Relatório Meta 2

Trabalho Prático - Sistemas Operativos

SOBay

Licenciaturas em Engenharia Informática
Sistemas Operativos

Trabalho Realizado por:

->Tiago Quintas, 2019128044, a2019128044@isec.pt

-> Carlos Andrade, 2019141270, 2019141270@isec.pt

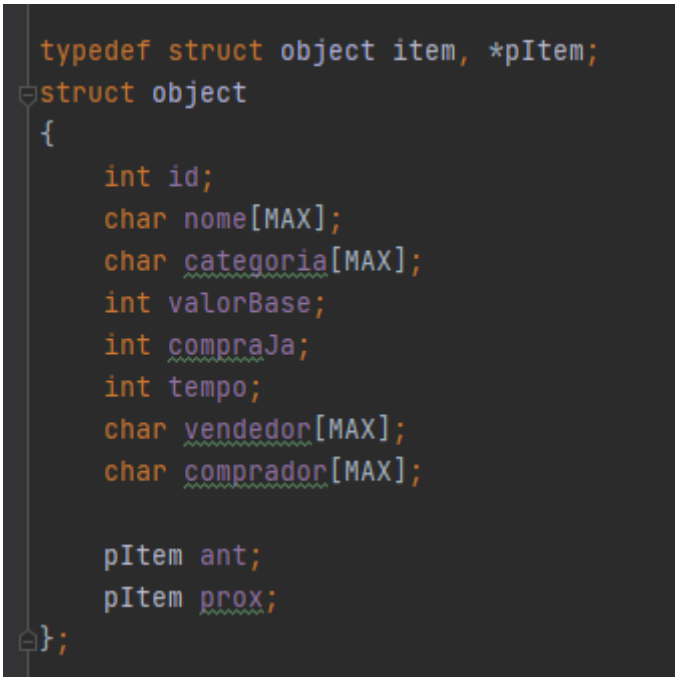
Índice

Introdução

Este relatório tem como base o trabalho prático da cadeira de Sistemas Operativos em 2022/2023, do segundo ano de Licenciatura em Engenharia Informática.
No âmbito da cadeia de Sistemas Operativos foi nos pedido um trabalho pratico com
o objetivo de implementar uma plataforma para gerir um sistema de leilões (“SOBay”) onde Neste relatório contem a informação relativa as estruturas dos vários elementos
utilizados e o código de funcionamento de cada um.

Estruturas Utilizadas

Estrutura itens, itemsList.h



```
typedef struct object item, *pItem;
struct object
{
    int id;
    char nome[MAX];
    char categoria[MAX];
    int valorBase;
    int compraJa;
    int tempo;
    char vendedor[MAX];
    char comprador[MAX];

    pItem ant;
    pItem prox;
};
```

Esta estrutura representa os itens colocados à venda pelos utilizadores. Sendo que todos itens estão armazenados são na forma de uma lista ligada.

id = Número de identificação do item

nome = nome do item

categoria = categoria da qual o item pertence

valorBase = valor base do item

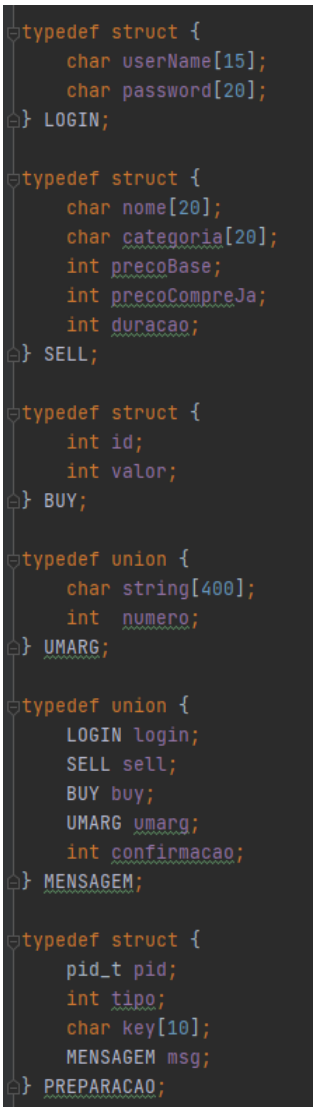
compraJa = valor para o item de imediato

tempo = tempo limite do leilão para o item

vendedor = nome do vendedor

comprador = nome de quem comprar o item

Estruturas Para Comunicação, globals.h



```c
typedef struct {
    char userName[15];
    char password[20];
} LOGIN;

typedef struct {
    char nome[20];
    char categoria[20];
    int precoBase;
    int precoCompreJa;
    int duracao;
} SELL;

typedef struct {
    int id;
    int valor;
} BUY;

typedef union {
    char string[400];
    int  numero;
} UMARG;

typedef union {
    LOGIN login;
    SELL sell;
    BUY buy;
    UMARG umarg;
    int confirmacao;
} MENSAGEM;

typedef struct {
    pid_t pid;
    int tipo;
    char key[10];
    MENSAGEM msg;
} PREPARACAO;
```

As diversas estruturas representadas acima são utilizadas para a comunicação entre o servidor e o cliente, cada uma contendo a para efetuar as operações necessárias. Estas estruturas estão contidas no globals.h, sendo utilizadas pelo frontend e o backend.

Todas as mensagens enviadas pelo servidor tem uma chave (key) associada para assegurar ao cliente que é o servidor a enviar as mesmas.

Estrutura GERAL, backend.h

```
typedef struct{
    int tempo;
    //int nPromotores;
    char CLIENT_FIFO_FINAL[100];    // CLIENTE123
    int nClientes;
    pItem itemsList;
    union sigval intMain;
    //cliente cli;
    //cliente cli[MAXCLI];    // STRUCT DE CLIENTES
    resposta rsp;   // STRUCT DE ENVIO DE DADOS
    //pthread_t clientes[MAXCLI];
    int fdServ;
    int stop;
    pthread_mutex_t *m;
} GERAL;
```

Esta estrutura contém variáveis gerais usadas por várias funções do backend.

Estrutura PROMOCAO, backend.h

```
typedef struct{
    char categoria[20];
    int desconto;
    int segundos;
    int ativo;
} PROMOCAO;
```

Esta estrutura contém a informação sobre uma promoção.

Estrutura PROMOTOR, backend.h

```
typedef struct {
    pid_t pid;
    char nome[50];
    int fdProms[2];
    int nPromocoes;
    PROMOCAO prom[MAXPROM];
    //pthread_mutex_t *mutex_promotor;
    int stop;   // SE ESTIVER EM FUNCIONAMENTO E FOR PARA O TERMINAR FICA A 1
    int ativo;  // SE ESTIVER EM FUNCIONAMENTO FICA A 1
    //GERAL geral;
} PROMOTOR;
```

Esta estrutura contém os dados relativos aos promotores.

Estrutura THREAD_PROMOTORES, backend.h

```
typedef struct {
    pthread_t tPromotor;
    PROMOTOR promotor;
    //GERAL geral;
} THREAD_PROMOTORES;
```

Estrutura utilizada para uma das etapas da execução dos promotores

Estrutura INFOS_CLIENTES, backend.h

```
typedef struct {
    pid_t pid;
    char username[MAX];
    int tempo;
    int ativo;
} INFOS_CLIENTES;
```

Estrutura que contem a informações dos clientes.

Estrutura COMANDOS, backend.h

```
typedef struct {
    pthread_t tPromotoresControl;
    THREAD_PROMOTORES prom[MAXPROM];
    pthread_t tClientes;
    pthread_t tCLientesHeartbeats;
    CLIENTES cliente;
} COMANDOS;
```

Estrutura relacionada com os comandos que permite a inicialização de certas threads.

Estrutura TEMPO, backend.h

```
typedef struct{
    int tempo;
    int stop;
    //GERAL geral;
    COMANDOS comandos;
    pthread_t tComandos;
} TEMPO;
```

Estrutura relacionada com os tempos que permite a inicialização da threat para os comandos.

Estrutura INICIALIZADOR, backend.h

```
typedef struct {
    TEMPO tempo;
    //GERAL geral;
    pthread_mutex_t mutex;
    //int stop;
} INICIALIZADOR;
```

Esta estrutura permite tempo que permite a inicialização da threat para os comandos.

Frontend.c

No frontend.c é efetuado todo o comportamento do lado do cliente. Serve para receber comandos do utilizador sendo que é feita a validação dos mesmos.

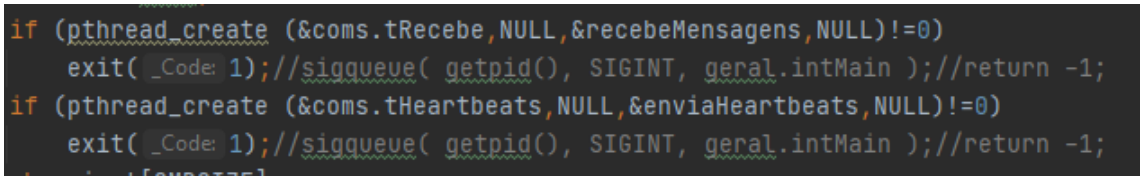
Para o frontend poder ser executado deverá se indicar o nome do utilizador e a password sendo que os mesmos são passados como argumento e é feita a verificação da existência dos mesmos, caso não sejam passados os parâmetros necessários será apresentado ao utilizador uma mensagem de erro ao utilizador. Esta funcionalidade está indica código abaixo.

```
if (argc < 3 || argc > 3) {
    printf( _Format: "Erro: Numero de argumentos errado! \nInsira o nome e a password para fazer login\n");
    exit( _Code: 1);
}
```

Após a verificação do número de argumentos é criado um FIFO para se poder comunicar com o Servidor (o processo backend), de modo que se possa validar os dados de login. Caso a resposta do servidor recuse o login vai continuar a ser pedido ao cliente os dados de login.

Threads Frontend

Uma vez que o cliente tenha efetuado login é criada uma *thread para receber mensagens do Servidor*, isto é, respostas aos comando e notificações e outra *thread para enviar um heartbeat* ao Servidor para indicar que o cliente continua em execução. Na thread inicial é pedido ao cliente os comandos a enviar ao servidor.



Todas as mensagens enviadas e recebidas são constituídas por estruturas. Consoante o tipo de mensagem é utilizada uma das estruturas já referidas neste relatório

Sinais Frotend

De modo a poder encerrar o frontend corretamente sem usar o comando exit é usado o sinal SIGINT

```
struct sigaction sa;
sa.sa_sigaction = sair;
sigaction(SIGINT, &sa, NULL);
```

Caso seja necessário desbloquear alguma das threads, é enviado um SIGUSR1 para a mesma. O tratamento deste sinal corresponde a uma função vazia, assim a thread vai tratar do sinal e avança a linha em que se encontrava.

```
struct sigaction act;
act.sa_sigaction = acorda;
act.sa_flags = SA_SIGINFO;
sigaction(SIGUSR1, &act, NULL);
```

Backend.c

No backend é efetuado todo o comportamento do lado do servidor. O Server recebe os comandos do frontend, gere os promotores, os itens em leilão e os clientes. Podendo também receber comandos do administrador. Só pode existir uma instância do mesmo, para assegurar isto é sempre verificado se o FIFO do servidor existe.

O tempo, os itens e os utilizadores são carregados no arranque do servidor e armazenados no final.

Threads Backend

De modo ao servidor poder executar várias tarefas em paralelo são utilizadas várias threads. Na thread inicial é feita a gestão do tempo.

A função recebeComando corre numa thread dedicada com o objetivo de receber comandos do administrador

```
if (pthread_create(&pTempo->tComandos, NULL, &recebeComando, &pTempo->comandos) != 0)
    sigqueue(getpid(), SIGINT, geral.intMain); // return -1;
```

De modo a se receber os diversos promotores foi criada uma thread dedicada para o mesmo.

```
if (pthread_create(&pComs->tPromotoresControl, NULL, &recebePromotores, &pComs->prom) != 0)
    sigqueue(getpid(), SIGINT, geral.intMain); // return -1;
```

Foi feita a criação de uma thread para inciar cada um dos promotores, sendo que é feita a criação de uma nova thread para cada um deles sendo que existe um limite max de promotores.

```
for (int i = 0; i < MAXPROM; i++)
{
    if (ptProms[i].promotor.ativo == 1 && ptProms[i].promotor.stop == 1)
    {
        ptProms[i].promotor.stop = 0;
        pipe(ptProms[i].promotor.fdProms);
        //printf("\nstop = %d\n", ptProms[i].promotor.stop);
        // printf("\nPROMOTOR %d\nMAXPROM %d \nnPromotores %d", i+1, MAXPROM, geral.nPromotores);
        if (pthread_create(&ptProms[i].tPromotor, NULL, &iniciaPromotores, &ptProms[i].promotor) != 0)
            /*return -1;*/ exit( Code: 1);
        //printf("\nPROXIMO\n");
    }
    // sleep(20);
}
```

Foi criada uma thread dedicada a receber os heartbeats dos clientes para confirmar se os mesmos ainda se encontram ativos.

```
if (pthread_create (&pComandos->tCLientesHeartbeats,NULL,&recebeHeartbeats,&pComandos->cliente)!=0)
     sigqueue( getpid(), SIGINT, geral.intMain );//return -1;
```

Existe uma thread para poder receber mensagem dos diversos clientes.

```
if (pthread_create (&pComandos->tClientes,NULL,&recebeClientes,&pComandos->cliente)!=0)
     sigqueue( getpid(), SIGINT, geral.intMain );//return -1;
```

Sinais Backend

Tanto o frontend como o backend estão em funcionamento num loop até um determinado momento. A flag de controlo é um stop, que se for "ativa" termina o loop de cada uma das thread e estas responsabilizam-se de limpar informações e terminar processos direcionados às mesmas. Porém, como no frontend foi indicado, as threads podem estar bloqueadas em certos pontos, então é enviado um SIGUSR1 a cada uma delas (às que possam estar bloqueadas) para estas concluírem o seu termino.

Variáveis de Ambiente

Foram nos fornecido vários ficheiros usados pelo backend contendo a informação dos promotores, utilizadores e dos itens. Os nomes destes ficheiros estão descritos em variáveis de ambiente de como ao backend os poder carregar. O backend verifica a existência destas variáveis de ambiente e no caso dos promotores verifica se existem os executáveis associados. A variável HEARTBEAT é utilizada pelo frontend sendo que representa os segundos de espera em que o frontend envia uma mensagem a sinalizar que está em execução ao servidor.

Para facilitar o processo de criação das mesmas, criamos o sript varsamb.sh:

$ varsamb.sh

```
1   FUSERS='backend/users/users.txt'
2   set FUSERS
3   export FUSERS
4
5   FITEMS='backend/items/items.txt'
6   set FITEMS
7   export FITEMS
8
9   FPROMOTERS='backend/promoters/proms.txt'
10  set FPROMOTERS
11  export FPROMOTERS
12
13  HEARTBEAT='60'
14  set HEARTBEAT
15  export HEARTBEAT
```

Conclusão

Ao longo deste trabalho pudemos aplicar os diversos conhecimentos de UNIX que temos vindo a aprender ao longo do semestre e expandir os nossos conhecimentos relativamente ao funcionamento de sistemas operativos baseados em UNIX.