Instituto Superior de Engenharia

Politécnico de Coimbra

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

Sistemas Operativos – Programação em C para Unix

Relatório de Trabalho Prático em Engenharia Informática

Autor

João Francisco de Matos Claro - 2017010293

Coimbra, Dezembro 2024

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

INTRODUÇÃO

Este trabalho, desenvolvido no âmbito da unidade curricular de Sistemas Operativos, tem como principal consolidar os conhecimentos adquiridos sobre mecanismos de comunicação entre processos no ambiente UNIX, utilizando a linguagem de programação C.

O projeto proposto envolve a criação de uma plataforma para envio e receção de mensagens curtas, organizada por tópicos e projetada para funcionar numa única máquina. A implementação explora as funcionalidades das funções biblioteca padrão da linguagem C, bem como os mecanismos de comunicação providênciados pelos named pipes (FIFOs).

ESTRUTURAS ESSENCIAIS

TDATA no manager

```
 1. typedef struct
 2. {
 3.     int stop;
 4.     int fd_manager;
 5.     int delta_time;
 6.     topic topic_list[TOPIC_MAX_SIZE];
 7.     int current_topics;
 8.     userData user_list[MAX_USERS];
 9.     int current_users;
10.     pthread_mutex_t *mutex_users;
11.     pthread_mutex_t *mutex_topics;
12. } TDATA;
13.
```

stop: permite indicar à thread que deve terminar o seu ciclo

fd_manager: permite fechar, de forma ordeira o file descriptor associado à receção de mensagens pelo manager

delta_time: inteiro para otimizar a atualização do tempo nas mensagens persistentes

topic_list: guarda a lista de tópicos existentes, em conjunto com o seu contador

user_list: guarda os utilizador atualmente logged in, em conjunto com o contador

mutexes: Os mutexes permitem salvaguarda de informação na escrita e leitura da mesma, garantido que a aplicação aguarda por outra thread que esteja a utilizar a memória associada, tal que:

- o mutex_users salvaguarda a informação no user_list
- o mutex_topics salvaguarda a informação no topic_list

Este estrutura é central para a execução do programa manager, tendo toda a informação requerida em qualquer momento.

Apesar da informação estar acessível em algumas funções, é atribuido o mutex associado sempre que estas realizarem alguma operação na estrutura. Garantindo que uma função lê informação sem esta estar a ser alterada em tempo reada em tempo real.

TOPIC

Estrutura para topicos

```
1. typedef struct
2. {
3.     char topic[TOPIC_MAX_SIZE];
4.     int subscribed_user_count;
5.     userData subscribed_users[MAX_USERS];
6.     int persistent_msg_count;
7.     msgData persist_msg[MAX_PERSIST_MSG];
8.     int is_topic_locked; // 1 is locked, 0 is NOT locked
9. } topic;
10.
```

topic: guarda o nome atribuido ao tópico

subscribed_users: guardar os útilizadores subscritos ao tópico associado, em conjunto com o seu contador

persist_msg: guarda as mensagem que tenham um tempo superior a 0, em conjunto com o contador

is_topic_locked: apenas recebe um valor binário, indica se o tópico aceita mensagens ou não

USERDATA

Estrutura para users

```
1. typedef struct
2. {
3.     pid_t pid;
4.     char name[USER_MAX_SIZE];
5. } userData;
6.
```

pid: guardar o id do processo do feed

name: guarda o nome que o user utilizou para fazer login

MSGDATA

Estrutura para mensagens

```
1. typedef struct
2. {
3.     char topic[TOPIC_MAX_SIZE];
4.     char user[USER_MAX_SIZE];
5.     int time; // Time until being deleted
6.     char text[MSG_MAX_SIZE];
7. } msgData;
8.
```

topic: nome

user: nome do utilizar que envia a mensagem

time: tempo de vida de uma mensagem persistente, 0 se a mesma não for persistente

text: guarda o texto enviado pelo utilizador

MSGTYPE

Enum para o tipo de mensagem a enviar/receber

```
 1. typedef enum
 2. {
 3.     LOGIN,
 4.     LOGOUT,
 5.     SUBSCRIBE,
 6.     UNSUBSCRIBE,
 7.     MESSAGE,
 8.     RESPONSE,
 9.     LIST,
10. } msgType;
11.
```

Como é traduzida diretamente num inteiro desde 0 ao número de items – 1, permite a identificação do tipo de mensagem de maneira fácil de ler, por parte do desenvolvedor, e de maneira simple de compara, por parte do programa.

Todas as mensagem enviadas do programa feed têm um *msgType* no inicio de cada estrutura.

TDADA NO FEED

```
1. typedef struct
2. {
3.     int stop;
4.     int fd_feed;
5.     char fifoName[100];
6.     userData user;
7. } TDATA;
8.
```

stop: permite indicar à thread que deve terminar o seu ciclo

fd_manager: permite fechar, de forma ordeira o file descriptor associado à receção de mensagens pelo manager

fifoName: guarda o nome do file descriptor para leitura pelo feed,

Como este altera devido ao id do processo, é também guardado na estrutura, para poder ser feito o unlink posteriormente

user: guarda o nome usada para login e o id do processo

MESSAGESTRUCT

Estrutura utilizada pelo feed para manter informação em bloco, é utilizada para enviar uma mensagem ao manager

```
1. typedef struct
2. {
3.     msgType type;
4.     userData user;
5.     int msg_size;
6.     msgData message;
7. } messageStruct;
8.
```

msg_size: tamanho da estrutura msgData a enviar

INFORMAÇÃO DEFINIDA NO INCIO DO PROGRAMA

No manager.h

```
1. #define MAX_PERSIST_MSG 5
2. #define MAX_DELTA_TIME 5 // seconds
```

MAX_PERSIST_MSG: indica o número máximo de mensagem persistentes que cada tópico pode guardar

MAX_DELTA_TIME: tempo máximo entre cada atualização

Este valor é utilizado para reduzir o tempo de cada mensagem persistente

helper.h

```
1. // paths for fifo files
2. #define MANAGER_FIFO "fifos/manager_fifo"
3. #define FEED_FIFO "fifos/feed_%d_fifo"
4.
5. #define MSG_MAX_SIZE 300  // Max size of a message
6. #define TOPIC_MAX_SIZE 20 // Max size of topics and topic name
7. #define USER_MAX_SIZE 100 // Max size of the user name
8.
9. #define MAX_USERS 10 // Max users
```

_FIFO: Nome atribuido aos named pipes respetivo

MAX_MSG_SIZE: indica o tamanho máximo a guardar pelo texto da mensagem

TOPIC_MAX_SIZE: indica o tamanho máximo do nome do tópico, bem como o número de tópicos que podem existir

Como o valor foi predifinido igual no enunciado, é possível utilizar a mesma variável

USER_MAX_SIZE: indica o tamanho máximo para o nome do utilizador

MAX_USERS: indica o número máximo de utilizadores logados

MACROS

```
// To remove '\n' from the string
12. #define REMOVE_TRAILING_ENTER(str) str[strcspn(str, "\n")] = '\0'
13.
14. // Calculates the size of msgData, already accounts for the '\0'
15. #define CALCULATE_MSGDATA_SIZE(str) TOPIC_MAX_SIZE + USER_MAX_SIZE + sizeof(int) +
strlen(str) + 1
16.
17. // Calculates the size of messageStruct
18. #define CALCULATE_MSG_SIZE(msgData_size) sizeof(msgType) + sizeof(userData) + sizeof(int) +
msgData_size
```

Permitem a redução de código para cálculo de tamanhos de mensagens.

REMOVE_TAILING_ENTER: recebe uma frase, e remove, caso exista, qualquer caracter do tipo "enter" dessa frase

O principal uso acontece quando existe informação inserida pelo user ou pelo administrado.

CALCULATE_MSGDATA_SIZE: facilita o calculo do tamanho da estrutura *msgData* quando enviada

CALCULATE_MSG_SIZE: apenas para a estrutura *messageStruct*

ARQUITETURA GERAL

Para o funcionamento completo do projeto, são necessários 2 programas, o manager e o feed.

Manager

Programa responsável pela gestão centralizada da receção e distribuição de mensagens. Este assegura que todas as mensagens enviadas para um tópico são encaminhadas para os utilizadores que subscreveram o tópico. O manager está organizado em 3 threads principais:

Main

Esta thread gere toda a comunicação com o administrador do programa.

Através da thread o administrador pode introduzir comando para remove utilizadores, bloquear e desbloquear tópicos, bem como monitorizar tópicos, mensagens e utilizadores.

handleFifoComunication

Esta thread gere toda a comunicação com o programa feed.

Recebe sempre, como primeira informação de pedido, o tipo do pedido, através de um campo do tipo msgType.

Dependendo deste campo, o restante da informação é lida, baseado nas estruturas definidas acima.

updateMessagerCounter

Esta thread atualiza o delta_time, acrescentando 1 por segundo e quando chega a um valor, definido por MAX_DELTA_TIME, o tempo de todas as mensagens é atualizado.

Para pedidos do tipo login ou logout, pode ir apenas uma estrutura do tipo userData.

Para pedidos de subscrição e de cancelar subscrição ou de listagem de tópicos, adiciona-se o nome do tópico.

Para pedidos de mensagem, existe um campo extra com o tamanho da mensagem, sendo esta de tamanho variável.

Feed

Programa utilizado pelos utilizadores para interagirem diretamente com a plataforma. Permite enviar mensagem para tópicos que estejam subscritos, subscrever ou cancelar a subscrição, e receber mensagens enviadas para os tópicos em que estão subscritos.

O feed está organizado em 2 threads:

Main

Esta thread gere toda a comunicação com o utilizador do programa.

Através da thread o user pode ver que topicos existem, subscrever ou cancelar a subscrição de um tópico e enviar mensagens.

handleFifoComunication

Esta thread gere toda a comunicação com o programa manager.

Recebe mensagens enviadas pelo manager, podendo estas serem confirmações ou informações sobre ações que tomou, ou mensagens de outros utilizadores.

Recebe sempre, como primeira informação, o tamanho da mensagem que foi enviada pelo servidor.

ESTRUTURA DO CODIGO

O ficheiro **helper.c** tem 2 funções de recurso ao manager e ao feed, por serem iguais a ambos ficheiros, e ao seu respetivo header, **helper.h**.

O **helper.h** tem a inicialização das funções acima indicadas, bem com das estruturas, variaveis e macros requeridas em ambos os ficheiros, manager e feed.

O ficheiro **feed.c** tem todo o código requerido para o funcionamento do mesmo, fazendo apenas recurso do header respetivo, **feed.h**, para inicialização de funções e de estruturas e o helper.h.

Quanto ao programa manager, este está dívido entre vários ficheiros, fazendo todos referencia ao ficheiro **manager.h**, onde estão inicializadas as funções, estruturas e macros necessárias para a sua execução.

- no **manager.c** estão apenas as threads **main** e **handleFifoCommunication,** onde é feita o redirecionamento, consoante pedidos que recebem.

Existem pedidos que não requerem redirecionamente, sendo tratadas pelas threads.

- no **files.c** estão 2 funções para ler e guardar no ficheiro de texto as mensagens persistentes que existam, e têm tempo util de vida.

- no **helper_users.c** existem funções de tratamento de utilizador, geralmente associado com a lista *user_list* e o *mutex_users,* mas não exclusivamente.

Aqui encontram-se funções para adicionar e remove o utilizador, ou para confirmar de que este se encontra em alguma lista específica.

- no **helper_topics.c** existem funções de tratamento de tópicos, geralmente associado com a lista *topic_list* e o *mutex_topics.*

Aqui encontram-se funções para adicionar e remover tópicos, para confirmar de que este exista or para bloquear e desbloquear tópicos.

- no **helper_messages.c** existem funções para receber novas mensagens, enviar respostas e mensagens ao utilizador, para remover e adicionar mesagens persistentes ou atualizar o tempo de vida destas.

MAKEFILE

```
headers = helper.h

executables = manager feed
manager_objects = manager.o files.o helper.o helper_messages.o helper_users.o helper_topics.o
feed_objects = feed.o helper.o

cflags = -Wall -Werror -pthread
export MSG_FICH = ./msg.txt

all: $(executables)

#link
manager: $(manager_objects)
	$(CC) -o $@ $^ $(cflags)

feed: $(feed_objects)
	$(CC) -o $@ $^ $(cflags)

#compile
%.o: %.c $(headers)
	$(CC) -c $<

#runs
run: manager feed
	./manager

startup:
	mkdir fifos
	touch msg.txt

clean:
	$(RM) *.o
	$(RM) $(executables)

.PHONY: all clean run1.
```

Com o intuito de facilitar a compilação dos programas que constituem o projeto, foi elaborado um *makefile*. Este permite não só compilar os programas, como remove os ficheiros objeto (.o) e executáveis de forma rápida e simples, e de preparar o ficheiro msg.txt e a pasta fifos, obrigaórios à execução do manager.

JUSTIFICAÇÃO DE OPÇÕES TOMADAS

No desenvolvimento deste trabalho, foram tomadas várias decisões que influenciaram a implementação e a execução da plataforma. Estas escolhas foram fundamentadas na simplicidade, robustez e alinahdas com os objetivos do projeto. Abaixo são justificadas as principais decisões:

A comunicação, seja de feed para manager ou de manager para feed, é realizada exclusivamente através dos named pipes respetivos. Esta abordagem garante um canal dedicado e organizado para cada fluxo de dados, minimizando conflitos

O único sinal tratado pelos programas feed e manager é o **SIGINT**. Este sinal é utilizado exclusivamente para informar o utilizador ou administrador sobre o modo correto de terminar o programa, garantindo a libertação apropriada de recursos, o encerramento seguro dos named pipes e a finalização ordenada das threads.

Contrariamente ao enunciado original, os utilizadores precisam de subscrever um tópico antes de enviar ou receber mensagens. Esta decisão foi tomada para evitar cenários incoerentes em que um utilizador interaja com tópicos aos quais não está associado, garantindo maior consistência no funcionamento do sistema.

Apesar de não ser 100% seguro no contexto de threads, o input recebido pelo administrador é tratado com a função *strtok* devido à sua simplicidade e eficiência.

Uma Solução alterativa seria a função *strtok_r,* sendo esta reentrante permite guardar o restante da informação a ser analisada pela função numa variável definida na execução.

Por outro lado, os inputs recebidos dos utilizadores através do feed são tratados com maior cautela, assegurando robustez adicional e evitando potenciais vulnerabilidades.

SEGURANÇA E VULNERABILIDADES

O programa termina sempre com um EXIT_FAILURE, mesmo quando o mesmo é encerrado corretamente pelo utilizador, e não havendo falhas. Isto é devido ao facto de que é encerrado com recurso à função *closeService* que garante: o término (*join*) das threads; o envio de uma mensagem de término a todos os utilizadores; guardar as mensagens em ficheiros; o desloqueio, caso exista, das threads em leitura; o fecho do file descriptor associado com a leitura; e a removeção do named pipes.

Isto garante o término ordeiro da aplicação, e da libertação de qualquer recurso utilizado.

Esta abordagem é utiliza pois, em todas as vezes que a função é chamada, à exceção através do respetivo comando pelo utilizador ou administrador, por um erro que possa ter acontecido durante a execução do programa.

Instituto Superior de Engenharia
Politécnico de Coimbra