Trabalho Prático

Disciplina Sistemas Operativos

Autores:

Daniel Duarte Dias Ferreira Albino – 2020134077

Nuno Alexandre Domingues – 2020109910

Engenharia Informática

Coimbra, janeiro de 2022

Índice

Índice de figuras

1. Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Sistemas Operativos, do curso de Licenciatura Informática, do Instituto Superior de Engenharia de Coimbra.

Tem como objetivo aplicar conhecimentos sobre a programação em linguagem C em ambiente Linux.

O trabalho em questão destina-se a simular um servidor “balcão” que atende utentes de um hospital e os encaminha para uma consulta.

Este relatório divide-se em três partes fundamentais: na primeira parte é apresentada uma descrição das estruturas de dados usadas; na segunda parte é realizada uma apresentação da interface do programa balcão, cliente e médico e por fim abordamos umas conclusões finais.

2. Estruturas de dados

2.1. Balcao

A estrutura de dados *Balcao* está presente no programa balcão e apresenta uma variável do tipo *char* onde vão ser escritos todos os comandos que o administrador do programa balcão quiser, a variável inteira *fk1* serve para abrir um *fork* na função de classificação do sintoma do utente, as variáveis *balc_to_clas* e *clas_to_balc* servem como modo de comunicação entre o programa classificador (fornecido pelos professores) e as variáveis de ambiente MaxMedicos (máximo de médicos) e MaxClientes (máximo de clientes).

```
typedef struct Balcao Balc, *pBalc;

struct Balcao{
    char comandos[50]; //Variavel onde é escrito os comandos do administrador
    int fk1, balc_to_clas[2],clas_to_balc[2],MaxMedicos,MaxClientes;
    //fork, files descriptors, vars ambiente
};
```

Figura 1 - struct Balcao

O administrador pode pedir ao balcão para executar várias ações como por exemplo:

- *utentes*: serve para ver os utentes em espera;

- *especialistas*: serve para ver os especialistas em espera;

- *delut <id>*: serve para apagar um utente em espera;

- *deles <id>*: serve para apagar um médico em espera;

- *freq <segundos>*: serve para alterar a frequência em que a lista de espera dos utentes e dos médicos é exibida pelo programa balcão.

2.2. ClienteMedico

A estrutura *ClienteMedico* está presente no programa balcão, médico e cliente e tem como finalidade a troca de informações entre o médico e o balcão e entre o cliente e o balcão e vice-versa. Apresenta uma variável do tipo *char clioumed* que tem como objetivo a identificação se a informação vem de um cliente ou de um médico ou se o balcão pretende comunicar com o médico ou com o cliente, de seguida estão presentes duas variáveis do tipo *char sintoma* e *resposta* que dizem respeito ao

sintoma do utente e á classificação determinada pelo *classificador*, a variável *char especialidade* informa, caso seja um médico, qual a sua especialidade. Esta estrutura apresenta ainda duas variáveis do tipo inteiro que são a *sair* que se estiver com o valor 1 quer dizer que aquele médico ou cliente saiu e a variável *espera* que diz respeito a se um cliente ou médico estão a espera para serem direcionados para uma consulta, e por fim uma variável *pid* que guarda o pid do cliente ou do médico.

```
typedef struct ClienteMedico CliMed, *pCliMed;

struct ClienteMedico{
    char clioumed[10];
    char sintoma[50],resposta[50]; //sintomas e resposta do classificador
    char especialidade[50];
    char nome[100];
    int sair;
    int espera;
    pid_t pid;
};
```

Figura 2 - struct ClienteMedico

2.3. InfoCliente e InfoMedicos

A estrutura *InfoCliente* e *InfoMedicos* estão presentes no programa balcão e guardam todas as informações do cliente e do médico em listas ligadas (por isso, é que é usado a variável *prox*).

Na estrutura *InfoCliente* temos o sintoma (variável *sintoma*), a especialidade a que lhe foi atribuída pelo *classificador* (variável *especialidade*), o nome do utente (variável *nome*), se está em espera ou não (variável *espera*) e por fim o pid do cliente (variável *pid*).

```
typedef struct InfoCliente ICli, *pICli;

struct InfoCliente{
    char sintoma[50],especialidade[50];
    char nome[100];
    int espera;
    pid_t pid; //pid do cliente
    pICli prox;
};
```

Figura 3 - struct Infocliente

Na estrutura *InfoMedico* temos a especialidade do médico (variável *especialidade*), o nome do médico (variável *nome*), se está em espera ou não (variável *espera*) e por fim o pid do médico (variável *pid*).

```
typedef struct InfoMedicos IMed, *pIMed;

struct InfoMedicos{
    char especialidade[50];
    char nome[100];
    int espera;
    pid_t pid; //pid do medico
    pIMed prox;
};
```

Figura 4 - struct InfoMedicos

2.4 DThreads

A estrutura *DThreads* é a estrutura de dados usada pelas threads do programa balcão com a variável *stop*. Esta serve para parar uma thread, a variável *freq* serve para de x em x segundos seja apresentada a informação dos clientes e médicos em espera, as variáveis *listaClientes* e *listaMedicos* são as listas ligadas que foram mencionadas anteriormente e a variável **m* que serve para que quando uma thread está a realizar uma determinada "tarefa" nada esteja a ser executando.

```
typedef struct DThreads DTh,*pDTh;

struct DThreads{
    int stop;
    int freq;
    int maxcli,maxmed;
    pICli listaClientes;
    pIMed listaMedicos;
    pthread_mutex_t *m;
};
```

Figura 5 - struct DThreads

2.5 Consulta

A estrutura *Consulta* esta presente nos três programas (balcão, cliente, médico) e tem como finalidade de enviar o id da consulta, neste caso será o número de consultas começando no número 1. Este id irá criar 2 fifos (*clientemedico1* e *medicocliente1*) que serão usados pelo programa cliente e médico para a sua comunicação.

```
typedef struct Consulta Con, *pCon;

struct Consulta{
    int id;
};
```

Figura 6 - struct Consulta

3. Interface

3.1 Balcão

Quando o balcão é iniciado as seguintes informações aparecem:

- Os clientes em espera, visto que não existem clientes o balcão apresenta a mensagem de que não existem Clientes;

- Os médicos em espera, visto que não existem médicos o balcão apresenta a mensagem de que não existem Médicos;

- A mensagem que possibilita ao administrador escrever um comando.

```
Ainda não existe Clientes!

Ainda não existe Medicos!

Introduza o comando (para sair introduza 'encerra'):
```

Figura 7 - Interface Balcão 1

Os comandos são tratados pelas seguintes funções:

1- Uma thread “trata” de ler o que é escrito pelo teclado:

```c
void *teclado(void *dados){
    DTh *pdados = (DTh*) dados;
    char varSair[100];
    do{
        fflush(stdin);
        printf("\nIntroduza o comando (para sair introduza 'encerra'): ");
        fgets(varSair,49,stdin);

        if(strcmp(varSair,"encerra\n") == 0){
            pdados->stop=0;
        }else{
            comandos(varSair,pdados);
        }
    }while(pdados->stop == 1);

    pthread_mutex_destroy(pdados->m);
    closeserver(pdados);
    pthread_exit(NULL);
}
```

Figura 8 - Thread Teclado

2- E esta função “trata” da validação dos comandos:

```c
void comandos(char *varSair,DTh *pdados){
    char c[50];
    int x,v=0;
    if(strcmp(varSair,"utentes\n") == 0){
        mostraInfoCliente(pdados->listaClientes);
        return;
    }
    if(strcmp(varSair,"especialistas\n") == 0){
        mostraInfoMedico(pdados->listaMedicos);
        return;
    }
    v = divideComando(varSair,c,&x);
    if(v==0){
        if(strcmp(c,"delut") == 0){
            if(x>0){
                if(pdados->listaClientes == NULL){
                    printf("\nAviso: Não existem Clientes, por isso não é possivel eliminar nenhum cliente\n");
                    return;
                }
                union sigval val;
                sigqueue(x,SIGINT,val);
                pdados->listaClientes = eliminaCliente(pdados->listaClientes,x);
                return;
            }
        }
        if(strcmp(c,"delesp") == 0){
            if(x>0){
                if(pdados->listaMedicos == NULL){
                    printf("\nAviso: Não existem Medicos, por isso não é possivel eliminar nenhum medico\n");
                    return;
                }
                union sigval val;
                sigqueue(x,SIGINT,val);
                pdados->listaMedicos = eliminaMedico(pdados->listaMedicos,x);
                return;
            }
        }
```

Figura 9 - Função Comandos

```
        if(strcmp(c,"freq") == 0){
            if(x>0){
                pdados->freq = x;
                return;
            }
        }
    }
    printf("\nComando invalido!\n");
    return;

}
```

Figura 10 - Continuação da Função Comandos

As duas figuras a baixo mostram o que acontece quando um cliente ou um médico iniciam o seu programa.

```
Novo Medico:
        Nome: artur, especialidade: oftalmologia
Ainda não existe Clientes!

Medicos

1

        Nome: artur
                Especialidade: oftalmologia

                PID: 1704
```

Figura 11 - Novo Médico

```
Novo Utente:
        Nome: manuel, classificação: oftalmologia 3
Utentes:

        Nome: manuel
                Sintoma: olho

                Especialidade: oftalmologia 3

                PID: 1729

Ainda não existe Medicos!
```

Figura 12 - Novo Cliente

Esta informação está presente no código abaixo:

1- Uma thread trata de mostrar a informação dos clientes e médicos em espera:

```c
void *temporizador(void *dados){
    pDTh pdados = (DTh*) dados;
    do{
        pthread_mutex_lock(pdados->m);

        mostraInfoCliente(pdados->listaClientes);
        mostraInfoMedico(pdados->listaMedicos);

        pthread_mutex_unlock(pdados->m);
        sleep(pdados->freq);
    }while(pdados->stop);
}
```

Figura 13 - Thread Temporizador

2- As listas ligadas onde estão presentes todas as informações dos clientes e dos médicos:

```c
pICli adicionaUtente(pICli listaUtentes, pCliMed c){
    pICli novo, aux;
    novo = malloc(sizeof(CliMed));
    if(novo == NULL){
        //Erro ao adicionar um novo cliente!
        printf("Erro ao adicionar um utente!\n");
        return listaUtentes;
    }
    preencheUtente(novo,c);
    if(listaUtentes==NULL){
        listaUtentes = novo;
    }else{
        aux=listaUtentes;
        while(aux->prox != NULL){
            aux = aux->prox;
        }
        aux->prox = novo;
    }
    return listaUtentes;
}

void preencheUtente(pICli novo,pCliMed c){
    strcpy(novo->sintoma,c->sintoma);
    strcpy(novo->especialidade,c->resposta);
    novo->pid = c->pid;
    strcpy(novo->nome,c->nome);
    novo->prox = NULL;
    novo->espera = c->espera;
}
```

Figura 14 - Função Adiciona Utente

```c
void mostraInfoCliente(pICli listaUtentes){
    if(listaUtentes == NULL){
            printf("\nAinda não existe Clientes!\n");
            return;
    }
    int espera;
    printf("\nUtentes: \n");
    while(listaUtentes!=NULL){
        espera = listaUtentes->espera;
        if(espera == 0){
            printf("\nO Utente %s está numa consulta!\n",listaUtentes->nome);
            listaUtentes = listaUtentes->prox;
        }
        if(espera==1){
            printf("\n\tNome: ");
            puts(listaUtentes->nome);
            printf("\t\tSintoma: ");
            puts(listaUtentes->sintoma);
            printf("\t\tEspecialidade: ");
            puts(listaUtentes->especialidade);
            printf("\n\t\tPID: ");
            printf("%d\n",listaUtentes->pid);
            fflush(stdout);
            listaUtentes = listaUtentes->prox;
        }
    }
}
```

Figura 15 - Função Mostra Cliente

```c
pIMed adicionaMedico(pIMed listaMedicos, pCliMed m){
    pIMed novo, aux;
    novo = malloc(sizeof(CliMed));
    if(novo == NULL){
        //Erro ao adicionar um novo cliente!
        printf("Erro ao adicionar um medico!\n");
        return listaMedicos;
    }

    preencheMedico(novo,m);

    if(listaMedicos==NULL){
        listaMedicos = novo;
    }else{
        aux=listaMedicos;
        while(aux->prox != NULL){
            aux = aux->prox;
        }
        aux->prox = novo;
    }

    return listaMedicos;
}

void preencheMedico(pIMed novo,pCliMed m){
    strcpy(novo->nome,m->nome);
    strcpy(novo->especialidade,m->especialidade);
    novo->pid=m->pid;
    novo->prox = NULL;
    novo->espera = m->espera;
}
```

Figura 16 - Adiciona Médico

```c
void mostraInfoMedico(pIMed listaMedicos){
    if(listaMedicos==NULL){
        printf("\nAinda não existe Medicos!\n");
        return;
    }
    printf("\nMedicos \n");
    while(listaMedicos!=NULL){
        printf("\n%d\n",listaMedicos->espera);
        if(listaMedicos->espera == 0){
            printf("\nO Medico %s está numa consulta!\n",listaMedicos->nome);
            listaMedicos = listaMedicos->prox;
        }
        if(listaMedicos!=NULL){
            printf("\n\tNome: ");
            puts(listaMedicos->nome);
            printf("\t\tEspecialidade: ");
            puts(listaMedicos->especialidade);
            printf("\n\t\tPID: ");
            printf("%d\n",listaMedicos->pid);
            fflush(stdout);
            listaMedicos = listaMedicos->prox;
        }
    }
}
```

Figura 17 - Mostra Médico

3.2 Cliente

No programa cliente, quando é iniciado aparece uma mensagem e o utente tem a possibilidade de escrever o seu sintoma, como exemplo foi usada a apalavra “olho”, e de seguida é apresentada a classificação do *classificador*. Logo após isso é possível sair do programa, informando ao balcão que este cliente saiu do sistema ou de esperar que seja atendido por um médico.

```
Bem vindo ao serviço MEDICALso Sr./Sra. manuel!
Indique o seu sintoma (para sair escrever 'sair'): olho

Resposta do classificador - [oftalmologia 3]

Espere para ser atendido. Se pretender sair escreva 'sair'
: sair

Voce saiu do sistema MEDICALso!
```

Figura 18 - Programa Cliente

A classificação é determinada pelo código abaixo, onde é usado um fork para a comunicação entre o balcão e o classificador:

```
void classifica(pCliMed c){
        Balc b;
        pipe(b.balc_to_clas);
        pipe(b.clas_to_balc);
        b.fk1 = fork();
        if(b.fk1<0){
            printf("ERRO: Erro no fork!\n");
            return;
        }
        else if(b.fk1 == 0){ //Filho
            close(STDIN_FILENO); //Fecha o stdin
            dup(b.balc_to_clas[0]); //liga o balc_to_clas ao stdin
            close(b.balc_to_clas[0]);
            close(b.balc_to_clas[1]);

            close(STDOUT_FILENO); //Fecha o stdout
            dup(b.clas_to_balc[1]); //liga o clas_to_balc ao stdout
            close(b.clas_to_balc[1]);
            close(b.clas_to_balc[0]);

            execl("classificador","classificador",NULL);
        }
        else if(b.fk1 > 0){//Pai
        close(b.balc_to_clas[0]);
        close(b.clas_to_balc[1]);

        write(b.balc_to_clas[1],c->sintoma,sizeof(c->sintoma));
        read(b.clas_to_balc[0],c->resposta,49);
        close(b.clas_to_balc[0]);
        if(c->resposta[strlen(c->resposta)-1] == '\n')
            c->resposta[strlen(c->resposta)-1] = '\0';
        }
}
```

Figura 19 - Classificação

3.3 Médico

No programa médico, logo após ser iniciado este tem 20 segundos para dar um "sinal de vida" escrevendo a palavra "aqui", caso contrário sai do sistema e o balcão é informado.

```
Bem vindo Sr./Sra. artur.
Aguarde ate que um utente lhe seja atribuido!
        Caso pretenda desistir da consulta escreva 'sair'!

Voce saiu do sistema MEDICALso!
```

Figura 20 - Programa Médico

Através do código abaixo podemos ver como é que a comunicação entre o balcão e o cliente e a comunicação entre o balcão e o médico são feitos:

```
int fdRecebe = open(SERVER_FIFO,O_RDWR);
if(fdRecebe == -1){
    printf("Erro!\n");
    exit(1);
}
int idU,idM;
do{
    size = read(fdRecebe,&cm,sizeof(cm));
    if(size>1){
            pthread_mutex_lock(&mutex);
            if(strcmp(cm.clioumed,"cli") == 0){
                if(cm.sair == 1){
                    listaUtentes=eliminaCliente(listaUtentes,cm.pid);
                    dados[0].listaClientes = listaUtentes;
                }else{
                    if(dados[0].maxcli<0){
                        cm.sair=1;
                        sprintf(CLIENTE_FIFO_FINAL,CLIENTE_FIFO,cm.pid);
                        int fdEnvio = open(CLIENTE_FIFO_FINAL,O_WRONLY);
                        int size2 = write(fdEnvio,&cm,sizeof(cm));
                        close(fdEnvio);
                    }else{
                        classifica(&cm);
                        cm.sair=0;
                        sprintf(CLIENTE_FIFO_FINAL,CLIENTE_FIFO,cm.pid);
                        int fdEnvio = open(CLIENTE_FIFO_FINAL,O_WRONLY);
                        int size2 = write(fdEnvio,&cm,sizeof(cm));
                        close(fdEnvio);
                        listaUtentes = adicionaUtente(listaUtentes,&cm);
                        dados[0].listaClientes = listaUtentes;
                        printf("Novo Utente:\n\tNome: %s, classificação: %s",cm.nome,cm.resposta);
                        fflush(stdout);
                    }
                }
            }
```

Figura 21 - Comunicação Balcão Cliente

```
if(strcmp(cm.clioumed,"med") == 0){
    if(cm.sair == 1){
        listaMedicos=eliminaMedico(listaMedicos,cm.pid);
        dados[0].listaMedicos = listaMedicos;
    }else{
        listaMedicos=adicionaMedico(listaMedicos,&cm);
        dados[0].listaMedicos = listaMedicos;
        printf("Novo Medico:\n\tNome: %s, especialidade: %s",cm.nome,cm.especialidade);
        fflush(stdout);
    }
}
pthread_mutex_unlock(&mutex);
```

Figura 22 - Comunicação Balcão Médico

4. Conclusões Finais

Em suma, este trabalho permitiu aprender e aplicar os conhecimentos da linguagem C em ambiente Linux dados em contexto de sala de aula. Assim, podemos afirmar que este trabalho nos permitiu aprofundar e explorar as versatilidades que o sistema operativo Linux nos proporciona e a fácil comunicação entre processos.

Este trabalho foi bastante útil para aplicar a matéria lecionada e compreendê-la melhor.