SISTEMAS OPERATIVOS

Trabalho Prático - SOBay

Docente: João Durães

Beatriz Isabel Inácio Maia – a2020128841 - LEI

João Miguel Duarte dos Santos – a2020136093 – LEI

Coimbra, 09 de Janeiro de 2023

Índice

1. Introdução

Este relatório é realizado no âmbito da unidade curricular de Sistemas Operativos do curso de Engenharia Informática do Instituto Superior de Engenharia de Coimbra (ISEC).

Este trabalho consiste na implementação de uma plataforma para gerir leilões ("SOBay"), o qual faz a gestão da comunicação entre os clientes envolvidos, gere os itens à venda, verifica os preços e determina quem arremata os itens. Para isto o sistema abarca 3 tipos de programas principais sendo eles o frontend, backend e promotor.

2. Estruturas e Organização

Neste trabalho usámos 3 header files (ficheiros .h) sendo eles: frontend.h, backend.h e utils.h (fora o ficheiro users_lib.h que foi fornecido pelos docentes)

2.1. frontend.h

```c
typedef struct tdados
{
    int pararThread;
    int threadComecou;
    pthread_mutex_t *trinco; // partilhado entre threads

    int pid;
    char nome[TAMANHO_STRING];
    char password[TAMANHO_STRING];
}TDados;

void executaComando(Utilizador dadosUtilizador, char* comando);
void fazerPedidoLogin(Utilizador dadosUtilizador);
void fazerPedido(Utilizador dadosUtilizador, char comando[TAMANHO_STRING]);
void lerResposta(int abrirFIFO_FRONTEND);

void *threadHEARTBEAT(void * dadosThread);
```

No frontend.h incluímos uma estrutura de dados para a thread que envia um sinal de X em X segundos (definido pela variável ambiente “HEARTBEAT”) e nesta incluímos uma variável para o começo e fim desta mesma thread, um trinco, o pid, nome e password do utilizador que está “ativo” e a enviar o sinal naquele preciso momento.

Existem também declarações de variáveis usadas pelo frontend.c para que seja possivel detetar o comando digitado pelo utilizador, enviar um pedido ao backend.c e ler a resposta enviada por este último.

2.2. utils.h

```c
#define TAMANHO_STRING 150
#define TAMANHO_STRING_XXL 1000
#define MAX_UTILIZADORES 20
#define MAX_PROMOTORES 10
#define MAX_ITEMS 30

#include <pthread.h>

typedef struct utilizador{
    int pid;
    char nome[TAMANHO_STRING];        // Guardar para enviar ao backend
    char password[TAMANHO_STRING];    // Guardar para enviar ao backend
}Utilizador;

Utilizador newUtilizador;        // nova estrutura para guardar o utilizador que tenta entrar

typedef struct pedido{
    int pid;
    char nomeUtilizador[TAMANHO_STRING];
    char passwordUtilizador[TAMANHO_STRING];

    char comando[TAMANHO_STRING];
    char resposta[TAMANHO_STRING_XXL];

    int controlo;   // 1 -> ACEITE   ||   0 -> RECUSADO   ||   -1 -> ERRO DO BACKEND
}Pedido;

Pedido newPedido;
Pedido respostaPedido;
```

Este ficheiro contém a declaração de constantes usadas durante o projeto tais como o número máximo de utilizadores, número máximo de promotores e o número de caracteres numa string.

A primeira estrutura “Utilizador” inclui o pid, o nome e a password do utilizador. Existe ainda uma variável estrutura “newUtilizador” do tipo “Utilizador” que vai ser responsável por guardar a informação de um novo utilizador que tenta entrar.

A segunda estrutura “Pedido” contém informações sobre um pedido feito ao backend, respetivamente, o pid do utilizador, o nome e password do utilizador, uma string que armazena o comando (caso este digite algum), uma que armazena a resposta do backend, e a variável controlo que pode ter 3 valores conforme o pedido seja aceite, recusado ou tenha ocorrido um erro no backend.

Contém ainda duas variáveis estrutura “newPedido” e “respostaPedido” do tipo “Pedido”.

2.3. backend.h

```c
void menos1Saldo_MetaUm(char * nome);

// FUNÇÕES PARA RESPONDER FRONTEND
bool venderItem(char * nomeVendedorNovoItem, char * nomeNovoItem, char * categoriaNovoItem,
                int precoInicialNovoItem, int precoCompreJaNovoItem, int durancaoNovoItem);

void testeItemsAdmin();
void testeItems();

void testeItemsCategoria(char * categoria);
void testeItemsVendedor(char * nome);
void testeItemsPrecoMax(int valor);
void testeItemsTempoMax(int tempo);
void mostrarSaldo(char * nome);
void adicionarSaldo(char * nome, int valor);
void licitarItem(char * nome, int id, int valor);

// COMANDOS -> FUNÇÕES ADMIN
void executaComandoAdmin(char * comando);
void testePromotor();
void testeUtilizadores();

// VARIÁVEIS DE AMBIENTE
int HEARTBEAT; // Tempo de resposta para garantir que o utilizador está vivo
char FPROMOTERS[25]; // Nome do ficheiro dos promotores
char FUSERS[25]; // Nome do ficheiro dos utilizadores
char FITEMS[25]; // Nome do ficheiro dos items
```

```c
void receberPedido(int abrirFIFO_BACKEND);

void resetResposta();

void *threadTempoSegundos(void * dadosThread);
void *threadPromotores(void * dadosThread);

int maxFDAberto(int numArgs, int **args);
```

No backend.h incluímos todas as declarações de funcções usadas pelo backend.c que vão desde ler o pedido recebido pelo frontend.c e gerir a resposta à própria até executar comandos digitados pelo administrador e ler novas promoções vindas do promotor ativo.

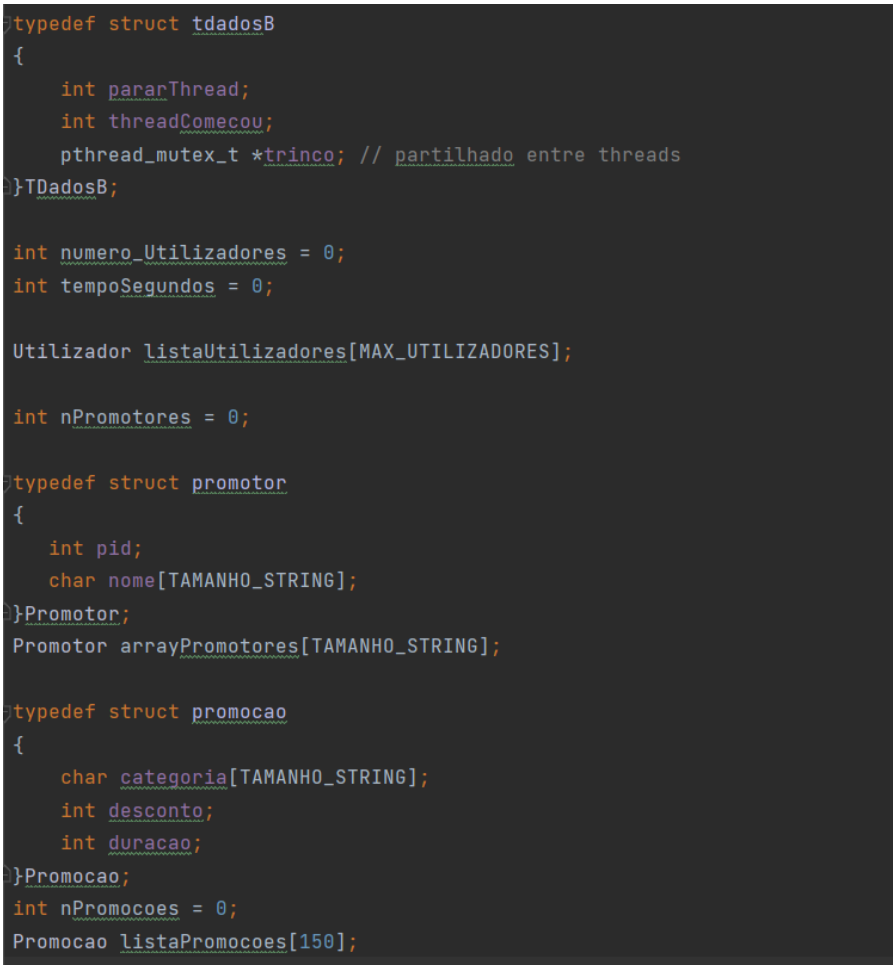
Existe também a declaração de 4 variáveis que são usadas como variáveis de ambiente, imprescindíveis ao funcionamento da aplicação.

```c
typedef struct items
{
    int id;
    int valInicial;
    int valCompreJa;
    int timeRestante;

    char nomeItem[TAMANHO_STRING];
    char nomeCategoria[TAMANHO_STRING];
    char nomeVendedor[TAMANHO_STRING];
    char nomeLicitador[TAMANHO_STRING];

    int valAntigo;
    int valNovo;
}Items;
Items listaItems[MAX_ITEMS];

int nItems = 0;
int readItems();
void saveItems();
```

Usámos uma estrutura “Items” para guardar a informação acerca de um item registado no leilão (items.txt) em que o id é único para cada item, e valInicial, valCompreJa, timeRestante, nomeItem, nomeCategoria, nomeVendedor e nomeLicitador são respetivamente o valor de licitação do item, o valor para comprar imediatamente o item, o tempo restante da disponibilidade do item no leilão, o nome do item, da sua categoria, do vendedor e do licitador (caso exista). A variável valAntigo e valNovo são referentes às promoções sendo o valAntigo o valor antes da promoção e o valNovo o novo valor do item.

Relativamente aos items existe também um array “listaItems” do tipo “Items” que como o nome indica, guarda a informação de cada item.

As funções readItems() e saveItems() são respetivamente, funções para ler o ficheiro “items.txt” e guardar os items atualmente ativos no leilão (devolve o numero de items que leu) e para guardar alterações nos items do leilão.



```c
typedef struct tdadosB
{
    int pararThread;
    int threadComecou;
    pthread_mutex_t *trinco; // partilhado entre threads
}TDadosB;

int numero_Utilizadores = 0;
int tempoSegundos = 0;

Utilizador listaUtilizadores[MAX_UTILIZADORES];

int nPromotores = 0;

typedef struct promotor
{
   int pid;
   char nome[TAMANHO_STRING];
}Promotor;
Promotor arrayPromotores[TAMANHO_STRING];

typedef struct promocao
{
    char categoria[TAMANHO_STRING];
    int desconto;
    int duracao;
}Promocao;
int nPromocoes = 0;
Promocao listaPromocoes[150];
```

Tal como no frontend.h, no backend.h temos também uma estrutura de dados para a thread (esta que vai gerir os segundos atuais desde o lançamento do backend e mecânica do tempo dos items e promocoes).

A variável “numero_Utilizadores” vai guardar o número de utilizadores atualmente ativos e a variável “tempo_Segundos” vai ser a responsável por guardar os segundos atuais desde o lançamento do backend.

Incluímos também um array “listaUtilizadores” do tipo “Utilizador” (derivada do utils.h) que serve para guardar a informação dos vários utilizadores atualmente ativos.

A variável “nPromotores” guarda o número de promotores atualmente ativos.

Existe ainda uma estrutura “Promotor” e “Promocao” que guardam respetivamente a informação de um promotor (pid e nome) e informação de uma promoção (devolvida por um promotor); guarda a categoria, desconto e duração desta.

Destas 2 estruturas surgem 2 arrays: “arrayPromotores” e “listaPromocoes” que consequentemente, guardam toda a informação de todos os promotores e promoções atualmente ativos/as.

3. Implementação

3.1. Makefile

```
default: all;

all: frontend.o backend.o
    gcc frontend.o -o frontend -l pthread
    gcc backend.o users_lib.o -o backend -l pthread

fronted.o: frontend.c
    gcc -c frontend.c

backend.o: backend.c
    gcc -c backend.c

clean:
    @echo "A limpar..."
    rm frontend backend frontend.o backend.o PIPE*
```

Para o sistema SOBay foi realizado o makefile que define um conjunto de tarefas a ser executadas aquanto da execução deste ficheiro. Serve para executar vários comandos ao mesmo tempo.

Temos a rule all que inclui a criação dos executáveis do frontend e backend e a rule clean que remove os executáveis criados assim como os pipes.

3.2. Threads

Neste projeto foram desenvolvidas 2 threads: uma para o frontend que realiza um envio de um “sinal de vida” para o backend (para que este garanta que está ainda ativo) de X em X segundos, sendo X definido pela variável ambiente “HEARTBEAT”.

A outra thread foi implementada no backend para que possa ser feita a gestão do tempo no leilão. Esta aumenta uma variável (responsável por guardar os segundos desde a primeira execução do leilão).

A utilização de threads neste projeto permite que o frontend e o backend possam executar várias tarefas de forma concorrente e assíncrona.

3.3. Selects

No código do frontend, o select é usado para permitir que o utilizador possa enviar um comando e ao mesmo tempo, ficar a aguardar uma resposta ou informação por parte do backend. O select é chamado dentro de um ciclo o que permite que o utilizador possa continuar a enviar comandos enquando não digitar “exit”.

O select vai ser usado para monitorizar se estes comandos vêm do teclado(stdin) ou do pipe frontend (que lê as respostas do backend).

No código do backend, o select é usado da mesma forma, mas para monitorizar dois pipes: o pipe do backend (que é usado para ler os pedidos dos frontends) e o unamed pipe (que é usado para ler as promoções dos promotores).

3.4. Comunicação através de named pipes

Tanto o backend como o frontend criam um named pipe, sendo o do frontend criado com um nome que inclui o seu pid, fazendo assim a distinção entre todos os utilizadores ativos.

O processo frontend envia pedidos através do pipe backend para o processo backend e o processo backend responde através do pipe frontend respetivo (usando o pid do pedido que recebeu) para o processo frontend. O processo frontend utiliza a função select() para "monitorizar" os dois named pipes ao mesmo tempo de modo a saber se há algum dado para ler ou enviar um pedido.

4. Conclusão

Podemos concluir que este trabalho foi importante para o aprofundamento do nosso conhecimento sobre este tema, uma vez que nos permitiu, através do ambiente Unix e seus comandos, e programação em C, perceber que podemos criar uma interação entre várias entidades com a comunicação entre processos através named pipes, unnamed pipes, selects e threads.