isec Engenharia

LICENCIATURA ENGENHARIA
INFORMÁTICA - CURSO EUROPEU

RELATÓRIO
SISTEMAS OPERATIVOS

PLATAFORMA DE MENSAGENS

PROJETO REALIZADO POR:
ANTÓNIO DOMINGOS GONÇALVES PEDROSO - 2021132042

ANO LETIVO 2024/2025

Conteúdos

Introdução

No âmbito da unidade curricular de Sistemas Operativos, foi proposto o desenvolvimento de uma plataforma de mensagens distribuídas, utilizando técnicas de programação em ambiente Unix/Linux. O presente projeto teve como objetivo a implementação de um sistema de comunicação em tempo real, onde múltiplos utilizadores podem enviar e receber mensagens organizadas por tópicos.

O objetivo central do projeto foi criar uma solução que demonstrasse a aplicação prática de conceitos fundamentais de sistemas operativos, nomeadamente a comunicação entre processos, gestão de recursos do sistema, named pipes e sincronização de processos concorrentes. A plataforma foi concebida para permitir uma interação simples e eficiente entre utilizadores, com funcionalidades como subscrição de tópicos e envio de mensagens persistentes e não persistentes.

Os dois principais componentes deste projeto (manager e feed) foram desenvolvidos para trabalhar em conjunto, proporcionando uma experiência de comunicação que simula sistemas de mensagens distribuídos em ambiente de linha de comandos. O programa manager atua como o núcleo do sistema, sendo responsável pela gestão de tópicos, utilizadores, distribuição de mensagens e interface para o administrador, enquanto o feed serve como interface para os utilizadores interagirem com a plataforma.

No decorrer deste relatório, serão apresentados os detalhes técnicos da implementação e as soluções implementadas para garantir o funcionamento correto e eficiente da plataforma.

Objetivos dos Programas

Manager

O programa manager representa o núcleo central da plataforma, com os seguintes objetivos principais:

1. **Gestão de recursos do sistema**
   a. Controlar o número máximo de utilizadores
   b. Gerir o número máximo de tópicos
   c. Limitar o número de mensagens persistentes por tópico
2. **Coordenação de comunicações**
   a. Intermediar todas as comunicações entre diferentes instâncias do feed
   b. Distribuir mensagens para todos os utilizadores subscritos num determinado tópico
   c. Garantir a entrega correta e eficiente de mensagens
3. **Gestão de mensagens**
   a. Processar mensagens persistentes e não persistentes
   b. Controlar a duração das mensagens persistentes
   c. Armazenar e recuperar mensagens persistentes de um ficheiro de texto
   d. Descartar mensagens expiradas
4. **Administração do sistema**
   a. Permitir comandos de administração
   b. Bloquear e desbloquear tópicos
   c. Remover utilizadores
   d. Encerrar a plataforma de forma controlada
5. **Sincronização e comunicação**
   a. Utilizar named pipes para comunicação entre processos
   b. Gerir múltiplas conexões simultâneas de feeds
   c. Tratar situações concorrentes de forma segura

Feed

O programa feed funciona como interface de utilizador, com os seguintes objetivos principais:

1. **Interface do utilizador**
   a. Permitir a identificação de um utilizador através de um username
   b. Fornecer uma interface de linha de comandos para interação com a plataforma
   c. Processar comandos do utilizador de forma interativa
2. **Comunicação com o manager**
   a. Enviar comandos e mensagens ao manager através de named pipes
   b. Receber notificações e mensagens do manager
   c. Lidar simultaneamente com entrada do utilizador e mensagens recebidas
3. **Funcionalidades de utilizador**
   a. Subscrever e desinscrever tópicos
   b. Enviar mensagens para tópicos (persistentes e não persistentes)
   c. Listar tópicos disponíveis
   d. Receber e apresentar mensagens de tópicos subscritos
4. **Gestão de sessão**
   a. Validar a conexão inicial com o manager
   b. Reagir a eventos como bloqueio de tópicos
   c. Encerrar a sessão de forma controlada quando solicitado
5. **Robustez**
   a. Verificar a existência do manager antes de iniciar
   b. Terminar se o manager for encerrado
   c. Tratar possíveis erros de comunicação ou execução

Estruturas de Dados

Pedido

Esta estrutura é usada para representar um pedido enviado por um utilizador ao servidor.

```
typedef struct
{
    char str[BUFFER_SIZE];
    char username[MAX_USERNAME_LENGHT];
    int pid;
} Pedido;
```

Resposta

Esta estrutura é usada para representar uma resposta do servidor a um pedido de cliente.

```
typedef struct
{
    char str[BUFFER_SIZE];
} Resposta;
```

Message

Esta estrutura armazena uma mensagem enviada por um utilizador para um tópico.

```
typedef struct
{
    char username[MAX_USERNAME_LENGHT];
    int duration;
    char body[MAX_MSG_LENGTH];
} Message;
```

Topic

Esta estrutura representa um tópico onde as mensagens são enviadas e os utilizadores podem se inscrever para receber as mensagens.

```
typedef struct
{
    char topic_name[MAX_TOPIC_LENGTH];
    int locked;
    int subscriber_count;
    int subscribers[MAX_USERS];
    Message messages[MAX_PERSISTENT_MESSAGES];
    int message_count;
} Topic;
```

ServerData

Esta estrutura armazena os dados principais do servidor, incluindo informações sobre os utilizadores, tópicos e as mensagens persistentes.

```
typedef struct
{
    int fd_srv;
    pid_t feed_pids[MAX_USERS];
    char feed_usernames[MAX_USERS][MAX_USERNAME_LENGHT];
    int feed_count;
    Topic topics[MAX_TOPICS];
    int topic_count;
    pthread_mutex_t lock;
} ServerData;
```

FeedData

Esta estrutura contém os dados necessários para a comunicação de um cliente com o servidor.

```
typedef struct
{
    int fd_cli, fd_srv;
    char fifo[30];
    char username[MAX_USERNAME_LENGHT];
    Pedido p;
    Resposta r;
} FeedData;
```

TALARM

Esta estrutura é usada para passar informações relacionadas com o “alarme” (um processo que executa periodicamente para atualizar tópicos e mensagens).

```
typedef struct
{
    ServerData *sd;
    int *running;
} TALARM;
```

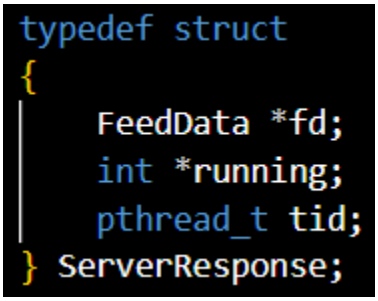
ClientRequest

Esta estrutura é usada para gerir os pedidos de clientes.

```
typedef struct
{
    ServerData *sd;
    int *running;
} ClientRequest;
```

ServerResponse

Esta estrutura é usada para gerir a resposta do servidor, permitindo a execução de threads para comunicação simultânea.



Funções

Manager

Funções principais

- void *doAlarm(void *pData);

Esta função executa uma tarefa periódica para atualizar a duração das mensagens e limpar tópicos sem mensagens ou subscritores. É executada numa thread separada enquanto a plataforma está em funcionamento.

- void *getClientRequest(void *pData);

Esta função é responsável por ler os pedidos dos clientes a partir do FIFO do servidor e processá-los. Ela é executada numa thread separada.

- int handle_manager_commands(ServerData *sd);

Função que processa os comandos do administrador via stdin. Esses comandos permitem que o administrador gira o servidor e as interações com os utilizadores.

Funcoes relacionadas com o utilizador

- void process_client_request(Pedido *p, ServerData *sd);

Processa os pedidos recebidos de um cliente, como msg, subscribe, unsubscribe.

- void notify_all_users(const char *message, ServerData *sd);

Envia uma mensagem a todos os utilizadores conectados à plataforma.

- int get_user_index(const char *username, ServerData *sd);

Retorna o índice do utilizador na lista de utilizadores conectados, dado o seu nome.

- void handle_user_removal(const char *username, int notify_users, ServerData *sd);

Remove um utilizador do servidor, desalocando os seus recursos e notificando os outros utilizadores (caso especificado).

- int remove_username(const char *username, ServerData *sd);

Remove um utilizador específico baseado no nome.

- int is_pid_duplicate(const int pid, ServerData *sd);

Verifica se o PID já está em uso na plataforma.

- int is_username_duplicate(const char *username, ServerData *sd);

Verifica se o nome já está em uso na plataforma.

- void display_connected_users(ServerData *sd);

Mostra uma lista de todos os utilizadores conectados, incluindo os nomes e os PIDs.

- int send_response_to_client(const char *fifo, const char *response);

Envia uma mensagem para um feed específico através do seu FIFO.

- int subscribe_to_topic(const char *username, Topic *topic, ServerData *sd);

Subscreve um utilizador num tópico, adicionando-o à lista de subscritores do tópico e enviando-lhe as mensagens persistentes presentes no tópico.

- int unsubscribe_from_topic(const char *username, Topic *topic, ServerData *sd);

Remove um utilizador da lista de subscritos de um tópico.

Funções relacionadas ao tópico

- void list_topics(ServerData *sd);

Mostra uma lista de tópicos disponíveis na plataforma, mostrando o seu estado, número de mensagens e número de subscritores.

- void update_topics(ServerData *sd);

Atualiza os tópicos, removendo aqueles sem mensagens ou subscritores.

- Topic *find_or_create_topic(const char *topic_name, int create_if_not_found, ServerData *sd);

Procura um tópico pelo nome. Se não for encontrado, cria um novo (caso o parâmetro create_if_not_found seja 1).

- int delete_topic(Topic *topic, ServerData *sd);

Apaga um tópico e desaloca os recursos associados ao mesmo.

- int lock_topic(Topic *topic);

Bloqueia um tópico, impedindo que novas mensagens sejam enviadas para ele.

- int unlock_topic(Topic *topic);

Desbloqueia um tópico, permitindo que novas mensagens sejam enviadas.

- void show_topic_messages(Topic *topic);

Mostra todas as mensagens de um tópico, incluindo o nome do utilizador que as enviou e a sua respetiva duração.

- void add_message_to_topic(Topic *topic, const Message *new_msg);

Adiciona uma nova mensagem a um tópico, caso o número máximo de mensagens persistentes não tenha sido atingido.

- void send_message_to_subscribers(Topic *topic, const Message *msg, ServerData *sd);

Envia uma mensagem para todos os utilizadores que estão subscritos num tópico.

- void update_message_duration(ServerData *sd);

Atualiza a duração das mensagens persistentes, removendo aquelas que expiram.

Funções auxiliares

- int load_messages_from_file(ServerData *sd);

Carrega mensagens persistentes do ficheiro definido na variável ambiente MSG_FICH, restaurando o estado anterior dos tópicos e mensagens da plataforma.

- void save_messages_to_file(ServerData *sd);

Guarda as mensagens persistentes no ficheiro, permitindo que o estado da plataforma seja restaurado numa execução futura.

- void handle_sigint(int sig);

Manipula o sinal SIGINT, permitindo que a plataforma seja apenas fechada de forma controlada e segura.

- void debug_server_state(ServerData *sd);

Mostra informações detalhadas para depuração sobre o estado da plataforma.

Feed

Funçoes principais

- void init_feed(FeedData *fd, const char *username);

Inicializia o feed para o cliente.

Funçõs de comunicação

- void create_and_open_fifos(FeedData *fd);

Cria e abre os FIFOs para comunicação entre cliente e “servidor”.

- void send_checkin(FeedData *fd);

Envia o comando CHECKIN para o servidor, efetuando o processo de login na plataforma.

- int handle_feed_commands(FeedData *fd, ServerResponse *sr);

Processa os comandos recebidos no stdin e envia-os à plataforma.

- void process_manager_request(Resposta *r, FeedData *fd, ServerResponse *sr);

Processa a resposta recebida da plataforma e mostra as mesmas.

- void *getServerResponse(void *pData);

Função executada numa thread que ouve e processa as mensagens do servidor enquanto o cliente está em funcionamento.

Funções auxiliares

void cleanup_and_exit(FeedData *fd);

Limpa os recursos alocados pelo cliente, como fechar os FIFOs e desligar-se da plataforma.

void handle_signal(int sig);

Manipula sinais, como SIGUSR1 e SIGINT, para controlar a execução do cliente.

Como Funciona a Comunicação

A comunicação entre o manager (servidor) e os feeds (clientes) é realizada através de named pipes (FIFOs), que é uma técnica comum em sistemas Unix/Linux para a comunicação entre processos. Passo a explicar o fluxo de comunicação em detalhe.

1. Criação de FIFOs

- Cada cliente (feed) cria o seu próprio FIFO único quando se liga ao servidor. Este FIFO é nomeado dinamicamente com base no PID do cliente (f_<PID>).
- O servidor (manager), por outro lado, tem um FIFO principal (f_manager) que ouve os pedidos dos feeds.

2. Fluxo de Comunicação (Cliente -> Servidor)

- Quando um cliente deseja interagir com o servidor (por exemplo, para enviar uma mensagem), ele envia um pedido através do seu FIFO dedicado para o servidor.
    - Exemplo: Um cliente envia o comando CHECKIN ao servidor com o seu nome de utilizador e PID através do seu FIFO (f_<PID>).
- O servidor lê esse pedido a partir do FIFO principal(f_manager). O servidor processa o comando e toma as ações necessárias, como verificar se o nome de utilizador já está em uso ou se o servidor está cheio.

3. Fluxo de Comunicação (Servidor -> Cliente)

- Depois de processar o comando, o servidor envia uma resposta ao cliente, indicando o resultado da operação (por exemplo, OK, #EXISTS, etc.). Esta resposta é enviada para o FIFO dedicado do cliente, garantindo que a comunicação seja isolada e personalizada para cada cliente.
    - Exemplo: O servidor pode responder com #EXISTS se o nome de utilizador já estiver em uso ou OK se o cliente realizou a conexão com sucesso.
- O cliente lê essa resposta a partir do seu FIFO e pode então tomar a ação apropriada, como exibir a mensagem ao utilizador ou executar uma nova operação, como inscrever-se em tópicos ou enviar mensagens.

4. Comunicação de Mensagens:

- Quando um cliente envia uma mensagem para um tópico, ele faz isso através do FIFO principal do servidor. O servidor, por sua vez, distribui essa mensagem para todos os clientes subscritos ao tópico, enviando a mensagem para cada um dos FIFOs dedicados de cada cliente subscrito.

    - Exemplo: Um cliente envia a mensagem "Ola" para o tópico “social". O servidor envia essa mensagem para todos os clientes subscritos ao tópico "social".
- Cada cliente subscrito recebe a mensagem do servidor através do seu FIFO e apresenta-a ao utilizador.

5. Gestão de Múltiplas Conexões e Sincronização

- O servidor deve ser capaz de gerir múltiplas conexões simultâneas de clientes.
    - A sincronização entre as threads é garantida através de mutexes (pthread_mutex_t), que são usados para proteger recursos partilhados, como a lista de tópicos e mensagens, evitando *race conditions* e garantindo a integridade dos dados.

6. Encerramento da Conexão

- Quando um cliente deseja terminar a sessão (comando exit), o cliente envia esse comando ao servidor via FIFO. O servidor então processa o pedido de logout e remove o cliente da plataforma.
    - O servidor pode então enviar uma mensagem de confirmação de logout para o cliente, indicando que a sessão foi encerrada.
- Fluxo: O cliente envia o comando sair, o servidor processa e remove o cliente, e responde ao cliente com a mensagem de encerramento.

7. Fechamento e Limpeza

- Depois de finalizar a interação com o servidor, o cliente fecha o seu FIFO e desconecta-se, enquanto o servidor mantém a sua operação até que o servidor seja encerrado.
    - Para garantir a limpeza de recursos, os FIFOs são fechados e removidos.

Conclusão

Agradeço ao professor João e a José pelo conhecimento transmitido ao longo do semestre. A orientação dada foi fundamental para o desenvolvimento deste projeto, e as explicações prestadas contribuíram para uma compreensão profunda dos conceitos envolvidos. O apoio contínuo recebido também foi essencial para a implementação do projeto.

Com este projeto, foram aplicados conceitos fundamentais de sistemas operativos. A implementação de uma plataforma de mensagens distribuídas permitiu aprofundar o conhecimento sobre o funcionamento de pipes, threads e a gestão de múltiplos utilizadores.

Além disso, foi possível aprofundar o entendimento sobre a importância de estruturas de dados eficientes para gerir recursos da plataforma, assegurando a integridade e eficiência, mesmo com a interação simultânea de vários clientes. Este projeto também proporcionou uma valiosa experiência prática em programação multithreaded e na interação cliente-servidor.

Este projeto representou uma excelente oportunidade para aplicar os conhecimentos adquiridos nas aulas, bem como os conhecimentos estudados de forma autónoma, desenvolvendo novas habilidades e aprofundando a compreensão sobre sistemas operativos.