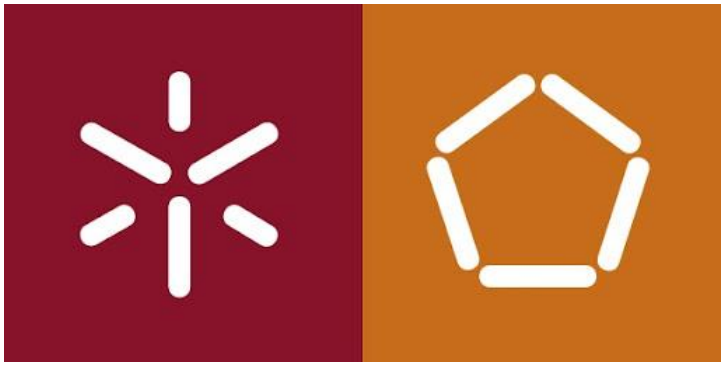

UNIVERSIDADE DO MINHO

LICENCIATURA EM ENGENHARIA INFORMÁTICA

SISTEMAS OPERATIVOS

Grupo

Diogo Costa a95147

Rodrigo Lopes a104440

Introdução:

No âmbito da Unidade Curricular de Sistemas Operativos foi proposto a realização de um projeto em que precisamos implementar um sistema de conexão entre um cliente e um servidor, sendo o cliente a interação principal que o utilizador tem com o programa onde disponibiliza a informação da tarefa que deseja realizar, sendo o servidor quem realiza essa tarefa e guarda o resultado num ficheiro.

Estruturas Utilizadas:

- **TASK:** é uma estrutura que serve para guardar a informação que o utilizador disponibiliza somente relativa à tarefa em si, ou seja, o nome/caminho que o programa tem de executar e os argumentos do programa. Desta forma temos um pointer que aponta para o nome do programa a ser executado, temos um array de pointers, cada elemento é uma string que representa um argumento do programa, com um tamanho máximo em 300, além disso possui ainda inteiro para definir o tipo de tarefa;
- **MSG:** é uma estrutura que serve para guarda toda a informação que o utilizador disponibiliza. Desta forma temos uma estrutura do tipo TASK que representa uma tarefa, como dito anteriormente, e temos 5 inteiros sendo que cada um representa, respetivamente, o identificador do cliente que enviará a mensagem ao servidor, o id (…), o tempo de execução estimado que é indicado pelo utilizador, o número de tarefas incluídas na mensagem e o tipo de pedido associado (para saber distinguir se a mensagem enviada é execute ou status);
- **WAITQUEUE:** esta é uma estrutura um pouco diferente das outras visto que ela serve para criar uma fila de espera para tarefas. Ela possui um pointer para uma estrutura TASK, ou seja, armazena uma referência para uma tarefa que está na fila de espera e um pointer para uma estrutura waitqueue que representa o próximo elemento na fila de espera, ou seja cada elemento aponta para o próximo elemento da fila utilizando assim listas ligadas. Utilizamos listas ligadas uma vez que trabalhamos com inserções e remoções, embora

as inserções sejam mais lentas, a remoção é muito mais eficiente o que acaba por compensar;

- **STATUS:** esta estrutura serve para guardar informações relativamente ao estado de cada tarefa. Elas podem ter 3 estados: em execução, em espera e completas. Desta forma temos 3 arrays de MSG, respetivamente, um que armazena as mensagens que estão em fila de espera, outro que armazena as mensagens que estão a ser executadas e outro que armazena as mensagens que estão concluídas. Além disso possui 3 inteiros, um para cada estado, que informa o tamanho;

Iniciar servidor:

Para iniciar o servidor (que vamos passar a chamá-lo de orchestrator), é preciso invocar o executável ./bin/orchestrator. Este executável tem de receber 2 argumentos, a pasta onde são guardados os ficheiros com o output de tarefas executadas e o número de tarefas que podem ser executadas em paralelo.
Criamos uma estrutura chamada status e inicializamos os campos waiting, running e completed size a 0. Depois criamos um pipe chamado orchestrator com as permissões 0600, ou seja o proprietário do arquivo pode ler e escrever no pipe, mas ninguém mais tem permissões.
De seguida abrimos esse pipe apenas para leitura e armazenamos isso em fd. Depois destes processos, o servidor entra em modo operacional.

Enviar pedido:

Quando queremos enviar uma tarefa para o orchestrator temos de recorrer ao executável ./bin/cliente. Para isto o executável utiliza de auxílio uma estrutura MSG que vai servir de mensagem do pedido.

Existem 2 tipos de operações que podemos fazer, ou enviar uma tarefa para o orchestrador fazer ou listar todas as tarefas que passaram pelo servidor.

Na opção execute criamos uma estrutura tarefa e armazenamos o segundo argumento no campo programa. A seguir cria uma string args e é preenchida com os restantes argumentos e o definimos o campo tipo como 1. Criamos uma estrutura mensagem, que é preenchida com a

tarefa, o ID do processo atual (através da função getpid()), o tempo esperado para a realização da tarefa e o tipo de pedido como 1. Abrimos então o FIFO para fazermos a comunicação entre o cliente e o orchestrator.
Na opção status, criamos uma estrutura chamada mensagem e uma estrutura status. Chamamos uma função que cria uma mensagem, uma função para criar um fifo com o nome do id do cliente (que é dado pela função getpid()), a função sendMsg envia a mensagem para o orchestrator. Após o programa orchestrator realizar a consulta de cada tarefa, o cliente lê o status atualizado através da função readStatus e dá print do estado atual das tarefas.

Receber pedido:

No orchestrador criamos uma estrutura status e inicializamos os campos waiting, running e completed a 0. Criamos um fifo e abrimos com permissão de leitura apenas. O programa entra aqui num loop infinito onde fica sempre à espera da entrega de tarefas vindas de clientes. Dentro do orchestrador temos uma função print que escreve o resultado num ficheiro de output, um função trata_pedido onde se o pedido for um execute de uma tarefa ele cria um processo filho destinado à realização da mesma, caso seja status ele abre um fifo e envia a mensagem do orchestrator para o cliente. Temos também uma função add_task que serve para aumentar a entrada waiting_size em 1 unidade, uma função exec_task que me retira a tarefa que estava a aguardar a inicialização e passa ao estado de execução e a função end_task que a ideia é a mesma mas retira a tarefa da execução e passa ao estado de concluída.

Observações:

Ao realizar o comando status obtivemos um problema porque não conseguíamos enviar o status atualizado do orchestrator para o cliente o programa orchestrator os valores da estrutura não eram enviados corretamente.