
Sistemas Operativos 21/22

# Trabalho Prático – Programação em C para UNIX

Docente: João Durães

Ano Letivo 2021/2022

**Trabalho feito por:**

Pedro Praça - 2020130980

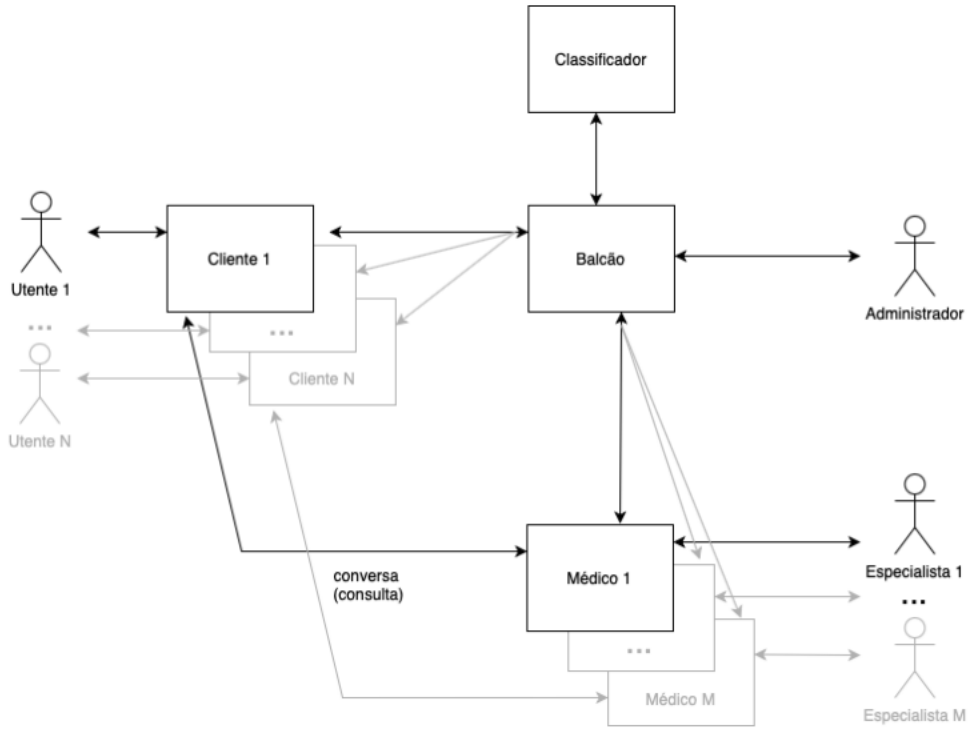
Francisco Simões - 2019133920

# Conteúdo

# Introdução

O trabalho prático consiste na implementação de um sistema para gerir o atendimento de clientes no estabelecimento médico, denominado de “MEDICALso”, que encaminha e medeia a interação entre clientes, médicos e balcão de atendimento.

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



## Estrutura de Dados

As estruturas de dados utilizadas para este trabalho estão todas declaradas no Balcao.h (tipo_medico, tipo_cliente, tipo_mensagem).

Quanto à estrutura *tipo_medico vai guardar as informações relativas a cada especialista*:

- pid
- nome
- especialidade
- pid_cliente – caso este valor seja 0 está à espera de clientes para atender, caso contrário está em consulta

No que toca à estrutra *tipo_cliente vai guardar as informações relativas a cada utente:*

- *Pid*
- *Nome*
- *Sintomas*
- *Especialidade recebida*
- *Prioridade atribuída*
- *Número de pessoas à sua frente – o valor vai alterando à medida que são adicionados clientes da mesma especialidade porém mais prioritários e estejam todos à espera de ser atendidos*

*Por fim, a última estrutura tipo_mensagem vai guardar as informações relativas a cada mensagem enviada:*

- *Pid*
- *Tipo – se é enviada pelo balcão, médico ou cliente*
- *Mensagem*
- *Nome*

## Struct tipo_medico

```
typedef struct {
    pid_t pid;
    char nome[BUFFER]; //nome cliente
    char especialidade[BUFFER];//especialidade obtida de cada cliente pós classificador
    pid_t pid_cliente;

} tipo_medico;
```

## Struct tipo_cliente

```
typedef struct {
    pid_t pid;
    char nome[BUFFER]; //nome cliente
    char sintomas[BUFFER]; // sintomas cliente
    char especialidade[BUFFER];//especialidade obtida de cada cliente pós classificador
    int prioridade; // prioridade cliente
    int npessoas_frente; // quantidade de pessoas a frente
} tipo_cliente;
```

## Struct tipo_mensagem

```
typedef struct {
    pid_t pid;
    int tipo;
    char mensagem[BUFFER]; // mensagem / sintomas do cliente / especialidade do medico
    char nome[BUFFER]; //nome especialista / nome cliente
} tipo_mensagem;
```

Quanto à alocação dinâmica de memória reservamos espaço para os clientes, especialistas e para a fila de espera de cada especialidade.

Para os clientes o seu tamanho máximo é o valor da variável de ambiente N;

Para os especialistas o seu tamanho máximo é o valor da variável de ambiente M;

E para as filas de espera de cada especialidade é o valor 5 (ESP_QUEUE);

```
clientes = malloc( _Size: sizeof(tipo_cliente) * N);//ALOCA NMAX de clientes no sistema
especialistas = malloc( _Size: sizeof(tipo_medico) * M);//ALOCA NMAX de medicos no sistema
oftalmologia = malloc( _Size: sizeof(tipo_cliente) * ESP_QUEUE);
neurologia = malloc( _Size: sizeof(tipo_cliente) * ESP_QUEUE);
estomatologia = malloc( _Size: sizeof(tipo_cliente) * ESP_QUEUE);
ortopedia = malloc( _Size: sizeof(tipo_cliente) * ESP_QUEUE);
geral = malloc( _Size: sizeof(tipo_cliente) * ESP_QUEUE);
```

## Pipes anónimos

Função, criada antes do loop do balcão, que recebe os arrays de inteiros correspondentes aos dois ***pipes***, ***pipe*** balcão->classificador e classificador->balcão. Esta cria dois ***pipes***, um ***pipe*** só com uma extremidade aberta, de leitura, c_b e um ***pipe***, só com uma extremidade aberta, de escrita, b_c. Isto é, o primeiro ***pipe*** é usado para o classificador escrever as mensagens para o balcão e por outro lado o segundo ***pipe***, b_c, serve para o balcão enviar mensagens para o classificador. De seguida cria um processo filho, onde se realiza o redireccionamento do stdin e stdout e posteriormente é lançado o classificador para comunicar através destes ***pipes***, com o balcão.

Já dentro do while do balcão, é criada uma outra função que recebe estes parâmetros por referência e que contém o processo pai onde é feita a escrita para o ***pipe*** de escrita de b_c e a leitura da especialidade obtida no ***pipe*** de leitura de c_b.

```c
int abre_pipes(int b_c[2], int c_b[2], int *pid) {

    int estado;
    if (pipe(b_c) == -1) { exit( _Code: 1); } // Erro ao criar pipe balcao-> classificador se devolver -1
    if (pipe(c_b) == -1) exit( _Code: 2); // Erro ao criar pipe classificador->balcao se devolver -1
    *pid = fork();
    if (*pid < 0) { exit( _Code: 3); } // Erro ao criar processo filho se menor 0

    if (*pid == 0) {//PROCESSO FILHO
        // fechar o stdin e duplicar b_c[0] para conseguir armazenar no canal b_c, na extremidade de escrita, os sintomas
        close(STDIN_FILENO);
        dup( _FileHandle: b_c[0]);
        close( _FileHandle: b_c[0]);
        close( _FileHandle: b_c[1]);
        // fechar o stdout e duplicar c_b[1] para conseguir ler no canal c_b, na extremidade de leitura, a especialidade e prioridade
        close(STDOUT_FILENO);
        dup( _FileHandle: c_b[1]);
        close( _FileHandle: c_b[1]);
        close( _FileHandle: c_b[0]);
        // executa o programa classificador
        printf( _Format: "EXECUTA CLASSIF");
        execl( _Filename: "./classificador",  _ArgList: "classificador", NULL);
        perror( _ErrMsg: "\nPrograma classificador não encontrado\n");
        return 4;
    }
    return 0;
}
```

```c
int classifica_sintomas(int *b_c, int *c_b, char *especialidade, int *pid, char *sintomas) {
    if (*pid > 0) {
        //fecho as extremidades já usadas em filho que não vão ser necessárias no processo pai
        close( _FileHandle: c_b[1]);
        close( _FileHandle: b_c[0]);

        //armazena os sintomas na extremidade de escrita b_c[1]
        write( _Filehandle: b_c[1], sintomas, strlen(sintomas));

        //envia a especialidade na extremidade de leitura c_b[0]
        read( _FileHandle: c_b[0], especialidade, BUFFER);
        especialidade[strlen(especialidade)] = '\0';
    }

    return 0;
}
```

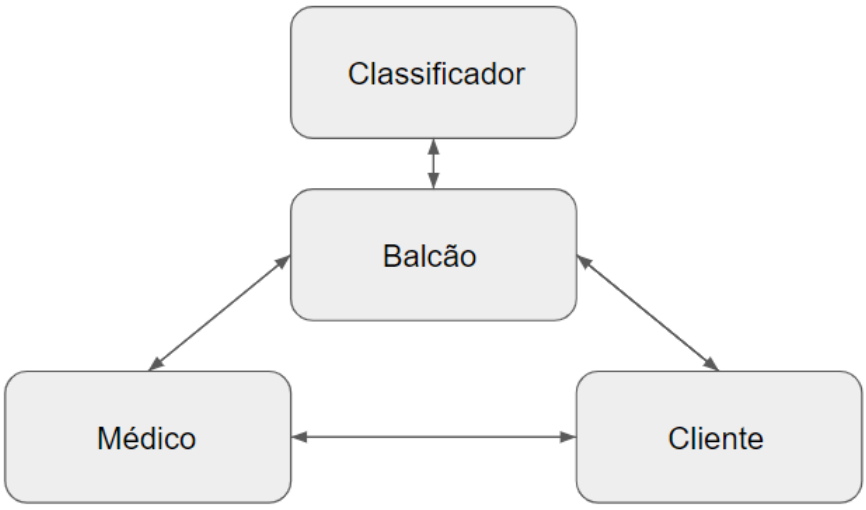
## Mecanismo de comunicação

O meio de comunicação usado pelo balcão, médico e o cliente é o ***named pipe***. Sempre que eles querem comunicar entre eles é chamada uma função ***envia_mensagem*** que escreve nos ***fifos*** uma estrutura ***tipo_mensagem*** que é comum a todos eles, visto que esta estrutura envia o ***pid*** de quem escreveu a mensagem, o tipo de mensagem que é(se é de chegada de um médico se é para esperar um cliente), a mensagem em si (tanto pode ser uma mensagem de aviso como podem ser os sintomas do cliente ou a especialidade do especialista) e por fim o nome (nome do cliente ou do especialista).

O cliente ao iniciar envia uma mensagem para o balcão com os seu sintomas e recebe dele uma especialidade, a prioridade e o número de pessoas que tem à sua frente na especialidade, aguardando por um médico lhe ser indicado e quando esse evento ocorre é iniciada uma consulta.

No médico, quando chega ao sistema aguarda até que um cliente lhe seja atribuído, até que quando este ocorre se inicia uma consulta.

Quando se reúnem todas as condições para que haja uma consulta o balcão envia ao cliente o ***pid*** do médico e o médico recebe o ***pid*** do cliente para falarem os dois através de ***pipes*** dentro dos seus ***selects***.



```c
void envia_mensagem(pid_t pid, tipo_mensagem mensagem, char *fifo_path) {
    int fifo_fd = 0;
    char fifo[BUFFER];
    if (strcmp(fifo_path, BALCAO_FIFO) == 0) {
        strcpy(fifo, BALCAO_FIFO);
    } else {
        sprintf(fifo, fifo_path, pid);
    }
    fifo_fd = open(fifo, O_WRONLY);
    if (write(fifo_fd, &mensagem, sizeof(tipo_mensagem)) != sizeof(tipo_mensagem)) {
        exit(EXIT_FAILURE);
    }
    close(fifo_fd);
}
```

## Select

Foram criados 3 ***selects*** neste projeto, um no cliente, um no médico e outro no balcão.

O ***select*** do balcão serve para o balcão conseguir estar à escuta tanto do teclado (STDIN_FILENO) como consiga receber mensagens do médico e do cliente.

O ***select*** do cliente serve tanto para receber mensagens do médico, quando está em consulta, como receber mensagens do balcão, quando for terminar abruptamente, por exemplo.

Por fim, o ***select*** do médico recebe mensagens do cliente, quando está em consulta, como recebe mensagens do balcão, quando for terminar abruptamente, por exemplo.

Quando acontece uma terminação abrupta no balcão o ***select*** retorna o valor –1 e por essa razão sempre que isso ocorre percorremos todos os médicos e clientes para os mandar encerrar.

```
FD_ZERO(&rfds);
FD_SET(STDIN_FILENO, &rfds);
FD_SET(b_fifo_fd, &rfds);
maxfd = STDIN_FILENO > b_fifo_fd ? STDIN_FILENO : b_fifo_fd;
retval = select(maxfd + 1, &rfds, NULL, NULL, NULL);
if (retval == -1) { // termina os medicos e os clientes
    for (int i = 0; i < n_especialistas; i++) {
        mensagem.tipo = TIPO_SAIDA_MEDICO;
        sprintf(mensagem.mensagem, _Format: "O balcão mandou encerrar o medico");
        envia_mensagem(especialistas[i].pid, mensagem, MEDICO_FIFO);
    }
    for (int i = 0; i < n_clientes; i++) {
        mensagem.tipo = TIPO_SAIDA_CLIENTE;
        sprintf(mensagem.mensagem, _Format: "O balcão mandou encerrar o cliente");
        envia_mensagem(clientes[i].pid, mensagem, CLIENTE_FIFO);
    }
```

## Comandos

O balcão permite ao administrador a escrita de comandos para ajustar e controlar o seu funcionamento. Os comandos podem ser escritos a qualquer altura, a par com todas as restantes atividades do balcão. Os comandos são os seguintes:

| |
|---|
| **utentes** – lista os utentes em fila de espera indicando qual a especialidade e qual a prioridade, e também os utentes atualmente a serem atendidos, em que especialidade e por qual especialista; |
| **especialistas** – indica a lista de especialistas conhecidos e estado atual (à espera, a atender o utente X); |
| **delut X** – remove o utente em espera X, informando-o através do seu programa cliente (que depois encerra); |
| **delesp X** – remove o especialista X, informando-o através do seu programa médico (que depois encerra). Válido apenas para especialistas que não estejam a atender nenhum utente no momento; |
| **freq N** – passa a apresentar a ocupação das filas de espera de N em N segundos; |
| **encerra** – termina o sistema, avisando os clientes, os médicos e o classificador. |

## Makefile

| Regras: | Dependências: | Operações: |
|---|---|---|
| all: Balcao Cliente Medico | | Compila todos os programas associados ao trabalho |
| Balcao: Balcao.o | gcc Balcao.o -o Balcao -pthread | Compila o objeto do Balcao |
| Balcao.o: Balcao.c Balcao.h Cliente.h | gcc – Balcao.c | Compila o programa Balcao |
| Cliente: Cliente.o | gcc –c Cliente.o -o Cliente | Compila o objeto Cliente |
| Cliente: Cliente.c Cliente.h | gcc –c Cliente.c | Compila o programa Cliente |
| Medico: Medico.o | gcc Medico.o -o Medico | Compila o objeto Medico |
| Medico.o: Medico.c Medico.h | gcc –c Medico.c | Compila o programa Medico |
| clean | rm *.o Balcao Cliente | Limpa todos os objetos |
| | rm -f /tmp/*_fifo | Limpa todos os fifos |

## Notas

Não foi conseguida a implementação de threads no trabalho. Não foi realizado o sinal de vida do médico nem do comando freq.

# Conclusão

Ao longo deste trabalho, deparámo-nos com vários desafios, estes desafios permitiu-nos colocar em prática conceitos importantes sobre a arquitetura, criação e desenvolvimento de programas para sistemas UNIX, dando-nos uma visão para os vários detalhes dessas atividades.