Relatório Trabalho Prático 2017/2018

Jorge Martins- 2015199193
José Basto- 2015229241

Introdução:

O trabalho prático realizado para a cadeira de Sistemas Operativos consiste na implementação um simulador das urgências de um hospital na linguagem de programação C. Este simulador será capaz de representar os processos de triagem e de atendimento que decorrem no hospital, utlizando mecanismos de gestão de processos, threads, comunicação e sincronização em Linux.

Funcionamento:

No inicio do trabalho prático é ligo o ficheiro config.txt e carregado para o sistema a informação necessaria para arrancar o programa, ou seja o numero de processos doutor, o numero de threads de triagem, a duração dos turnos dos processos doutores e o numero maximo de pessoas na fila de mensagens, utilizando a função le_config() colocando toda a informação numa estrutura Config.

Em seguida é criado o named pipe, pela função cria_pipe(), é criado a Message queue, pela função cria_mq(), é criado a fila de espera usando uma lista ligado, cria_fila(), é criado a memoría partilhada, criar_memoria_partilhada(), são criados todos os mecanismismos de sincronização, ou seja, os semaforos doutorFim, Triagem, Atendimento, o phtread_mutex_t mutexListaLigada, e o phtread_mutex_t mutex Depois são criados os processos doutor e as threads necessarias ao funcionamento.

Os processos filhos doutores receberão uma função para fazer o respetivo atendimento.

Para alem das threads de triagem, serão criadas duas threads adicionais nas duas primeiras posições do array de threads, na posição 0, está a thread para ler a informação enviada para o named_pipe. Na posição 1 está uma thread responsável pela criação de novos processos doutores no final do seu respetivo turno. As restantes posições vão conter threads de triagem.

Ao criar as threads, elas recebem uma função worker para executarem. A thread na posição 0, está responsável por estar à escuta de informação vinda pelo named_pipe. Afinção worker chama-se

le_pipe(). Esta função está responsável por por abrir o named_pipe, ler a informação e coloca num buffer, que é um array de chars. Esta informação pode ser de 3 tipos, “nome tempo_triagem tempo_atendimento prioridade” se estiver a ser tratado um uníco paciente, pode receber um grupo de pessoas, “numero_pessoas empo_triagem tempo_atendimento prioridade”, ou pode ser um comando especial para alterar o numero de threads, “TRIAGE=numero”. Para tratar esta informação é usada a função strtok() para separar os vários campos e colocados numa estrutura paciente caso não seja o comando para alterar as threads de triagem. Para distingir os grupos das pessoas sozinhas, utlizando a função isalpha() numa condição if, vemos que ou são pacientes sozinhos ou o comando “TRIAGE”.
Em seguida se o token for igual à palavra “TRIAGE”, é guardado na variável nova_threads o valor e depois se o numero for maior do que o numero de theads que está na config, é feito um realoc a ao array de threads para aumentar o tamanho e criado mais threads nessas posiçoes. Se não é feito um realloc para diminuir o tamanho e é cancelado as threads necessárias.
Se o token não for igual à palavra “TRIAGE” serão colocados numa estrututa paciente os dados do paciente e inseridos na lista ligada. A diferença dos grupos é que o nome das pessoas será “grupo” mais o numero de chegada, e é utilizado um ciclo for para inserir o numero de pessoa desse grupo. Em seguida será feito um sem_post para o semaforo Triagem que se emcontra a 0, e iniciará a função worker das threads de triagem, chamada triagem().

Esta função está a espera parada com um semaforo e só começa a trabalhar quando no final da função le_pipe() faz um post. Esta função vai ao segundo nó da lista ligada, porque o primeiro é sempre o cabeçalho e copia a informação que esta na estrutura paciente para a estrututa da message queue Mymsd, escreve na memoria partilhada as estatisticas da trriagem e manda para a message queue. Depois de mandar elimina o nó que tem esse paciente, sem perder a informação do resto dos pacientes.
Em seguida chama a função ver_MQ() que verifica os paciente que estão na message queue, se estiver mais de 80% do maximo da fila cria um doutor extra. Depois faz um post para função trabalha_doc() que é responsável pelo atendimento.

A função trabalha_doc() recebe as mensagens da message queue e verifica o numero de pessoas que lá estão se for menos de 80% do maximo da fila liberta o doutor extra. Em seguida faz o atendimento, se o tempo de atendimento do paciente for maior que o tempo do turno o doutro tem que esperar que o paciente acabe a consulta, se não no final do turno sai o doutor. É escrito na memoria partilhada as estatisticas da consulta e é feito em post para o a função substutuiDoutor().

Esta função é a worker as thread que está na posição 1 no array de threads e é responsavel por criar novos processos doutores quando o turno deles acaba, ou seja sempre que um doutor acaba o turno ele sustitui o seu id por -1 no array de pid que está na memoria partilhada. Assim esta thread

deteta esse evento e cria nessa posição outro processo doutor e passa-lhe a função trabalha_doc() para poder atender pacientes.

O programa ao receber um sinal SIGINT vai chamar a função termina() que por sua vez chama a função cleanup() que é responsavel por esperar o final do processos e acabar esses processos, terminar todas as threads, apagar a message queue, destruir todos os semáforos e mutexs, destruir a zona de memoria partilhada, fechar o pipe, libertar os arrays das threads e destruir a lista ligada da fila de espera.

Após a receção de um sinal SIGUSR1 é chamada a função print_stats() que vai mostrar no ecrâ todas a estatisticas até ao momento.

Para Mandar um paciente tem que abrir outro terminal na diretoria do programa e escrever no terminal “echo “nome tempo_triagem tempo_atendimento prioridade”>input_pipe”.