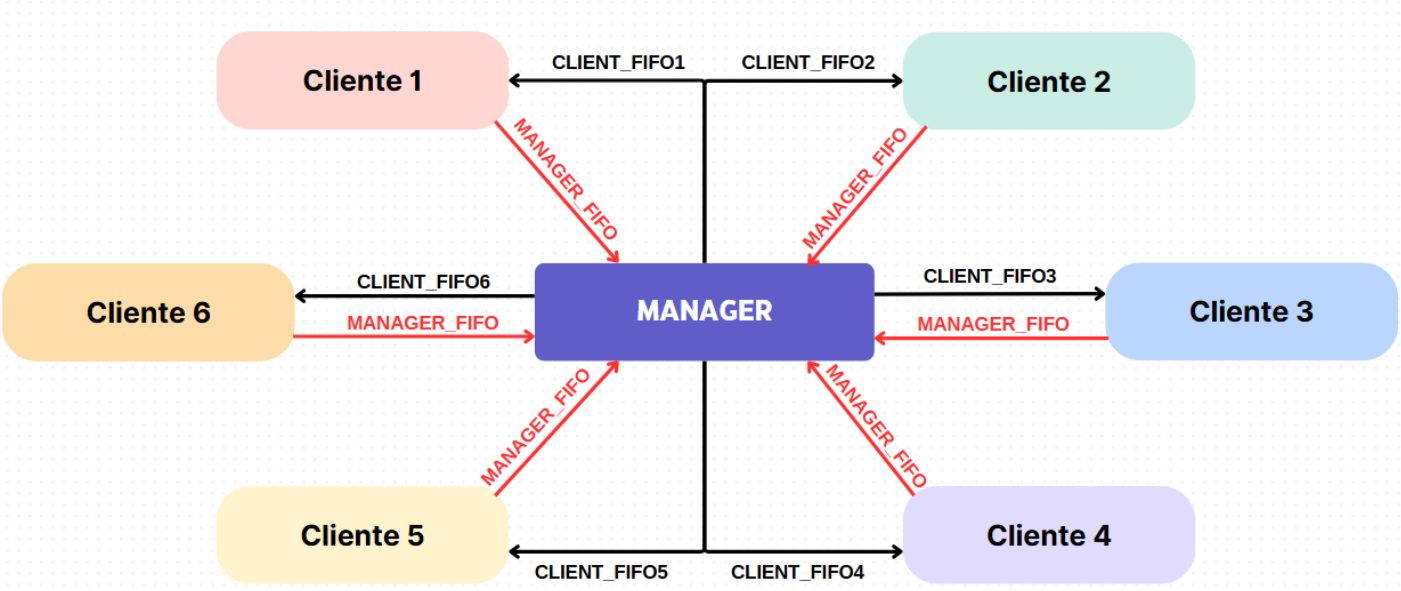
isec Engenharia

# Sistemas Operativos 24/25

## Plataforma de envio e receção de mensagens curtas através de tópicos




Trabalho realizado por:

Tiago Cabral, 2018020685

# Índice

# 1. Introdução

Este relatório apresenta o desenvolvimento de um trabalho prático realizado no âmbito da unidade curricular de Sistemas Operativos. O trabalho consistiu na implementação de uma plataforma de envio e receção de mensagens curtas, organizadas por tópicos, utilizando a linguagem de programação C e o sistema Unix (Linux). A solução desenvolvida explorou diversos mecanismos e recursos do sistema operativo abordados durante as aulas, tais como named pipes, sinais, threads, selects, etc.

A plataforma implementada é composta por dois programas principais: o manager, responsável pela gestão centralizada de tópicos, mensagens e utilizadores, e o feed, utilizado pelos clientes para interagir com a plataforma. O sistema permite a criação dinâmica de tópicos, envio de mensagens persistentes e não-persistentes, bem como a subscrição de tópicos por vários utilizadores. Para além disso, a solução integra funcionalidades para gestão de mensagens com tempo de vida definido e recuperação de mensagens persistentes através de ficheiros, garantindo uma operação robusta e eficiente.

O objetivo principal deste trabalho foi consolidar os conhecimentos sobre sistemas operativos e práticas de programação em C, promovendo o uso de APIs do sistema Unix e assegurando o funcionamento correto e ordenado dos processos e recursos envolvidos. Este relatório documenta a estratégia adotada, as decisões de implementação, a arquitetura do sistema e os desafios enfrentados durante o desenvolvimento, bem como os resultados obtidos.

# 2. Arquitetura da plataforma

A plataforma de mensagens foi projetada para permitir a interação de múltiplos clientes com um servidor, utilizando uma arquitetura cliente-servidor. O servidor, denominado manager, é responsável por gerir os tópicos criados pelos clientes, controla a subscrição dos mesmos nos respetivos tópicos, distribui as mensagens recebidas pelos clientes subscritos no respetivo tópico, armazena em memória mensagens com um determinado tempo de vida e é também o responsável por controlar a vida útil dessas mensagens. Os clientes, através do programa feed, enviam e recebem mensagens de forma interativa. A comunicação entre os clientes e o servidor é feita através de named pipes, garantindo a comunicação síncrona entre processos.

## 2.1. Estruturas utilizadas

A plataforma utiliza diversas estruturas de dados para gerir as informações de tópicos, mensagens e utilizadores.

No manager, cada tópico é representado por uma estrutura contendo o nome do tópico, um array que contém os identificadores (PID's) dos clientes subscritos, o total de clientes subscritos, três arrays para armazenar, no máximo, 5 mensagens persistentes por tópico, respetivos usernames e tempo de vida da mensagem, um contador para o número de mensagens persistentes e uma flag sobre o estado do tópico (bloqueado ou desbloqueado).

```
16
17  typedef struct {
18      char nome_topico[20];
19      int pid_clientes[10];
20      int numClientes;
21      char msg_persistentes[5][300];
22      int tempo[5];
23      char usernames[5][20];
24      int numPersistentes;
25      int bloqueado;
26  } Topico;
27
```

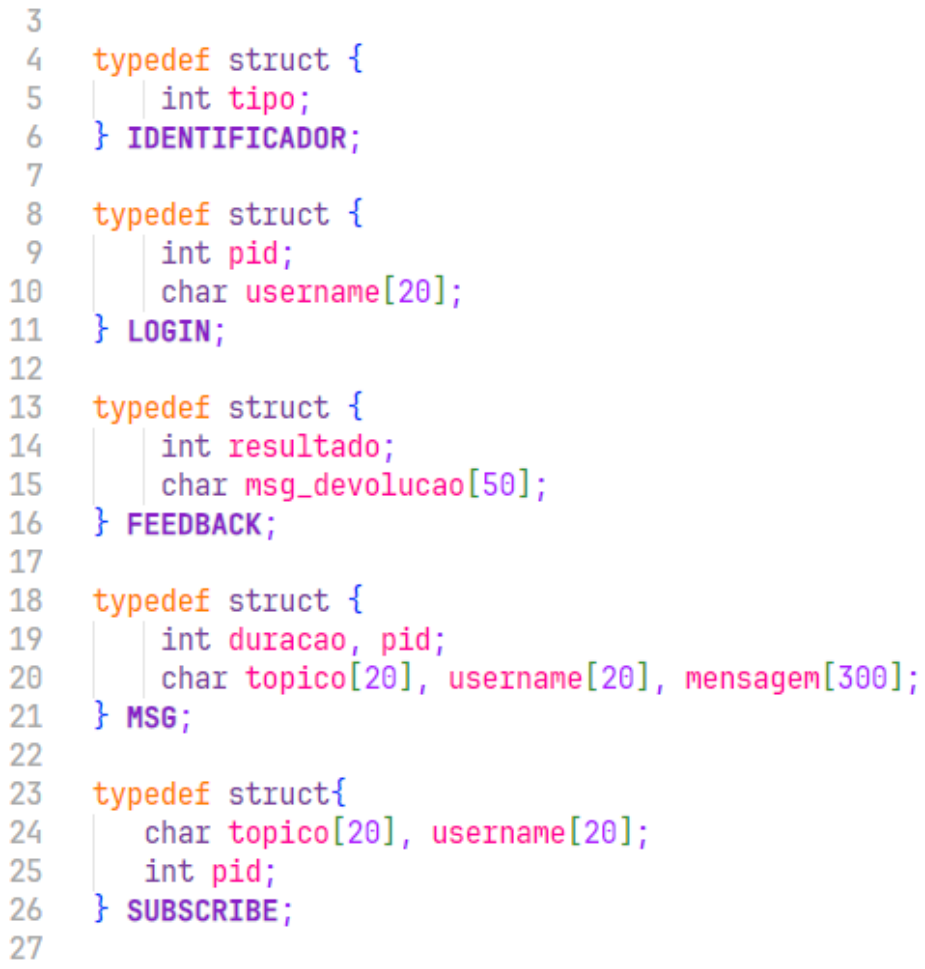
Figura 2.1 – Estrutura "Topico"

A estrutura “ServerData” armazena todos os tópicos, contém um contador para os mesmos, guarda os nomes, PID’s dos utilizadores ativos e tem um contador para os mesmos, tem um ponteiro para um mutex, útil para controlar situações de race conditions, e é constituído ainda por uma flag que irá ser usada nas threads e um inteiro que se refere ao file descriptor responsável por receber estruturas enviadas pelos clientes.

```
28
29  #define MAX_TOPICOS 20
30  #define MAX_CLIENTES 10
31  typedef struct {
32      Topico topicos[MAX_TOPICOS];
33      int numTopicos;
34      char usernames[MAX_CLIENTES][20];
35      int pids[20];
36      int numCli;
37      pthread_mutex_t *m;
38      int lock;
39      int fd;
40  } ServerData;
41
```

Figura 2.2 – Estrutura “ServerData”

Para além destas, existem outras estruturas comuns ao manager e feed. A estrutura “IDENTIFICADOR” funciona como uma flag que é enviada para o recetor da estrutura saber o que deve processar de seguida. A estrutura “LOGIN” é enviada do feed para o manager no momento da sua criação, com o objetivo de registar o mesmo na plataforma. A estrutura “FEEDBACK” serve para retornar o parecer de uma determinada ação efetuada no manager para o feed. A estrutura “MSG” é utilizada para escrever mensagens no feed e para estas serem registadas e partilhadas pelo manager. Por fim, a estrutura “SUBSCRIBE” é utilizada para subscrever um cliente num determinado tópico.



```c

typedef struct {
    int tipo;
} IDENTIFICADOR;

typedef struct {
    int pid;
    char username[20];
} LOGIN;

typedef struct {
    int resultado;
    char msg_devolucao[50];
} FEEDBACK;

typedef struct {
    int duracao, pid;
    char topico[20], username[20], mensagem[300];
} MSG;

typedef struct{
   char topico[20], username[20];
   int pid;
} SUBSCRIBE;

```

Figura 2.3 – Estruturas usadas para a comunicação entre feed-manager e manager-feed

## 2.2. Manager

O manager é o processo central que coordena a plataforma de mensagens. A sua arquitetura foi projetada para garantir o bom funcionamento da mesma, aproveitando os recursos do sistema operativo Unix. O programa foi estruturado de forma a usar threads para a execução simultânea de tarefas, garantindo a paralelização e a eficiência do sistema.

Foram utilizadas quatro threads principais no manager:

1. Processamento de inputs do utilizador: a função main é responsável por ficar à "escuta" do teclado, validando e reagindo aos comandos inseridos pelo administrador da plataforma.
2. Processamento de Named Pipes: Responsável por ler e interpretar os comandos enviados pelos clientes através de named pipes. Esta thread recebe estruturas de dados dos clientes e processa as respetivas ações, como login, envio

de mensagens, subscrição em tópicos, guarda mensagens persistentes, etc.

3. Desconto de Tempo: Esta thread gere o tempo de vida das mensagens persistentes. A cada segundo, ela desconta uma unidade ao tempo de vida das mensagens guardadas em memória e verifica se o tempo de alguma dessas mensagens expirou, removendo-as caso essa condição se verifique.
4. Gestão de Tópicos: A função desta thread é monitorizar e limpar tópicos que não tenham clientes subscritos ou mensagens persistentes. Ela verifica periodicamente, de 10 em 10 segundos, se há tópicos que podem ser removidos para liberar recursos.

O uso de pthread_mutex é fundamental para garantir a sincronização das operações, especialmente quando se lida com estruturas de dados compartilhadas entre as threads (neste caso, a estrutura ServerData), evitando “race conditions”.

## 2.3. Feed

O feed é o programa utilizado pelos clientes para interagir com a plataforma de mensagens. Foi desenvolvido para garantir uma interface simples e eficiente, utilizando select’s para a gestão de múltiplos descritores de arquivos.

O select é utilizado no programa feed para monitorizar a entrada de texto no terminal da parte do utilizador, e também as mensagens recebidas do servidor. Isso permite que o cliente leia comandos de entrada enquanto “escuta” novas mensagens do servidor de forma assíncrona, sem bloquear o processo principal.

Cada vez que o cliente executa um comando (como enviar uma mensagem, subscrever ou desinscrever de um tópico), o feed envia essa informação através de um named pipe para o manager, que então processa esse pedido. Se o cliente receber uma mensagem de outro utilizador ou do próprio servidor (por exemplo, feedback relativo ao comando enviado), ele é notificado de forma imediata.

## 2.4. Comunicação

A comunicação entre os processos (feed e manager), como já referido anteriormente, é realizada através de named pipes. O feed cria um pipe para a comunicação com o manager, com o objetivo de

receber estruturas de dados do mesmo. Cada cliente tem um named pipe exclusivo associado ao seu PID, permitindo que o servidor envie mensagens específicas para cada cliente. O manager tem o seu próprio named pipe, partilhado entre todos os clientes que o utilizam para comunicar com o manager.

- Feed: O feed envia comandos para o manager, como login, envio de mensagens, subscrições, etc., através do named pipe do manager (é utilizada a função write para escrever no mesmo). O named pipe do cliente também está ativamente à espera de que o manager envie alguma estrutura de dados.
- Manager: O manager processa os dados recebidos dos clientes através do seu named pipe, e gere tópicos. mensagens, envia as respostas de volta aos clientes, entre outras ações, através dos named pipes exclusivos para cada cliente.

Este modelo de comunicação baseado em pipes permite uma interação eficiente e síncrona entre os processos, garantindo que as mensagens sejam entregues rapidamente e sem bloqueios indesejados.

# 3. Implementação

O feed é inicializado com o nome de utilizador passado como argumento pela linha de comandos. Assim, a estratégia utilizada para efetuar o login dos utilizadores passou por enviar imediatamente a estrutura “LOGIN” preenchida com o username passado pela linha de comandos e com o PID correspondente ao utilizador ainda antes de entrar em qualquer ciclo. Após este envio, é então aberto o named pipe do cliente, cuja leitura será efetuada num select destinado a essa função, já dentro de um ciclo.

Nesse ciclo, o select responsável por “escutar” os inputs no teclado da parte do utilizador guarda numa variável “buffer” o que o utilizador escreve. A partir daí, é feita uma filtragem do que esse input significa. Para cada uma das possibilidades, o processo é sempre o mesmo: preencher a estrutura “IDENTIFICADOR” com um tipo diferente para cada situação e enviar a mesma para o manager. Isto faz com que o manager consiga identificar o tipo de estrutura que deve ler de seguida. Desde modo, o passo seguinte do feed é enviar a estrutura correspondente ao tipo preenchida para o manager poder executar as suas tarefas com toda a informação necessária para tal

```c

    if (FD_ISSET(0, &read_fds)) {
        int fd_envia = open(MANAGER_FIFO, O_WRONLY);
        if (fd_envia == -1) {
            printf("Erro ao abrir o FIFO do servidor");
            return 1;
        }
        char buffer[350];
        if (fgets(buffer, sizeof(buffer), stdin)) {
            buffer[strcspn(buffer, "\n")] = 0;// Remove o '\n' do final da string se houver

            if (sscanf(buffer, "msg %19s %d %[^\n]s", msg.topico, &msg.duracao, msg.mensagem) == 3) {
                id.tipo = 2;
                msg.pid = getpid();
                strcpy(msg.username, argv[1]);
                if (write(fd_envia, &id, sizeof(id)) == -1) {
                    printf("Erro ao escrever no FIFO do servidor\n");
                    close(fd_envia);
                    unlink(CLIENT_FIFO_FINAL);
                    return 2;
                }
                if (write(fd_envia, &msg, sizeof(msg)) == -1) {
                    printf("Erro ao escrever no FIFO do servidor\n");
                    close(fd_envia);
                    unlink(CLIENT_FIFO_FINAL);
                    return 2;
                }
            }
            else if (strcmp(buffer, "topics") == 0) {
                id.tipo = 3;
                if (write(fd_envia, &id, sizeof(id)) == -1) {
                    printf("Erro ao escrever no FIFO do servidor\n");
                    close(fd_envia);
                    unlink(CLIENT_FIFO_FINAL);
                    return 2;
                }
            }
            else if (sscanf(buffer, "subscribe %19s", sub.topico) == 1) {
                sub.pid = getpid();
                id.tipo = 4;
                if (write(fd_envia, &id, sizeof(id)) == -1) {
                    printf("Erro ao escrever no FIFO do servidor\n");
                    close(fd_envia);
                    unlink(CLIENT_FIFO_FINAL);
                    return 2;
                }
                if (write(fd_envia, &sub, sizeof(sub)) == -1) {
                    printf("Erro ao escrever no FIFO do servidor\n");
                    close(fd_envia);
                    unlink(CLIENT_FIFO_FINAL);
                    return 2;
                }
            }
            else if (sscanf(buffer, "unsubscribe %19s", sub.topico) == 1) {
                id.tipo = 5;
                sub.pid = getpid();
                if (write(fd_envia, &id, sizeof(id)) == -1) {
                    printf("Erro ao escrever no FIFO do servidor\n");
                    close(fd_envia);
                    unlink(CLIENT_FIFO_FINAL);
                    return 2;
                }
                if (write(fd_envia, &sub, sizeof(sub)) == -1) {
                    printf("Erro ao escrever no FIFO do servidor\n");
                    close(fd_envia);
                    unlink(CLIENT_FIFO_FINAL);
                    return 2;
                }
            }
            else if (strcmp(buffer, "exit") == 0) {
                id.tipo = 6;
                if (write(fd_envia, &id, sizeof(id)) == -1) {
                    printf("Erro ao escrever no FIFO do servidor\n");
                    close(fd_envia);
                    unlink(CLIENT_FIFO_FINAL);
                    return 2;
                }
                if (write(fd_envia, &login, sizeof(login)) == -1) { //info necessaria para saber qual remover
                    printf("Erro ao escrever no FIFO do servidor\n");
                    close(fd_envia);
                    unlink(CLIENT_FIFO_FINAL);
                    return 2;
                }
                printf("\nA encerrar cliente...\n");
                fflush(stdout);
                flag = 1;
                sleep(1);
            }
            else {
                printf("Comando inválido. Tente novamente.\n");
                continue;
            }
        }
    }
```

Figura 3.1 – Feed: select responsável por receber inputs do teclado

O manager, tal como anteriormente mencionado, tem uma thread cuja responsabilidade é receber e processar todas as estruturas de dados recebidas no seu named pipe, chamada “processaNamedPipes”.

A lógica desta thread é bastante simples: primeiro é recebida a estrutura “IDENTIFICADOR”. Essa estrutura é lida, e consoante o número inteiro que a variável “id” contenha é decidido o que irá ser feito. Assim, através de um switch, o manager processa um novo login, uma nova mensagem, uma nova subscrição, entre outros, chamando as funções necessárias para cada um dos casos.

```
68  void* processaNamedPipes(void* aux) {
69      ServerData* serverData = (ServerData*)aux;
70
71      int size;
72      IDENTIFICADOR id;
73      int flag1 = 0, flag2 = 0;
74
75      while (serverData->lock == 0) {
76          size = read(serverData->fd, &id, sizeof(id));
77          if (size > 0) {
78           switch(id.tipo){
79              case 1: {
80                 LOGIN login;
81                 read(serverData->fd, &login, sizeof(login));
82                 novoLogin(&login, serverData);
83                 break;
84              }
85              case 2: {
86                 MSG msg;
87                 SUBSCRIBE sub = {0};
88                 read(serverData->fd, &msg, sizeof(msg));
89                 flag1 = analisaTopico(&msg, serverData);
90
91                 if (flag1 == 0){
92                    flag2 = guardaPersistentes(&msg, serverData);
93
94                    if (flag2 == 0){
95                       subscreveCliente(&msg, &id, &sub, serverData);
96                       distribuiMensagem(&msg, serverData);
97                    }
98                 }
99                 break;
100             }
101             case 3: {
102                char nome[20] = "\0";
103                mostraTopicos(serverData, nome);
104                break;
105             }
106             case 4: {
107                MSG msg = {0};
108                SUBSCRIBE sub;
109                read(serverData->fd, &sub, sizeof(sub));
110                subscreveCliente(&msg, &id, &sub, serverData);
111                break;
112             }
113             case 5:{
114                SUBSCRIBE sub;
115                read(serverData->fd, &sub, sizeof(sub));
116                unsubscribe(&sub, serverData);
117                break;
118             }
119             case 6: {
120                LOGIN login;
121                read(serverData->fd, &login, sizeof(login));
122                int flag = 0;
123                apagaUsername(login.username, serverData, flag);
124                break;
125             }
126             case 7: {
127                LOGIN login;
128                read(serverData->fd, &login, sizeof(login));
129                int flag = 1;
130                apagaUsername(login.username, serverData, flag);
131                break;
132             }
133             default:
134                printf("\nTipo n existe");
135          }
136       }
137    }
138    close(serverData->fd);
139    unlink(MANAGER_FIFO);
140    return NULL;
141 }
```

Figura 3.2 - Manager: thread “processaNamedPipes”

É importante assinalar que a lógica de criação de tópicos foi definida para criar um tópico quando é enviada a primeira mensagem para esse tópico. Deste modo, quando o cliente envia um tipo=2 (corresponde a uma mensagem), é chamada a função “analisaTopico” no manager. Esta é a função responsável por verificar se o tópico já existe ou não. Caso não exista, se houver espaço em memória para criar um tópico, esse tópico é criado. Esta lógica implica que, caso o cliente faça “subscribe” de um tópico que não exista, é prontamente notificado que esse tópico ainda não foi criado.

Várias destas funções, consoante o sucesso ou insucesso das mesmas, devolvem feedback aos clientes. Funções como “subscreveCliente” ou “distribuiMensagem”, enviam múltiplas mensagem persistentes no momento da subscrição (caso existam) ou distribuem uma nova mensagem recebida por todos os clientes subscritos no tópico dessa mesma mensagem, respetivamente.

A lógica do envio destas informações é exatamente a mesma do feed para o manager: primeiro é enviada a estrutura “IDENTIFICADOR” e, de seguida, é enviada a estrutura corresponde à situação. Como consequência, o feed tem de estar pronto para receber e processar estas imações. Como é possível verificar no excerto de código abaixo, o segundo select no feed é responsável por receber as estruturas enviadas pelo manager. À semelhança da thread “processaNamedPipes” no programa manager, primeiro é recebida a estrutura “IDENTIFICADOR” e, a partir daí, são lidas as estruturas correspondentes a cada tipo.

```
224     //Escuta named pipe
225     else if(FD_ISSET(fd_recebe, &read_fds)){
226        int size = read(fd_recebe, &id, sizeof(id));
227        if (size > 0) {
228           switch(id.tipo){
229              case 1:{
230                 FEEDBACK feedback;
231                 read(fd_recebe, &feedback, sizeof(feedback));
232                 printf("%s", feedback.msg_devolucao);
233                 if(feedback.resultado == 0){
234                    close(fd_recebe);
235                    close(fd_envia);
236                    unlink(CLIENT_FIFO_FINAL);
237                    return 0;
238                 }
239                 break;
240              }
241              case 2:{
242                 MSG msg;
243                 read(fd_recebe, &msg, sizeof(msg));
244                 printf("\nNova mensagem! user:[%s] topico:[%s] msg:[%s]", msg.username, msg.topico, msg.mensagem);
245                 fflush(stdout);
246                 break;
247              }
248              case 3: {
249                 FEEDBACK feedback;
250                 read(fd_recebe, &feedback, sizeof(feedback));
251                 printf("\n%s", feedback.msg_devolucao);
252                 fflush(stdout);
253                 break;
254              }
255              case 4:{
256                 printf("\nManager encerrado. A encerrar cliente...\n");
257                 fflush(stdout);
258                 flag = 1;
259                 sleep(1);
260                 break;
261              }
262              case 5:{
263                 printf("\nRemovido pelo administrador. A encerrar cliente...\n");
264                 fflush(stdout);
265                 flag = 1;
266                 sleep(1);
267                 break;
268              }
269           }
270        }
271        printf("\n");
272     }
273  } while (flag == 0);
```

Figura 3.3 - Feed: select responsável pela leitura do named pipe do cliente

No manager, a thread main funciona de forma muito semelhante ao select responsável por receber inputs do teclado do programa feed, ilustrado na figura 2.4. Nessa thread, é possível remover utilizadores da plataforma, cujo feed é notificado e encerrado, e os outros utilizadores conectados também são notificados que o utilizador foi desconectado. Para além disso, é possível mostrar todos os tópicos ativos, mostrar um determinado tópico, bloquear e desbloquear um determinado tópico (bloqueia o envio de mensagens, mas ainda é possível um cliente subscrever e anular a sua subscrição nesse tópico), mostrar todos os utilizadores ativos e, por fim, fechar o manager, notificando e encerrando todos os clientes.

Outra funcionalidade importante do manager passa por guardar todas as mensagens persistentes num ficheiro de texto, que está na variável ambiente "MSG_FICH", antes do manager ser encerrado. Esta feature é cumprida através da função "guardaPersistentesFicheiro" mostrada abaixo. Primeiro, é feita uma verificação para a existência dessa variável ambiente. De seguida, é aberto o ficheiro para escrita e, por fim, são escritas as mensagens (1 por linha) com o seguinte formato: *<nome do tópico> <username do autor> <tempo de vida restante> <corpo da mensagem>.*

```
812  void guardaPersistentesFicheiro(ServerData* serverData) {
813      char *nome_ficheiro = getenv("MSG_FICH");
814      if (nome_ficheiro == NULL) {
815          fprintf(stderr, "Erro: Variável de ambiente MSG_FICH não definida.\n");
816          return;
817      }
818
819      FILE *f = fopen(nome_ficheiro, "w");
820      if (f == NULL) {
821          perror("Erro ao abrir o ficheiro para escrita");
822          return;
823      }
824
825      for (int i = 0; i < serverData->numTopicos; i++) {
826        for (int j = 0; j < serverData->topicos[i].numPersistentes; j++){
827          if (serverData->topicos[i].tempo[j] > 0) {
828              fprintf(f, "%s %s %d %s\n",
829                      serverData->topicos[i].nome_topico,
830                      serverData->topicos[i].usernames[j],
831                      serverData->topicos[i].tempo[j],
832                      serverData->topicos[i].msg_persistentes[j]);
833          }
834        }
835      }
836      fclose(f);
837      printf("Mensagens persistentes guardadas em %s.\n", nome_ficheiro);
838  }
```

Figura 3.4 – Manager: função que guarda mensagens persistentes em ficheiro de texto

No início do programa manager, é sempre chamada a função "recuperaPersistentesFicheiro", que abre o ficheiro de texto para leitura e armazena as mensagens persistentes em memória. Naturalmente, na primeira vez que o manager é lançado o ficheiro de texto está vazio, mas quando é reaberto verifica sempre se existem mensagens para guardar em memória e efetua essa ação.

A thread "descontaTempo", de 1 em 1 segundo, percorre todos os tópicos existentes na plataforma e, se houver mensagens persistentes, desconta o seu tempo numa unidade. Para além disso, faz uma verificação se o tempo chegou a 0. Caso essa condição se verifique, essa mensagem é eliminada e utiliza-se a estratégia de mover todas as mensagens, usernames e tempo uma posição para a esquerda nos seus arrays correspondentes, decrementando também o contador de mensagens persistentes por tópico.

```
143 void* descontaTempo(void *aux){
144     ServerData *serverData = (ServerData *) aux;
145     int i, j, k;
146     while (serverData->lock == 0){
147         sleep(1);
148         pthread_mutex_lock(serverData->m);
149         for (i = 0; i < serverData->numTopicos; i++) {
150             if (serverData->topicos[i].numPersistentes > 0){
151                 for (j = 0; j < serverData->topicos[i].numPersistentes; j++){
152                     serverData->topicos[i].tempo[j]--;
153                     if (serverData->topicos[i].tempo[j] == 0){
154                         for (k=j; k < 4; k++){
155                             serverData->topicos[i].tempo[k] = serverData->topicos[i].tempo[k+1];
156                             strcpy(serverData->topicos[i].msg_persistentes[k], serverData->topicos[i].msg_persistentes[k+1]);
157                             strcpy(serverData->topicos[i].usernames[k], serverData->topicos[i].usernames[k+1]);
158                         }
159                         strcpy(serverData->topicos[i].msg_persistentes[k], "\0");
160                         strcpy(serverData->topicos[i].usernames[k], "\0");
161                         serverData->topicos[i].tempo[k] = 0;
162                         serverData->topicos[i].numPersistentes--;
163                     }
164                 }
165             }
166         }
167         pthread_mutex_unlock(serverData->m);
168     }
169 }
```

Figura 3.5 – Manager: thread "descontaTempo"

Por fim, a thread "gereTopicos" verifica se os tópicos têm utilizadores subscritos e/ou mensagens persistentes associadas. Caso contrário, esse tópico é eliminado da plataforma, possibilitando a criação de novos tópicos. Esta thread é importante tendo em conta o limite de 5 tópicos em memória no manager.

```c
void* gereTopicos(void* aux) {
    ServerData* serverData = (ServerData*)aux;
    int i, j;
    while (serverData->lock == 0){
      if (serverData->numTopicos > 0){
         sleep(10);
         for (i = 0; i < serverData->numTopicos; i++) {
            if (serverData->topicos[i].numClientes == 0 && serverData->topicos->numPersistentes == 0){
               pthread_mutex_lock(serverData->m);
               for (int j = 0; j < serverData->numTopicos; j++){
                  serverData->topicos[j] = serverData->topicos[j+1];
               }
               serverData->numTopicos--;
               pthread_mutex_unlock(serverData->m);
            }
         }
      }
    }
}
```

Figura 3.6 - Manager: thread “gereTopicos”

# 4. Conclusão

O desenvolvimento desta plataforma de envio e receção de mensagens proporcionou uma experiência valiosa na utilização dos recursos do sistema operativo Unix, como named pipes e threads, select’s e sinais. Através da implementação do programa manager, foi possível gerir múltiplos tópicos, clientes e mensagens de forma eficiente, utilizando sincronização e controle de tempo de vida das mensagens persistentes.

O uso de select no programa feed possibilitou uma interface interativa e responsiva para os utilizadores, permitindo que eles enviassem mensagens e se inscrevessem em tópicos enquanto recebiam notificações do manager. A implementação foi bem-sucedida ao garantir que múltiplos clientes pudessem interagir simultaneamente sem interferências ou race conditions, graças ao uso de mutexes e threads no manager.

A comunicação através de named pipes foi crucial para permitir a comunicação eficiente entre o servidor e os clientes, assegurando que as mensagens fossem entregues de forma ordenada e sem perdas.

Em suma, a plataforma atingiu os objetivos propostos, oferecendo uma solução robusta e eficiente para a troca de mensagens em tempo real, adequada para um ambiente de múltiplos utilizadores e tópicos.