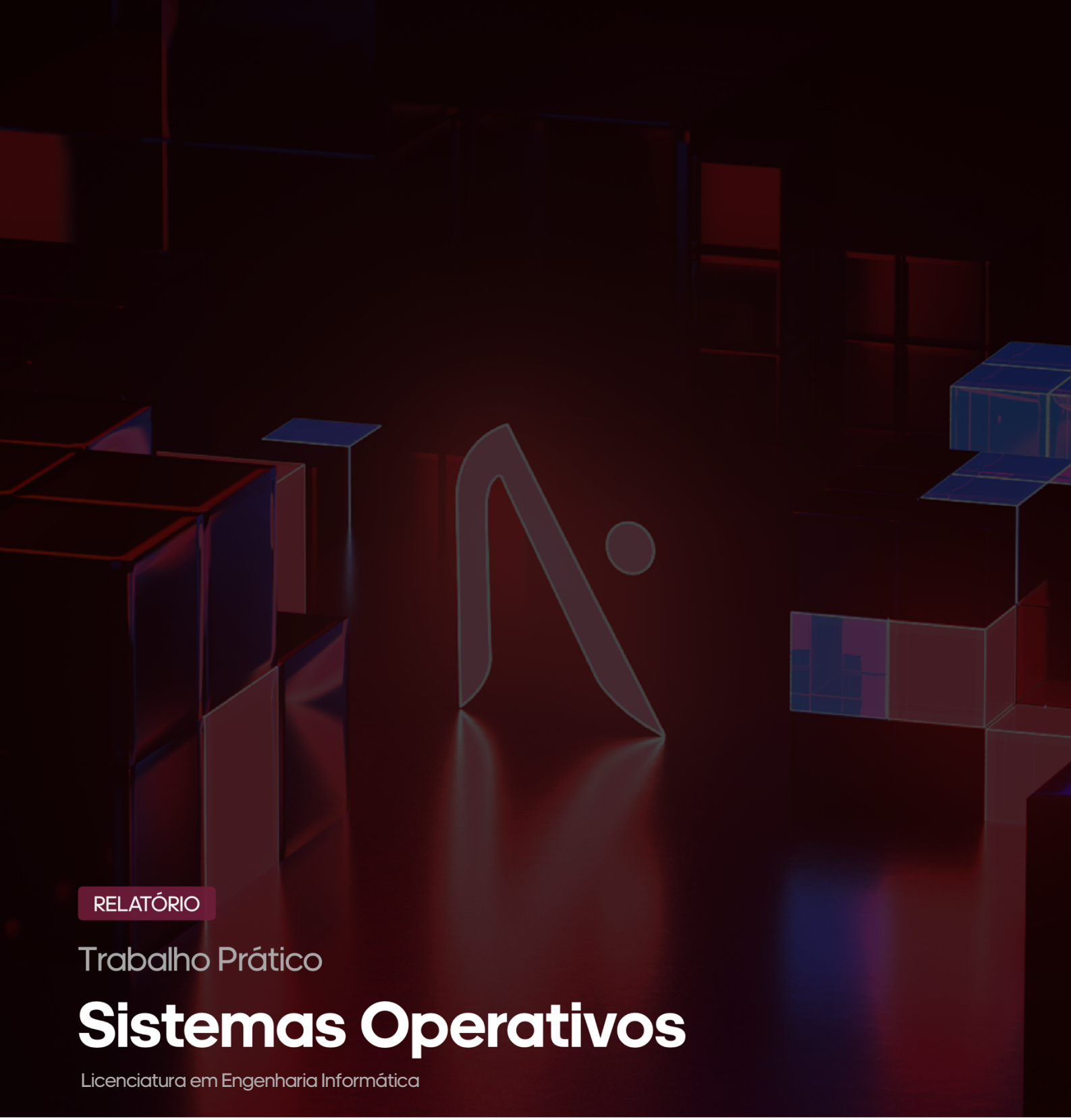

RELATÓRIO

Trabalho Prático

Sistemas Operativos

Licenciatura em Engenharia Informática

O trabalho prático de Sistemas Operativos consiste na implementação de um sistema para gerir atendimento de clientes em estabelecimentos médicos.

Trabalho realizado por:

Tomás Gomes Silva - 2020143845

Tomás da Cunha Pinto - 2020144067

Índice

Introdução

O trabalho prático de Sistemas Operativos consiste na criação de um sistema para gerir o atendimento de clientes em estabelecimentos médicos na linguagem de programação C e ambiente Unix.
Existem várias entidades, como por exemplo: o balcão, que gere a comunicação entre os utentes e os médicos, o utente, que dá entrada no sistema com um sintoma e espera que lhe seja atribuído um médico, e o médico, que vai atender um utente e irá ter uma conversa com este.

Implementação

Variáveis de Ambiente

No ficheiro *envar.sh* temos definidas 3 variáveis de ambiente:

- **MAXCLIENTES:** indica o número máximo de clientes conectados
- **MAXMEDICOS:** indica o número máximo de médicos conectados
- **MEDICALSO_RUN:** indica se o balcão está online

Estas variáveis de ambiente são utilizadas pelos vários programas para controlar certos aspetos dos mesmos. No caso de não estarem definidas ou estarem mal definidas, os programas apresentam uma mensagem de erro e fecham, impedindo assim o utilizador de efetuar qualquer outra ação.

```c
char *mso_state = getenv("MEDICALSO_RUN");
if(mso_state == NULL){
    printf("[MÉDICO]\nO balcão está fora de serviço\n");
    return 0;
}
```

```
export MAXCLIENTES=5
export MAXMEDICOS=5
export MEDICALSO_RUN=1
```

Estruturas de Dados

No que toca às estruturas de dados, foi necessário criar 3 estruturas diferentes para armazenar informações relativas ao balcão, médicos e clientes.

A estrutura do balcão armazena os seguintes dados:

- **unpipeBC, unpipeCB:** arrays de inteiros que vão guardar os pipes para comunicação entre o balcão e o classificador e vice-versa
- **nClientesMax, nMedicosMax:** número máximo de clientes e médicos
- **nClientesAtivos, nMedicosAtivos:** número de médicos e clientes ativos
- **medicos, clientes:** array de estruturas do tipo Medico e Cliente, respetivamente, que são utilizadas para armazenar os médicos e os clientes conectados

Já as estruturas dos médicos e dos clientes são bastante semelhantes. Ambas guardam o ID do processo (PID), bem como o nome do cliente ou médico.
O que distingue as duas é que a estrutura dos clientes guarda os sintomas descritos e a análise fornecida pelo médico, enquanto que a estrutura dos médicos apenas guarda a sua especialidade.

```c
typedef struct Balcao {
    int unpipeBC[2], unpipeCB[2];
    int nClientesMax, nMedicosMax;
    int nClientesAtivos, nMedicosAtivos;
    struct Medico medicos[MAX];
    struct Cliente clientes[MAX];
} balcao, *balcao_ptr;
```

```c
typedef struct Medico {
    int pid;
    char nome[MAX];
    char especialidade[MAX];
} medico, *medico_ptr;
```

```c
typedef struct Cliente {
    int pid;
    char nome[MAX];
    char sintomas[MAX];
    char analise[MAX];
} cliente, *cliente_ptr;
```

Comunicação balcão-classificador

Para poder haver comunicação entre o balcão e o classificador foi necessário criar 2 pipes: um para comunicar do balcão para o classificador e outro para comunicar no sentido contrário.

Houve necessidade de criar um processo filho de modo a que o classificador pudesse correr ao mesmo tempo que o balcão. O *if(pid == 0) ... else ...* serve para controlar que parte do código é corrido e por quem.

Se o pid for igual a zero é o processo filho a correr, por outro lado, se o pid não for igual a zero, então é o pai a correr. Há duas possibilidades:

- Se for o filho a correr, o STDIN e o STDOUT do processo filho são fechados para usarmos os pipes como meio de comunicação em vez de usarmos o I/O stream, duplicamos os pipes no processo filho e corremos o programa classificador sem passar nenhum argumento adicional.
- Se for o pai a correr, fechamos os pipes inúteis no lado do pai, nomeadamente o read do Balcão->Classificador e o write do Classificador->Balcão.

```
balcao b;
pipe(b.unpipeBC);
pipe(b.unpipeCB);
int pid = fork();
```

```
if(pid == 0){
    // Child
    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    dup(b.unpipeBC[0]);
    dup(b.unpipeCB[1]);
    execvp("../classificador", NULL);
```

```
} else {
    // Parent
    close(b.unpipeBC[0]);
    close(b.unpipeCB[1]);
}
```

Após a criação dos pipes já podemos então criar um ciclo “infinito” para perguntar ao utilizador quais os seus sintomas. Escrevemos para o pipe Balcão->Classificador utilizando a função write(), o conteúdo existente na variável sintomas onde foi armazenado o input do utilizador e lemos de seguida a resposta do classificador utilizando a função read() onde escrevemos para a variável análise a especialidade para o qual o utente irá ter de ser encaminhado.

```
while (1){
    strcpy(analise,"");
    printf("\nIndique os seus sintomas (debug): ");
    fgets(sintomas, sizeof(sintomas), stdin);
    sintomas[strlen(sintomas) - 1] = '\0';
    strcat(sintomas, "\n");

    write(b.unpipeBC[1], sintomas, strlen(sintomas));
    int tmp = read(b.unpipeCB[0], analise, MAX);
    analise[tmp-1]= '\0';
    printf("O classificador retornou: %s", analise);
    fflush(stdout);
    fflush(stdin);
}
wait(NULL);
return 0;
```

Conclusão

O tópico Conclusão encontra-se em desenvolvimento. Aqui serão descritas as conclusões que tirámos ao longo do trabalho.