execute o comando **`concordia-ativar`**. Quando um utilizador corre este comando, é executado o processo para adicionar o utilizador ao grupo concordia, daí a necessidade de **setuid** neste ficheiro.

```
pid_t pid = fork();
if (pid == 0) {
    execlp("usermod", "usermod", "-aG", "concordia", pw->pw_name, NULL);
    perror("execl");
} else if (pid < 0) {
    perror("fork");
} else {
    wait(NULL);
}
```

Após pertencer ao grupo são criadas também as pastas *concordia-username/inbox* e *concordia-/archive* na home do usuário. E recorrendo ao uso de ACLs definimos que o servidor tem permissões `rwx` sobre estas pastas e que o utilizador tem apenas `r-x` sobre a pasta *concordia-/inbox* e `rwx` sobre a pasta *concordia-/archive*. Por fim comunica ao servidor uma mensagem "concordia-ativar" para que o servidor o reconheça como um novo usuário na nossa base de dados.

Quando o utilizador normal executa o comando para **`listar mensagens`**, este processo agrega todas as mensagens da pasta *Inbox* e apresenta-as ao utilizador, no caso de a opção [-a] estar ativa apresenta também a data de receção, o remetente e o tamanho da mensagem.

Já no processo de ler a mensagem, a mensagem é apresentada no *stdout* e posteriormente movida para a pasta archive.

Quanto ao processo de responder a mensagens, o utilizador vai fazer um pedido ao servidor. O Servidor recebe o **username** do destino e da mensagem a enviar, este escreve a mensagem na pasta *Inbox* do destino.

Finalmente a funcionalidade dos grupos, para que um utilizador externo não tivesse forma de ver a que grupos os outros utiliazdores pertencem, optamos por armazenar a informação sobre os grupos numa pasta onde só o servidor tem permissões de escrita e leitura. A execução do comando `concordia-grupo-criar` faz um pedido ao servidor para criar um novo ficheiro de grupo. Quem cria o grupo torna-se administrador deste, querendo com isto dizer, que o servidor aceita apenas os pedidos deste administrador quando se trata de adicionar novos membros.

Quanto aos comandos de enviar mensagens para grupos, o servidor itera sobre os elementos pertencentes ao grupo e executa um `concordia-enviar` individual para cada um deles.

3.3. Demonstração

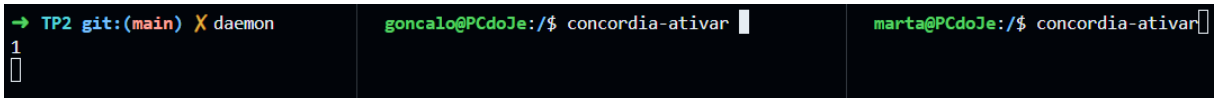


Figura 2: Exemplo de ativação de utilizadores no sistema

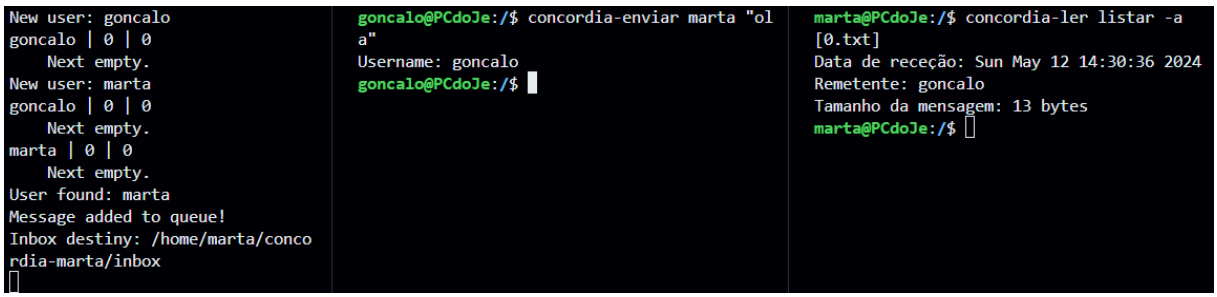


Figura 3: Exemplo enviar mensagem

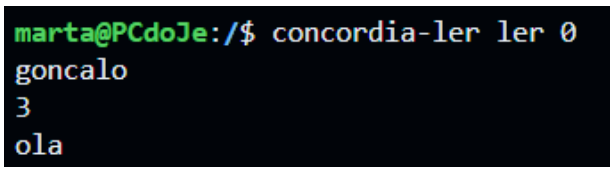


Figura 4: Exemplo de ler mensagem

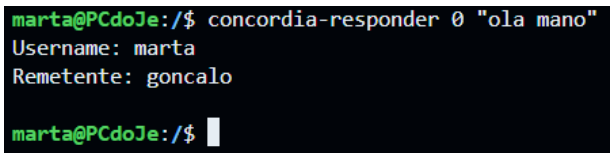


Figura 5: Exemplo de responder a mensagem

4. Reflexão Crítica

Consoante as decisões feitas, esta secção focar-se-á em aprofundar as decisões tomadas ao longo do desenvolvimento de todo este projeto.

4.1. Aspetos funcionais e de segurança

A equipa de trabalho compreendeu a questão do serviço sustentar uma comunicação assíncrona com os utilizadores a partir do momento que se "desativam" definitivamente do serviço, portanto deixam de estar disponíveis para a comunicação, daí a necessidade de uma *hashtable* pertencente ao "daemon" que armazenasse os utilizadores que estavam ativos e se atualizasse consoante as ações dos mesmos. Contudo, à última da hora surgiu a dúvida sobre a definição do conceito, por isso decidimos manter a definição que compreendemos. Assim, o envio de mensagens é apenas permitido para utilizadores destino que se encontrem ativos.

Decidimos definir as informações dos grupos dentro de uma pasta para qual só o "daemon" tem acesso, pois sentimos-nos limitados em termos de tempo para encontrar outras alternativas. Porém esta abordagem permite que o servidor saiba diretamente para que utilizadores necessita de enviar as mensagens de grupo, e igualmente não necessita de aceder aos grupos para saber quem são os seus criadores. Adicionalmente, não achamos que seria vantajoso a criação dos grupos no sistema Linux.

No enunciado é mencionado seguir uma abordagem similar ao *qmail*, por isso, após um pequeno estudo de como poderíamos implementar esse aspeto, introduzimos a *queue*. Consideramos que esta traz vantagens na não-manipulação das mensagens enviadas pelos outros utilizadores, pois esta não é gerida pelos processos-filhos que agem sobre as respostas aos pedidos dos utilizadores, esta é na mesma gerida pelo "daemon" quando tiver mensagens. Contudo, não temos total certeza da boa implementação desta funcionalidade. Esta *queue* inicialmente também estaria focada a fornecer os id's às mensagens mas altera-mos esse método. Tentamos, igualmente, cumprir com regras do *qmail* como códigos reduzidos, sem *setuids* e tentar utilizar o mínimo a nível do *root*.

O plano inicial deste trabalho era a implementação de o serviço "daemon" como sendo de facto o "demónio", no entanto, apesar de considerarmos e termos feito toda a sua estrutura para o alcançar, tivemos dificuldades na implementação do *systemctl* necessário para o executar.

Para ter comunicação contínua entre o servidor e a aplicação, de modo a receber "feedback" sobre tudo o que se passa de ambos os lados do serviço, seria necessária uma conexão constante entre ambos, pois o utilizador só deve correr os comandos e só deve saber o que se passa no programa ao receber as informações vindas do servidor através do *FIFO*.

4.2. Permissões e mecanismos complementares de segurança

Complementamos os mecanismos de segurança mencionados anteriormente e as permissões, com o uso de listas de controlo de acesso (ACL), permitindo adicionar uma camada de granularidade e controlo sobre quem pode aceder e executar determinadas operações no serviço. Isto é essencial para reduzir o risco de comprometimento da integridade e da confidencialidade do serviço.

O uso da função **execl**, que exige um absolute PATH, ao contrário de **execlp** que confia na variável de ambiente PATH para localizar o executável foi também uma medida adicional de segurança. O **execl** é mais seguro porque nos garante que estamos a utilizar o executável certo, protegendo a aplicação nos casos em que um atacante tente manipular maliciosamente esta variável de ambiente.

Assim sendo, utilizamos o comando **execlp** apenas com binários comuns do sistema operativo. Para além disso, alguns comandos, como `deluser` e o `adduser` por exemplo, podem estar em sítios diferentes consoante a distribuição do sistema operativo.

4.3. Modularidade e encapsulamento das componentes de software

Procuramos melhorar o nosso projeto aplicando o máximo de modularidade e encapsulamento. Nos programas de certos comandos, como o código das componentes era reduzido, não sentimos necessidade de melhorar algum destes parâmetros. Contudo, para os comandos que o utilizador fazia acessos

diretamente às suas pastas, leitura de mensagens por exemplo, consideramos vantajoso passar as funções de leitura dos ficheiros para um outro programa. Do lado do servidor "daemon", houve mais ajustes a realizar, transferindo as *structs* como a *hashtable* dos utilizadores e da *queue* das mensagens para os seus respetivos ficheiros juntamente com as suas funções de manipulação das mesmas. Também passamos as funções de manipulação dos grupos para um ficheiro individual, numa tentativa de melhorar o encapsulamento. Apesar destas alterações do lado do "daemon", consideramos que poderíamos ter melhorado um pouco a sua modularidade e encapsulamento, no entanto, algumas das funções que lá ficaram necessitavam de acesso à *queue* e tiveram que permanecer lá, pois era uma variável global, e outras estavam relacionadas com estas mesmas. Sendo assim, poderiamos ter melhorado o encapsulamento mas requeria alterar várias coisas e houve escassez de tempo.

4.4. Ferramentas de teste e debugging

Para testar todo o programa e o seu bom funcionamento, foi fulcral a realização de muitos testes, não foram feitos testes unitários de componentes mas houve uma atenção especial aos constantes testes do serviço, ou seja, correr os vários programas múltiplas vezes e em diversas situações. A ferramenta adotada para detetar casos de erros nestes mesmos testes foram flags de compilação, mais conhecidos como "*prints*" ao longo do desenvolvimento. A sua utilização permitiu detetar em que passos do programa os erros ocorrem.

5. Conclusão

Este segundo trabalho-prático de Segurança de Sistemas de Informação, mostrou-se trabalhoso e complexo, levando a um elevado número de desafios e adversidades, principalmente tendo em conta a limitação do tempo, apesar disso, sentimos que fomos capazes de ultrapassá-las e ao mesmo tempo fomos capazes de consolidar os diversos conhecimentos adquiridos lecionados na unidade curricular.

Sentimos diversas dificuldades durante as várias fases do desenvolvimento. Nomeadamente com a utilização de permissões nas pastas e ficheiros, no entanto consideramos que superamos algumas destas. Também tivemos empasses na ativação do Daemon e na implementação dos grupos, consideramos a abordagem final suficiente mas talvez não seja a pretendida.

Dado assim por concluído o trabalho, consideramos que este possui muita margem para crescimento e melhoria no futuro. Gostavamos de ter focado mais na segurança e em alguns detalhes que não conseguimos abordar, como a utilização de registo de eventos com *syslog*.