 t

Licenciatura em Engenharia Informática

# Índice

[Índice 3](#_Toc92742925)

[Introdução 4](#_Toc92742926)

[Implementação 5](#_Toc92742927)

[Variáveis de Ambiente 5](#_Toc92742928)

[Ligação ao Balcão 5](#_Toc92742929)

[Estruturas de Dados 6](#_Toc92742930)

[Threads 7](#_Toc92742931)

[Sinais de Vida 8](#_Toc92742932)

[Aceitar médicos/clientes 9](#_Toc92742933)

[Consola do Administrador 11](#_Toc92742934)

[Utentes 11](#_Toc92742935)

[Especialistas 11](#_Toc92742936)

[Delut 12](#_Toc92742937)

[Delesp 12](#_Toc92742938)

[Freq 13](#_Toc92742939)

[Encerra 13](#_Toc92742940)

[Comunicação balcão-classificador 14](#_Toc92742941)

[Comunicação médico-cliente 15](#_Toc92742942)

[Conclusão 17](#_Toc92742943)

# 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.

Uma imagem com texto

Descrição gerada automaticamente

# Implementação

## Variáveis de Ambiente

No ficheiro *envar.sh* temos definidas 2 variáveis de ambiente:

* **MAXCLIENTES:** indica o número máximo de clientes conectados
* **MAXMEDICOS:** indica o número máximo de médicos conectados

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.

Uma imagem com texto

Descrição gerada automaticamente

## Ligação ao Balcão

Quando o balcão é iniciado são criados 3 pipes que servem para receber informação proveniente dos médicos e utentes. Para além disso servem o propósito de indicar se o balcão está a funcionar ou não.

Uma imagem com texto, captura de ecrã, ecrã

Descrição gerada automaticamente

Os médicos e os utentes ao abrirem tentam abrir o pipe **MEDICALso** e se conseguirem isso indica que o balcão está a funcionar e podem continuar o processo de ligação, caso contrário o balcão encontra-se fechado e o programa é interrompido.

Uma imagem com texto

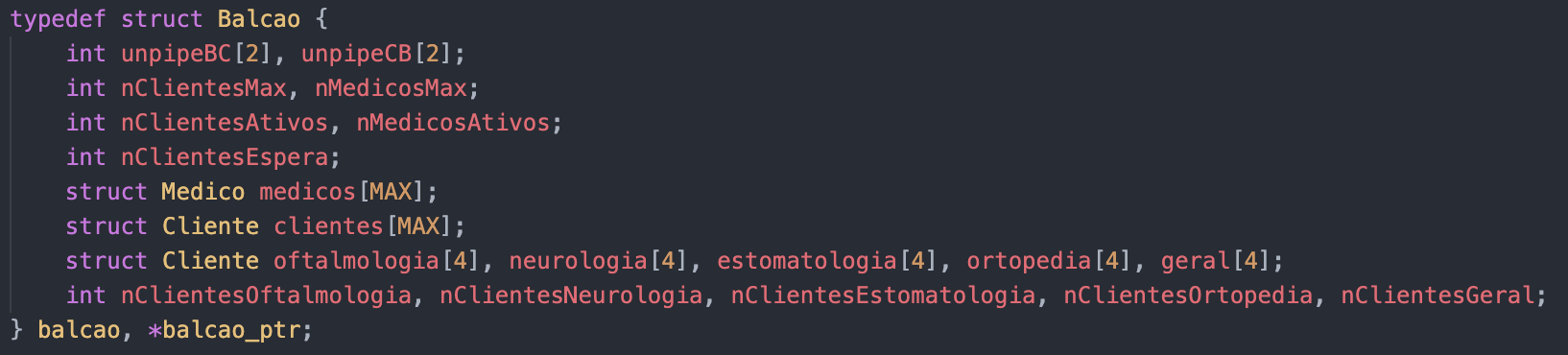
Descrição gerada automaticamente

## 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
* **oftalmologia, neurologia, estomatologia, ortopedia, geral:** arrays de estruturas do tipo Cliente que são utilizadas para guardar os clientes atualmente em espera para cada especialidade
* **nClientesOftalmologia, nClientesNeurologia, nClientesEstomatologia, nClientesOrtopedia, nClientesGeral:** número de clientes em espera para cada especialidade



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.

A estrutura do Médico tem um campo que guarda a sua especialidade para que o balcão saiba que clientes há de atribuir ao médico. Para além disso tem duas variáveis que servem como flags, o **alive** (para os sinais de vida) e o **ocupado** (para o balcão saber se o médico está em consulta ou não).

A estrutura do Cliente guarda os sintomas descritos pelo utente, a análise devolvida pelo classificador e a posição do utente na lista de espera para a especialidade.

Ambas as estruturas contam com um campo que guarda o pipe do cliente (no caso do médico) e o pipe do médico (no caso do cliente) para que estes possam comunicar durante a consulta.

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente

## Threads

Para que os programas possam fazer várias coisas ao mesmo tempo é necessário criar threads. Neste trabalho foram necessárias várias threads nos vários programas:

* **Cliente**
  + Ler de um pipe uma mensagem enviada pelo médico
  + Escrever para um pipe uma mensagem para o médico
* **Médico**
  + Enviar sinais de vida ao balcão
  + Ler de um pipe uma mensagem enviada pelo utente
  + Escrever para um pipe uma mensagem para o utente
* **Balcão**
  + Aceitar os médicos
  + Aceitar os clientes
  + Atualizar o estado dos médicos para os sinais de vida
  + Remover médicos que não deram sinal de vida
  + Obter comandos do administrador
  + Mostrar as filas de espera de X em X segundos

Para criar uma thread utilizamos a função pthread\_create em que os últimos dois parâmetros correspondem à função que queremos executar numa thread separada do programa principal e os argumentos que queremos passar a essa própria função.

Uma imagem com texto

Descrição gerada automaticamente

Para esperar que uma thread termine utilizamos o pthread\_join e desta forma o programa não termina sem que todas as threads tenham sido terminadas.



No caso particular do comando “freq” que é executado pelo administrador para mostrar as filas de espera, é necessário cancelar a thread anterior para que não hajam várias threads para o mesmo propósito a correr de modo a impedir que fossem mostradas as filas de espera várias vezes constantemente. A função pthread\_cancel cancela uma thread existente.



Nos próximos tópicos serão explicadas as várias threads existentes no programa detalhadamente.

## Sinais de Vida

Os sinais de vida são uma componente importante em sistemas de estrutura cliente-servidor. O servidor necessita de saber se um cliente ainda se encontra ativo ou se este foi terminado abruptamente sem ter tido oportunidade de informar o servidor. Neste trabalho foram implementados sinais de vida apenas no médico utilizando threads.

As funções threadVida e enviaSinalVida consistem num ciclo que é executado a cada X segundos (definido na constante SINAL\_VIDA) que envia uma mensagem por um pipe para o balcão a indicar que esse médico ainda se encontra ativo.

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto, ecrã, captura de ecrã, prateado

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamenteO balcão ao receber esse sinal (na thread **updateVivos**) vai procurar pelo PID do médico na lista de médicos ativos e vai atualizar a sua variável alive para 1. Desta forma, quando o ciclo da thread **removeMortos** executar, o médico não vai ser desconectado nesse ciclo pois o alive encontra-se a 1.

No caso do médico ter sido desconectado sem avisar, o seu alive passará a 0 e desta forma, no próximo ciclo da thread **removeMortos**, o médico irá ser removido do sistema.

Todos os médicos ainda conectados têm a sua variável alive colocada a 0 para que sejam obrigados a enviar um sinal de vida outra vez.

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente

## Aceitar médicos/clientes

Para fazer a distinção entre os dois tipos de utilizadores que se podem conectar ao balcão, criámos 2 pipes extra: o pipe **MEDICALsoMED** (para os médicos) e o pipe **MEDICALsoCLI** (para os utentes).

Cada médico tem o seu pipe que está na forma MEDICO[%d] em que o %d indica qual é o PID do médico. Guardamos isso no **MEDICO\_FIFO\_FINAL**.

Uma imagem com texto, ecrã

Descrição gerada automaticamente

Antes de conectarmos o médico ao balcão definimos todos os campos da sua estrutura.

Uma imagem com texto

Descrição gerada automaticamente

Finalmente abrimos o pipe do balcão e escrevemos a estrutura que contém as informações desse médico.

Uma imagem com texto

Descrição gerada automaticamente

Do lado do balcão temos uma thread a correr (**aceitaMedicos**) que trata de ler do pipe **MEDICALsoMED** e aceita os médicos se o limite não tiver sido atingido anteriormente, adicionando-os ao array de médicos.

Uma imagem com texto

Descrição gerada automaticamente

No caso do cliente o processo é o mesmo exceto a parte depois de ser aceite pelo balcão.

Existe um passo adicional que consiste em classificar os sintomas descritos pelo utente, devolvê-los ao utente e adicionar esse mesmo utente ao array que representa a fila da especialidade para a qual ele necessita de ir.

São usados unnamed pipes para comunicar com o balcão como descrito no tópico [**Comunicação balcão-classificador**](#_Comunicação_balcão-classificador). Escrevemos no write end do pipe do classificador os sintomas e lemos do read end do pipe do balcão a análise dos sintomas descritos que corresponde a uma especialidade e uma prioridade.

Uma imagem com texto

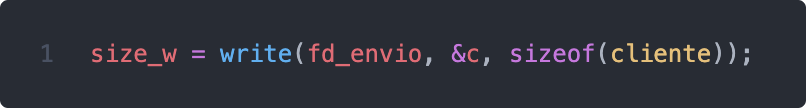
Descrição gerada automaticamente

Posteriormente, adicionamos esse utente à lista de espera para a sua especialidade e ordenamos conforme as prioridades. No final devolvemos uma estrutura semelhante à que o cliente nos enviou só que os campos especialidade, prioridade e análise vêm preenchidos com informação pertinente.

Uma imagem com texto

Descrição gerada automaticamente

Os ciclos for ordenam os utentes por ordem de prioridade mas um dos problemas neste algoritmo é que não tem em consideração a ordem de chegada depois de reordenar o array.



## Consola do Administrador

A consola do administrador é utilizada pelo utilizador do programa balcão e serve para efetuar comandos para obter informação sobre médicos e clientes, filas de espera e até desconectar médicos e utentes.

Para interpretarmos os comandos usámos a função strtok que divide uma string utilizando um delimitador (token) que neste caso é um espaço. O strtok aponta para a localização onde se encontra esse delimitador; com isto podemos adicionar a palavra até essa localização ao array args que vai ser utilizado mais à frente para saber que comando é que o administrador quer executar.

Em args[0] encontra-se o comando que o administrador quer executar e nos índices seguintes são os restantes argumentos necessários para os vários comandos existentes.

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente

### Utentes

O comando utentes mostra todos os utentes conectados ao balcão que estão em consulta ou em lista de espera.

Uma imagem com texto

Descrição gerada automaticamente

### Especialistas

O comando especialistas mostra todos os especialistas conectados ao balcão.

Uma imagem com texto

Descrição gerada automaticamente

### Delut

O comando delut permite ao administrador desconectar um utente que esteja em lista de espera. Para executar este comando o administrador precisa de saber o PID do utente que quer desconectar. Enviamos um **SIGINT** ao processo com o PID introduzido pelo administrador e o programa cliente trata de fechar de forma ordeira.

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

### Delesp

O comando delesp permite ao administrador desconectar um especialista que não esteja atualmente em consulta. É necessário indicar o PID do especialista que pretendemos desconectar e caso este exista no sistema é removido do array de médicos e é enviado um **SIGINT** ao programa do médico que faz com que este termine.

Uma imagem com texto

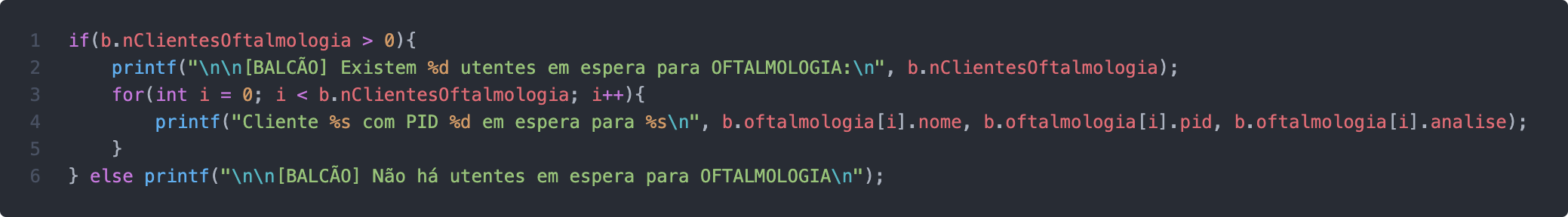
Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

### Freq

O comando freq permite definir um intervalo de tempo para mostrar as filas de espera de todas as especialidades consecutivamente. O administrador define esse intervalo de tempo e é criada uma thread que vai executar a função mostraFilas a cada X segundos



É necessário cancelar a thread anterior para que não hajam várias threads sempre que o comando freq é executado. É suposto haver apenas uma thread a mostrar as filas de espera.

Uma imagem com texto

Descrição gerada automaticamente

### Encerra

O comando encerra serve para fechar o balcão e consequentemente todos os pipes e conexões a este associados e também o classificador.

A sequência de encerramento começa por enviar um “#fim” ao classificador para que este feche. Seguidamente são executados 6 for loops que tratam de desconectar todos os médicos e utentes independentemente de estarem ou não em consulta.

Uma imagem com texto

Descrição gerada automaticamente

Finalmente, são fechados os unnamed pipes com o classificador, é libertada a memória guardada para os comandos e são eliminados todos os pipes relativos ao balcão.

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente

## 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.

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente

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.

Uma imagem com texto

Descrição gerada automaticamente

## Comunicação médico-cliente

A comunicação entre o médico e o cliente, assim como maior parte da comunicação inter-processos, é efetuada com recurso a named pipes. Quando um programa cliente ou médico são abertos são criados pipes no formato CLIENTE[%d] e MEDICO[%d] respetivamente em que o %d é substituído pelo PID desse processo.

Quando um utente é atendido pela primeira vez por um médico são trocadas informações relativas um ao outro e uma dessas informações é precisamente o nome do pipe ao qual se devem conectar para falar.

Tanto no programa médico como no programa cliente temos duas threads que servem para ler mensagens de um pipe e escrever mensagens para um pipe.

Uma imagem com texto

Descrição gerada automaticamente

Como podemos ver na imagem de cima, a função readMensagem recebe como parâmetro a estrutura do médico recebida por pipes. Com essa estrutura podemos abrir o pipe do médico em modo de leitura apenas e em modo não bloqueante para que possamos estar constantemente à espera de uma mensagem do médico.

O ciclo while lê do pipe do cliente se houver alguma coisa para ler e dá print na mensagem. Se a mensagem for um “adeus” o cliente fecha.

Para enviar uma mensagem temos a função writeMensagem que recebe também a estrutura do médico para que o cliente saiba para onde é que tem de enviar a sua mensagem. Sendo assim, abrimos o pipe do médico em modo de escrita apenas e caso essa operação tiver corrido com sucesso entramos num ciclo que nos permite ler do teclado uma mensagem e enviar para o pipe do médico a mensagem.

Uma imagem com texto

Descrição gerada automaticamente

O procedimento do lado do médico é exatamente o mesmo que foi descrito aqui para o cliente.

As funções fecharCliente/fecharMedico servem para fechar todos os pipes abertos anteriormente e eliminar o pipe criado no início do programa para o cliente/médico.

Uma imagem com texto

Descrição gerada automaticamente

# Conclusão

Com a realização deste trabalho apercebemo-nos dos sistemas incríveis que podemos criar para solucionar problemas da vida real, neste caso em particular, a gestão de um hospital. Este trabalho foi uma excelente oportunidade para melhorarmos as nossas competências na linguagem de programação C e conseguimos aprender imenso sobre o modo de funcionamento dos sistemas baseados em Unix.

