Sistemas Operativos

MEDICALso

Docente:

João Durães

Alunos:

Paulo Henrique Figueira Pestana de Gouveia nº 2020121705 – LEI

João Filipe Silva de Almeida nº 2020144466 – LEI

Coimbra, 16 de Janeiro de 2022

Índice

1.Introdução

Neste relatório apresentamos a arquitetura assim como o funcionamento do nosso trabalho prático.

Este trabalho prático de sistemas operativos consiste na implementação de um sistema para gerir o atendimento de clientes em estabelecimentos médicos. O sistema, denominado MEDICALso, destina-se a correr em ambiente Unix linha de comandos e irá mediar a interação entre doente, médico e balcão de atendimento.

2.Arquitetura

Para este trabalho usamos uma arquitetura servidor, cliente e médico, em que o servidor é designado por balcão, o cliente e o médico serão os utilizadores para o simular o funcionamento do programa.

Os clientes usam um cliente(shell) e médicos usam um médico(shell) , ambos interagem com o mesmo balcão, pois este é único no programa inteiro, podem no entanto haver vários médicos e clientes.

O balcão é o centro do nosso programa, é responsável por todas as manutenções de dados, recebe informação dos clientes e médicos que serve para estabelecer a conexão correta entre o programa cliente e médico.

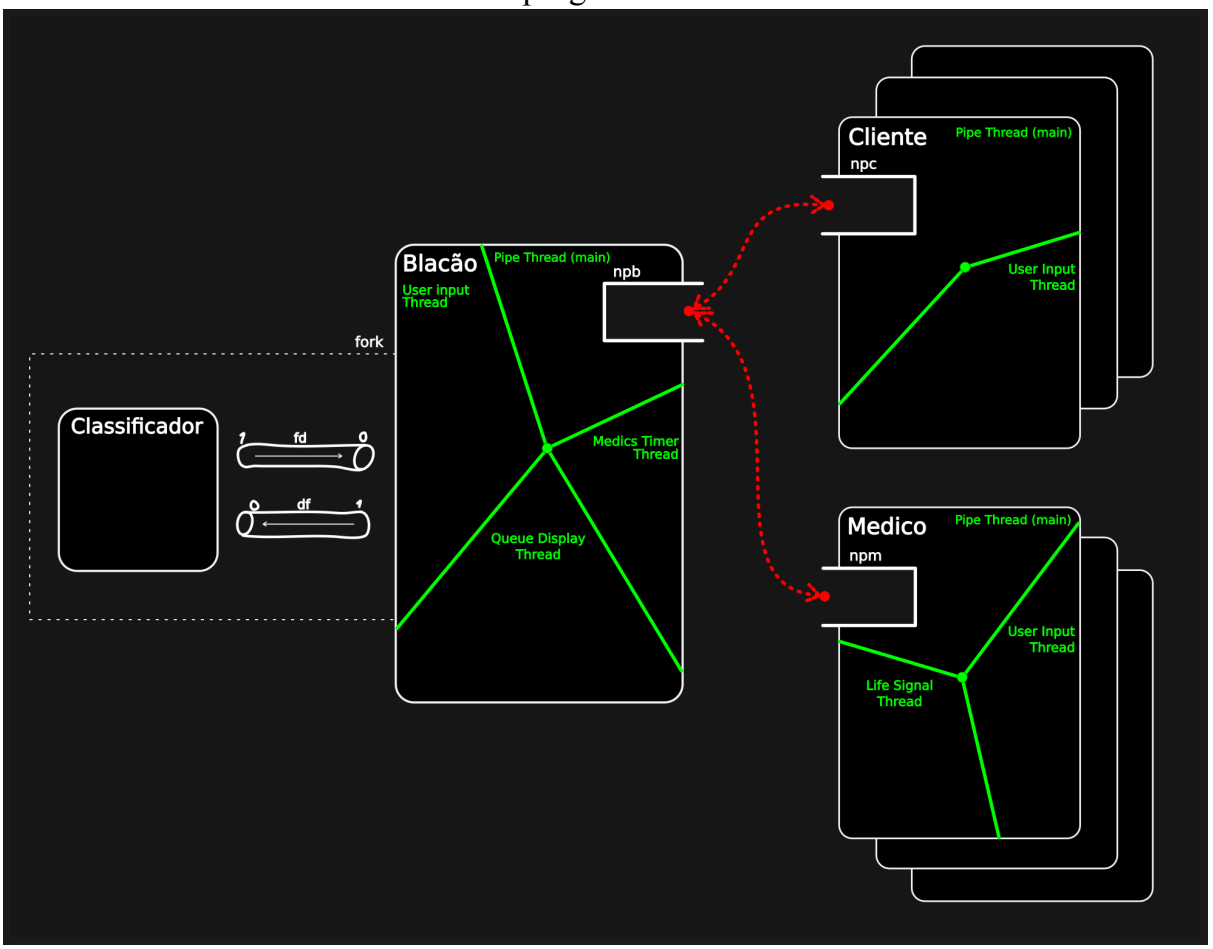

Imagem 1: Arquitetura

2.1 Estruturas

Existem dois tipos de estruturas predominantes no trabalho: as estruturas para envio de mensagens e as estruturas para partilha de dados entre threads.

Todas as estruturas com a finalidade de mandar mensagens entre processos estão no ficheiro structs.h e são partilhadas entre todos os programas.

As restantes estruturas são exclusivas aos ficheiros onde são usadas e são portanto declaradas adequadamente.

A estrutura global_info existe para aglomerar os dados que foram feitos globais de modo a poderem ser terminados de forma ordeira quando é chamado o término do programa através de sinais como o CTRL+C

2.2 Threads

Os threads, como mostrado na imagem 1, têm variadas finalidades. As principais funcionalidades dos threads ao longo de todo o projeto foram:

- Leitura de dados do utilizador;
- Leitura e tratamento de dados dos named pipes

Foram também usados threads também em outras situações que mostraram precisar da sua implementação, nomeadamente:

- O envio e recebimento do sinal de vida do médico;
- Display das filas de espera

Para bloqueio e proteção dos dados, trancamo-los através de um mutex. Utilizamos apenas um mutex que irá destrancar e trancar todos os dados das listas ligadas do balcão. Mais adequadamente teríamos no mínimo dois mutexes, uma para a lista ligada dos médicos e um para a lista ligada dos clientes, idealmente poderíamos até ter um mutex para cada cliente e médico na estrutura, a nossa implementação ficou assim limitada devido à falta de tempo para a implementar.

3.Funcionamento

O balcão é iniciado primeiramente, pois sem ele, os clientes e os médicos não conseguem ser iniciados devido a não ser possível serem guardados no sistema. Ao iniciar o balcão é verificado se um FIFO designado por “balcao_fifo” está criado na pasta tmp, pois só uma instância do balcão deverá estar iniciada, caso não tenha encontrado, é então criado um. Este FIFO é criado de forma READ/WRITE para prevenir o bloqueio no open, é usado para receber informação do cliente e médico necessária para iniciar o nosso programa e futuramente mandar informação de volta

para os mesmos. São atribuídos máximos de clientes e médicos que o balcão tolera através de variáveis de ambiente e limite de 5 na fila de espera.

Após o seu início, entramos na etapa de colocar clientes e médicos no nosso sistema em que a ordem não afeta o decorrer do programa.

Explicando primeiramente o nosso programa cliente, ao ser iniciado, são passados como argumentos, o nome e os sintomas desse cliente. É criado também FIFO designado "cliente _%d_fifo", em que o inteiro será o PID único desse mesmo cliente de forma a distinguir de outro cliente, é usada sempre que seja preciso enviar informação para o cliente. Esses dados são então guardados numa struct que servirá para enviar os dados para o balcão através de um Após o envio dessa estrutura pelo FIFO do balcão, criámos então uma maneira de ele ao saber o tamanho da estrutura que está a lidar, vai prosseguir de forma correta, umas das maneiras que foi discutida em aula teórica.

Recebida a informação do cliente, o balcão usa um fork para a comunicação com o classificador (fornecido pelos professores) através de dois unnamed pipes, um para mandar a informação (“fd”) pelo stdin do balcão para o classificador, e outro para mandar a informação do classificador para o balcão pelo stdout (“df”). O classificador é quem dá a informação ao balcão de que prioridade esse cliente tem e que especialidade se deve dirigir. Dados que são guardados numa estrutura em conjunto com a informação sobre a lista de espera de clientes presentes nessa especialidade, e de médicos no sistema. Foram usadas listas ligadas para a organização dos clientes do e dos médicos no sistema, para a facilitar entrada e saída deles no sistema, e display de dados. Caso não exista nenhum médico, o cliente ficará à espera que um deles dessa especialização entre no sistema.

Passando então para o funcionamento dos médicos, funciona de forma semelhante aos clientes, com um argumento extra da sua especialização, é também criado um FIFO designado "med _%d_fifo" com um PID único associado ao médico.
Envia essa informação para o balcão através do FIFO (do balcão) para o meter na lista ligada de médicos, e fica à espera então que lhe seja atribuído um cliente da mesma especialização.

Havendo então um cliente e um médico da mesma especialidade, o balcão percorre a lista para ver a prioridade de cada cliente e quem está primeiro, após feita essa seleção, o balcão conecta esse cliente e o médico através dos FIFOS específicos desses mesmos.

A forma de como é feita a comunicação é através de threads que estão a ler inputs feitos tanto pelo cliente como pelo médico, que serão mandados através dos FIFOS. Essa thread é chamada antes da conexão, caso tanto o cliente como médico decidam sair do sistema antes da consulta. Dentro da consulta tanto o médico, tanto o cliente podem dar a consulta como terminada ao dar input de “adeus”, de seguida o balcão encarrega de retirar o cliente do sistema, e meter o médico como disponível.

Após a saída de um cliente ou médico é feita uma reconstrução da lista ligada, e retirados os FIFOS criados para os mesmos.

O funcionamento do resto das threads, que servem mais para display de dados, e simulação de um admin no balcão.

No balcão temos 3 threads para além do main, um deles serve para user inputs, que é a execução de comandos indicados no enunciado do trabalho e encerramento do mesmo, de um cliente ou médico. Outro será para uma frequência pré definida ou alterado pelo admin, dar display da lista de espera dos cliente de cada especialização, e por último receber um sinal de vida de 20 em 20 segundos dos médicos do sistema, frequência que pode ser alterada, algo não aperfeiçoado no trabalho, sendo algo extra que tentamos implementar.

Esse sinal de vida é um thread criado nos médicos, que serve para indicar ao balcão que este médico ainda está operacional.

Por último temos os sinais, esses foram usados para fechar e libertar tudo caso feche do terminal os programas através do CTRL + C.

4.Conclusão

Planeamos bastante cedo a forma de organização do projeto para evitar reestruturações futuras.

Criámos uma pasta src para os ficheiros de código e header files e uma pasta dist para os executáveis, excepto o classificador.

Compilamos e executamos o programa através de um Makefile.

Na implementação final do programa não foram encontrados nenhum bugs que pusessem em causa o funcionamento do programa.

Foi dada especial atenção para que o programa, quando terminado, termina todas as threads em execução e liberta toda a memória usada para que não haja nenhuma *memory leak.*

Usámos plataformas como o Discord, o Github e o GoogleDocs para efeitos de colaboração, e para produtividade ambos instalamos e realizamos o trabalho na Linux distro Manjaro com o Visual Studio Code.