Departamento de Engenharia Informática e de Sistemas Instituto Superior de Engenharia de Coimbra Licenciatura em Engenharia Informática

Sistemas Operativos 2024/2025

Relatório de Trabalho Prático

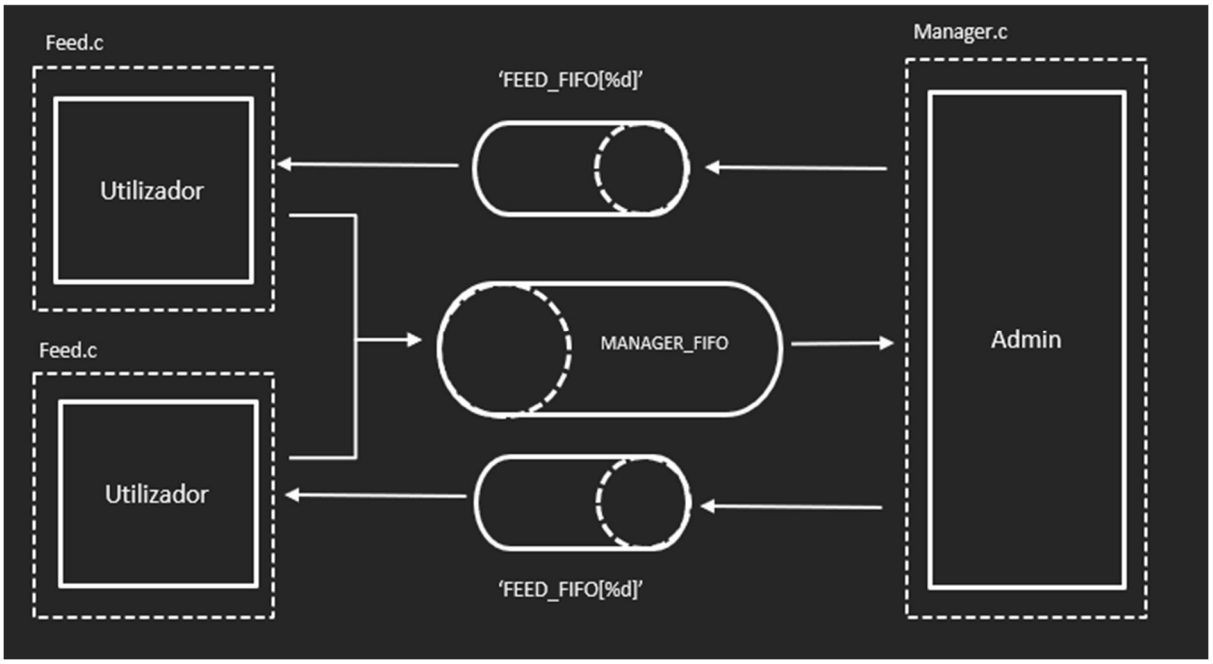
Martim Oliveira 2022132041

Ricardo Duarte 2022137878

Índice

Introdução

O trabalho prático aborda os conhecimentos do Sistema Unix, em linguagem C, através da utilização de mecanismos deste Sistema Operativo, estes abordados nas aulas teóricas e práticas, tais como sinais, pipes, threads, mutex's, entre outros.
O trabalho consiste em uma plataforma de envio e receção de mensagens, estas organizadas por tópicos. Um utilizador pode enviar mensagens para um determinado tópico e receber mensagens do mesmo.
Um programa "manager" faz a gestão dos tópicos e mensagens, o outro programa "feed" efetua o envio e receção de mensagens, a comunicação é feita através de (named pipes), este programa "feed" é usado pelos utilizadores, podendo existir diversos "feeds".
O nosso programa "manager" aceita um máximo de 10 utilizadores, 20 tópicos e cada tópico pode ter no máximo 5 mensagens persistentes, estas que ficam guardadas e enquanto tiverem tempo são vistas por novos utilizadores, quando o programa é encerrado, se existir alguma mensagem persistente é guardada num ficheiro.



Programas

Tal como dito na Introdução, temos os programas “manager.c” e “feed.c”, e .h (headers) adjacentes, mas para além desses 2, temos outros que explicaremos mais à frente, tais como, “utils.h”, “processocom.c” e “makefile”.

Manager.c

Tal como dito o programa manager, faz o processamento de toda a informação, tal como Utilizadores, tópicos e mensagens.
Para efetuar o processamento, o manager usa 5 funções, main(principal), acorda, abre pipes, decrementa e ler pipe, para além dessas ainda conta com a “ajuda” do ficheiro processocom, tendo este outras funções, tais como, “guarda_mensagens_persistentes”, “load_mensagens_persistentes”, “processa_comando_manager”, “processa_comando_feed”, estas ajudam com a informação recebida.

Inicialmente o main declara as suas variáveis, para futura utilização. Após isso, tenta abrir o ficheiro onde tem as mensagens e tópicos guardados, se existirem guarda. Após isso cria o pipe para receber informação, liga no modo O_RDWR(escrita e leitura) de modo a não ficar bloqueado quando não tem utilizadores, apesar de nunca escrever.

```
fd_manager_pipe = open(ManPipe, O_RDWR); //aberto para leitura e escrita para nao ficar preso
if (fd_manager_pipe == -1) {
    perror("Erro ao abrir o FIFO");
    return -1;
}
printf("FIFO '%s' aberto com sucesso.\n", ManPipe);
```

Após, cria 2 threads, uma para leitura dos pipes e a outra para decrementar as mensagens.

```
// Prepara os dados para a thread ler o pipe
thread_data[0].fd_manager_pipe = fd_manager_pipe;
thread_data[0].manager = &manager;
thread_data[0].m = &mutex;
thread_data[0].trinco = 0;

// Cria a thread para ler do pipe
if (pthread_create(&tid_pipe[0], NULL, ler_pipe, (void *)&thread_data[0]) != 0) {
    perror("Erro ao criar thread para ler do pipe");
    close(fd_manager_pipe);
    unlink(ManPipe);
    return -1;
}

// Prepara os dados para a thread tratar das mensagens persistentes
thread_data[1].manager = &manager;
thread_data[1].m = &mutex;
thread_data[1].trinco = 0;

// Cria a thread para decrementar a duracao das mensagens persistentes
if (pthread_create(&tid_pipe[1], NULL, decrementa, (void *)&thread_data[1]) != 0) {
    perror("Erro ao criar thread para decrementar timer de mensagens");
    close(fd_manager_pipe);
    unlink(ManPipe);
    return -1;
}
```

Após isso fica em ciclo, à espera de comandos introduzidos pelo “admin”, quando este escreve “close” termina o programa.

Feed.c

O programa Feed.c é bastante simples, fazendo apenas a leitura de inputs vindas do teclado/stdin para enviar para o programa Manager, e, com uso de selects faz a leitura de mensagens vindas do Manager, sejam estas: mensagens de outros feeds ou respostas do Manager a comandos vindos do Feed.

```c
if (FD_ISSET(0, &fds)) { // teclado
    // limpa os campos para evitar lixo
    msg.corpo[0] = '\0';
    msg.comando[0] = '\0';
    msg.pid = getpid();

    fgets(corpo, sizeof(corpo) - 2, stdin);
    sscanf(corpo, "%s %[^\n]", msg.comando, msg.corpo);

    // validacao de comando
    while (is_invalid_command(msg.comando)) {
        printf("Comando inválido, tente novamente.\nCMD> ");
        fflush(stdout);
        fgets(corpo, sizeof(corpo) - 2, stdin);
        sscanf(corpo, "%s %[^\n]", msg.comando, msg.corpo);
    }



    if (strcmp(msg.comando, "exit") == 0) {
        strcpy(msg.corpo, "Eu vou sair!");
    }

    // mostra o comando introduzido
    printf("\nEnvia comando: %s\n", msg.comando);
    if(strcmp(msg.corpo,"\0")==0){}else{
        printf("Corpo da mensagem: %s\n", msg.corpo);}

    // Envia a mensagem para o manager
    if (write(fd_mngr_fifo, &msg, sizeof(msg)) == -1) {
        perror("Erro ao enviar mensagem para o servidor");
        close(fd_mngr_fifo);
        close(fd_feed_fifo);
        unlink(feedpipe_final);
        exit(1);
    }
}
```

```c
if (FD_ISSET(fd_feed_fifo, &fds)) { // pipe
    size = read(fd_feed_fifo, &msg2, sizeof(msg2));
    if (size > 0) {
        if(msg2.fg1==1){
        printf("Foste expulso pelo administrador.\n");
        fflush(stdout);
        close(fd_feed_fifo);
        close(fd_mngr_fifo);
        unlink(feedpipe_final);
        exit(0);}

        printf("\nMensagem do Manager:\n%s\n", msg2.corpo);
        printf("CMD> ");
        fflush(stdout);
    } else {
        printf("O manager foi fechado, vou encerrar.\n");
        close(fd_feed_fifo);
        close(fd_mngr_fifo);
        unlink(feedpipe_final);
        exit(0);
    }
}
```

Utils.h

O ficheiro utils.h serve para definir variáveis a usar no feed e manager, inclusão de bibliotecas e definição de estruturas.

Manager

```
typedef struct Manager man;
struct Manager{
    usr utilizadores[MAXUSERS];
    int nusers;
    int ntopicos;
    tp topicos[MAXTOPICS];
};
```

Estrutura do Manager, esta tem as estruturas relativamente aos utilizadores e aos tópicos, esta é a estrutura “principal”, já que contém as estruturas necessárias para o projeto.

User

```
typedef struct User usr;
struct User {
    char nome_utilizador[20];
    pid_t pid;
    int nsubscritos;
    Sub subscrito[MAXTOPICS];
};
```

Estrutura relativamente aos Utilizadores, esta guarda na estrutura Sub, o nome do tópico e o pid do utilizador subscrito. Mesmo que não esteja subscrito a nenhum tópico a estrutura guarda o pid do utilizador para usar também nos pipes do utilizador

Topic

```c
typedef struct Topic {
    char topico[20];
    TMensagem conteudo[MAXMSGS];
    int npersistentes;
    int lock;
    int ninscritos;
    pid_t inscritos[MAXUSERS];
} tp;
```

Estrutura onde guarda as mensagens do tópico, se este está disponível, e os pid's dos utilizadores subscritos.

Mensagem

```c
typedef struct Mensagem msg;
struct Mensagem{
    char comando[15];
    char corpo[650];
    pid_t pid;
    int fg1;
};
```

Estrutura que serve de comunicação entre os programas, os 2 programas, manager e feed, têm acesso a esta estrutura e usam-na para comunicar entre si.

TData

```c
typedef struct {
    int fd_manager_pipe;
    int trinco;
    man *manager;
    pthread_mutex_t *m;
    usr *usr;
} TData;
```

Estrutura para o uso das threads, aqui está um ponteiro para mutex, importante, para o programa manager, conseguir efetuar o processamento de 2, ou mais comandos, quer seja do pipe ou pelo admin, sem este ficar "preso"

Conclusão

O desenvolvimento deste trabalho prático permitiu a aplicação de diversos conceitos fundamentais do sistema Unix, incluindo sinais, pipes, threads e mutex's, em uma solução prática e funcional. A implementação dos programas "manager.c" e "feed.c", juntamente com os "headers" auxiliares e o arquivo "makefile", demonstrou a importância da integração entre diferentes componentes e técnicas de programação para a criação de um sistema robusto e eficiente.

A plataforma desenvolvida atendeu aos requisitos definidos, permitindo o envio e a receção de mensagens organizadas por tópicos, com persistência de dados e controle eficiente de recursos, como o número máximo de utilizadores, tópicos e mensagens. Além disso, a possibilidade de interação simultânea entre utilizadores, com suporte a múltiplas threads, garantiu a escalabilidade e o correto funcionamento da plataforma.

Por fim, este trabalho não apenas consolidou o conhecimento técnico, mas também reforçou significativamente a compreensão das capacidades e desafios do desenvolvimento em ambientes Unix.