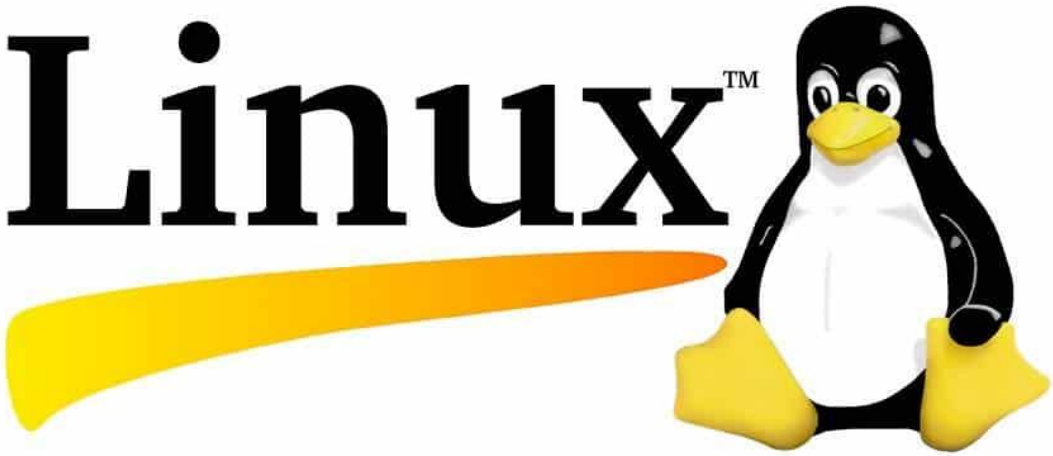
Instituto Superior de Engenharia de Coimbra

Sistemas Operativos



Trabalho Prático

Programação em C para UNIX

Licenciatura em Engenharia Informática

2024 / 2025

Nuno Tomás Paiva

a2023137363@isec.pt

Rui Martins dos Santos

a2023145822@isec.pt

1. Introdução

O trabalho consiste em criarmos uma plataforma de troca de mensagens entre vários utilizadores para um único servidor e esses puderem escrever sobre diversos tópicos.

O nosso trabalho está organizado em 3 ficheiros.

No ficheiro *feed.c,* é a parte do utilizador no nosso trabalho, onde o utilizador consegue "comunicar" com o servidor os comandos que quer executar. O utilizador pode fazer as seguintes ações: dar login, saber a lista de tópicos existentes, mandar uma mensagem, inscrever-se e desinscrever-se de um tópico e sair do servidor.

O servidor por sua vez, é o nosso *manager.c*, ficheiro que tem como função ser o servidor de vários programas do tipo feed, neste caso vários utilizadores em simultâneo. O manager é obrigado a comunicar ao feed se os seus comandos foram executados com sucesso ou não. O manager é o administrador de todo o trabalho podendo fazer os seguintes comandos: saber a lista de utilizadores, remover um utilizador, saber a lista de tópicos, listar as mensagens existentes em cada tópico, bloquear e desbloquear um tópico e encerrar a plataforma.

O ficheiro util.h contém todas as structs e os includes utilizados no trabalho. Optamos por chamar structs dentro de structs por acharmos que ficava melhor organizado e depois de falarmos com um dos professores no seu gabinete.

2. Comunicação entre Feed e Manager

Do lado do feed, fazemos write no pipe de uma estrutura sempre dividido em 2 partes, uma estrutura Header e uma estrutura com os dados da operação que queremos fazer.

Exemplo de um write no pipe de uma Mensagem (MESSAGE_CLI):

```
typedef struct{
        HEADER header; -> estrutura Header
        MESSAGE message; -> estrutura com os dados da operação que
} MESSAGE_CLI                                          queremos fazer
```

Do lado do manager, primeiramente, lê se a estrutura Header, que contém o comando a receber, e depois, ao usar um switch…case com esse comando recebido, escolhe o tipo de estrutura a ler, neste caso, uma estrutura Message.

Optámos por fazer o envio e receção de informações desta maneira, de forma a enviar o mínimo tamanho necessário.

Já o Manager, do seu lado, envia uma estrutura também dividida em 2 partes.

Exemplo de um write no pipe de uma Mensagem (MESSAGE_SERV):

```
typedef struct{
        int cmd; -> comando para saber qual o tipo de informação que vai receber
        MESSAGE message; -> estrutura com os dados da operação que
} MESSAGE_SERV                                         queremos fazer
```

Optámos por criar uma estrutura SERV e não utilizar a estrutura CLI recebida de forma a minimizar, novamente, o tamanho reenviado pois senão iríamos estar a enviar dados que não iríamos utilizar.

3. Uso de Threads

Optámos por usar threads no Feed e no Manager para conseguirmos fazer várias coisas ao mesmo tempo.

FEED

No Feed temos 2 threads, a thread da função main que essencialmente serve para o teclado, ou seja, enviar informações para o servidor. A outra thread serve exclusivamente para receber os dados enviados pelo Manager.

MANAGER

No Manager temos 3 threads, sendo uma Main para tratar receber e enviar informações para o Feed, outra thread para os comandos admin, enumerados mais acima, e ainda uma thread para ir decrementando de 1 em 1 segundo de todas as mensagens persistentes.

A estratégia que utilizamos foi a de a thread do **timer** decrementar o tempo de cada mensagem persistente usando o sleep e atualizando a cada sleep de 1 segundo a nossa estrutura de mensagens persistentes que está dentro da estrutura **CONTEXT_TOPICS**.

O **Admin** faz comandos de atualização dos dados do **manager** (estrutura app context) mas em 2 casos em que ela também escreve no pipe do **Feed**. Sendo essas vezes, no uso do comando **Remove** e no uso do comando **Close**.

Nós fazemos esta comunicação no **Close** de forma ao **Manager**, caso existam clientes ativos, escreve no pipe de todos os clientes o comando exit, ou seja, de que a sessão foi encerrada.

Já no comando **Remove**, escreve apenas no pipe do cliente que foi removido da plataforma, novamente através do comando exit.

4. Encerramento do Programa de Forma Ordeira

FEED

A estratégia que usamos para terminar o **Feed** de forma ordeira através do comando exit, foi a seguinte: Escrevemos no pipe do servidor que queremos dar exit, ele lê essa informação e escreve de volta para o pipe do **Feed** essa informação, o **Feed** lê esse comando na **thread** de exit e envia um sinal através do **pthread_kill** para a função main, faz pthread_exit. A função main como teve o sinal enviado, o **scanf** não vai ser efetuado corretamente e, portanto, quebra o ciclo com o **break**.

Com o **pthread_join**, a função main espera que a thread que recebe os dados saia de forma ordenada e depois o main, consequentemente, também sai de forma ordenada com o **pthread_exit**.

Fazemos este procedimento de forma a conseguir entrar em contacto com a **thread** e assim desligá-la ordeiramente.

MANAGER

Encerramos o programa todo de forma ordeira através do comando **Close** que verifica se há utilizadores ativos, caso haja, escreve no pipe dos utilizadores o comando exit de maneira que estes saiam de forma ordeira, envia um sinal com o pthread_kill para a função main (semelhante ao que fazemos no **Feed**) e altera o valor da variável que usamos para controlar o while de forma que este acabe e faça **pthread_exit**.

Semelhante ao que fizemos no **Feed**, a função main espera pelas threads com o **pthread_join** e por fim dá **pthread_exit**.

5. Conclusão

O objetivo deste trabalho foi aprofundar os conceitos de sistemas operativos, especialmente no contexto UNIX, através da aplicação prática de comunicação entre processos, sincronização com threads e manutenção de estados consistentes.

Com a realização do projeto, adquirimos competências valiosas que nos preparam para desafios reais no desenvolvimento de aplicações, consolidando nosso aprendizado em uma das unidades curriculares mais importantes para o nosso percurso.