Relatório Trabalho Prático - Meta 2

sobay

Licenciatura em Engenharia Informática
Sistemas Operativos
2022/2023

Rodrigo Cruz .: 2019122799@isec.pt

Índice

struct.h

Neste ficheiro estão contidas todas as bibliotecas necessárias para a correta execução do programa bem como as constantes de controlo e as estruturas partilhadas entre backend e frontend.

Constantes de controlo

MAX_SIZE 256 - Tamanho máximo de dados.
MAX_SIZE_FIFO 32 - Tamanho máximo do nome do FIFO.
MAX_USERS 20 - Limite máximo de utilizadores.
MAX_PROMOTORES 10 - Limite máximo de promotores em execução.
MAX_ITEMS 30 - Limite máximo de itens em leilão.

BKND_FIFO “bknd” - Nome do pipe do backend.
FRND_FIFO “user_%d” - Nome do pipe do frontend, o “%d” será substituído pelo PID do frontend.
HB_FIFO “hbfd” - Nome do pipe para envio do heartbeat para o backend.

Estruturas

Utilizador

Estrutura responsável por armazenar os dados dos utilizadores.
Na variável nome é guardado o nome do utilizador, a password e a sua palavra-passe, no saldo o saldo atual do utilizador, a variável valid serve para devolver o resultado do login (1-Credenciais corretas, 0-Credenciais incorretas), pid guarda o pid do frontend do qual está a aceder o utilizador e por fim a variável expiresAt guarda o tempo no qual expira o heartbeat do mesmo.

```
typedef struct Utilizador User;
struct Utilizador{
    char nome[MAX_SIZE];
    char password[MAX_SIZE];
    int saldo;
    int valid;
    int pid;
    int expiresAt;
}
```

Promoção

Estrutura responsável por guardar os dados das promoções recebidas do promotor, de modo a imprimir depois as mesmas no ecrã, é mostrado a categoria , desconto que é o valor da promoção e a duração que se trata do tempo restante até ao final da promoção.

```
typedef struct Promocao Promo;
struct Promocao{
    char categoria[MAX_SIZE];
    int desconto;
    int duracao;
};
```

Item

Esta estrutura é responsável por guardar todos os itens, de forma a posteriormente colocá-los à venda, é composta por id, nome, categoria, bid (licitação), buynow (compre já), tempo restante que o item está à venda, vendedor e licitador.

```
typedef struct Item Item;
struct Item{
    int id;
    char nome[MAX_SIZE];
    char categoria[MAX_SIZE];
    int bid;
    int buyNow;
    int tempo;
    char vendedor[MAX_SIZE];
    char licitador[MAX_SIZE];
    Promo promocao;
};
```

Cliente_Administrador

Estrutura utilizada para comunicação entre o backend e frontend.

```
typedef struct Cliente_Administrador CA;
struct Cliente_Administrador{
    char word[MAX_SIZE];
    char secWord[MAX_SIZE];
    int number;
    int secNumber;
    User ut;
    Item it[MAX_ITEMS];
};
```

utils.c

Contém a função de imprimir itens compartilhados entre os programas backend e frontend.

Backend

backend.h

Contém as variáveis que vão ser utilizadas para armazenar as variáveis de ambiente, o tempo, e o id sequencial atribuído aos itens. Existe ainda a variável FINIT que contém o nome do ficheiro de inicialização da plataforma. Contém ainda as estruturas responsáveis por passar os dados para as threads.

Variáveis globais

```
char *FUSERS;
char *FITEMS;
char *FPROMOTERS;
char *FINIT = "init.txt";
int HEARTBEAT;
int TEMPO;
int PROX_ID;
```

Estruturas

Estrutura para passar dados para a thread de gestão de tempo.

```
typedef struct ThreadTempoData TD;
struct ThreadTempoData{
	User *connUt;
	int *nConnUt;
	Item *listIt;
	int *nlistIt;
	pthread_mutex_t *ptrinco;
	pthread_mutex_t *ptrinco_promos;
	int para;
};
```

Estrutura para passar dados para a thread de resposta a pedidos.

```
typedef struct ThreadRequestRespondeData RR;
struct ThreadRequestRespondeData{
	User *connUt;
	int *nConnUt;
	Item *listIt;
	int *nlistIt;
	int fd_sv_fifo;
	pthread_mutex_t *ptrinco;
	int para;
};
```

Estrutura que guarda promotores ativos.

```
typedef struct Promotor Promotor;
struct Promotor{
	int pid;
	int threadNumber;
	char designacao[MAX_SIZE];
};
```

Estrutura para passar dados para a thread de gestão dos promotores.

```
typedef struct ThreadPromotorData PD;
struct ThreadPromotorData{
	Promotor *promotor;
	Promo *promocao;
	int *nPromocoes;
	Item *listIt;
	int *nlistIt;
	pthread_mutex_t *ptrinco;
	int para;
};
```

Funcionamento - backend.c

No início da sua execução o backend vai verificar se as variáveis de ambiente FUSERS, FPROMOTERS, FITEMS e HEARTBEAT foram declaradas e guardá-las se tal se verifique.
De seguida verifica se o ficheiro de inicialização existe através da função initPlataforma, se existir vai ler o ficheiro e modificar os valores do TEMPO e do PROX_ID para os valores lidos de forma a continuar o estado da última execução. Se não encontrar, irá criar o ficheiro de inicialização. Tenta ainda carregar o ficheiro de utilizadores de forma a certificar-se de que pode prosseguir a execução.
Posto isto, faz a criação do seu pipe e abre-o para leitura e escrita. De forma a garantir que os heartbeats são tratados a tempo foi também criado um pipe apenas para enviar sinais heartbeat para o backend, assim, à semelhança do pipe anterior, também este é criado e aberto para leitura e escrita.
De forma a evitar saídas inesperadas do sistema é nesta altura que o comportamento do sinal SIGINT vai ser alterado para que quando acontecer ao invés de sair sem avisar o programa envia a todos os utilizadores conectados a informação de que vai encerrar e manda-os encerrar também libertando todos os recursos iniciados.
Posto isto vão então ser criadas as diversas threads, em primeiro lugar são inicializadas as estruturas de dados a passar para as respetivas threads e de seguida são criadas as threads pela seguinte ordem, thread de gestão de login, thread de gestão de heartbeats e finalmente a thread de gestão de tempo. Desta forma o backend já pode lidar de forma simultânea com os diversos pedidos que lhe chegam.
Os comandos de administrador são tratados na thread principal do programa.
Quando o comando de executar promotores é detectado o backend vai verificar o ficheiro que contém a lista de promotores a executar e guarda o nome de cada um deles numa lista. De seguida compara a lista obtida com os promotores que estão atualmente em execução,

sendo que quando detecta que um dos operadores da lista já está em execução esse é retirado da lista de forma a ficar com uma lista apenas com os promotores que são realmente para executar/continuar em execução. Depois compara a nova lista gerada com os promotores atualmente ativos, caso o promotor não esteja contido nessa nova lista o backend manda-o encerrar, se estiver contido deixa-o continuar em execução. Manda então por fim executar os promotores que ainda não estão em execução mas que estão contidos na lista gerada sendo que cada promotor será executado numa thread diferente (uma thread por promotor) o backend inicia a estrutura de dados a enviar para a thread de gestão de promotores e preenche a lista de promotores ativos com o promotor executado e o número da thread que lhe foi atribuída de forma a que quando for preciso terminar o promotor saber qual das estruturas alterar de modo a sair do ciclo de execução da thread.
Quando um promotor lança uma promoção esta é lida e posteriormente é percorrida a lista de itens em leilão e comparada a categoria da promoção com a categoria do item, se coincidirem são então guardados os dados da promoção no item e feitos os cálculos para os valores de licitação e “compre já”.
A comunicação entre backend e frontend é feita com base na estrutura CA (Cliente_Administrador) sendo que é na variável word dessa estrutura que é enviado o código/palavra da ação que se pretende executar, sendo essa palavra lida pelo destino e tratada em conformidade.
Sempre que um item expira, é licitado ou comprado o backend percorre a lista de clientes conectados e envia-lhes a indicação do sucedido.
Quando o administrador pede que o backend termine, todos os recursos são libertados, todos os clientes ligados são alertados e é feita a atualização dos ficheiros de itens e de estado da plataforma (ficheiro de inicialização).

Frontend

Funcionamento - frontend.c

Ao iniciar o frontend é feita a verificação do número de argumentos recebidos, se o número de argumento for diferente de 3 então o programa é encerrado de imediato. Caso contrário é verificado se o backend está em execução e se tal se confirmar vai abrir o pipe do backend e de heartbeats em modo de escrita. De seguida vai criar o seu pipe de acordo com o seu pid de forma a poder receber comunicações e modificar o comportamento do sinal SIGINT para garantir que os recursos utilizados são libertados e que os pipes de comunicação são fechados ao detectar o encerramento inesperado.
Envia então as credenciais de acesso para o backend de forma a que os dados sejam validados e aguarda resposta. Quando o backend da autorização de acesso envia para além da variável valid com o valor 1 de forma a permitir o acesso o intervalo de tempo de heartbeat ao qual o frontend deve enviar o seu sinal de vida. Esse valor é guardado e é feita a inicialização da estrutura que envia os dados para a thread de heartbeat e criação dessa mesma thread cuja função é a de somente informar de que está conectado.
De forma a decidir o que fazer a seguir é implementado o mecanismo de select. Este mecanismo funciona como uma espécie de escuta que é adicionada aos canais que queremos saber se tiveram alterações, neste caso foi colocado no pipe do frontend de forma a detetar a receção de mensagens e no canal de *input* do teclado (file descriptor 0). Desta forma, quando o utilizador insere um comando é possível detectar esse acontecimento e indicar que deve ser dada prioridade aos dados recebidos do teclado e de igual forma detectar se foi recebida alguma mensagem no canal reservado para comunicação com o frontend e posteriormente tratar essa informação.
Quando o utilizador pede para encerar o frontend este envia uma mensagem ao backend a indiciar que se vai desconectar e por fim liberta todos os recursos utilizados tal como a thread e encerra.