Sistemas Operativos
Programação em C para UNIX

Ano Letivo 2022/23

Filipe Fernandes/ a2020134826

Licenciatura de Engenharia Informática

Índice

1. Introdução

Neste relatório iremos proceder à analise e interpretação do trabalho prático da cadeira de Sistemas Operativos, que consistiu em arquitetar e implementar uma plataforma para gerir um sistema de leilões, o qual faz a gestão da comunicação entre clientes envolvidos, gere os itens à venda, verifica os preços e determina quem arremata os itens.

O trabalho prático foi concretizado em linguagem C, para a plataforma UNIX (Linux), usando os mecanismos do sistema operativo abordados.

2. Arquitetura Geral

Neste trabalho foi optado usar o modelo Cliente – Servidor, onde o Servidor será considerado como o Backend e os Clientes considerados como Frontend.

Numa fase inicial, assim que o nosso Cliente(Frontend) é executado, é realizado um processo de autenticação, onde as credênciais do Cliente são revistas pelo Servidor(Backend). Através de um NamedPipe o Cliente envia uma estrutura com todas as suas informações. Nessa altura, o nosso Servidor, já se encontra com uma thread pronta para receber a estrutura do nosso Cliente e fazer a devida autenticação com recurso a uma biblioteca (users_lib), fornecida pelos Professores.

Após a análise dos dados do Cliente, é enviada uma resposta para o mesmo, com a aprovação ou reprovação do seu Login. Caso o Login do Cliente seja aprovado o mesmo ganha acesso aos comandos capazes de vender/comprar/obter informação de itens e gerir capital.

Nessa mesma fase inicial, mas na perspectiva do nosso Servidor, após a sua execução, são obtidas todas as informações necessárias através de ficheiros com essa função, que serão abordados mais à frentre. Após obter a informação necessária o Servidor está apto para lançar as suas thread's responsáveis pelo seu funcionamento.

Na Figura 1, podemos observar a Arquitetura pensada para este projeto.

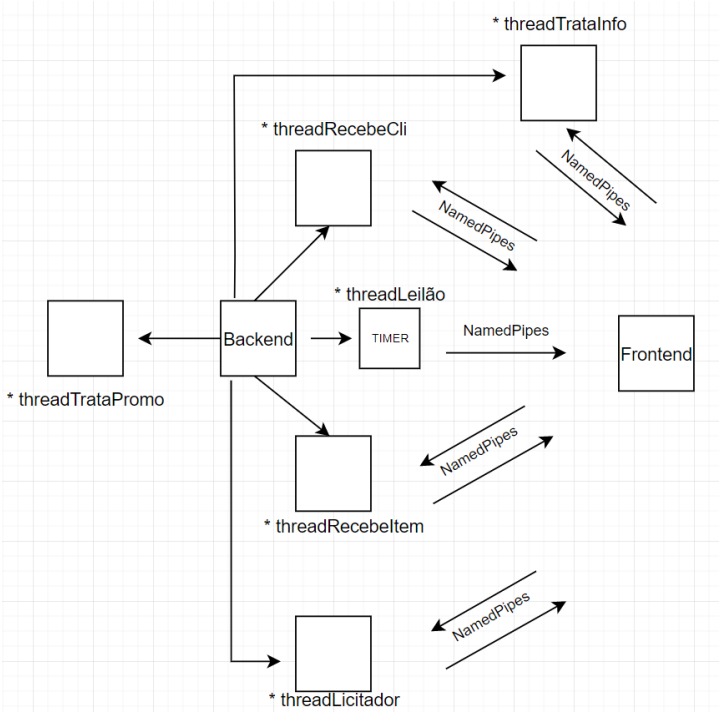

Figura 1

É possível observar na Figura, todas as threads pertencentes ao nosso Servidor, bem como os respectivos NamedPipes usados para receber/enviar informação para os nossos Clientes.

2.1. Estruturas de Dados

2.1.1. Estrutura do Backend

A estrutuda do Backend, tem um papel fundamental em todo o projeto, pois a mesma tem que ser capaz de armazenar diferentes tipos de informação proveniente dos Clientes.

Na Figura 2, podemos observar a estrutura do Backend.

```
typedef struct backendCliente backCli, *backCli_ptr;
struct backendCliente{

  int *numCli;
  int maxCli;
  int *numItens;
  int maxItens;
  int *numPromo;
  int maxPromo;
  int stop;

  int **pipe_arr;

  pid_t* prom_pids;

  char** promo_names;

  user_ptr array_clientes;

  item_ptr array_itens;

  pthread_mutex_t *trinco;
};
```

Figura 2

De todas as variáveis presentes nesta estrutura, existem 4 que desempenham um papel fundamental no que diz respeito a armazenamento de informação.

A primeira variável é “user_ptr array_cliente”, como o próprio nome indica, esta variável trata-se de um ponteiro para uma estrutura do tipo Cliente. Esta variavél é responsável pelo armazenamento das estruturas do tipo Cliente de todos os Clientes que se encontrem online.

A segunda variável é “item_ptr array_itens”, como o próprio nome indica, esta variável trata-se de um ponteiro para uma estrutura do tipo Item. Esta variável é responsável pelo armazenameno das estruturas do tipo Item, de todos o Itens que se encontram a ser leiloados pelo Backend.

A terceira variável é "int **pipe_arr", esta variável trata-se de um array bidimensional, cuja responsabilidade é a atribuição e redirecionamento stdout de e para um pipe anónimo para cada Promotor lançado.

A quarta variável é "pid_t* prom_pids", esta variável é responsável por armazenar todos os pid's dos Promotores lançados.

2.1.2. Estrutura do Frontend

A estrutuda do Frontend, contém todas as informações relativamente ao Cliente.

Na Figura 3, podemos observar a estrutura do Frontend.

```
typedef struct User user, *user_ptr;
struct User{

    char nome[50];
    char pass[50];
    char msg[50];
    int saldo;

    pid_t pid;

};
```

Figura 3

3. Backend

Nesta parte do relatório vamos abordar algumas das que foram as estratégias e implementações mais importantes do Projeto.

3.1.1. Ficheiro FITEMS

O Ficheiro fitems.txt é o ficheiro onde são armazenados os itens que se encontravam em leilão antes do encerramento do Backend.

Assim que o Backend é executado, recorremos ao ficheiro fitems.txt, para armazenar num array dinâmico “array_itens” todos os itens que se encontrarem lá presentes.

Na Figura 4, podemos observar, caso existam itens no ficheiro, o armazenamento dos mesmos na estrutura do Backend, mais propriamento no array de Itens.

```
FILE *file = fopen(FITEMS, "r");
if (file == NULL)
{
  printf("\n[ERRO] Ficheiro fitems.");
  exit(1);
}

while(fgets(buffer, sizeof(buffer), file) != NULL){

  if(numItens == 0){

  backCliData.array_itens = malloc(sizeof(item));
  if(backCliData.array_itens == NULL){
    printf("\n[ERRO] Alloc de memoria array_itens.");
    free(backCliData.array_itens);
    exit(1);
  }
  } else  {

    item_ptr aux;
    aux = realloc(backCliData.array_itens, sizeof(item)*(numItens + 1));
    if(aux == NULL){
      printf("\n[ERRO] realloc de memoria aux.");
      free(aux);
      exit(1);
    }

    backCliData.array_itens = aux;

  }

  int value;
  sscanf(buffer, "%d %s %s %d %d %d %s %s",
  &backCliData.array_itens[numItens].id,
  backCliData.array_itens[numItens].name,
  backCliData.array_itens[numItens].catg,
  &value,
  &backCliData.array_itens[numItens].buyN,
  &backCliData.array_itens[numItens].time,
  backCliData.array_itens[numItens].userOwner,
  backCliData.array_itens[numItens].userLbuyer);

  if(strcmp(backCliData.array_itens[numItens].userLbuyer,"-")==0){// Cas
    backCliData.array_itens[numItens].value = value;
    backCliData.array_itens[numItens].lastBid = 0;
  }else{ //Caso haja Licitador
    backCliData.array_itens[numItens].lastBid = value;
    backCliData.array_itens[numItens].value = 0;
  }
```

Figura 4

3.1.2. *threadRecebeCli

Esta Thread, como o prórpio nome diz, é responsável por receber as estruturas do Clientes através de um NamedPipe.

Numa fase inicial, esta Thread recebe uma estrutura de um Cliente, de seguida será feita uma validação dessa estrutura, onde será autenticado o nome e a password do Cliente com o auxílio das funções Biblioteca fornecidas pelos Professores. Consoante o resultado da validação do Cliente, a mesma estrutura será enviada de volta, mas com uma pequena alteração, mais propriamente na variável "char msg[50]", nesta variável estará presente o resultado da validação das informações do Cliente. Assim que o Cliente recebe de novo a sua estrutura, será feita uma análise à variável alterada na estrutura, para que dessa forma seja possível decidir se o Login é válido ou não no lado do Frontend.

Caso as informações presentes na estrutura do Cliente, sejam válidas, antes do envio da estrutura de volta, a mesma é guardada na estrutura do Backend, mais propriamente no array dinâmico "array_clientes" com o auxílio da função "pthread_mutex_lock " e "pthread_mutex_unlock " uma vez que está a ser realizada uma alteração ao nível da estrutura.

Na Figura 5, podemos observar a receção da estrutura através de um NamedPipe, a sua validação, o seu armazenamento e o seu envio de volta.

```
while (backCliData->stop) {
  // Lê credenciais do Cliente
  res_size = read(fd_back_fifo, &cli_dados, sizeof(user));

  if(res_size != sizeof(user)){
    printf("\n[ERRO] Ler do fifo do backend.");
  }

  if(strcmp(cli_dados.msg,"V")==0){
    loadUsersFile(FUSERS);
    if (isUserValid(cli_dados.nome, cli_dados.pass) == 1){
    // Testar para evitar mais que uma log de um Cliente
    for(int i=0; i<(*backCliData->numCli); i++){
      if(strcmp(backCliData->array_clientes[i].nome,cli_dados.nome)==0) // Ver se já está logado
        flag=1;
    }

    if(!flag){
      //Guardar novo Cliente e os seus dados
        pthread_mutex_lock(backCliData->trinco);
        auxCliente_ptr = add_cli(backCliData->numCli, backCliData->array_clientes, &cli_dados);
        backCliData->array_clientes = auxCliente_ptr;
        pthread_mutex_unlock(backCliData->trinco);

      //Responder ao Cliente
        sprintf(nome_fifo_cli,CLIENT_FIFO,cli_dados.pid);
        strcpy(cli_dados.msg,"VALIDO");
        fd_cli_fifo = open(nome_fifo_cli, O_WRONLY);
        if(fd_cli_fifo == -1){
          printf("\n[ERRO] Abrir fifo do cliente.");
          exit(1);
        }
```

Figura 5

3.1.3. *threadLeilao

Esta Thread, desempenha também um papel importante no que diz respeito à funcionalidade do Backend, pois é responsável por manipular o tempo de cada leilão. O Cliente não tem qualquer contacto com esta Thread, ele apenas recebe notificações da mesma quando um Item é vendido, quando faltam 10 segundos para o término do leilão de um determinado Item ou quando algum Licitador comprou o Item ao valor Compra já.

O funcionamento desta Thread consiste na iteração contínua do array dinâminco “array_itens”, que como o nome diz, é uma variável que armazena todos os itens que estão a ser leiloados de momento. Assim que o tempo desse Item termina, ou seja, chega a zero, o primeiro passo é aceder ao ficheiro fitems e com o auxílio de um ficheiro secundário vamos transcrever todos os itens presentes no ficheiro fitems com a exceção do Item cujo tempo terminou. Independentemente do Item estar presente ou não no ficheiro, o mesmo é iterado se existirem lá Itens.

Caso o tempo do Item não tenha chegado ao seu final, são testadas outras condições, sendo uma delas testar se o Item foi comprado ao valor Compra já. E se a condição for verdadeira, o primeiro passo, é remover o dinheiro licitado pelo Cliente comprador do Item e adiciona-lo à conta do Cliente vendedor. Ambos o comprador como o vendedor são avisados de imediato, após isso o Item é removido do array.

Na Figura 6, podemos observar a transcrição dos Itens presentes no ficheiro fitems.txt para o ficheiro temporário e por fim a reposição dos dados do Itens no ficheiro original com exceção do Item cujo tempo terminou, caso ele esteja no ficheiro.

```
while(fgets(buffer, sizeof(buffer), file)){

  strcpy(line,buffer);
  word = strtok(buffer, " ");
  int num = atoi(word);

  if (num != backCliData->array_itens[b].id){
    fprintf(temp_file, "%s", line);
  }

}

fclose(file);
fclose(temp_file);

file = fopen(FITEMS, "w");
temp_file = fopen("temp.txt", "r");

while (fgets(buffer, sizeof(buffer), temp_file)) {
  fprintf(file, "%s", buffer);
}

fclose(file);
fclose(temp_file);
```

Figura 6

Na Figura 7, podemos observar a condição de teste para saber se algum Cliente licitou um Item ao preço de Compra já.

```
else if(backCliData->array_itens[b].buyN <= backCliData->array_itens[b].lastBid){ // Valor compra Já
  if(strcmp(backCliData->array_itens[b].userLbuyer,"-")!=0){
    loadUsersFile(FUSERS);
    if(updateUserBalance(backCliData->array_itens[b].userLbuyer,getUserBalance(backCliData->array_itens[b].user
      printf("\n[ERRO] Remover dinheiro de Cliente.\n");
    }
    if(updateUserBalance(backCliData->array_itens[b].userOwner,getUserBalance(backCliData->array_itens[b].userO
    printf("\n[ERRO] Adicionar dinheiro ao Cliente.\n");
    }
    saveUsersFile(FUSERS);

    printf("\n[COMPRE JA]-[ATIVADO]-[%s]-[%d]-[%s]\n",
    backCliData->array_itens[b].name,
    backCliData->array_itens[b].buyN,
    backCliData->array_itens[b].userLbuyer);
```

Figura 7

3.1.4. Lançamento de Promotores

Esta implementação consiste no lançamento do Promotores que se encontram presentes no ficheiro fpromoters.txt.

O primeiro passo é saber o número de promotores presentes no ficheiro, caso haja algum, passamos à inicialização do nosso array dinâmico “char** promo_names”. De seguida vamos também inicializar o nosso array bidimensional de pipes anónimos. É importante dizer que cada Promotor lançado terá o seu pipe anónimo.

Após a inicialização de todos os array dinâmicos necessários, vamos proceder à criação de processos filho com a finalidade de lançar cada promotor, dentro de cada processo filho é realizado um redirecionamento stdout para o pipe que lhe for atribuido.

Na Figura 8, podemos observar a implementação do que foi explicado anteriormente.

```c
//Lançar Promotores presentes no ficheiro
FILE* fp = fopen(FPROMOTERS, "r");
if (fp == NULL) {
  perror("Error opening file");
  return 1;
}

if(feof(fp)) {
  printf("\n[FPROMOTERS VAZIO]\n");
} else {

  char line[50];

  int str_len=50;

  //Contar o número de promotores presentes no ficheiro
  while (fgets(line, sizeof(line), fp) != NULL)
  {
    numPromo++;
  }

  if(numPromo!=0){

    //Caso haja mais que 10 promotores no ficheiro
    if(numPromo>backCliData.maxPromo){
      printf("\n[FPROMOTERS EXCEDE O NUMERO MAXIMO DE PROMOTORES]\n");
      numPromo=backCliData.maxPromo;//Ficamos só com o 10
    }

    rewind(fp);
```

```c
  //Alocar memória para o array de pid's
  backCliData.prom_pids = malloc(numPromo * sizeof(pid_t));

  //Lançar o promotores
  for(int c=0; c<numPromo; c++){

    pid_t pid = fork();
    if(pid == 0){
      close(1);
      dup(backCliData.pipe_arr[c][1]); // Redirecionamento
      close(backCliData.pipe_arr[c][0]);
      close(backCliData.pipe_arr[c][1]);
      execl(backCliData.promo_names[c], backCliData.promo_names[c], NULL);
      printf("\n[ERRO] Impossivel executar [%s].\n",backCliData.promo_names[c]);
      exit(1);
    } else if(pid > 0){ // Vamos guardar o pid do Promotor executado no array
      close(backCliData.pipe_arr[c][1]);
      backCliData.prom_pids[c] = pid;
      printf("\nPromotor: [%s] executado com sucesso.\n",backCliData.promo_names[c]);
    } else {
      printf("\n[ERRO] Criar child process.\n");
      return 1;
    }

  }

} else {
  fclose(fp);
  printf("\n[FPROMOTERS VAZIO]\n");
}

pthread_create(&thread_promo,NULL,threadTrataPromo,&backCliData);
```

Figura 8

4. Conclusão

Ao longo deste trabalho, deparei-me com problemas e desafios que foram essenciais para a minha evolução como programador. Fui obrigado a utilizar um pensamento crítico capaz de resolver qualquer problema que surgisse.

O desenvolvimento deste trabalho, testou constantemente as minhas capacidades e conhecimentos, podendo dizer sem qualquer dúvida que foi uma experiência enriquecedora.