Instituto Politécnico de Coimbra

Instituto Superior de Engenharia de Coimbra

Licenciatura em Engenharia Informática

Unidade Curricular de Sistemas Operativos

Mauro Roque Jesus - 2019130796

Rúben Rodrigues Almeida - 2019130955

Coimbra 2021/22

Índice

Introdução

O trabalho prático de Sistemas Operativos consistia na implementação de um sistema para gerir atendimento de clientes em estabelecimentos médicos. Tendo interação entre o doente (utente), médico e o balcão de atendimento sendo este o responsável por todo o sistema. O projeto foi realizado em linguagem C em plataforma Unix (Linux).

O sistema pretendido é composto por 4 programas principais: cliente, médico, balcão e classificador.

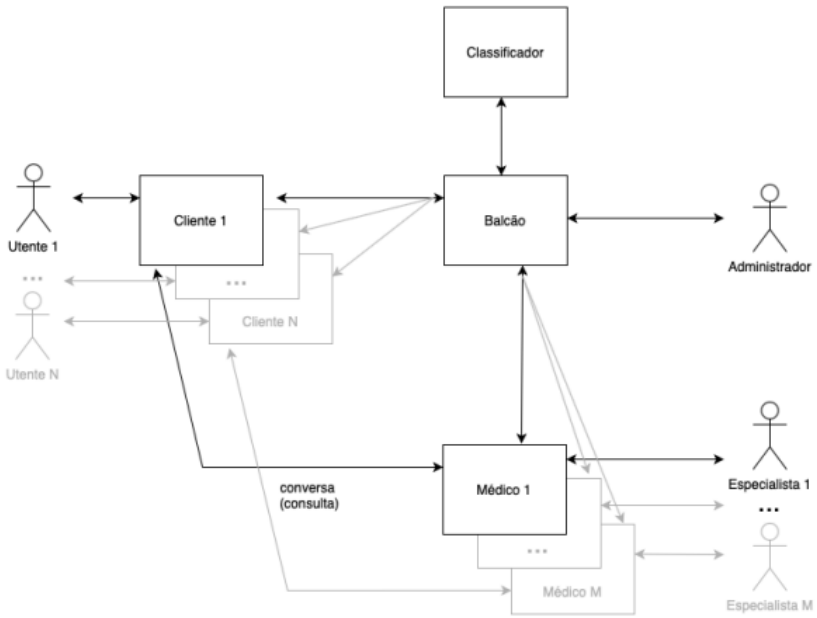


Figura 1 - Funcionamento do Sistema MEDICALso

Funções do Sistema: Balcão

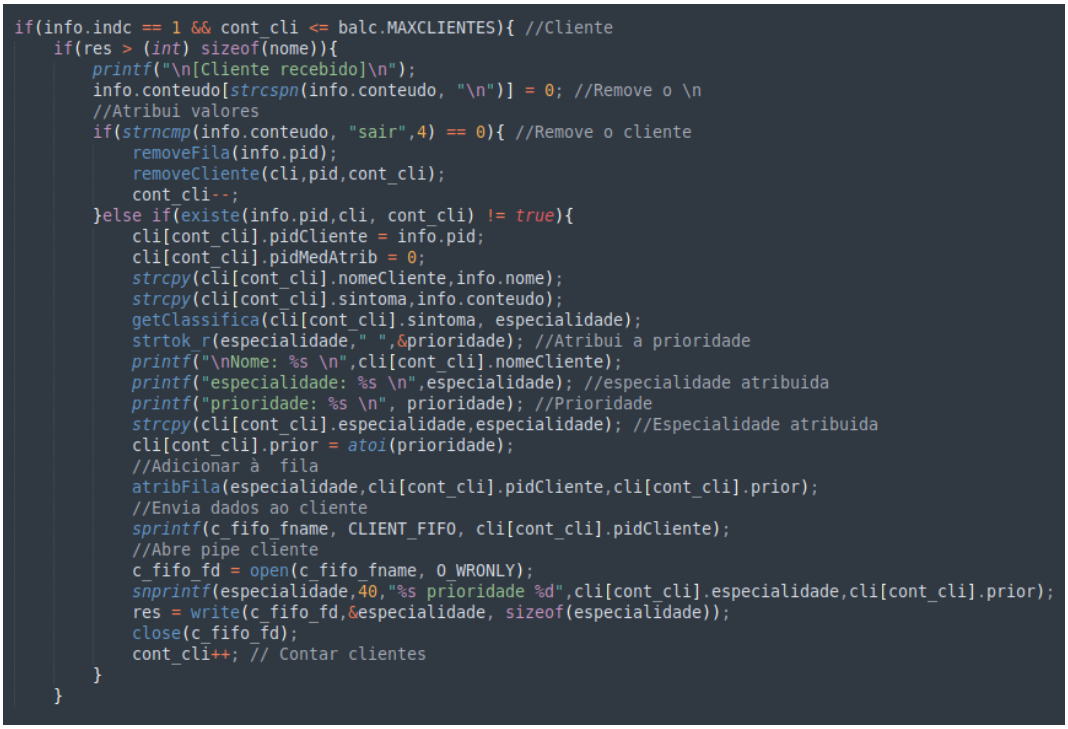
O programa “balcão” é o primeiro a ser executado do sistema e é feito essa verificação em todos os outros programas necessários ao sistema.

A sua inicialização é feita através do comando “./balcao”, e fica à espera de novos comandos introduzidos pelo utilizador ou então por uma nova entrada de um utente ou um médico no sistema. Os comandos disponíveis ao balcão são os seguintes: comando “utentes” fazendo listagem de todos os utentes no sistema estando ou não em consulta, comando “especialistas” faz a listagem de todos os médicos no sistema distinguindo aqueles que estão ou não a fazer uma consulta, comando “delut PID” remove o utente do sistema pelo PID inserido, comando “delesp PID” remove o médico do sistema pelo PID inserido, comando “freq SEGUNDOS” passa a mostrar as filas atuais de segundos em segundos

inseridos e o comando “encerra” termina o sistema avisando todos os utentes e médicos atualmente inseridos no sistema.

Inserção de utentes e médicos no sistema

Após ser inicializado uma nova estancia correspondente a um novo cliente envia os dados ao balcão que verifica através do seu indicador (1 – cliente ou 2 – medico) se trata de um cliente ou médico, após isso verifica se o comando inserido foi “sair” sendo assim elimina esse cliente do sistema senão verifica se esse cliente recebido já existe no existe e senão envia o sintoma ao classificador que irá retornar a especialidade e a prioridade que será enviado ao cliente e adiciona os dados desse novo cliente à estrutura Utente.



```
if(info.indc == 1 && cont_cli <= balc.MAXCLIENTES){ //Cliente
    if(res > (int) sizeof(nome)){
        printf("\n[Cliente recebido]\n");
        info.conteudo[strcspn(info.conteudo, "\n")] = 0; //Remove o \n
        //Atribui valores
        if(strncmp(info.conteudo, "sair",4) == 0){ //Remove o cliente
            removeFila(info.pid);
            removeCliente(cli,pid,cont_cli);
            cont_cli--;
        }else if(existe(info.pid,cli, cont_cli) != true){
            cli[cont_cli].pidCliente = info.pid;
            cli[cont_cli].pidMedAtrib = 0;
            strcpy(cli[cont_cli].nomeCliente,info.nome);
            strcpy(cli[cont_cli].sintoma,info.conteudo);
            getClassifica(cli[cont_cli].sintoma, especialidade);
            strtok_r(especialidade," ",&prioridade); //Atribui a prioridade
            printf("\nNome: %s \n",cli[cont_cli].nomeCliente);
            printf("especialidade: %s \n",especialidade); //especialidade atribuida
            printf("prioridade: %s \n", prioridade); //Prioridade
            strcpy(cli[cont_cli].especialidade,especialidade); //Especialidade atribuida
            cli[cont_cli].prior = atoi(prioridade);
            //Adicionar à  fila
            atribFila(especialidade,cli[cont_cli].pidCliente,cli[cont_cli].prior);
            //Envia dados ao cliente
            sprintf(c_fifo_fname, CLIENT_FIFO, cli[cont_cli].pidCliente);
            //Abre pipe cliente
            c_fifo_fd = open(c_fifo_fname, O_WRONLY);
            snprintf(especialidade,40,"%s prioridade %d",cli[cont_cli].especialidade,cli[cont_cli].prior);
            res = write(c_fifo_fd,&especialidade, sizeof(especialidade));
            close(c_fifo_fd);
            cont_cli++; // Contar clientes
        }
    }
```

Figura 2 – Inserção de Utentes e verificações

Se for recebido um novo médico, o balcão verifica se foi recebido o comando “sair” eliminando assim esse médico do sistema, se recebeu o comando “acabou” termina a consulta em que esse médico se encontrava apagando após isso o cliente que esteve na consulta e colocando novamente o médico disponível para novas consultas seguindo todas as verificações. Se recebeu o comando “alarme” enviado de 20 em 20 segundos do médico para o balcão lembrando que este ainda se encontra no sistema, para fazer a inserção de um novo médico verifica se a especialidade recebida pelo médico segue os padrões do sistema adiciona assim o novo médico.

```
}else if(info.indc ==  2 && cont_med <= balc.MAXMEDICOS){ //Medico
    if(res >  (int)sizeof(nome)){
        printf("\n[Medico recebido]\n");
        info.conteudo[strcspn(info.conteudo, "\n")] = 0; //Remove o \n
        if(strncmp(info.conteudo, "sair",4) == 0){
            removeMedico(med, info.pid,cont_med);
            cont_med--;
        }else if(strncmp(info.conteudo, "acabou",6) == 0){ //Comando para acabar a consulta
            printf("\nAcabou a consulta do Médico: %d \n", info.pid);
            for(int i=0;i<cont_med;i++){
                if(med[i].pidMedico == info.pid && med[i].pidUtenAtrib != 0){
                    //Remove o cliente
                    removeCliente(cli,med[i].pidUtenAtrib,cont_cli);
                    cont_cli--;
                    med[i].pidUtenAtrib = 0; //Medico volta a ficar disponivel
                }
            }
        }else if(strncmp(info.conteudo,"alarme",6) == 0){
            printf("\nO médico %d ainda se encontra presente no sistema.\n",info.pid);
        }else if(verificaEsp(info.conteudo) == true){
            //Atribui valores
            med[cont_med].pidMedico = info.pid;
            med[cont_med].pidUtenAtrib = 0;
            strcpy(med[cont_med].nomeMedico,info.nome);
            strcpy(med[cont_med].especialidade,info.conteudo);
            printf("\nNome: %s ",med[cont_med].nomeMedico);
            printf("\nEspecialidade: %s \n",med[cont_med].especialidade);
            //Envia dados ao medico
            //sprintf(c_fifo_fname, MEDICO_FIFO, med[cont_med].pidMedico);
            //c_fifo_fd = open(c_fifo_fname, O_WRONLY);
            //res = write(c_fifo_fd,&especialidade, sizeof(especialidade));
            //close(c_fifo_fd);
            cont_med++; //Contar medicos
        }
    }
}
```

Figura 3 - Inserção de Médicos e verificações

A distinção feita entre receber comandos do teclado ou então receber novas mensagens de utentes ou médicos foi realizado usando um select que realiza essa distinção entre o teclado e o namedpipe usado para a comunicação.

Verificação de Consultas

```c
//Verifica consultas e alerta utentes e médicos
for(int i= 0;i<cont_cli;i++){
    for(int j=0;j<cont_med;j++){
        if(strcmp(cli[i].especialidade,med[j].especialidade) == 0){ //Há clientes e médicos com a mesma especialidade
            if(cli[i].pidMedAtrib == 0 && med[j].pidUtenAtrib == 0){ //Estão disponiveis
                //printf("\nEntrei\n");
                cli[i].pidMedAtrib = med[j].pidMedico;
                med[j].pidUtenAtrib = cli[i].pidCliente;
                printf("\nPID-Medico: %d\n", med[j].pidMedico);
                printf("\nPID-Cliente: %d\n",cli[i].pidCliente);
                //Informa cliente
                snprintf(comando,30,"consulta %d-%d",med[j].pidMedico,cli[i].pidCliente);
                sprintf(c_fifo_fname, CLIENT_FIFO, cli[i].pidCliente);
                c_fifo_fd = open(c_fifo_fname, O_WRONLY);
                res = write(c_fifo_fd,&comando, sizeof(comando));
                close(c_fifo_fd);
                //Informa medico
                sprintf(c_fifo_fname, MEDICO_FIFO, med[j].pidMedico);
                c_fifo_fd = open(c_fifo_fname, O_WRONLY);
                res = write(c_fifo_fd,&comando, sizeof(comando));
                close(c_fifo_fd);
                removeFila(cli[i].pidCliente); //Remove cliente da fila
            }
        }
    }
}
```

Figura 4 - Verificação de realização de consultas

Através do código mencionado na fig.4 é feita a verificação de possíveis novas consultas, começa por verificar se há clientes e médicos com a mesma especialidade, após isso verifica o campo cli.pidMedAtrib ou med.pidCliAtrib que será diferente de 0 se o cliente ou o médico tiver um pid associado estando em consulta nesse momento em consulta. Se passarem todas essas verificações começa o processo de consulta, é enviado para o cliente e médico o pid correspondente ao utilizador com quem irá realizar a consulta e após isso remove o cliente da fila onde se encontra. A consulta funciona como um diálogo entre utente e médico podendo um deles sair da consulta através do comando “adeus”.

Remover utentes ou médicos do sistema

```c
void removeCliente(utente cli[], int pid, int count_cli){
    int i=0,j = 0, pos = -1;
    for(i = 0;i<count_cli;i++){
        if(cli[i].pidCliente == pid){
            pos = i;
            break;
        }
    }
    if(pos != -1){
        for(i = pos; i<count_cli-1;i++){
            cli[i] = cli[i+1];
        }
    }else{
        printf("\nCliente não encontrado.\n");
    }

}

void removeMedico(medico med[], int pid, int count_med){
    int i=0,j = 0,pos = -1;
    for(i = 0;i<count_med;i++){
        if(med[i].pidMedico == pid){
            pos = i;
            break;
        }
    }
    if(pos != -1){
        for(i = pos;i<count_med-1;i++){
            med[i] = med[i+1];
        }
    }else{
        printf("\nMedico não encontrado.\n");
    }
}
```

Figura 5 - Remover utentes ou médicos

Através das duas funções demonstradas acima é realizado o processo de remoção de utentes ou médicos do sistema sendo este avisado e terminado o seu processo.

Atribuição de fila ao utente

```
void atribFila(char *especialidade, int pid_cl, int prior){
    int i = 0,pos = -1;
    if(strncmp(especialidade,"oftalmologia",12) == 0){
        if(oftal[MAX_FILA-1][0] == 0){ //Verifica se há espaço na fila
            if(oftal[0][0] != 0){
                for(i = 0;i<MAX_FILA;i++){
                    if(oftal[i][1] >= prior && oftal[i][0] != 0){ //Verifica se a prior maior ou igual que a do vetor
                        pos = i;
                    }
                }
                if(pos != -1){
                    for(int i=pos;i<MAX_FILA-1;i++){
                        if(oftal[i+1][0] != 0){
                            oftal[i+1][0] = oftal[i][0];
                            oftal[i+1][1] = oftal[i][1];
                        }else{
                            oftal[i+1][0] = oftal[i][0];
                            oftal[i+1][1] = oftal[i][1];
                            break;
                        }
                    }
                    oftal[pos][0] = pid_cl;
                    oftal[pos][1] = prior;
                }else if(pos == -1){
                    for(i=0;i<MAX_FILA;i++){
                        if(oftal[i][0] == 0){ //Se estiver vazio
                            oftal[i][0] = pid_cl;
                            oftal[i][1] = prior;
                            break;
                        }
                    }
                }
            }else{
                oftal[0][0] = pid_cl;
                oftal[0][1] = prior;
            }
        }else{
            printf("\nA fila para esta especialidade  está  cheia, volte mais tarde.");
        }
```

Figura 6 - Função de atribuição de fila a utentes

A função “atribFila” é responsável por atribuir um lugar na fila a cada utente, tendo 5 filas distintas para cada tipo de especialidade autorizada, sendo o lugar na fila dependente da prioridade atribuída a cada utente, sendo uma prioridade menor atribuída aos primeiros lugares na fila. Sempre que um utente sair do sistema, é removido o seu lugar da fila e os seguintes passam um lugar para a frente.

Estruturas usadas pelo sistema

```c
typedef struct fifo_balcao fifo_info;
struct fifo_balcao{
    int indc;
    pid_t pid;
    char nome[MAX_NAME];
    char conteudo[MAX_CONT];
};

typedef struct especialista medico;
struct especialista{
    pid_t pidMedico;                    //Processo Médico
    char nomeMedico[MAX_NAME];          // Nome Médico
    char especialidade[MAX_NAME];       //Especialidade MÉDICO
    int pidUtenAtrib;                   //PID do Utente atribuido para a consulta (0 s/ utente)
};

typedef struct cliente utente;
struct cliente{
    pid_t pidCliente;                   //Processo Cliente
    char nomeCliente[MAX_NAME];         //Nome Cliente
    char sintoma[MAX_NAME];             //Sintoma Cliente
    char especialidade[MAX_NAME];       //Especialidade atribuida pelo classificador
    int prior;                          //Prioridade Atribuida pelo Classificador
    int pidMedAtrib;                    //PID do Médico atribuido para a consulta (0 s/medico)
};

typedef struct servidor balcao;
struct servidor{
    pid_t pid;                          //Processo Balcao
    int MAXCLIENTES;                    //Limite de Clientes
    int MAXMEDICOS;                     //Limite de Medicos
};
```

Figura 7 – Estruturas

Na figura 7 estão presentes todas as estruturas que achamos serem necessárias para a implementação deste sistema.

A estrutura “fifo_balcao” é responsável por armazenar a informação necessária para ser enviado para o balcão dos utentes ou médicos. O campo “indc” pode ser 1 sendo indicativo para os utentes e 2 sendo para os médicos é através deste indicador que o balcão faz a distinção entre ambos, recebe também o pid do processo que fez o envio, o nome de utente ou médico e o seu conteúdo que poderá ser o sintoma no caso do utente e a especialidade no caso do médico ou então o comando “sair” que é possível por ambos.

A estrutura “especialista” é responsável por armazenar os dados necessários dos médicos e a estrutura “cliente” armazena os dados dos utentes.

A estrutura “servidor” é responsável por armazenar os dados necessários para o balcão tendo o pid do balcão e os dois valores inteiros que são obtidos pelas duas variáveis de ambiente “MAXCLIENTES” e “MAXMEDICOS” que indicam o número máximo de clientes e médicos no sistema em simultâneo.

Obtenção das variáveis ambiente

```c
//obtem variavel de ambiente MAXCLIENTES utilizada no balcao
int getMAXCLIENTES(){
    int value;
    char *var = getenv("MAXCLIENTES");

    if (var == NULL){   //caso a variavel de ambiente não foi definida avisa e termina
        fprintf(stderr, "balcao: Variavel de ambiente \"MAXCLIENTES\" desconhecida!\n");
        return -1;
    }

    if ((sscanf(var, "%d", &value)!=1) || value < 1){// caso a leitura para o inteiro falhe
        fprintf(stderr, "balcao: Variável de ambiente \"MAXCLIENTES\" tem um valor incorreto: %s\n",var);
        fprintf(stderr, "balcao: A variável de ambiente \"MAXCLIENTES\" deve conter um valor inteiro maior do que 1.\n");
        return -1;
    }

    return value;
}


//obtem variavel de ambiente MAXMEDICOS utilizada no balcao
int getMAXMEDICOS(){
    int value;
    char *var = getenv("MAXMEDICOS");

    if (var == NULL){   //caso a variavel de ambiente não foi definida avisa e termina
        fprintf(stderr, "balcao: Variavel de ambiente \"MAXMEDICOS\" desconhecida!\n");
        return -1;
    }

    if ((sscanf(var, "%d", &value)!=1) || value < 1){// caso a leitura para o inteiro falhe
        fprintf(stderr, "balcao: Variável de ambiente \"MAXMEDICOS\" tem um valor incorreto: %s\n",var);
        fprintf(stderr, "balcao: A variável de ambiente \"MAXMEDICOS\" deve conter um valor inteiro maior do que 1.\n");
        return -1;
    }


    return value;
}
```

Figura 8 - Programa getEnvVars.c

Foi criado este programa auxiliar responsável apenas pela obtenção das variáveis ambiente e sua transformação em valores inteiros, estas duas funções são chamadas dentro do programa balcão e posteriormente adicionadas à estrutura do balcão.

Funções do Sistema: Utentes

No programa “cliente” sendo executado através do comando “./cliente NOME” introduzindo o nome do cliente através da linha de comandos, começa por pedir o sintoma ao cliente que após ser lido envia essa informação ao balcão e em seguida recebe a especialidade atribuída e a sua prioridade.

Ficando assim à espera de ser atendido por um médico da sua especialidade.

O cliente distingue novos comandos introduzidos pelo teclado ou então comunicações feitas pelo balcão ou o médico associado através de um select.

Se o cliente estiver numa consulta o namedpipe associado ao seu recetor será do médico com quem se encontra em consulta e poderá sair da consulta através do comando “adeus” e após isso será eliminado do sistema e o seu processo terminará.

Funções do Sistema: Médicos

No programa “medico” sendo executado através do comando “./medico NOME ESPECIALIDADE” introduzindo o nome do médico através da linha de comandos e a sua especialidade, envia de seguida ao cliente essa informação que irá verificar e adicionar o médico ao sistema. Podendo após isso enviar o comando “sair” ao balcão eliminando-o do sistema.

O médico à semelhança do cliente também distingue os comandos através de um select, funcionando da mesma forma para a consulta e após acabar a consulta o médico envia automaticamente o comando “acabou” que irá ser reconhecido pelo balcão como o final da consulta, eliminando o cliente e colocando o médico como disponível novamente.

Notificação de presença enviada para o balcão

```c
void lembrete(int num, siginfo_t *info, void *uc){
    //Mandar mensagem ao balcao
    int s_fifo_fd;  /* identificador do FIFO do servidor */
    s_fifo_fd = open(SERVER_FIFO, O_RDWR); /* bloqueante */
    int med_indc = 2;
    int pid = getpid();
    fifo_info infob; //Escrita fifo (servidor)
    infob.indc = med_indc;
    infob.pid = pid;
    strcpy(infob.nome,"medico");
    strcpy(infob.conteudo,"alarme");
    write(s_fifo_fd,&infob,sizeof(fifo_info)); //Indicador de Médico
    //Avisou o balcão que ainda se encontra presente
    //printf("\nMensagem enviada ao balcao.\n");
}
```

Figura 9 - Enviar lembrete ao balcão

Através da função mencionada na fig.9 o médico envia um lembrete ao balcão a cada 20 segundos utilizando o sistema de alarmes indicando que ainda se encontra disponível no sistema. Na figura seguinte é feita a configuração da função alarme.

```c
//Tratar sinal
struct sigaction act;
act.sa_sigaction = lembrete;
act.sa_flags = SA_SIGINFO;
sigaction(SIGALRM,&act,NULL);
```

Figura 10 - Configuração de Sinal de Alarme

Ficheiro Makefile

Como pedido tivemos que realizar um ficheiro makefile para ser possível compilar todos os ficheiros necessários para o sistema através de um único ficheiro.

```
all: balcao.o cliente.o medico.o getEnvVars.o
	cc balcao.o getEnvVars.o -o balcao
	cc cliente.o -o cliente
	cc medico.o -o  medico

getEnvVars.o: getEnvVars.c
	cc -c getEnvVars.c

balcao.o: balcao.c structs.h getEnvVars.c
	cc -c balcao.c

medico.o: medico.c medico.h structs.h
	cc -c medico.c

cliente.o: cliente.c cliente.h structs.h
	cc -c cliente.c

clear:
	rm *.o balcao cliente medico getEnvVars
```

Figura 11 - Ficheiro Makefile

Interrupções por parte dos Utentes ou Médicos

```
void sair(int num, siginfo_t *info, void *uc){
    char    c_fifo_fname[25];
    int s_fifo_fd;
    s_fifo_fd = open(SERVER_FIFO, O_RDWR);
    int med_indc = 2;
    int pid = getpid();
    sprintf(c_fifo_fname, MEDICO_FIFO, pid);
    fifo_info infob; //Escrita fifo (servidor)
    infob.indc = med_indc;
    infob.pid = pid;
    strcpy(infob.nome,"medico");
    strcpy(infob.conteudo,"sair");
    write(s_fifo_fd,&infob,sizeof(fifo_info)); //Indicador de Médico
    unlink(c_fifo_fname);
    exit(0);
}
```

Figura 12 - Sair (Comando Ctrl+C)

Sempre que um Utente ou Médico decidirem terminar de forma inesperada o seu processo através do Comando Ctrl+C, é enviado a mensagem “sair” ao balcão que irá eliminar esse utilizador e fechará assim o seu processo sem ocorrer nenhum tipo de interrupção para o resto do sistema. O exemplo da fig.12 está no ambiente de médico mas a função do utente é semelhante a esta.

Implementações

Após revermos o projeto realizado e executar vários testes achamos que todos os requisitos pedidos foram implementados. Desde mecanismos de comunicação e outros, os comandos e as suas respostas seguindo os modelos pedidos.

Conclusão

Após acabarmos o projeto de Sistemas Operativos podemos retirar algumas conclusões como grupo gerimos bem o nosso tempo dedicado para a realização deste projeto e que conseguimos chegar a um resultado que reflete o nosso trabalho e foi um ótimo desafio.

Conseguimos também aumentar o nosso conhecimento em linguagem C em ambiente Unix e preparar-nos da melhor maneira para o exame final.