Instituto Superior
de Engenharia
Politécnico de Coimbra

DEPARTAMENTO DE / DEPARTMENT OF
INFORMÁTICA E SISTEMAS

Sistemas Operativos 24/25
Trabalho Prático - Programação em C para UNIX

Autor / Author

Daniel Duarte Silva – a2023144551,

Engenharia Informática

Autor / Author

Guilherme Alexandre Neves Martins – a2023144573,

Engenharia Informática

Coimbra, dezembro de 2024

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

ÍNDICE

1 INTRODUÇÃO

Este trabalho apresenta o desenvolvimento de uma plataforma de comunicação para envio e receção de mensagens curtas organizadas por tópicos. O sistema é composto por dois programas: o **"feed"**, utilizado pelos utilizadores para interagir com a plataforma, e o **"manager"**, responsável por gerir a receção e distribuição das mensagens.

Os utilizadores podem subscrever tópicos para receber mensagens relacionadas com os seus interesses. Para a comunicação entre programas, são utilizados **FIFOs** (pipes nomeados), assegurando o envio e receção corretos das mensagens. O objetivo foi criar um sistema simples e funcional para a troca de mensagens organizadas.

2 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado com 7 ficheiros:

3 ficheiros ".h"

2 ficheiros ".c"

Makefile

1 ficheiro .txt (criado ou a ser criado com o nome que esta numa variável ambiente)

2.1 Explicação individual de cada ficheiro e respetivas funções

2.1.1 util.h

Esta função é responsável por todas as definições das variáveis globais (Limites definidos no enunciado) e também das estruturas do projeto (pedido, resposta, mensagensP, topicos, servidor_data)

Este código implementa a base de um servidor de mensagens baseado em tópicos, utilizando FIFOs (tubos nomeados) para comunicação entre clientes e o servidor, e threads para suportar múltiplos clientes ao mesmo tempo. As principais estruturas são:

1. pedido: Representa uma solicitação de um cliente (e.g., login, envio de mensagem, etc.).

2. resposta: Representa a resposta do servidor para o cliente.
3. mensagensP: Representa mensagens associadas a um tópico.
4. topicos: Gere a informação de um tópico, incluindo mensagens e utilizadores inscritos.
5. servidor_data: Contém os dados globais do servidor, como a lista de utilizadores, tópicos e mutexes para evitar conflitos em operações concorrentes.

Funcionamento básico:

- Os clientes enviam comandos ao servidor via FIFOs.
- O servidor processa os comandos (e.g., criar tópicos, enviar mensagens, subscrever) e responde através de FIFOs dedicados aos clientes.
- Os mutexes garantem a segurança e consistência no acesso simultâneo aos dados partilhados.

Codigo:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/select.h>
#include <pthread.h>

#define FIFO_SRV "tubo"
#define FIFO_CLI "f_%d"

// tipos
#define TYPE_LOGIN "login"
#define TYPE_MESSAGE "message"
#define TYPE_LOGOUT "logout"
#define TYPE_SUBSCRIBETOPIC "subscribe_topic"
#define TYPE_UNSUBSCRIBETOPIC "unsubscribe_topic"
#define TYPE_SHOWTOPICS "show_topics"

// tamanhos
#define TAM 20
```

```c
#define MAX_CLI 10
#define MAX_TCS 20
#define MAX_MSG_PRS 5
#define MAX_MSG_CORP 300

typedef struct{
    char type[TAM];
    char nome[TAM];
    char str[400];
    int pid;
}pedido;

typedef struct{
    char res[200];
    char type[TAM];
}resposta;

typedef struct{
    char topico[TAM];
    char nome[TAM];
    char corpo[MAX_MSG_CORP];
    int tempo;
}mensagensP;

typedef struct{
    pedido users[MAX_CLI];
    char nome[TAM];
    mensagensP msgPrs[MAX_MSG_PRS];
    int conta_msgPrs;
    int conta_user;
    int bloqueado;
}topicos;

typedef struct {
    int fd;
    pedido users[MAX_CLI];
    topicos topics[MAX_TCS];
    int conta_users;
    int conta_topics;
    pthread_mutex_t mutex_users;
    pthread_mutex_t mutex_topics;
} servidor_data;
```

2.1.2 manager.h

2.1.2.1 FIFO'S e Comunicação entre Processos (manager e feed)

As funções responsáveis pela criação e manipulação dos FIFO´s capazes de comunicar entre processos:

- construir_fifo_cliente: Gera o nome do FIFO de um cliente usando o PID.

```
void construir_fifo_cliente(char *fifo, int pid) {
    sprintf(fifo, FIFO_CLI, pid);
}
```

- abrir_fifo_cliente: Abre o FIFO para escrita, verificando se ainda existe.

```
int abrir_fifo_cliente(char *fifo) {
    if (access(fifo, F_OK) != 0) {
        fprintf(stderr, "FIFO do cliente %s não existe mais\n", fifo);
        return -1;
    }

    int fd = open(fifo, O_WRONLY);
    if (fd == -1) {
        fprintf(stderr, "Falha ao abrir FIFO do cliente %s para escrita!\n",
fifo);
    }
    return fd;
}
```

- enviar_resposta: Envia uma mensagem (tipo resposta) para o FIFO do cliente.

```
int enviar_resposta(char *fifo, resposta *resp) {
    int fd = abrir_fifo_cliente(fifo);
    if (fd == -1) {
        return -1;
    }

    int r = write(fd, resp, sizeof(resposta));
    if (r != sizeof(resposta)) {
        fprintf(stderr, "Erro ao enviar resposta para o cliente\n");
    }
    close(fd);
    return r;
}
```

2.1.2.2 Manipulação de Utilizadores:

Neste tópico existe um conjunto de funções responsáveis por gerirem os utilizadores conectados ao manager e estes são:

- mostrar_users: Mostra todos os utilizadores online (nome e PID).

```
void mostrar_users(pedido *users, int conta_users) {
    if(conta_users > 0){
        printf("Utilizadores Online:\n");
        for (int i = 0; i < conta_users; i++) {
            printf("\t(NOME) %s  , (NºPROCESSO) %d \n", users[i].nome,
users[i].pid);
        }
    }else{
        printf("[SEM UTILIZADORES REGISTADOS]\n");
    }
}
```

- procuraUser: Busca um utilizador pelo nome e retorna o índice na lista.

```
int procuraUser(pedido *users, int conta_users, char *nome) {
    int pid;
    if(strcmp(nome, "") == 0 ){
        printf("[COMANDO INCOMPLETO]\n");
        return -1;
    }
    for (int i = 0; i < conta_users; i++) {
        if (strcmp(users[i].nome, nome) == 0) {
            return i;
        }
    }
    printf("[SERVIDOR] -> UTILIZADOR NAO EXISTENTE\n");
    return -1;
}
```

- user_login: Adiciona um utilizador à lista se não existir e se houver espaço.

```
void user_login(pedido *users, int *conta_users, pedido p, resposta *resp){
    int flag = 0;
    for(int i = 0 ; i < *conta_users ; i++){
        if(strcmp(users[i].nome, p.nome) == 0){
            flag = 1;
            break;
        }
    }

    if(flag == 0 && *conta_users < MAX_CLI){
        users[(*conta_users)++] = p;
        strcpy(resp->res , "REGISTADO");
        strcpy(resp->type, "");
    }else{
        strcpy(resp->res, flag ? "JA EXISTE" : "LIMITE DE USERS CHEIO");
        strcpy(resp->type , TYPE_LOGOUT);
    }
}
```

- remover_cliente: Remove um cliente da lista de utilizadores e tópicos em que está inscrito, enviando notificações para outros utilizadores.

```
void remover_cliente(topicos* topics, pedido* users, int* conta_users, int*
conta_topics, char* nome, resposta* resp) {

    int i, j, k, fd_cli;
    char fifo_cli[20];

    i  = procuraUser(users, (*conta_users) , nome);
    snprintf(resp->res, sizeof(resp->res), "[NOTIFICAÇÃO] Utilizador '%s' saiu
da plataforma.", nome);

    for (int u = 0; u < (*conta_users); u++) {
        if(users[u].pid != users[i].pid){
             construir_fifo_cliente(fifo_cli, users[u].pid);
             fd_cli = abrir_fifo_cliente(fifo_cli);
             enviar_resposta(fifo_cli, resp);
        }

    }

    // Remover o cliente da lista de utilizadores
    strcpy(resp->res, "[LOGOUT] concluído!");
    strcpy(resp->type, TYPE_LOGOUT);
    if (i != -1 && i >= 0) {
        users[i] = users[(*conta_users) - 1];
        memset(&users[(*conta_users) - 1], 0, sizeof(pedido));
        (*conta_users)--;
    }

    // Remover o cliente dos tópicos em que está inscrito
    for (j = 0; j < *conta_topics; j++) {
        k = procuraUserTopico(topics[j], nome);
        if(k != -1 && k >= 0){
           topics[j].users[k] = topics[j].users[topics[j].conta_user - 1];
           memset(&topics[j].users[topics[j].conta_user - 1], 0,
sizeof(pedido));
           topics[j].conta_user--;
        }

        if(topics[j].conta_user == 0 && topics[j].conta_msgPrs == 0){
            printf("[TOPICO DESTRUIDO] %s \n" , topics[i].nome);
            topics[i] = topics[*conta_topics - 1];
            memset(&topics[*conta_topics - 1], 0, sizeof(topicos));
            (*conta_topics)--;
        }
    }
}
```

- remover_todos_clientes: Desconecta todos os utilizadores, removendo seus FIFOs.

```
void remover_todos_clientes(pedido *users, int *conta_users, resposta resp) {
    int fd_cli;
    char fifo_cli[TAM];
    strcpy(resp.res, "[SERVIDOR ENCERROU]");
    strcpy(resp.type, TYPE_LOGOUT);
    for (int i = 0; i < *conta_users; i++) {
        construir_fifo_cliente(fifo_cli, users[i].pid);
        fd_cli = abrir_fifo_cliente(fifo_cli);
        enviar_resposta(fifo_cli, &resp);
        close(fd_cli);
        unlink(fifo_cli);
    }
    *conta_users = 0; // Atualiza a quantidade de utilizadores para 0
}
```

2.1.2.3 Manipulação de Tópicos

Neste tópico há um conjunto de funções responsáveis pelo gerenciamento dos tópicos e os respetivos utilizadores associados (adicionar/remover…)

- mostrar_topics: Lista todos os tópicos com informações sobre bloqueio e número de utilizadores.

```
void mostrar_topics(topicos *topics , int conta_topics){
    if(conta_topics > 0){
        printf("Topicos:\n");
        for(int i = 0 ; i < conta_topics ; i++){
            printf("\t(TOPICO %d:) %s",i , topics[i].nome);
            if(topics[i].bloqueado){
                printf("\t bloqueado: ativado");
            }else{
                printf("\t bloqueado: desativado");
            }
            printf("\t NºUsers: %d\n" , topics[i].conta_user);
        }


    }else{
        printf("[SEM TOPICOS REGISTADOS]\n");
    }
}
```

- procuraTopico: Busca um tópico pelo nome e retorna seu índice.

```
int procuraTopico(topicos *topics, int conta_topics, char *nome) {
    if(strcmp(nome, "") == 0 ){
        printf("[COMANDO INCOMPLETO]\n");
        return -1;
    }
    for (int i = 0; i < conta_topics; i++) {
        if (strcmp(topics[i].nome, nome) == 0) {
            return i;
        }
    }
    printf("\n[SERVIDOR] -> TOPICO NAO EXISTENTE\n");
    return -1;
}
```

- criar_topico: Cria um novo tópico (se não exceder o limite).

```
int criar_topico(topicos* topics , int* conta_topics ,const char *nome_topico)
{
    if ((*conta_topics) < MAX_TCS) {
        strncpy(topics[(*conta_topics)].nome, nome_topico, TAM - 1);
        topics[(*conta_topics)].conta_msgPrs = 0;
        topics[(*conta_topics)].conta_user = 0;
        topics[(*conta_topics)].bloqueado = 0;
        return (*conta_topics)++;
    } else {
        printf("[SERVIDOR] Limite de tópicos excedido.\n");
        return -1;
    }
}
```

- adicionar_mensagem_a_topico: Adiciona mensagens persistentes a um tópico.

```
void adicionar_mensagem_a_topico(topicos* topics,int indice_topico, mensagensP
mensagem) {
    topicos *topic = &topics[indice_topico];
    if (topic->conta_msgPrs < MAX_MSG_PRS) {
        topic->msgPrs[topic->conta_msgPrs++] = mensagem;
    } else {
        printf("[SERVIDOR] Limite de mensagens persistentes no tópico '%s'
excedido.\n", topic->nome);
    }
}
```

- bloquearTopico e desbloquearTopico: Alteram o estado de bloqueio de um tópico.

```
void bloquearTopico(topicos *topics , int pos){
    topics[pos].bloqueado = 1;
}

void desbloquearTopico(topicos *topics , int pos){
    topics[pos].bloqueado = 0;
}
```

- sair_topico: Remove um utilizador de um tópico. Se o tópico ficar vazio (sem utilizadores e mensagens), ele é excluído.

```
void sair_topico(topicos *topics, int *conta_topics, pedido p, resposta *resp)
{
    int i, j;

    for (i = 0; i < *conta_topics && strcmp(topics[i].nome, p.str) != 0; i++);
    if (i == *conta_topics) {
        strcpy(resp->type, TYPE_UNSUBSCRIBETOPIC);
        strncpy(resp->res, "[SERVIDOR] Tópico não existente\n", sizeof(resp-
>res) - 1);
    } else {
        for (j = 0; j < topics[i].conta_user &&
strcmp(topics[i].users[j].nome, p.nome) != 0; j++);
        if (j == topics[i].conta_user) {
            strcpy(resp->type, TYPE_UNSUBSCRIBETOPIC);
            strncpy(resp->res, "[SERVIDOR] Utilizador não encontrado no
tópico\n", sizeof(resp->res) - 1);
        } else {
            topics[i].users[j] = topics[i].users[topics[i].conta_user - 1];
            memset(&topics[i].users[topics[i].conta_user - 1], 0,
sizeof(pedido));
            topics[i].conta_user--;

            strcpy(resp->type, TYPE_UNSUBSCRIBETOPIC);
            strncpy(resp->res, "[SERVIDOR] Saiu do tópico\n", sizeof(resp-
>res) - 1);

            if (topics[i].conta_user == 0 && topics[i].conta_msgPrs == 0) {
                printf("[TOPICO DESTRUIDO] %s \n" , topics[i].nome);
                topics[i] = topics[*conta_topics - 1];
                memset(&topics[*conta_topics - 1], 0, sizeof(topicos));
                (*conta_topics)--;

            }
        }
    }
}
```

2.1.2.4 Mensagens Persistentes

Neste tópico temos 2 funções responsáveis pelas mensagens persistentes, são elas:

- mostrar_msg_persistente: Exibe mensagens persistentes de um tópico.

```
void mostrar_msg_persistentes(topicos topics){
    if(topics.conta_msgPrs > 0){
        printf("TOPICO %s:\n" , topics.nome);
        for(int i = 0; i <  topics.conta_msgPrs ; i++){
            printf("\t[Mensagem %d] %s\n" , i , topics.msgPrs[i].corpo);
        }
    }else{
        printf("[NAO TEM MENSAGENS PERSISTENTES]\n");
    }
}
```

- broadcast_users: Envia uma mensagem para todos os utilizadores de um tópico, exceto o emissor.

```
void broadcast_users(topicos topic , char *corpo , int pid , char* nome) {
    char fifo_cli[TAM];
    resposta resp = {0};

    for (int i = 0; i < topic.conta_user; i++) {
        if (topic.users[i].pid != pid) {
            construir_fifo_cliente(fifo_cli, topic.users[i].pid);
            sprintf(resp.res, "[%s][%s]: %s",topic.nome,nome, corpo);
            strcpy(resp.type, TYPE_MESSAGE);
            enviar_resposta(fifo_cli, &resp);
        }
    }
}
```

2.1.2.5 Comandos do Servidor

A função verificaCMD processa comandos do manager. Comandos suportados:

- users: Exibe a lista de utilizadores conectados.
- remove <nome>: Remove um utilizador específico.
- topics: Mostra os tópicos criados.
- lock<nome> e unlock<nome>: Bloqueiam/desbloqueiam um tópico.
- show<nome>: Mostra mensagens persistentes de um tópico.
- close: Fecha o servidor , devendo fechar os clientes tambem

```c
void verificaCMD(char *str, pedido *users , topicos *topics , int
*conta_users, int *conta_topics) {
    char cmd1[20] = {0};
    char cmd2[20] = {0};
    char fifo_cli[20];
    int fd_cli;
    resposta resp = {0};
    sscanf(str, "%19s %399[^\n]",cmd1, cmd2);

    if (strcmp(cmd1, "users") == 0) {
        mostrar_users(users, *conta_users);
    } else if (strcmp(cmd1, "remove") == 0) {
        int i = procuraUser(users, *conta_users, cmd2);
        if (i >=  0) {
             construir_fifo_cliente(fifo_cli, users[i].pid);
             fd_cli = abrir_fifo_cliente(fifo_cli);
             strcpy(resp.res, "[LOGOUT] concluído!");
             strcpy(resp.type, TYPE_LOGOUT);
             sleep(1);
             enviar_resposta(fifo_cli, &resp);
             remover_cliente(topics,users, conta_users,conta_topics, cmd2,
&resp);

        }else{
            printf("[UTILIZADOR NAO EXISTE!]\n");
        }
    } else if (strcmp(cmd1, "topics") == 0) {
        mostrar_topics(topics, *conta_topics);
    }else if(strcmp(cmd1 , "lock") == 0) {
        int pos = 0;
        if ((pos = procuraTopico(topics,*conta_topics, cmd2)) >= 0) {
            bloquearTopico(topics,pos);
        }
    } else if (strcmp(cmd1 , "unlock") == 0) {
        int pos = 0;
        if ((pos = procuraTopico(topics, *conta_topics, cmd2)) >= 0) {
            desbloquearTopico(topics, pos);
        }
    } else if (strcmp(cmd1 , "show") == 0){
        int pos = 0;
        if ((pos = procuraTopico(topics, *conta_topics, cmd2)) >= 0) {
            mostrar_msg_persistentes(topics[pos]);
        }
    }else {
        printf("Comando desconhecido: %s\n", cmd1);
    }
}
```

2.1.2.6 Gerenciamento de Inscrições

Neste tópico existe conjuntos de funções responsáveis pelo gerenciamento de entrada e saída de usuários nos tópicos.

- subscreve_topico: Inscreve um utilizador em um tópico. Se o tópico não existir, cria um novo.

```
void subscreve_topico(topicos *topics, int *conta_topics, pedido p, resposta
*resp) {
    int i, j;

    // Procurar pelo tópico
    for (i = 0; i < *conta_topics && strcmp(topics[i].nome, p.str) != 0; i++);
    if (i == (*conta_topics)) {
        if (*conta_topics < MAX_TCS) {
            // Criar novo tópico
            strcpy(topics[(*conta_topics)].nome, p.str);

            topics[(*conta_topics)].conta_msgPrs = 0;
            topics[(*conta_topics)].bloqueado = 0;

            // Adicionar usuário ao novo tópico
            strcpy(topics[(*conta_topics)].users[0].nome, p.nome);
            topics[(*conta_topics)].users[0].pid = p.pid;
            topics[(*conta_topics)].conta_user = 1;

            (*conta_topics)++; // Incrementar número total de tópicos

            // Resposta
            strcpy(resp->type, TYPE_SUBSCRIBETOPIC);
            strncpy(resp->res, "[SERVIDOR] Tópico criado e usuário
adicionado\n", sizeof(resp->res) - 1);
        } else {
            strcpy(resp->type, TYPE_SUBSCRIBETOPIC);
            strncpy(resp->res, "[SERVIDOR] Limite de tópicos excedido\n",
sizeof(resp->res) - 1);
        }
    } else {
        for (j = 0; j < topics[i].conta_user &&
strcmp(topics[i].users[j].nome, p.nome) != 0; j++);
        if (j == topics[i].conta_user) {
            if (topics[i].conta_user < MAX_CLI) {
                strcpy(topics[i].users[j].nome, p.nome);
                topics[i].users[j].pid = p.pid;
                topics[i].conta_user++;

                strcpy(resp->type, TYPE_SUBSCRIBETOPIC);
```

```
                strncpy(resp->res, "[SERVIDOR] Foi adicionado ao tópico já
existente\n", sizeof(resp->res) - 1);
                // Enviar mensagens persistentes ao cliente
                if (topics[i].conta_msgPrs > 0) {
                    char fifo_cli[TAM];
                    construir_fifo_cliente(fifo_cli, p.pid);
                    int fd_cli = abrir_fifo_cliente(fifo_cli);
                    if (fd_cli != -1) {
                        for (int k = 0; k < topics[i].conta_msgPrs; k++) {
                            resposta msg_persistente = {0};
                            strcpy(msg_persistente.type, TYPE_MESSAGE);
                            snprintf(msg_persistente.res,
sizeof(msg_persistente.res), "[%s][%s]
%s",topics[i].nome,topics[i].msgPrs[k].nome ,topics[i].msgPrs[k].corpo);
                            enviar_resposta(fifo_cli, &msg_persistente);
                        }
                        close(fd_cli);
                    }
                }
            } else {
                strcpy(resp->type, TYPE_SUBSCRIBETOPIC);
                strncpy(resp->res, "[SERVIDOR] Limite de usuários no tópico
excedido\n", sizeof(resp->res) - 1);
            }
        } else {
            strcpy(resp->type, TYPE_SUBSCRIBETOPIC);
            strncpy(resp->res, "[SERVIDOR] Tópico já existente e já lhe
pertence\n", sizeof(resp->res) - 1);
        }
    }
}
```

- sair_topico: Remove um utilizador de um tópico. Se o tópico ficar vazio, ele é excluído.

```
void sair_topico(topicos *topics, int *conta_topics, pedido p, resposta *resp)
{
    int i, j;

    for (i = 0; i < *conta_topics && strcmp(topics[i].nome, p.str) != 0; i++);
    if (i == *conta_topics) {
        strcpy(resp->type, TYPE_UNSUBSCRIBETOPIC);
        strncpy(resp->res, "[SERVIDOR] Tópico não existente\n", sizeof(resp-
>res) - 1);
    } else {
        for (j = 0; j < topics[i].conta_user &&
strcmp(topics[i].users[j].nome, p.nome) != 0; j++);
        if (j == topics[i].conta_user) {
            strcpy(resp->type, TYPE_UNSUBSCRIBETOPIC);
```

```
            strncpy(resp->res, "[SERVIDOR] Utilizador não encontrado no
tópico\n", sizeof(resp->res) - 1);
        } else {
            topics[i].users[j] = topics[i].users[topics[i].conta_user - 1];
            memset(&topics[i].users[topics[i].conta_user - 1], 0,
sizeof(pedido));
            topics[i].conta_user--;

            strcpy(resp->type, TYPE_UNSUBSCRIBETOPIC);
            strncpy(resp->res, "[SERVIDOR] Saiu do tópico\n", sizeof(resp-
>res) - 1);

            if (topics[i].conta_user == 0 && topics[i].conta_msgPrs == 0) {
                printf("[TOPICO DESTRUIDO] %s \n" , topics[i].nome);
                topics[i] = topics[*conta_topics - 1];
                memset(&topics[*conta_topics - 1], 0, sizeof(topicos));
                (*conta_topics)--;

            }
        }
    }
}
```

- mostrar_topicos_utilizadores: Prepara a resposta do servidor com todos os tópicos e seus detalhes (nome ,número de mensagens persistentes, estado de bloqueio).

```
void mostrar_topicos_utilizadores(topicos *topics, int conta_topics, resposta
*resp) {
    strcpy(resp->type, TYPE_SHOWTOPICS);
    strncpy(resp->res, "\n----------------\nTópicos Ativos\n----------------
\n", sizeof(resp->res) - 1);

    for (int i = 0; i < conta_topics; i++) {
        char buffer[20];
        sprintf(buffer, "%d", topics[i].conta_msgPrs);
        strncat(resp->res, topics[i].nome, sizeof(resp->res) - strlen(resp-
>res) - 1);
        strncat(resp->res, "\t", sizeof(resp->res) - strlen(resp->res) - 1);
        strncat(resp->res, "N.Mensagems Persistentes: ", sizeof(resp->res) -
strlen(resp->res) - 1);
        strncat(resp->res, buffer, sizeof(resp->res) - strlen(resp->res) - 1);
        strncat(resp->res, "\t", sizeof(resp->res) - strlen(resp->res) - 1);
        strncat(resp->res, "Estado: ", sizeof(resp->res) - strlen(resp->res) -
1);
        strncat(resp->res, topics[i].bloqueado ? "Bloqueado" : "Desbloqueado",
sizeof(resp->res) - strlen(resp->res) - 1);
        strncat(resp->res, "\n", sizeof(resp->res) - strlen(resp->res) - 1);
    }
```

```
    if (conta_topics == 0) {
        strncat(resp->res, "[SERVIDOR] Nao temos topicos registados\n",
sizeof(resp->res) - strlen(resp->res) - 1);
    }
}
```

2.1.2.7 Estruturas de Dados

As estruturas de dados apresentadas a seguir foram retiradas do ficheiro util.c já referido anteriormente.

- pedido: Representa um utilizador com nome, PID e string de comando adicional.
- resposta: Representa uma mensagem enviada pelo manager para o feed.
- tópicos: Representa um tópico com:
    - Nome do tópico.
    - Estado de bloqueio.
    - Lista de utilizadores inscritos.
    - Lista de mensagens persistentes.

2.1.3 manager.c

Pequena explicação do funcionamento cíclico deste ficheiro (main)

- **Inicialização:**
    - O servidor verifica se já existe um FIFO aberto (FIFO_SRV).
    - Cria o FIFO do servidor e abre em modo leitura/escrita.
- **Criação de Threads:**
    - handle_commands: Lida com comandos administrativos.
    - handle_requests: Processa os pedidos dos clientes.
    - handle_tempMsg: Gerencia a expiração de mensagens.
- **Encerramento:**
    - O comando "close" finaliza o servidor:
        - Remove clientes conectados.
        - Salva as mensagens persistentes no ficheiro.
        - Fecha e limpa os recursos.

2.1.3.1 Funções Implementadas

- **iniciar_manager:**
  - Inicializa o servidor e carrega mensagens persistentes de um ficheiro cujo caminho é definido pela variável de ambiente MSG_FICH.
  - Lê mensagens do ficheiro e as organiza por tópicos.
  - Cria tópicos inexistente e insere as mensagens nos tópicos correspondentes.

```
void iniciar_manager(void *arg) {
    servidor_data *servidor = (servidor_data *)arg;
    char *ficheiro_log = getenv("MSG_FICH");
    if (ficheiro_log == NULL) {
        printf("[SERVIDOR] Variável de ambiente MSG_FICH não está
definida\n");
        return;
    }

    FILE *file = fopen(ficheiro_log, "r+");
    if (file == NULL) {
        perror("[SERVIDOR] Erro ao abrir o ficheiro de log");
        return;
    }

    mensagensP mensagem;
    while (fscanf(file, "%19s %19s %d %299[^\n]", mensagem.topico,
mensagem.nome,&mensagem.tempo,mensagem.corpo) == 4) {
        int indice_topico = procuraTopico(servidor->topics, servidor-
>conta_topics, mensagem.topico);
        if (indice_topico == -1) {
            indice_topico = criar_topico(servidor->topics , &servidor-
>conta_topics ,mensagem.topico);
            if (indice_topico == -1) {
                continue;
            }
        }
        adicionar_mensagem_a_topico(servidor->topics ,indice_topico,
mensagem);
    }

    fclose(file);
    printf("[SERVIDOR] Mensagens carregadas do ficheiro.\n");
}
```

- **escrever_ficheiro:**

- Salva as mensagens persistentes no mesmo ficheiro (MSG_FICH).
- É usada para persistir o estado das mensagens antes do encerramento do servidor.

```
void escrever_ficheiro(void *arg) {
    servidor_data *servidor = (servidor_data *)arg;
    char *ficheiro_log = getenv("MSG_FICH");

    if (ficheiro_log == NULL) {
        printf("[SERVIDOR] Variável de ambiente FICHEIRO_LOG não está
definida.\n");
        return;
    }

    FILE *file = fopen(ficheiro_log, "w");
    if (file == NULL) {
        perror("[SERVIDOR] Erro ao abrir o ficheiro de log para escrita");
        return;
    }

    // Percorrer todos os tópicos e escrever as mensagens persistentes no
ficheiro
    for (int i = 0; i < servidor->conta_topics; i++) {
        for (int j = 0; j < servidor->topics[i].conta_msgPrs; j++) {
            mensagensP *msg = &servidor->topics[i].msgPrs[j];
            fprintf(file, "%s %s %d %s\n",
                    msg->topico,
                    msg->nome,
                    msg->tempo,
                    msg->corpo);
        }
    }

    fclose(file);
    printf("[SERVIDOR] Mensagens persistentes foram salvas no ficheiro
'%s'.\n", ficheiro_log);
}
```

2.1.3.2 Threads Criadas

- **handle_commands**
  - Responsável por gerenciar comandos digitados pelo administrador.
  - Comando suportado:
    - "close": Encerra o servidor, removendo clientes e limpando recursos.
  - Realiza ações no servidor usando mutexes para garantir consistência.

```
void *handle_commands(void *arg) {
    servidor_data *servidor = (servidor_data *)arg;
    char str[40];

    while (1) {
        if (fgets(str, sizeof(str), stdin) != NULL) {
            str[strcspn(str, "\n")] = '\0'; // Remover newline
            if(strcmp(str,"") == 0){continue;}
            if (strcmp(str, "close") == 0) {
                pthread_mutex_lock(&servidor->mutex_users);
                pthread_mutex_lock(&servidor->mutex_topics);
                remover_todos_clientes(servidor->users, &servidor-
>conta_users, (resposta){0});
                pthread_mutex_unlock(&servidor->mutex_topics);
                pthread_mutex_unlock(&servidor->mutex_users);
                close(servidor->fd);
                printf("[SERVIDOR] Encerrando...\n");
                break;
            }
            pthread_mutex_lock(&servidor->mutex_users);
            pthread_mutex_lock(&servidor->mutex_topics);
            verificaCMD(str, servidor->users, servidor->topics, &servidor-
>conta_users, &servidor->conta_topics);
            pthread_mutex_unlock(&servidor->mutex_topics);
            pthread_mutex_unlock(&servidor->mutex_users);

            printf("[COMANDO EXECUTADO] %s\n", str);
        }
    }
    return NULL;
}
```

- **handle_request**
    - Gerencia pedidos de clientes lidos do FIFO do servidor (servidor.fd).~
    - Suporta diferentes tipos de pedidos, como:
        - TYPE_LOGIN: Login de um cliente.
        - TYPE_SHOWTOPICS: Mostra tópicos disponíveis.
        - TYPE_SUBSCRIBETOPIC: Inscrição em um tópico.
        - TYPE_UNSUBSCRIBETOPIC: Cancelar inscrição em um tópico.
        - TYPE_LOGOUT: Logout de um cliente.
        - TYPE_MESSAGE: Postar mensagens em tópicos.

        PS: Estes tipos de pedidos podem ser encontrados no ficheiro util.h já mencionado em cima.
    - O tratamento de mensagens verifica:
        - Formato e tamanho.
        - Existência do tópico.
        - Autorização do Cliente para escrever no tópico.
    - Mensagens persistentes são salvas no tópico se tempo > 0.

```
void *handle_requests(void *arg) {
    servidor_data *servidor = (servidor_data *)arg;
    char fifo_cli[20];

    while (1) {
        resposta resp = {0};
        pedido p = {0};

        if (read(servidor->fd, &p, sizeof(p)) > 0) {
            construir_fifo_cliente(fifo_cli, p.pid);

            pthread_mutex_lock(&servidor->mutex_users);
            pthread_mutex_lock(&servidor->mutex_topics);

            if (strcmp(p.type, TYPE_LOGIN) == 0) {
                user_login(servidor->users, &servidor->conta_users, p, &resp);
            } else if (strcmp(p.type, TYPE_SHOWTOPICS) == 0) {
                mostrar_topicos_utilizadores(servidor->topics, servidor-
>conta_topics, &resp);
            } else if (strcmp(p.type, TYPE_SUBSCRIBETOPIC) == 0) {
                subscreve_topico(servidor->topics, &servidor->conta_topics, p,
&resp);
            } else if (strcmp(p.type, TYPE_UNSUBSCRIBETOPIC) == 0) {
```

```
                sair_topico(servidor->topics, &servidor->conta_topics, p,
&resp);
            } else if (strcmp(p.type, TYPE_LOGOUT) == 0) {
                remover_cliente(servidor->topics, servidor->users, &servidor-
>conta_users , &servidor->conta_topics, p.nome , &resp);
            } else if (strcmp(p.type, TYPE_MESSAGE) == 0) {
                char *topico = NULL, *tempo_str = NULL, *corpo = NULL;
                int tempo;

                // Tokenizar a string usando strtok
                topico = strtok(p.str, " ");
                tempo_str = strtok(NULL, " ");
                corpo = strtok(NULL, "");


                // Validação dos tokens
                if (!topico || !tempo_str || !corpo) {
                    printf("Erro: Formato inválido da mensagem.\n");
                    pthread_mutex_unlock(&servidor->mutex_topics);
                    pthread_mutex_unlock(&servidor->mutex_users);
                    continue;
                }

                // Converter tempo para inteiro
                tempo = atoi(tempo_str);
                if (tempo < 0) {
                    printf("Erro: Tempo inválido.\n");
                    pthread_mutex_unlock(&servidor->mutex_topics);
                    pthread_mutex_unlock(&servidor->mutex_users);
                    continue;
                }

                // Verificar limites
                if (strlen(topico) > 20 || strlen(corpo) > 300) {
                    printf("Erro: Tamanho de tópico ou corpo excede o limite
permitido.\n");
                    pthread_mutex_unlock(&servidor->mutex_topics);
                    pthread_mutex_unlock(&servidor->mutex_users);
                    continue;
                }

                if(tempo == 0 || tempo > 0){
                    int pos = procuraTopico (servidor->topics, servidor-
>conta_topics, topico);
                    if(pos >= 0){
                        int pid = procuraUserTopico(servidor->topics[pos] ,
p.nome);
                        if(pid == 0 || pid > 0){
                            if(!servidor->topics[pos].bloqueado){
```

```
                                broadcast_users(servidor->topics[pos] , corpo
, servidor->topics[pos].users[pid].pid , p.nome);
                                if(tempo > 0){
                                    mensagensP msgP = {0};
                                    strcpy(msgP.topico , topico);
                                    strcpy(msgP.nome , p.nome);
                                    strcpy(msgP.corpo , corpo);
                                    msgP.tempo = tempo;
                                    if(servidor->topics[pos].conta_msgPrs < 5){
                                        servidor->topics[pos].msgPrs[servidor-
>topics[pos].conta_msgPrs++] = msgP;
                                    }

                                }
                            }else{
                                 strcpy(resp.type, TYPE_MESSAGE);
                                 strncpy(resp.res, "[SERVIDOR] Tópico
Bloqueado\n", sizeof(resp.res) - 1);
                            }
                        }
                    }
                }

                pthread_mutex_unlock(&servidor->mutex_topics);
                pthread_mutex_unlock(&servidor->mutex_users);
                enviar_resposta(fifo_cli, &resp);
            }
        }
        return NULL;
}
```

- **handle_tempMSG**
    - Monitora mensagens persistentes e decrementa o tempo restante (tempo) de cada uma.
    - Remove mensagens cujo tempo expira (tempo<=0).
    - Executa periodicamente a cada segundo (sleep(1)).
    - Destrói o tópico quando este tiver sem utilizadores subscritos e sem mensagens persistentes

```
void *handle_tempMsg(void *arg) {
    servidor_data *servidor = (servidor_data *)arg;
    while (1) {
        pthread_mutex_lock(&servidor->mutex_topics);
        for (int i = 0; i < servidor->conta_topics; i++) {
            for (int j = 0; j < servidor->topics[i].conta_msgPrs; j++) {
                servidor->topics[i].msgPrs[j].tempo--; // Decrementar o tempo
                if (servidor->topics[i].msgPrs[j].tempo <= 0) {
```

```
                    for (int k = j; k < servidor->topics[i].conta_msgPrs - 1;
k++) {
                        servidor->topics[i].msgPrs[k] = servidor-
>topics[i].msgPrs[k + 1];
                    }
                    servidor->topics[i].conta_msgPrs--;
                    j--;
                }
            if(servidor->topics[i].conta_user == 0 && servidor-
>topics[i].conta_msgPrs == 0){
                    printf("[TOPICO DESTRUIDO] %s\n" , servidor-
>topics[i].nome);
                    servidor->topics[i] = servidor->topics[servidor-
>conta_topics - 1];
                    memset(&servidor->topics[servidor->conta_topics - 1] , 0 ,
sizeof(topicos));
                    servidor->conta_topics--;
             }


            }


        }
        pthread_mutex_unlock(&servidor->mutex_topics);
        sleep(1);
    }

    return NULL;
}
```

2.1.3.3 Sincronização

- **Mutexes** (pthread_mutex_lock e pthread_mutex_unlock):
    - Garante acesso exclusivo aos arrays compartilhados de utilizadores e tópicos.
    - Previne inconsistências causadas por múltiplas threads acessando os mesmos recursos.

2.1.3.4 Funções Auxiliares:

As funções referidas a seguir estão especificadas no ficheiro manager.h já referido acima.

- procuraTopico: Verifica se um tópico existe.
- criar_topico: Cria um novo tópico.

- adicionar_mensagem_a_topico: Insere uma mensagem no tópico correspondente.
- broadcast_users: Envia uma mensagem a todos os usuários inscritos em um tópico.
- enviar_resposta: Responde ao cliente via FIFO criado dinamicamente.

2.1.4 feed.h

2.1.4.1 Operações relacionadas com FIFO do Servidor

- verificar_fifo_servidor: Verifica se o FIFO do servidor existe.

```c
int verificar_fifo_servidor(const char *fifo_srv) {
    if (access(fifo_srv, F_OK) == -1) {
        return 0;
    }
    return 1;
}
```

- conectar_servidor: Abre o FIFO do servidor para escrita.

```c
int conectar_servidor(const char *fifo_srv) {
    int fd = open(fifo_srv, O_WRONLY);
    if (fd == -1) {
        perror("Erro ao abrir FIFO do servidor");
        return -1;
    }
    printf("Conectado ao servidor (fd = %d)\n", fd);
    return fd;
}
```

2.1.4.2 Operações Relacionadas com FIFO do Cliente

- criar_fifo_cliente: Cria um FIFO exclusivo para o cliente usando o **PID**.

```c
int criar_fifo_cliente(char *fifo_cli, size_t tamanho) {
    snprintf(fifo_cli, tamanho, FIFO_CLI, getpid());
    if (mkfifo(fifo_cli, 0600) == -1) {
        perror("Erro ao criar FIFO do cliente");
        return 0;
    }
    printf("FIFO do cliente: %s\n", fifo_cli);
    return 1;
}
```

- abrir_fifo_cliente: Abre o FIFO do cliente para leitura e escrita.

```c
int abrir_fifo_cliente(const char *fifo_cli) {
    int fd_cli = open(fifo_cli, O_RDWR);
    if (fd_cli == -1) {
        perror("Erro ao abrir FIFO do cliente");
        return -1;
    }
    printf("FIFO aberto: %s\n", fifo_cli);
    return fd_cli;
}
```

2.1.4.3 Comunicação manager-feed

- enviar_pedido: Envia um pedido ao servidor através do FIFO.

```
void enviar_pedido(int fd, pedido *p) {
    printf("pid: %d type: %s\n" , p->pid , p->type);
    int r = write(fd, p, sizeof(*p));
    if (r != sizeof(*p)) {
        fprintf(stderr, "Erro ao enviar para o servidor\n");
        exit(1);
    }
}
```

- ler_resposta: Lê a resposta do servidor pelo FIFO do cliente.

```
void ler_resposta(int fd_cli, resposta *res) {
    int r = read(fd_cli, res, sizeof(*res));
    if (r != sizeof(*res)) {
        fprintf(stderr, "Erro ao ler resposta do servidor\n");
        exit(1);
    }
}
```

2.1.4.4 Login

- realizar_login: Realiza o login do cliente no servidor.

```
void realizar_login(int fd, int fd_cli, const char *nome, const char
*fifo_cli) {
    // Criar e enviar o pedido de login
    pedido p = { .pid = getpid(), .type = TYPE_LOGIN };
    strcpy(p.nome, nome);
    enviar_pedido(fd, &p);

    // Ler a resposta do servidor
    resposta res;
    ler_resposta(fd_cli, &res);

    // Verificar a resposta
    if (strcmp(res.type, TYPE_LOGOUT) == 0) {
        fprintf(stderr, "Erro: %s\n", res.res);
        close(fd_cli);
        unlink(fifo_cli);
        exit(1);
    }

    printf("Login realizado com sucesso! Bem-vindo, %s.\n", nome);
```

```
}
```

2.1.4.5 Estruturas Utilizadas

- Pedido
- Resposta

  Ps: Estas estruturas já foram referidas em cima no ficheiro util.c

2.1.5 feed.c

O programa realiza as seguintes etapas principais:

1. Verificação do FIFO do servidor: Garante que o servidor está em execução.
2. Criação de FIFO para o cliente: Cria um FIFO exclusivo para comunicação com o servidor.
3. Realização de login: Realiza o login do cliente no servidor.
4. Loop interativo: Permite ao cliente enviar comandos e receber respostas do servidor.
5. Finalização e limpeza: Remove o FIFO do cliente e encerra a conexão.

2.1.5.1 Estrutura do Código

- Validação de argumentos
  - Verifica se o nome do utilizador foi passado como argumento ao programa.

```
    if (argc != 2) {
        fprintf(stderr, "Falta parametros! Exemplo: ./feed <nome>\n");
        return -1;
    }
```

- Verificação do Servidor
  - Usa a função verificar_fifo_servidor para checar se o servidor está ativo (FIFO do servidor existe).
  - Encerra o programa se o servidor não está disponível.

```
    if (!verificar_fifo_servidor(FIFO_SRV)) {
        fprintf(stderr, "N\u00e3o existe servidor!\n");
        return 1;
    }
```

- Conexão com o servidor
  - Conecta ao FIFO do servidor usando conectar_servidor.
  - Obtém um descritor de arquivo (fd) para o FIFO do servidor.

```
    Fd = conectar_servidor(FIFO_SRV);
    if (fd == -1) {
        return 1;
    }
```

- Criação do FIFO do Cliente
    - Cria um FIFO exclusivo para o cliente usando criar_fifo_cliente.
    - Abre o FIFO do cliente com abrir_fifo_cliente para permitir leitura e escrita.

```
    if (!criar_fifo_cliente(fifo_cli, sizeof(fifo_cli))) {
        close(fd);
        return 1;
    }

    fd_cli = abrir_fifo_cliente(fifo_cli);
    if (fd_cli == -1) {
        unlink(fifo_cli);
        close(fd);
        return 1;
    }
```

- Login do Cliente
    - Realiza o login do utilizador no servidor:
        - Envia o nome do utilizador para autenticação.
        - Recebe uma resposta indicando sucesso ou falha.

```
    strncpy(str, argv[1], sizeof(str) - 1);
    realizar_login(fd, fd_cli, str, fifo_cli);
```

2.1.5.1.1 Loop Interativo

Este é responsável por permitir que o utilizador interaja com o servidor.

- Usa select para monitorar dois eventos:
    - Comandos.
    - Mensagens recebidas do servidor pelo FIFO do cliente.

```
do {
        FD_ZERO(&fds);
        FD_SET(0, &fds);
        FD_SET(fd_cli, &fds);
        fflush(stdin);
        n = select(fd_cli + 1, &fds, NULL, NULL, NULL);
```

2.1.5.1.1.1 Entrada do Utilizador

- Lê comandos digitados pelo usuário e interpreta-os.

- Comandos suportados:
    - exit: Encerra o cliente
    - topics: Lista os tópicos disponíveis
    - subscribe <tópico>: Inscreve-se em tópico
    - msg <mensagem>: Envia uma mensagem para um tópico.

```
// Enviar pedido ao servidor
r = write(fd, &p, sizeof(p));
if (r != sizeof(p)) {
      fprintf(stderr, "[ERRO] Falha ao enviar pedido\n");
}
```

2.1.5.1.1.2 Resposta do Servidor

- Lê e interpreta a resposta do servidor:
    - Se o tipo for **logout**, encerra o cliente.
    - Caso contrário, exibe a mensagem retornada pelo servidor.

```
if (FD_ISSET(fd_cli, &fds)) {
           r = read(fd_cli, &res, sizeof(res));
           if (r == sizeof(res)) {
               if (strcmp(res.type, TYPE_LOGOUT) == 0) {
                   printf("\n%s\n", res.res);
                   break; // Encerrar cliente
               } else {
                   printf("%s\n", res.res);
               }
           }
       }
```

2.1.5.1.1.3 Finalização e Limpeza

- Fecha os descritores de arquivo.
- Remove o FIFO do cliente com unlink.

```
  printf("CLIENTE SAIU\n");
   close(fd_cli);
   unlink(fifo_cli);
   close(fd);
```

3 CONCLUSÃO

O sistema desenvolvido conseguiu atingir o objetivo de oferecer uma plataforma eficiente para o envio e receção de mensagens organizadas por tópicos. Através dos programas "feed" e "manager", os utilizadores têm a capacidade de se inscrever em tópicos e receber mensagens de forma simples e eficaz.

O uso de **FIFOs** assegurou uma comunicação eficiente e garantiu a entrega correta das mensagens. Por outro lado, a implementação de **threads** e **select** possibilitou que tanto o servidor como os utilizadores executassem várias tarefas simultaneamente, melhorando a performance e a capacidade de resposta do sistema.

Este trabalho demonstrou como conceitos fundamentais de sistemas operativos, como comunicação entre processos, **threads** e **select**, podem ser aplicados de forma eficaz no desenvolvimento de soluções funcionais para a troca de mensagens em tempo real, garantindo a gestão adequada de tópicos e a comunicação concorrente entre múltiplos utilizadores.

Instituto Superior de Engenharia
Politécnico de Coimbra