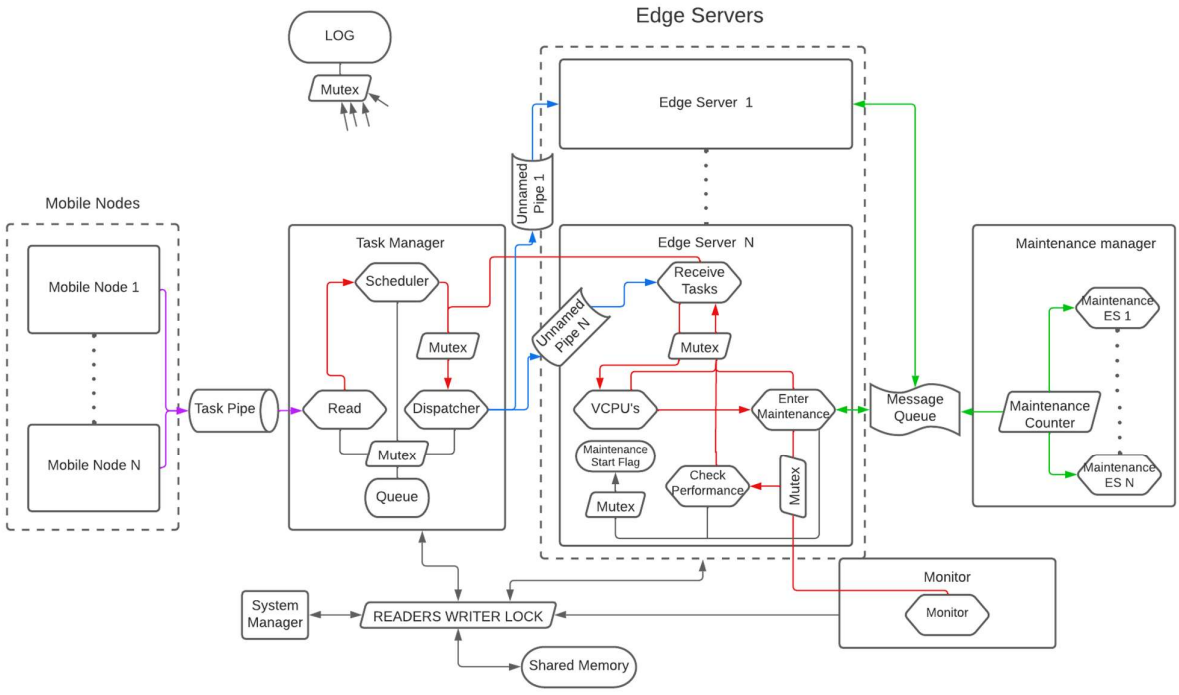
Trabalho prático – Simulador de *Offloading* de Tarefas no *Edge* (Meta final)

Projeto de Sistemas Operativos 2021/2022

Gonçalo Almeida
nº 2020218868
Departamento de Engenharia Informática
Universidade de Coimbra
Coimbra, Portugal
gfalmeida@student.dei.uc.pt
Cerca de 70 horas despendidas

João Santos
nº 2020218995
Departamento de Engenharia Informática
Universidade de Coimbra
Coimbra, Portugal
jbsantos@student.dei.uc.pt
Cerca de 75 horas despendidas



Legenda:

- Formas:
  - Hexágono – Threads
  - Elipse – Recursos
  - Retângulo – Processos
  - Paralelogramo – Semáforos / Mutexes
- Setas:
  - Vermelho – Variáveis de condição
  - Azul – Unnamed pipes
  - Roxo – Named pipe
  - Verde – Message Queue
  - Preto – Acesso a recursos

Breve explicação do projeto

Mobile Node

Gera tarefas para *offloading*, tendo em conta os parâmetros, abre um *named pipe* (TASK PIPE) para escrita e envia as tarefas pela *TASK PIPE* para o *Task Manager*. O id de cada tarefa é baseado no *pid* do *mobile node* e o número de pedidos.

Log

Ficheiro onde estão guardadas todas as informações acerca do programa. Por motivos de sincronização, é usado um *mutex* para não haver conflitos de escrita neste ficheiro entre entidades.

Monitor

Controla o nível de performance dos *Edge Server* de acordo com as regras estabelecidas. Isto é, através de uma *thread*, este altera a *performance change flag* presente na *shared memory,* quando o *task manager* o notifica que houve alterações, e notifica os *Edge Servers* que a mesma foi alterada. Os valores da *performance flag* podem ser 1 e 2, que correspondem, respetivamente, aos modos *normal* e *high performance* dos *edge servers*.

Maintenance Manager

Coloca os *Edge Servers (ES)* em manutenção por um número aleatório de segundos. Ou seja, existe uma *thread* de manutenção para cada *ES* e esta envia uma mensagem para o *ES* correspondente para iniciar-se a manutenção; a seguir espera que o *ES* lhe envie a mensagem de que já pode começar a manutenção ou de que esta vai ser abortada; por fim, se puder entrar em manutenção, a *thread* faz um *sleep*, e, no final deste, envia uma mensagem para o *Edge Server* a dizer que a manutenção terminou e faz outro *sleep*, por um número aleatório de segundos, para intervalar manutenções. Para evitar conflitos entre diferentes *ES* e *threads* de manutenção na *message queue*, consideramos que todos os *edge servers* e *threads* de manutenção têm tipos diferentes de mensagens, baseados nos ids de cada um.

System Manager

Lê o ficheiro de configurações e arranca todo o sistema. Isto é, cria o ficheiro de log, a *shared memory*, a *TASK PIPE*, a *message queue* e os processos do *Task Manager*, *Maintenance Manager* e *Monitor*. Guarda também todos os *Edge Servers* na *shared memory*, e cria alguns dos semáforos *mutexes* que são utilizados nos seus processos filho. Além disso, ao receber o sinal SIGTSTP, imprime as estatísticas do simulador, que estão guardadas na *shared memory.*

Task manager

Recebe comandos através da *task pipe*: se for "STATS", imprime as estatísticas do simulador, que estão guardadas na *shared memory*; se for "EXIT", envia um sinal SIGINT para o *system manager*, acabando o programa de forma controlada; se for uma tarefa, adiciona-a à fila de tarefas e notifica o *scheduler*. O *scheduler* começa por verificar tarefas expiradas, e depois reavalia as prioridades (a prioridade vai corresponder ao número de tarefas com menos tempo restante que a tarefa em causa mais um), e, finalmente, notifica o *dispatcher*. O *dispatcher*, quando é ativado (continua desativado enquanto não houverem vCPUs livres), pesquisa um *vCPU* que consiga executar a tarefa mais prioritária da fila de tarefas a tempo, envia-a para o *edge server*, através da *unnamed pipe* correspondente, esperando que este atualize o seu estado. É de notar que o acesso à fila de tarefas é feito através de um *mutex*.

Edge Servers

Cada *edge server* tem um nível de performance entre 0 e 2 (0-*Stopped*, 1-*Normal*, 2-*High Performance*). A *thread receive_tasks* espera que um dos *vCPUs* fique livre (depende também do nível de performance); notifica, de seguida, o *task manager* e espera por uma tarefa; ao recebê-la, atualiza o estado do *vCPU* que vai executar a tarefa na *shared memory* e confirma que recebeu a tarefa. A *thread* do *vCPU* que recebeu a tarefa, simula a sua execução da mesma, fazendo um *sleep* pelo tempo que iria demorar a executá-la (baseado na sua capacidade de processamento e número de instruções da tarefa). Quando a *thread enter_maintenance* recebe uma mensagem de início de manutenção, através da *message queue*, muda o nível de performance para 0 (a *thread receive_tasks* deixa de receber tarefas), espera que as tasks acabem (se o simulador for fechar enquanto está nesta fase, envia para o *maintenance manager* uma mensagem para abortar a manutenção), e confirma a manutenção ao *maintenance manager*. Ao receber uma nova mensagem a indicar o fim de manutenção, restaura o nível de performance tendo em conta a *performance change flag* da *shared memory* e confirma o fim da manutenção. A *thread change_performance* muda o nível de performance do *edge server* quando o monitor o notifica que mudou a *performance change flag* (tal não acontece se estiver em manutenção).

Shared memory

A *shared memory* contém informações que os processos necessitam de partilhar. Para maximizar o acesso à *shared memory* sem corrupção de dados, implementámos um sistema *readers-writer-lock*, isto é, várias entidades podem ler da *shared memory* simultaneamente, mas, quando é necessário escrever nesta, apenas essa entidade tem acesso. Para fazer algum encapsulamento à *shared memory*, cada recurso é acedido através de uma função *get* e atualizado através de uma função *set*.

Fim do simulador

O *system manager*, ao receber um sinal SIGINT (premindo *Ctrl+C* na consola ou enviando "EXIT" para a *task pipe*), envia o sinal SIGUSR1 para os outros processos (os *edge servers* recebem-nos através do *task manager*) e limpa os recursos utilizados. Cada um destes vai cancelar as suas *threads* através de *pthread_cancel* (em secções críticas, as *threads* bloqueiam o seu cancelamento) e acordar *threads* que estejam em *pthread_cond_wait,* limpando os recursos e saindo de seguida.