Departamento de Engenharia Informática e de Sistemas

Instituto Superior de Engenharia de Coimbra

Licenciatura em Engenharia Informática

Sistemas Operativos 2021/2022

Relatório do Trabalho Prático de Sistemas Operativos - *MEDICALso*

META 2

André Lopes - 2019139754

Samuel Tavares - 2019126468

ÍNDICE

INTRODUÇÃO

Este Trabalho tem como objetivo pôr em prática a matéria lecionada nas aulas de Sistemas Operativos.

Este consiste em implementar uma comunicação entre diversos processos, com parecenças a um balcão de atendimento médico, onde um cliente se dirige ao balcão e é posteriormente encaminhado a um médico disponível.

Serão implementadas classificação e prioridade de sintomas, filas de espera, consultas entre especialista e utente, tudo isto por meio “remoto”.

Estrutura de Dados

As estruturas de dados necessárias para o correto funcionamento dos programas (medico, cliente e balcao) foram distribuídas pelos respetivos ficheiros header files.

No ficheiro medico.h, temos uma estrutura relativa aos dados dos médicos, tais como o nome, a sua área de especialidade, um inteiro que vai corresponder ao seu id. Caso este seja 0, o medico ainda não existe, caso contrário o medico existe e o seu id corresponde ao seu PID. Existe também uma variável inteira que identifica se o médico está a atendar algum utente ou se está inativo.

No ficheiro cliente.h, existe uma estrutura que vai guardar a informação dos clientes, tais como o nome, os sintomas apresentados, a área de especialidade que lhe foi atribuída, assim com a sua prioridade. Tal com o médico, o cliente apresenta um inteiro que vai corresponder ao seu id. Caso este seja 0, o cliente ainda não existe, caso contrário o cliente existe e o seu id corresponde ao seu PID. Existe também uma variável inteira que indica a posição do utente na lista de espera relativa à especialidade que lhe foi atribuída.

No ficheiro balcao.h encontra-se uma estrutura organizada por 4 inteiros: número de clientes, número de médicos, maxClientes e maxMedicos (guardam as variáveis de ambiente), e 2 arrays de inteiros para a fila de espera e número de especialistas, estas guardam o número de fila dos clientes ligados e o número de especialistas existentes em cada área. No header, existe também um enum que vai facilitar a identificação das especialidades quando estas foram atribuídas aos clientes/médicos.

Em relação à alocação da memória das estruturas, esta vai ser alocado de forma estática. Como são definidas duas variáveis que indicam o tamanho máximo dos arrays que vão guardar os clientes e médicos, e visto que o tamanho deles não vai ser alterado, não se achou necessária a alocação dinâmica das mesmas.

Relativamente a estruturas relacionadas com a comunicação entre o balcão, o medico e o cliente, estas serão referidas durante o processo de explicação dos métodos implementados para cumprir essa comunicação.

```
enum especialidades{
    oftalmologia = 0,
    neurologia = 1,
    estomatologia = 2,
    ortopedia = 3,
    geral = 4
};
```

```c
typedef struct balcao Balcao, *pBalcao;
struct balcao{
    int maxClientes;
    int maxMedicos;
    int numClientes;
    int numMedicos;
    int numEspecialistas[MAXESPECIALIDADES]; // numero de especialistas por area
    int filaEspera[MAXESPECIALIDADES][5]; //numero de pessoas em fila de espera por area
    int tempoFila;
    pthread_mutex_t mutex;

    //Exemplo:
    //filaEspera[0] - 1 utente - oftalmologia
    //filaEspera[1] - 3 utentes (na fila de espera) - neurologia
    // ...

};
```

```c
typedef struct medico Medico, *pMedico;
struct medico{

    char nome[30];
    char especialidade[100];
    int id;
    // 0 - medico nao existe
    // > 0, - medico existe e o id corresponde ao seu pid
    int estado;
    // 0 - parado
    // 1 - a trabalhar
    pid_t clienteAtender;
    long isAlive;
};
```

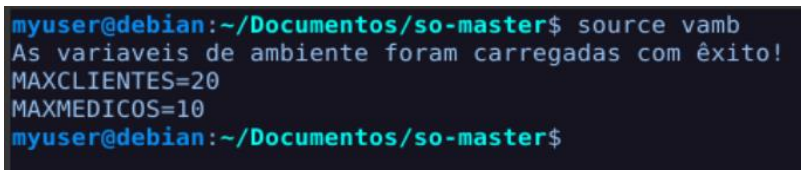
```c
typedef struct cliente Cliente, *pCliente;
struct cliente{

    char nome[30];
    char sintomas[100];
    char areaEspecialidade[100];
    int prioridade;
    int id;
    // 0 - cliente nao existe
    // > 0, - cliente existe e o id corresponde ao seu pid
    int posicaoListaEspera;
    int atendido;
    pid_t medicoAtribuido;
};
```

VARIÁVEIS DE AMBIENTE

Foram criadas duas variáveis de ambiente, as variáveis **MAXCLIENTES** e **MAXMEDICOS** que vão guardar o valor máximo de clientes e médicos que podem estar ligados em simultâneo.

As variáveis de ambiente encontram-se num ficheiro auxiliar denominado vamb, e vão ser recebidas pelo balcao através da função getenv, e também a função sscanf para converter para valores inteiros.

Nota: Para ter acesso às variáveis de ambiente, é necessário executar o comando "source vamb" no terminal, ou exportá-las diretamente através da Shell.



PIPES ANÓNIMOS

Foram criados 2 Pipes **b2c**(balcao para classificador) e **c2b**(classificador para balcao), sendo que b2c realiza a escrita e c2b encarrega-se da leitura.

Para isso foi criado um fork(), onde o filho fecha o canal de leitura, duplica o b2c de leitura, fecha ambos os canais do b2c e o canal de escrita, duplica o c2b de leitura e fecha ambos os seus canais, executando o classificador no final.

Por sua vez, o pai fecha o canal de leitura do b2c e o canal de escrita do c2b, em seguida envia os sintomas através de um write com o pipe b2c e recebe em resposta a classificação por meio de um read com o pipe c2b, eventualmente fechando o canal de escrita do b2c e o de leitura do c2b.

É também efetuado um waitpid() de forma a evitar que o classificador fique aberto como um processo anónimo.

```
if (pid == 0) { //Filho

    close( fd: 0);
    dup( fd: b2c[0]);
    close( fd: b2c[0]);
    close( fd: b2c[1]);
    close( fd: 1);
    dup( fd: c2b[1]);
    close( fd: c2b[0]);
    close( fd: c2b[1]);

    execl( path: "./classificador",  arg: "classificador", NULL);
}
```

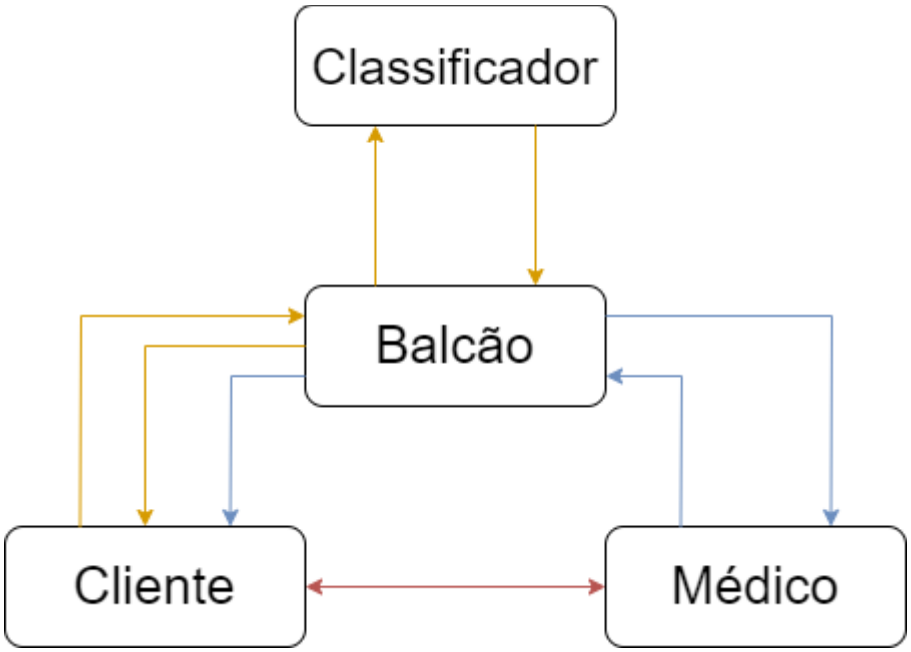
```
tam = write( fd: b2c[1], sintomas, strlen(sintomas));
tam1 = read( fd: c2b[0], especialidade_Prioridade,  nbytes: sizeof(especialidade_Prioridade) - 1);
```

```
waitpid(pid, &estado,  options: 0);
```

Comunicação entre Processos

O processo de comunicação começa com a ligação de um balcão, este trocará mensagens com médicos e clientes posteriormente iniciados, esta comunicação relativa a mensagens irá utilizar pipes que enviam e recebem estruturas por meio de write() e read().

O cliente ao iniciar envia uma mensagem de ativação para o balcão por meio de um pipe para o "FIFO" do balcão, após estabelecida a ligação, o balcão responde ao cliente, perguntando os seus sintomas, após recebidos este comunica com um processo denominados de classifica que atribuirá uma especialidade e um nível de prioridade que serão enviados ao balcão de novo, de modo que este guarde os ficheiros e os envie para o cliente. Posteriormente a este processo o cliente ficará à espera de que um médico lhe seja atribuído, quando isso vier a acontecer, este receberá uma mensagem do balcão com o pid do médico, de forma a iniciar a consulta.



Por outro lado, um processo parecido acontece quando um médico entra, sendo que este envia uma mensagem ao balcão por meio de pipes com o seu nome e especialidade, por sua vez, o balcão, recebe a informação e trata de atribuir um cliente ao médico se este possuir a mesma especialidade, o balcão atribuirá sempre o cliente que estiver em primeiro na fila de espera.

Para que a comunicação entre o cliente e o médico seja estabelecida, o balcão irá enviar o pid de um para o outro, estes passam a operar de forma autónoma do balcão, trocando mensagens entre si, apenas voltando a comunicar diretamente com o balcão quando a consulta se der por terminada.

Estruturas dedicadas a mensagens:

```c
//Estrutura da mensagem: cliente -> balcao
typedef struct {
    pid_t   pid;
    int medico_cliente; // medico = 0; cliente = 1
    char nome[30];
    int  atendido; //1 - a atender/ser atendido
    char especialidade[30];
    char    mensagem[MAX];
} Mensagem_utilizador_para_Balcao;
```

```c
//Estrutura da mensagem de resposta do balcao
typedef struct {
    pid_t pid_balcao;
    pid_t pid_medico;
    pid_t pid_cliente;
    char    mensagem[MAX];
} Mensagem_Balcao;
```

```c
//Estrutura da mensagem entre cliente e medico

typedef struct {
    char    mensagem[MAX];
} Consulta;
```

SELECT

Foram elaborados 5 Selects ao longo deste projeto, sendo 2 no cliente, 2 no médico e 1 no balcão, estes estão divididos por dois tipos, sendo estes utilizados de forma similar, mas com intuitos diferentes.

Os Selects servem para que o programa consiga ler do teclado e dos pipes em simultâneo, permitindo ao utilizador escrever comandos ao mesmo tempo que recebe mensagens oriundas de outros processos. 3 deles servem estritamente para isso, os restantes dois são utilizados para permitir ao medico e ao cliente sair do programa a qualquer momento da sua execução.

```
while (1) {
    FD_ZERO(&read_fds);
    FD_SET(0, &read_fds);//para comandos
    FD_SET(balcao_fifo_fd, &read_fds);//para Pipes

    nfd = select(balcao_fifo_fd + 1, &read_fds, NULL, NULL, NULL);

    ...
```

Threads e Sinal de Vida

Foram usadas 3 threads para o funcionamento de certas funcionalidades pedidas no enunciado. A thread **isAlive** usada no programa Médico, é responsável por enviar um sinal de vida para o Balcão, fazendo a escrita de uma mensagem a cada 20 segundos (com recurso à funcão **sleep()**) para o FIFO do Balcão. Essa mensagem está a ser recebida dentro do select do programa principal, que está a fazer a leitura tanto do teclado, como do pipe do Balcão. Quando essa mensagem é recebida, o médico guarda na variável isAlive, o tempo em milissegundos em que recebeu essa mensagem.

```c
void *isAlive(){

    while(1){
        Mensagem_utilizador_para_Balcao mens_med_balc;
        mens_med_balc.pid = getpid();
        mens_med_balc.medico_cliente = 0; // 0 = medico
        mens_med_balc.atendido = -1;  // -1 = sinal de vida

        //Enviar sinal de vida para o balcão
        write(balcao_fifo_fd, &mens_med_balc, sizeof(mens_med_balc));
        sleep(20);
    }
}
```

```c
//SINAL DE VIDA DO MEDICO
if(mens_para_balc.medico_cliente == 0 && mens_para_balc.atendido == -1){
        fprintf(stderr, "[INFO] Sinal de vida recebido do Médico [%d]\n", mens_para_balc.pid);
        for (int i = 0; i < balcao.maxMedicos; i++)  {
            if(listaMedicos[i].id == mens_para_balc.pid){
                    listaMedicos[i].isAlive = tempoAtual();
                    break;
            }
        }
}
```

Para além disso, no Balcão, existe um thread **verificaSinaisVida** sempre a correr que está constantemente a atualizar os valores relativos aos tempos em que as mensagens foram enviadas pelos médicos e a verificar se estes deixaram de dar sinal de vida. Caso isso aconteça, os mesmos são removidos da lista de Médicos. Notar, que estes dados são recebidos como argumentos, numa estrutura que aloca tanto o balcão, como a lista de médicos e o mutex. Tendo em conta que os dados que estão a ser usados e alterados nas threads, são partilhados por outras funções, há necessidade de proteger essa parte crítica. O **lock** do mutex, é usado para garantir a exclusividade da thread a esses dados (Início da parte crítica), e o **unlock** do mutex é usado para libertar essa exclusividade (Fim da parte crítica).

```c
void *verificaSinaisVida(void *dadoss){

    Dados *dados = (Dados *) dadoss;

    while(1){
        //Inicio de uma parte critica
        pthread_mutex_lock(&dados->mutex);

            for (int i = 0; i < dados->balcao->maxMedicos; i++)  {
                if(dados->listaMedicos[i].id!=0){
                    if(dados->listaMedicos[i].isAlive==0){
                        dados->listaMedicos[i].isAlive=tempoAtual();
                    }else if(tempoAtual() - dados->listaMedicos[i].isAlive > 21000){
                        fprintf(stderr, "[INFO] Médico [%d] deixou de enviar sinais de vida\n",  dados->listaMedicos[i].id);
                        removerMedico(dados->balcao, dados->listaMedicos, dados->listaMedicos[i].id);
                    }
                }
            }

        pthread_mutex_unlock(&dados->mutex);
        //Fim de uma parte critica
    }
}
```

Para além destas 2 threads, existe a thread **mostraListaXSec** que vai receber como argumentos, a estrutura com os dados do balcão. Esta thread assim como a anterior, também faz o lock e unlock do mutex, pois os dados que estão na parte crítica, estão a ser utilizados em funções. Para além disso, ela tem um sleep de x segundos (30 por Default), conforme o que o utilizador referir no freq x. A cada x segundos mostra a lista de espera de cada especialidade ao administrador.

```c
void *mostraListaXsec(void *balcaoo){
    Balcao *balcao = (Balcao *) balcaoo;

    while(1){
        //Inicio de uma parte critica
        pthread_mutex_lock(&balcao->mutex);
        printf("\n[INFO] Tempo de frequência: %d\n", balcao->tempoFila);
        mostraListasEspera(balcao);
        pthread_mutex_unlock(&balcao->mutex);
        sleep(balcao->tempoFila);
        //Fim de uma parte critica
    }
}
```

COMANDOS

Neste trabalho foram implementados alguns comandos, sendo a maior parte destinada ao uso único da parte do médio, com a exceção do comando “sair” que tem como utilizadores alvo o médico e o cliente.

Lista de Comandos:

- utentes -> Lista todos os clientes ligados ao balcão;
- especialistas -> Lista todos os médicos ligados ao balcão;
- delut X -> Elimina um cliente, que se encontre em fila de espera, identificado através do pid;
- delesp X-> Elimina um médico que se encontre na fase de aguardar cliente, através da identificação do seu pid;
- freq N -> Altera a frequência com que a lista de espera é demostrada no balcão, atribuindo o número de segundos de intervalo entre demostrações;
- encerra -> Termina o processo do balcão, desligando todos os médicos e clientes em simultâneo;
- help -> Mostra todos os comandos possíveis por parte do balcão;
- sair -> Termina o processo do medico/cliente.

CONCLUSÃO

Com a realização deste trabalho, ganhamos experiência na utilização de mecanismos que permitem a comunicação entre diferentes processos, na utilização de threads que são bastante úteis e práticas num futuro dia-a-dia de um programador. Para além disso, é um bom método de estudo para o exame teórico/prático.