Instituto Superior
de Engenharia
Politécnico de Coimbra

Sistemas Operativos

Licenciatura em Engenharia Informática

2024/2025

João Pedro Silveira da Costa N.º2022143368
Juliana Silva Teixeira N.º2022143379

Índice

1. Introdução

O trabalho prático teve como objetivo a criação de uma plataforma para envio e receção de mensagens curtas, organizadas por tópicos. Nesta plataforma, um utilizador pode enviar mensagens associadas a um tópico específico e subscrever um ou mais tópicos. Sempre que uma mensagem é enviada para um determinado tópico, todos os utilizadores que o tenham subscrito recebem essa mensagem. O sistema desenvolvido funciona como intermediário entre os clientes (feeds) e o administrador (manager), controlando a troca de mensagens entre ambos. A implementação foi realizada em linguagem C, para a plataforma UNIX (Linux), utilizando os mecanismos do sistema operativo explorados nas aulas. Na gestão dos recursos do sistema, demos prioridade à utilização de chamadas do sistema operativo em face ao uso de funções de biblioteca.

2. Estruturas de Dados

3.1. Struct DataUser

```
typedef struct
{
    pid_t pid;
    char name[100];
} DataUser;
```

Estrutura que armazena os pid e o username do feed para que seja passada junto com o feedComands, tendo assim o manager sempre o conhecimento de quem enviou o comando.

3.2. Struct FeedCommands

```
typedef struct
{
    int id;
    char command[100];
    char topic[20];
    int duration;
    char message[300];
    DataUser user;
} FeedCommands;
```

Estrutura utilizada para armazenar toda a informação do comando do user para enviar para o manager e assim fazer a lógica no manager.

3.3. Struct TopicUsers

```
typedef struct
{
    // Users inscritos neste topico
    int numUsers;
    DataUser users[MAX_USERS];
} TopicUsers;
```

Estrutura que guarda o número de users subscritos no manager até ser atingido o número máximo de users conectados.

3.4. Struct DataMessageP

```
typedef struct
{
    // User que escreveu o topico
    char name[100];
    int duration;
    char msg[300];
} DataMessageP;
```

Estrutura que trata da mensagem permanente num tópico, com o nome do user, duração da mensagem e a mensagem. Esta estrutura está inserida dentro de um array e armazena todas as mensagens persistentes do respetivo tópico.

3.5. Struct TopicMessagesP

```
typedef struct
{
    int id;
    // Mensagens permanentes neste topico
    int num_msgP;
    DataMessageP messagesP[MAX_MSG_P];
} TopicMessagesP;
```

Estrutura utilizada para quando o user subscreve um tópico, esta estrutura armazena todas as mensagens desse tópico para enviar para o feed respetivo.

3.6. Struct DataTopic

```
typedef struct
{
    char name[20];
    char status[60];
    int num_msgP;
} DataTopic;
```

Estrutura utilizada em um array e armazena o nome do tópico e o seu status se está bloqueado ou desbloqueado e o número de mensagens persistentes.

3.7. Struct TopicInfos

```
typedef struct
{
    int id;
    int numTopics;
    DataTopic topics[MAX_TOPICS];
} TopicInfos;
```

Estrutura que guarda o número de tópicos criados até atingir o número máximo de tópicos. Esta estrutura é utilizada para no comando topics enviar todos os tópicos existentes no manager.

3.8. Struct DataDados

```
typedef struct
{
    FeedCommands feedCommands;
    TopicUsers topicUsers;
    TopicMessagesP topicMessagesP;
    TopicInfos topicInfos;
} DataDados;
```

Estrutura onde engloba todas as outras necessárias, ou seja, é a estrutura usada para a comunicação entre o feed e o manager.

3.9. Struct Topic

```
typedef struct
{
    DataTopic dataTopic;
    TopicUsers topicUsers;
    TopicMessagesP topicMessagesP;
    pthread_mutex_t *m;
} Topic;
```

Estrutura utilizada como um array para armazenar todas as informações sobre cada tópico existente.

3. Manager

O manager é o responsável pela gestão de tópicos e users na plataforma. Atua como um servidor que controla a criação, inscrição e comunicação dentro dos tópicos.

Entrada:

Quando um novo utilizador se tenta conectar ao sistema, o manager lê as informações enviadas pelo FIFO (SERVER_FIFO) e valida o username, verificando se o mesmo ainda não existe. Caso o user seja conectado, o Manager, adiciona-o ao vetor storedUsers, inicializando o seu PID e username.

```
// Verifica os users que se querem conectar
if (dados.user.id == 1)
{
    int found = 0;
    int emptyUserSlotIndex = -1;

    for (int i = 0; i < numStoredUsers; i++)
    {
        if (strcmp(storedUsers[i].name, dados.user.name) == 0)
        {
            found = 1;
            break;
        }

        // Verifica se há um slot vazio (pid == 0 e name == '\0')
        if (storedUsers[i].pid == 0 && strcmp(storedUsers[i].name, "\0") == 0 && emptyUserSlotIndex == -1)
        {
            emptyUserSlotIndex = i;
        }
    }

    if (found == 1 || numStoredUsers >= MAX_USERS)
    {
        // Envia ao usuario a flag=1 | rejeitado
        dados.user.flag = 1;
        printf("\n<Server_Info> Usuário %s rejeitado.\n", dados.user.name);
        sprintf(CLIENT_FIFO_FINAL, CLIENT_FIFO, dados.user.pid);
        fd_response = open(CLIENT_FIFO_FINAL, O_WRONLY);
        write(fd_response, &dados, sizeof(dados));
        close(fd_response);
    }
    else
    {
        if (emptyUserSlotIndex != -1)
        {
            // Substitui o usuário no slot vazio
            strcpy(storedUsers[emptyUserSlotIndex].name, dados.user.name);
            storedUsers[emptyUserSlotIndex].pid = dados.user.pid;
        }
        else
        {
            // Armazena o novo usuário na próxima posição disponível
            strcpy(storedUsers[numStoredUsers].name, dados.user.name);
            storedUsers[numStoredUsers].pid = dados.user.pid;
            numStoredUsers++;
        }

        printf("\n<Server_Info> Usuário %s conectado.\n", dados.user.name);
        int countUsers = 0;
        for (int i = 0; i < numStoredUsers; i++)
        {
            if (strcmp(storedUsers[i].name, "\0") != 0)
            {
                countUsers++;
            }
        }
        printf("<Server_Info> %d Usuários conectados.\n", countUsers);

        // Altera variáveis para indicar aceitação
        dados.user.flag = 0;
        dados.user.id = 0;
        sprintf(CLIENT_FIFO_FINAL, CLIENT_FIFO, dados.user.pid);
        fd_response = open(CLIENT_FIFO_FINAL, O_WRONLY);
        write(fd_response, &dados, sizeof(dados));
        close(fd_response);
    }
}
```

Saída:

O user pode ser desconectado pelo comando remove ou pode-se desconectar a ele próprio com o comando exit.

Tópicos:

O Manager controla a criação e o status dos tópicos. Estes podem ser bloqueados ou desbloqueados, permitindo ou limitando as mensagens. As mensagens persistentes são guardadas em um ficheiro txt.

Comandos do Manager:

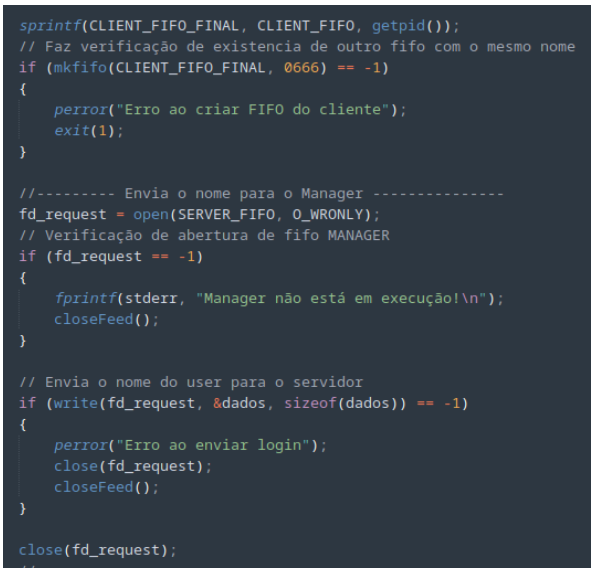
- **users:** apresenta a lista dos utilizadores ativos**;**
- **remove:** eliminar um utilizador existente e informá-lo que foi removido, assim como todos os utilizadores ativos;
- **topics:** apresenta a lista dos tópicos existentes, assim como o número de mensagens persistentes respetivas;
- **show:** apresenta a lista de mensagens de um determinado tópico;
- **lock:** bloqueia um tópico, não sendo possível enviar mensagens para esse tópico, porém é possível subscrevê-lo;
- **unlock:** desbloqueia o tópico;
- **close:** encerra todos os feeds e informando-os do sucedido e de seguida termina o manager.

4. Feed

O Feed fornece uma interface de comandos para que os clientes enviem mensagens e coordenem a sua inscrição nos tópicos.

Inicialização:

O Feed começa por verificar a existência do Manager, enviando um pedido de conexão pelo FIFO (SERVER_FIFO), de seguida cria o seu próprio FIFO (CLIENT_FIFO) para receber as respostas.



```
sprintf(CLIENT_FIFO_FINAL, CLIENT_FIFO, getpid());
// Faz verificação de existencia de outro fifo com o mesmo nome
if (mkfifo(CLIENT_FIFO_FINAL, 0666) == -1)
{
    perror("Erro ao criar FIFO do cliente");
    exit(1);
}

//--------- Envia o nome para o Manager --------------
fd_request = open(SERVER_FIFO, O_WRONLY);
// Verificação de abertura de fifo MANAGER
if (fd_request == -1)
{
    fprintf(stderr, "Manager não está em execução!\n");
    closeFeed();
}

// Envia o nome do user para o servidor
if (write(fd_request, &dados, sizeof(dados)) == -1)
{
    perror("Erro ao enviar login");
    close(fd_request);
    closeFeed();
}

close(fd_request);
```

Comandos do Feed:

- **topics:** apresenta a lista de todos os tópicos, assim como o número de mensagens persistentes e se este está bloqueado ou desbloqueado**;**
- **msg:** envia uma mensagem para um determinado tópico;
- **subscribe:** subscrever um tópico;
- **unsubscribe:** deixa de subscrever o tópico;
- **exit:** encerra o feed e informa o manager.

5. Makefile

Regra	Dependência	Funcionalidade
all	feed manager	Compila todos os programas associados ao trabalho
manager	manager.o	Compila o programa manager
manager.o	manager.c	Compila o objeto manager
feed	feed.o	Compila o programa feed
feed.o	feed.c	Compila o objeto feed
clean		Limpa todos os ficheiros originados pelo makefile
clean-manager		Limpa os ficheiros do manager originados pelo makefile
clean-feed		Limpa os ficheiros do feed originados pelo makefile

6. Desafios na Implementação do Trabalho

Durante a implementação, foi utilizada uma estrutura global para o processamento dos dados. No entanto, foi ponderada a opção de separar o processo e adotar a leitura por envelope como alternativa. Contudo, essa abordagem resultou em diversos erros, o que dificultou a sua implementação de forma eficaz. Este desafio levou à decisão de manter a estratégia inicial da estrutura global, uma vez que a implementação da leitura por envelope não se mostrou viável, sendo esta a solução mais estável e adequada para garantir a precisão e consistência do trabalho.

7. Conclusão

Ao longo deste trabalho, enfrentámos e superámos diversos desafios, o que se revelou uma excelente oportunidade para consolidar os conteúdos abordados nas aulas de Sistemas Operativos. Este projeto permitiu-nos aplicar na prática conceitos essenciais relacionados com a arquitetura, criação e desenvolvimento de programas para sistemas UNIX, proporcionando-nos uma compreensão mais aprofundada dos vários pormenores envolvidos.