Relatório – Projeto SO
Meta Final

Problema:

Este projeto tem como objetivo simular um sistema de offloading de tarefas computacionais no edge computing, com interação entre os clientes e o servidor, onde o servidor precisa de responder aos pedidos que o cliente faz. Para isso, é necessário um sistema que distribua corretamente os recursos e os pedidos existentes, utilizando diversos conceitos relacionados com a cadeira, como threads, processos, semáforos, memória partilhada, ...

Funcionamento do programa:

O programa inicia com a chamada da função **SystemManager()**, que envolve a criação das estruturas e recursos que necessitamos para este projeto (shared memory, mutexes, pipes, message queue e uma lista ligada para controlar as tarefas). Lemos também os dados do ficheiro "ConfigFile.txt" relativos aos servers (número de servers e a descrição de cada um), e colocamos essa informação na shared memory.

As tarefas/pedidos vão ser criados pelo utilizador com a execução do ficheiro "mobileNode.c" (consola diferente), que fica recetivo a comandos relativos aos pedidos e os envia pelo named pipe para o **taskManager()**.

Em seguida, criamos os 3 processos "principais" (**taskManager()**, **monitor()** e **maintenance()**). O processo **taskManager()** cria os servers, recebe a informação das tarefas que vai gerir, e envia as mesmas para o server destinado através do unnamed pipe. Este tem duas threads, scheduler e dispatcher, a primeira é responsável por verificar se existem tarefas cujo prazo máximo de execução ja tenha passado(recebe um sinal vindo do TaskManager() para trabalhar, sempre que chegue uma tarefa), e a segunda, verifica se a tarefa tem tempo para ser executada, se sim, enviará para o Server disponível(recebe um sinal vindo do TaskManager() para trabalhar, e este trabalhará se tiver disponivel um VCPU e se tiver 1 tarefa à espera na fila.

O processo **monitor()** controla o número de VCPU ativos dos servers para execução das tarefas, havendo 2 modos de performance que dependem de várias condições (Normal Performance e High Performance). Através de uma variável condição, que verifica através de um sinal dado ao fim de cada execução ou um pedido é eliminado.

Finalmente, o processo **maintenance()** é responsável pela manutenção dos servers trocando informação através da message queue. Este que verifica se existem menos de 3 servidores em manutenção no caso de existir 2 não irá colocar outro em manutenção.

O programa pode receber o sinal SIGTSTP (utilizador prime CRTL + Z), que imprime as estatísticas, e termina com a captação do sinal SIGINT (o utilizador tem que premir CTRL + C para terminar o programa).

Shared Memory:

Esta zona de memória foi criada de modo a partilhar um conjunto de informações relevantes da simulação offloading entre todos os processos envolvidos. Deste modo, quando um processo altera uma determinada variável da shared memory, essa alteração é também visível em outros processos.

Named Pipe:

Canal que permite a comunicação entre o Mobile Node e o Task Manager. Através do named pipe são enviados o número de instruções/pedidos e o tempo máximo de execução dos mesmos.

O Task Manager vai receber essa informação e armazená-la na shared memory. Consoante o número de slots livres, o Task Manager vai colocar o pedido na lista de pedidos, ou, pelo contrário, eliminá-lo, uma vez que a lista está cheia de tarefas.

Unnamed Pipe:

Canal que permite a comunicação entre o Task Manager e cada um dos servers. As tarefas vão ser enviadas pelo unnamed pipe dedicado a cada um dos servers. A tarefa vai ser recebida no server respetivo e executada no VCPU do server que estiver disponível.

Desta forma, a informação é gerida pela função **createServers()** que vai receber as tarefas pelo unnamed pipe e executá-las em algum VCPU.

Message Queue:

O processo **maintenance()** vai periodicamente fazer a manutenção dos servidores. Essa comunicação é estabelecida por uma message queue, que avisa o server respetivo que vai entrar em manutenção. Com a receção dessa informação, o server termina primeiro as suas tarefas e, depois disso, a manutenção é feita num determinado intervalo de tempo aleatório.

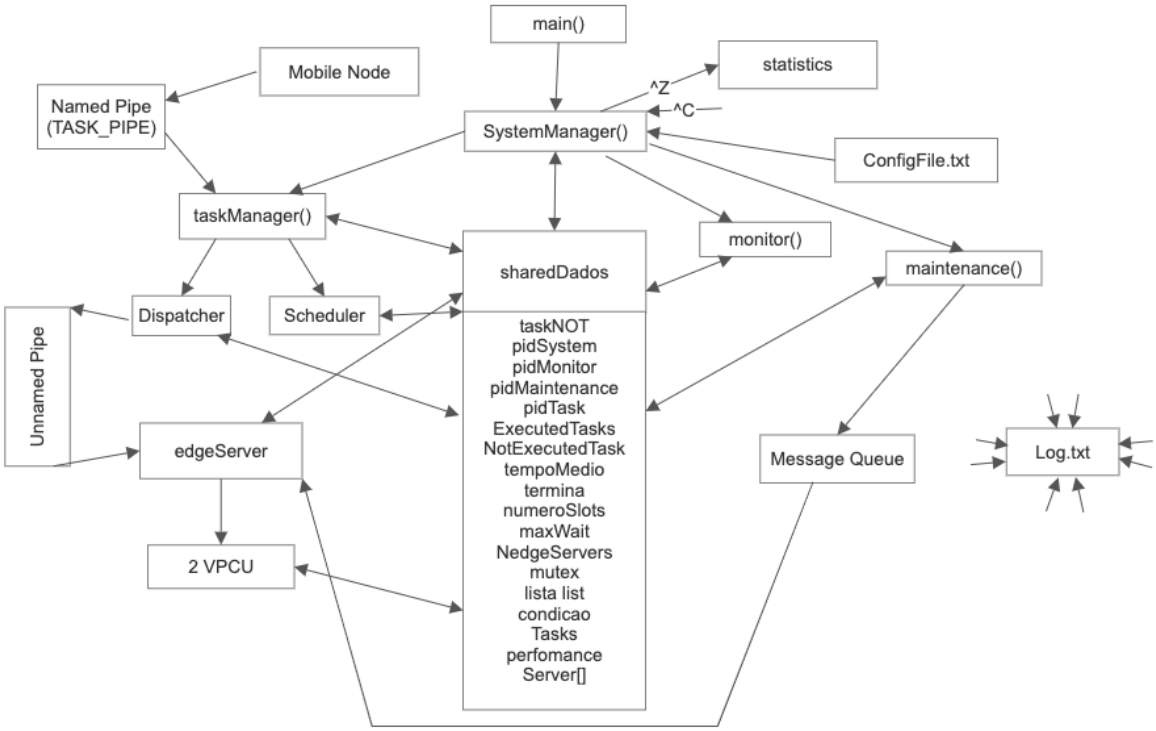
Sinais:

- **SIGTSTP:** imprime as estatísticas;
- **SIGINT:** termina o programa. Aguarda que as tarefas que foram enviadas através do unnamed pipe terminem. Após estas concluírem imprime as estatísticas e liberta/remove todos os recursos utilizados.
- **SIGTERM:** Envia um sinal ao Monitor.
- **SIGURS1:** Envia um sinal ao dispatcher e ao Monitor.
- **SIGURS2**: Envia um sinal à função que destroi_cond está que se refere se as últimas tarefas a serem enviadas para os Servers foram concluídas antes de fechar o programa.
- Ignoramos todos os outros sinais.

Mecanismos de sincronização:

- Mutex de controlo das threads (maintenance, monitor, scheduler, dispatcher,destroir,Servers,VCPU's);
- Semáforo de controlo de escrita do ficheiro log.txt;
- Semáforo de controlo de escrita na shared memory;
- Variáveis de condição nos processos monitor e maintenance;
- Variável de condição para os servers (na execução das tarefas nos VPCUS e para manutenção);
- Semáforo para controlo das threads VPCU;
- Captação dos sinais **SIGINT** para terminar o programa e **SIGTSTP** para imprimir as estatísticas do programa.

Diagrama do projeto:



Horas de trabalho:

-Bruno Sequeira – 100h

-Diogo Tavares - 70h

Trabalho realizado por:

- Bruno Eduardo Machado Sequeira nº 2020235721(PL8)

- Diogo Rafael Melo Tavares nº 2020236566(PL7)