Instituto Superior de Engenharia de Coimbra

Licenciatura em Engenharia Informática

Sistemas Operativos

2021/2022

Relatório do Trabalho Prático

Meta Final

Carolina Ferreira – 2018018459

José Francisco Pereira Barra Marques – 2018016626

Conteúdo

cliente.h

O header file cliente.h tem representada a estrutura base do cliente, guardando dados que são pedidos diretamente ao utilizador, nomeadamente as suas informações pessoais como o nome, sintomas indicados e ainda dados devolvidos pelo balcão também inerentes a si como a especialidade para onde se deve dirigir, o grau de prioridade e um "número de senha" que corresponde, na verdade, ao seu PID.

```
typedef struct cliente{
    char nome[40];
    char sintomas[70];
    char especialidade[30];
    int prioridade;
    int numero_senha; //pid
}Cliente;
```

medico.h

O header file medico.h possui a estrutura para guardar as informações relevantes do médico para o sistema. Nessa estrutura constam as variáveis "nome", "especialidade" clínica, "disponibilidade" do tipo bool para indicar se o médico se encontra disponível ou em consulta e uma "id_medico" para guardar o seu PID.

```
typedef struct medico{
    char nome[40];
    char especialidade[70];
    bool disponibilidade;
    int id_medico;
}Medico;
```

balcao.h

O balcão.h contém a definição de 4 estruturas:

```
typedef struct all_meds{
    Medico *especialistas;
    int tamanho;
}AllMeds;
```

- uma estrutura "AllMeds" que na verdade é um array de estruturas do tipo "Medico" com uma variável int "tamanho" que guarda o número total de estruturas no array, ou seja, o número total de especialistas;

```
typedef struct filas{
    Cliente *geral;
    int geralTam;
    Cliente *totais;
    int toTam;
    Cliente *oftalmologia;
    int oftTam;
    Cliente *neurologia;
    int neuTam;
    Cliente *estomatologia;
    int estTam;
    Cliente *ortopedia;
    int ortTam;
    //5 é o numero maximo de fila por especialidade
}Filas;
```

- uma estrutura "Filas" para armazenar as estruturas de todos os utentes que entram no sistema e ficam à espera de atendimento. Todos os utentes são armazenados num array conjunto para facilitar funções responsáveis por mostrar no balcão todos os utentes em espera num dado momento, mas, a sua informação é também guardada num array

específico da especialidade onde será atendido. Para cada um dos arrays de estruturas, associa-se uma variável do tipo inteiro que guarda o tamanho atual de cada array, traduzindo-se no número de utentes de cada fila;

- uma estrutura "Consulta" para passar os utentes das filas de espera para outro array de utentes em consulta no momento;

```
typedef struct consulta{
    Cliente *atendido;
    int tam;
}Consulta;
```

- uma estrutura "MSG"

```
typedef struct msg{
    int pid;
    char mensagem[40];
}MSG;
```

balcao.c

```
int main(int argc, char *argv[],char *envp[]){
    int bc[2];
    int cb[2];
    int fd,fdm,fd_retorno,fdb;
    char resposta[30];
    char str[30],cmd[30];
    char *espec;
    char *com;
    pipe(bc);
    pipe(cb);
```

Na imagem ao lado podemos ver um "int bc[2]" e um "int cb[2]" que têm como intenção criar dois pipes para leitura e escrita bidirecionais (no seu conjunto) sendo o primeiro no sentido balcão->classificador e o segundo no sentido classificador->balcão.

Recorrendo à função fork(), o processo do balcão cria um novo processo para ficar pronto a receber a informação do cliente através dos pipes, executar o programa "classificador" e escrever/redirecionar para esse canal os sintomas recebidos.

Named Pipes

O mecanismo de comunicação entre clientes, balcão e médicos é o named pipe.

```
//Verificar se existe um Fifo criado anteriormente
    if(access(FIFO_SERV, F_OK) == 0){
        printf("[ERRO] O fifo ja existe!\n");
        exit(1);
    }
//Criar o FIFO
    mkfifo(FIFO_SERV2,0700);
     fdm= open(FIFO_SERV2,O_RDWR);

    mkfifo(FIFO_SERV, 0700);
     fd = open(FIFO_SERV, O_RDWR);

    printf("Criei o fifo...\n");
     //Abrir o fifo
```

O tratamento dos pipes do stdin e do stdout e é feito da seguinte forma:

```
//tratamento do pipe stdin
close(0);
dup(bc[0]);
close(bc[0]);
close(bc[1]);

//tratamento do pipe stdout
close(1);
dup(cb[1]);
close(cb[1]);
close(cb[0]);
```

- O stdin do filho passa a ser diretamente o bc[0]
- O stdout do filho passa a ser diretamente o cb[1]

Fifos

Foram construídos 6 FIFOs, dois para cada um dos ficheiros. O FIFO_SERV e o FIFO_SERV2 servem para a comunicação entre cliente e balcão e entre médico e balcão de modo a este saber com qual dos utilizadores está a contactar. Por sua vez o FIFO_CLI serve para receber a especialidade e o grau de prioridade proviniente do balcão, função semelhante a que tem o FIFIO_MED mas desta vez para o médico. Os restantes 2 FIFOs(FIFO_CLIC e FIFO_MEDC) servem para fazer a comunicação entre balcão e médico. De modo a poder ser suportada mais que uma ligação é colocado o pid do utilizador na atribuição do nome ao FIFO.

```
#define FIFO_SERV "balc"
#define FIFO_CLI "utente%d"
#define FIFO_MED "espec%d"
#define FIFO_SERV2 "balcmed"
#define FIFO_MEDC "especcom%d"
#define FIFO_CLIC "utentecom%d"
```

Funções

No que diz respeito a funções foram adicionadas, a *AdicionaUtente()* vai adicionar um cliente a uma fila de espera total e à fila de espera da sua respetiva especialidade clínica caso a

lotação não esteja cheia. A inserção está a ser feita de forma sequencial, mas como existem diferentes graus de prioridade é chamada a função Reordena() que coloca os utentes com mais prioridade nos primeiros lugares da fila. Tal como existe a AdicionaUtente() também existe a *AdicionaEspecialista()* que coloca um especialista num array de médicos, caso este não esteja lotado. Quando existe a atribuição de um médico com um cliente, junção essa verificada através da função *UtenteASerAtendido()* é chamada a função *AdicionaConsulta().* Quando é adicionada uma consulta o cliente é transportado para uma fila diferente passando a estar localizado na fila de consultas ativas, fila essa que poderá ser consultada no *ListaUtentes()*. Quando se chega ao fim da consulta o utente é removido através da função *RemoveUtente()*. Tal como o utente é também possível remover um médico através da função *RemoveEspecialista()*. Além destas funções enumeradas em cima existem também as funções contaUtentes() e ContaEspecialistas() de um determinado ramo médico. A primeira retorna o número de utentes à frente de um determinado utente e a segunda retorna o número de especialistas no sistema num dado momento.

```
17 > int ContaEspecialistas(AllMeds a,Cliente c){…
24   }
25
26 > void MudaDisponibilidade(AllMeds *a,int id_medico){…
36   }
37
38 > void AdicionaConsulta(Consulta *s,Filas f,int ncliente){…
59   }
60
61 > int contaUtentes(Filas f,Cliente c,int maxclientes){…
76   }
77
78 > void ListaEspecialistas(AllMeds a){…
87   }
88
89 > Cliente* reordena(Cliente *lista,int tam){…
27   }
28
29 > int AdicionaUtente(Filas *f,Cliente *cliente,int maxclientes){…
38   }
39
40   /*
41 > int getPosicao(Cliente *c,int tam, int pidAux){…
49   }*/
50
51 > int AdicionaEspecialista(AllMeds *a,Medico *m,int maxmedicos){…
71   }
72
73 > void RemoveEspecialista(AllMeds *a,int n){ …
```

varambiente.sh

Este ficheiro contém apenas a criação das variáveis de ambiente que estabelecem o número máximo de médicos e de clientes.

```
#!/bin/bash
MAXCLIENTES="25"
MAXMEDICOS="5"
export MAXCLIENTES
export MAXMEDICOS
```

Makefile

O makefile que possui os targets de compilação “all” (compilação de todos os programas), “cliente” (compilação do programa cliente), “balcao” (compilação do programa balcão), “medico” (compilação do programa médico) e “clean” (eliminação de todos os ficheiros temporários de apoio à compilação e dos executáveis).

```
all:    cliente balcao medico clean

cliente: cliente.o
        gcc cliente.o -o cliente

cliente.o: cliente.c cliente.h
        gcc cliente.c -c

balcao: balcao.o
        gcc balcao.o -o balcao

balcao.o: balcao.c balcao.h
        gcc balcao.c -c

medico: medico.o
        gcc medico.o -o medico

medico.o: medico.c medico.h
        gcc medico.c -c

clean:
        rm balc rm balcmed rm espec* rm utente*
```