Relatório SSI Grupo 50

Martim Redondo – a100664

Jéssica Cunha – a100901

Tiago Moreira – a100541

Introdução:

Este projeto consiste em garantir a segurança dos usuários, do sistema de arquivos e do próprio programa. Isso envolve assegurar que apenas processos autorizados pelo programa tenham acesso aos arquivos específicos utilizados ou criados por ele. Além disso, é fundamental que o código seja executado com o mínimo de permissões possíveis, reduzindo assim a margem de manobra para possíveis invasores. Por fim, o Concórdia deve ser capaz de recuperar o estado do programa em caso de falha, garantindo assim a continuidade do serviço. Estes são os fatores cruciais para o projeto.

Apesar de termos plena consciência de quais os pontos importantes e no que devíamos focar. Não conseguimos atingir tais objetivos. A arquitetura do nosso código consiste em alguns ficheiros: user.c onde, intuitivamente, se percebe que está tudo relacionado com os users, server.c onde está tudo relacionado com o server e msg.c onde estão algumas funções e structs que nos ajudarão no processo de troca de mensagens entre o user e o server.

Já com a nossa arquitetura estruturada desta maneira há descumprimentos de dois objetivos que são cruciais para o projeto. Em primeiro lugar é de salientar que desta maneira vai haver partes do nosso código que vão ter permissões que não deveria, pois, todas as funções que sejam do user vão estar num mesmo ficheiro. O outro problema advém do facto de que desta maneira, caso ocorra um problema no programa de enviar mensagens, todo o user deixará de funcionar, automaticamente, o que é errado, pois a falha de um serviço não devia levar à falha dos outros subsequentes.

Arquitetura funcional:

Não iremos incluir no relatório uma arquitetura estrutural por não fazer sentido, visto que temos todas as funções todas no mesmo ficheiro, seria algo demasiado simples e que não agrega nenhum conhecimento sobre a nossa implementação, em contra partida iremos apresentar uma arquitetura comportamental o mais detalhista possível para se perceber melhor a ideia geral do programa.

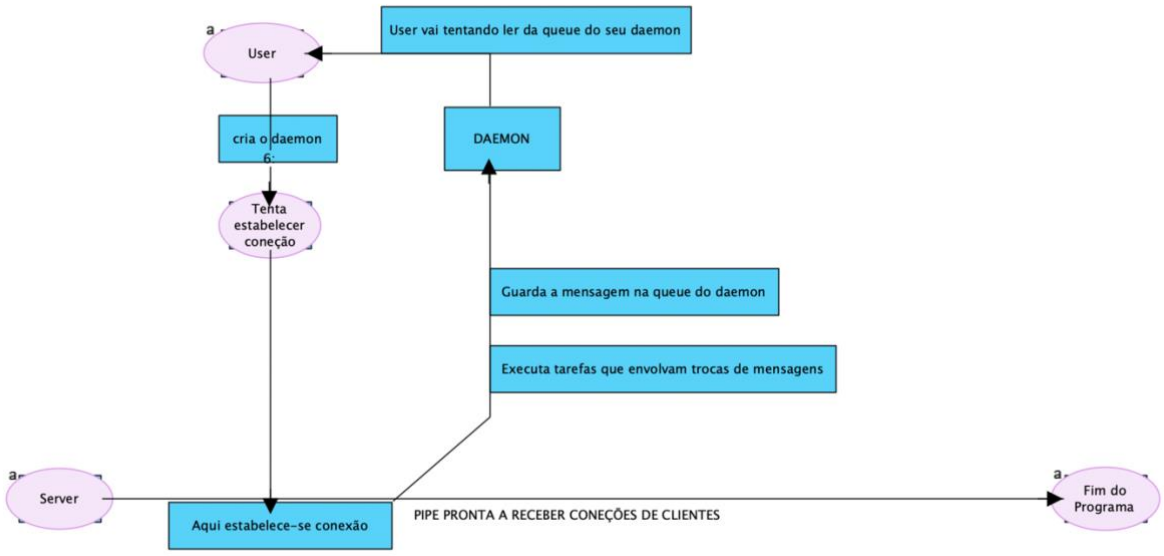

Figura 1- Arquitura comportamental

Quanto às estruturas usadas, só usamos duas estruturas sendo ambas muito simples, o motivo para tal é porque não implementamos a troca de email entre o servidor e o cliente, mas só a troca de mensagens entre o cliente e o servidor. São coisas diferentes. Pois, o que nos temos é a capacidade de o servidor conseguir dizer ao cliente se alguma das suas tarefas que passou pelo servidor fora bem resolvida. Em contrapartida, a troca de emails é algo mais elaborado, onde seriam precisas estruturas próprias para armazenar os emails dos users e também alguma forma de controlar o acesso para impedir que haja quaisquer problemas. Sendo assim, as únicas estruturas que temos são:

```
typedef struct {
    int msquid; // Identificador da fila de mensagens
    char username[256]; // Nome de usuário
} ClientInfo;
```

Figura 2 - Struct 1 (usada no server)

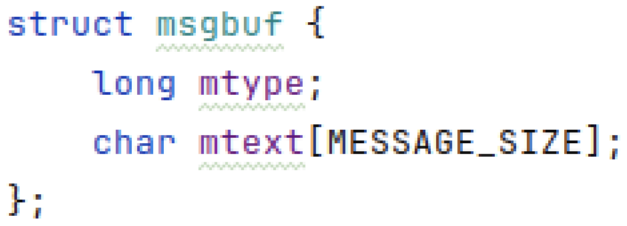

Figura 3 - Struct 2 (usada nas msg)

A primeira struct é usada para armazenar o msquid da queue e o username do user que se acabara de logar, isto é importante para as funções de mandar mensagens através do daemon, visto que é a forma que usamos para termos acesso a tal daemon.

A outra struct é usada para ler da queue as mensagens e, também, para inicializar a queue.

Decisões/implementações e melhorias:

Durante a implementação deste projeto, nos deparamos com inúmeras dificuldades. Apesar de termos uma compreensão clara do que precisava ser feito, executar as tarefas tornou-se extremamente desafiador. Consequentemente, tomamos várias decisões incorretas ao longo do processo para garantir que tivéssemos algo para apresentar.

Inicialmente, optamos por priorizar a implementação relacionada à criação e remoção de grupos. Para realizar essas ações, é necessário ter permissões adequadas, como o acesso ao comando "sudo". Como tal, elaboramos um script que atribuiria permissões de "sudo" a todos os usuários, facilitando assim a criação de grupos. No entanto, percebemos que essa abordagem comprometeria a segurança do sistema, e optamos por descartá-la. Portanto, a única maneira de executar essas operações é como usuário root; qualquer outra tentativa resultará em um erro de permissão.

Uma melhoria potencial para essa implementação seria agrupar todas as funções que exigem as mesmas permissões em um único arquivo e atribuir as permissões apropriadas a esse arquivo. Infelizmente, enfrentamos dificuldades para implementar essa abordagem.

Em seguida, abordamos a ativação e desativação de usuários, o que foi relativamente simples, pois não exigia permissões específicas além das atribuídas durante a criação do diretório. O mesmo princípio se aplica à remoção/desativação de usuários.

Passando para o último ponto que tivemos tempo de implementar, o daemon. Foi muito complicado perceber como funcionava este tipo de sistema, ainda agora não estamos completamente confiantes sobre o seu bom funcionamento. Mas a ideia por trás da implementação do daemon e o motivo foi a seguinte.

Primeiro, o cliente escreve o que quer no stdin, caso seja uma task que tenha de ser executada pelo servidor, esta vai ser mandada pelo socket para o servidor. Nele, vai haver o processar da tarefa e a obtenção de uma resposta. No nosso código, as únicas duas tarefas que o servidor consegue fazer são ativar e desativar um usário, então as respostas serão a responder se correu tudo bem ou não. Enquanto isso, o Daemon vai estar continuamente a ler o conteúdo da queue à espera de uma mensagem e vai imprimei-la no ecrã.

O problema é que isto está ainda bastante incompleto, pois há várias coisas a serem melhoradas, primeira coisa a ser melhorada é que a queue é partilhada por todos os users, por algum motivo, mesmo havendo um msquid diferente para conseguirmos diferenciar as diferentes queues, o daemon assume que é sempre a mesma, então caso dois clientes estejam a pedir para ser ativados, ambos vão ler as mensagens um do outro. Outro problema, advém do facto de que apesar do daemon estar sempre a ler a queue e perceber que algo foi lá colocado a mensagem da queue só será impressa na próxima iteração, ou seja, vai haver sempre delay de uma mensagem.

Uma forma de melhorar esta parte é implementar de forma direita a queue para cada utilizador, pois cada utilizador teira acesso, unicamente, às suas mensagens, como deveria ser. Para resolver o problema do delay era reescrever a função que nos dá o print no terminal, podemos tentar aplicar uma política de SD aqui, sendo que podemos dar LOCK no stdout até que o conteúdo da queue seja devidamente impresso, sendo que isto seria só uma ideia.

Estas são as ideias e possíveis melhorias por trás do nosso código, contudo é de realçar que houve partes que não foram implementadas, sendo elas a troca de mensagens, esta seria a parte mais crítica em termos de segurança. A nossa ideia para implementar esta parte seria termos, primeiro, uma queue que consiga guardar emails pendentes, essa queue não poderia estar no daemon do user que receberia a mensagem nem no daemon do user que enviou a mensagem, pois esses daemon's só corre quando o user está ligado, e a queue tem de existir em qualquer momento. Então a ideia poderia ser ter um segundo daemon que corre no servidor com a queue e os clientes podem ir lá buscar as informações quando quiserem ler os seus emails. Outro problema que viria associado a estas tarefas seria a forma como a mensagem seria escrita na queue. Pois, sabemos que não é confiável o user escrever diretamente na queue. Logo, teria de haver uma série de verificações.

Reflexão/Conclusão:

Como dito até agora, há abordagem mais correta seria separar as funções por ficheiros e atribuir as respetivas permissões. Quando ao que foi feito por nós, não achamos que haja nenhuma parte que deva ser realçada, o trabalho está muito fraco em termos de implementação, como já referido houve muitas dificuldades neste segundo projeto. O tempo também foi pouco devido à sobrecarga de trabalhos e testes. Ainda para acrescentar todos os membros do grupo têm MacOs, o que complicou o processo.