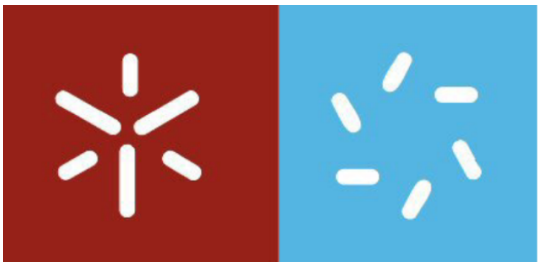

UNIVERSIDADE DO MINHO

LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

Sistemas Operativos

Trabalho prático - Grupo 27

Moisés Ferreira	Rita Machado	José Martins
A97020	A102508	A102928

1. Introdução

O presente relatório refere-se ao trabalho prático proposto na unidade curricular de Sistemas Operativos na Licenciatura em Ciências da Computação da Escola de Ciências da Universidade do Minho.

Foi proposto a criação de um serviço de orquestração de tarefas num ambiente computacional. O objetivo principal deste projeto é desenvolver um sistema capaz de receber e gerir solicitações de execução de tarefas por parte dos utilizadores, atribuindo identificadores únicos e escalonando a execução de forma eficiente. A interação entre o cliente (responsável por submeter as tarefas) e o servidor (encarregado de executá-las e monitorizar o seu progresso) constitui o foco deste trabalho, que visa explorar conceitos como execução encadeada de programas, processamento em paralelo e avaliação de políticas de escalonamento.

2. Arquitetura

A aplicação adota a arquitetura Cliente/Servidor, o que implica na existência de dois programas distintos: o Cliente, encarregado de enviar Tarefas ao Servidor; e o Servidor, encarregado de executar comandos de "bash" e consultar os programas em execução, enviando os resultados de volta ao Cliente.

3. Funcionalidades

1. Execução encadeada de programas:
   Permite a criação de pipelines de programas para execução sequencial, tal como o operador "|".

2. Submissão de tarefas individuais e pipelines de programas:
   Possibilita ao utilizador submeter pedidos de execução de tarefas individuais, indicando tempo estimado e programas a executar, assim como pipelines de programas.

3. Consulta do estado das tarefas:
   Permite ao utilizador verificar o estado das tarefas em execução, agendadas e concluídas.

4. Definição do número de tarefas em paralelo:
   Permite ao utilizador especificar o número de tarefas a serem executadas simultaneamente, o que otimiza o algoritmo de escalonamento e a velocidade de processamento.

5. Avaliação de políticas de escalonamento:
   Permite ao utilizador desenvolver testes para avaliar a eficiência da política de escalonamento implementada e compara-a com outras políticas para melhorar a experiência dos utilizadores.

4. Cliente

A função do arquivo client.c é atuar como o cliente que interage com o servidor para submeter pedidos de execução de tarefas e consultar o status das mesmas.

- Interface com o utilizador:
  O arquivo client.c tem a lógica para interagir com o utilizador por meio da linha de comando, que permite que ele envie comandos para solicitar a execução de tarefas e consultar o status das mesmas.

- Envio de Pedidos ao Servidor:
  Através do client.c, o utilizador pode enviar pedidos de execução de tarefas individuais ou pipelines de programas para o servidor. Esses pedidos são enviados via pipes nomeados para o servidor.

- Recebimento de Respostas do Servidor:
  Após enviar um pedido de execução de tarefa, o client.c aguarda a resposta do servidor, que inclui um identificador único da tarefa. Essa informação é então apresentada ao utilizador.

- Consulta de Status das Tarefas:
  Além de enviar pedidos de execução, o client.c também permite que o utilizador consulte o status das tarefas em execução, em espera e terminadas. Essas consultas são enviadas ao servidor para processamento.

- Tratamento de Comandos e Argumentos:
  O client.c é responsável por interpretar os comandos e argumentos fornecidos pelo utilizador, que garante que os pedidos são formatados corretamente antes de serem enviados ao servidor.

- Apresentação de Resultados ao Usuário:
  Após receber as respostas do servidor, o client.c apresenta as informações relevantes ao utilizador, como identificadores de tarefas, programas em execução e tempos de execução.

- Interação com o Servidor:
  O client.c estabelece a comunicação com o servidor para enviar e receber informações sobre as tarefas em execução, o que permite uma interação eficaz entre o utilizador e o sistema de orquestração de tarefas.

5. Orchestrator

O arquivo orchestrator.c atua como o servidor responsável por receber os pedidos de execução de tarefas dos clientes, escalonar e executar essas tarefas, além de fornecer informações sobre o status das tarefas em execução.

- Inicialização do Servidor:
  O arquivo orchestrator.c é responsável por inicializar o servidor que fica a aguardar conexões dos clientes para receber pedidos de execução de tarefas.

- Recepção de Pedidos dos Clientes:
  O servidor aguarda a chegada de pedidos de execução de tarefas dos clientes por meio de pipes.

- Escalonamento de Tarefas:
  Após receber um pedido de execução de tarefa, o servidor no orchestrator.c é responsável por escalonar as tarefas dos clientes e leva em consideração a duração estimada de cada tarefa fornecida pelos usuários.

- Execução de Tarefas:
  O servidor no orchestrator.c executa os programas das tarefas conforme especificado, o que garante que os programas são executados como em um ambiente Linux normal. O output e error dos programas são redirecionados para arquivos com o identificador da tarefa correspondente.

- Registro de Informações:
  Após a conclusão de uma tarefa, o servidor registra as informações relevantes, como o identificador da tarefa e o tempo de execução real, desde o momento em que recebe o pedido até a conclusão do programa.

- Consulta de Status das Tarefas:
  O servidor no orchestrator.c também é responsável por atender às consultas dos clientes sobre o status das tarefas em execução, em espera e terminadas, e fornece informações detalhadas sobre cada uma.

- Comunicação com o Cliente:
  O servidor mantém a comunicação com os clientes por meio de pipes nomeados, responde aos pedidos de execução de tarefas e consultas de status de forma eficiente e precisa.

6. Estrutura de dados

Como neste projeto estamos a usar dois sistemas diferentes, o client e o orchestrator, criamos estas estruturas para simular as mensagens que trocam ente sim. Sendo os pedidos as ,mensagens que o client envia ao servidor e as tarefas a solução a esse pedido.

Estrutura de dados que é utilizada para representar um pedido do sistema.

```c
struct Request{
    char command[300];
    int eta; //tempo estimado de conclusao
    int pid; //pid do processo
    int commandType;
    // Flag caso queriamos o status do programa
    int isStatus;
}request;
```

Estrutura de dados para representar a tarefa.

```
struct Task{
    int taskId; //identificador unico da tarefa
    // 0-> not started, 1-> running, 2-> finished
    int eta; //tempo estimado de conclusao
    int pid; //pid do processo
    int status; //indica o status da tarefa
    float timeElapsed; //epresenta o tempo decorrido desde o inicio da execucao da tarefa
    int isStatus;
    int commandType;
    char command[300]; //armazena o comando associado a tarefa
}task;
```

Estrutura de dados usada para representar um comando no sistema.

```
struct Command{
    char* args[300];
    char* path;
}command;
```

Estrutura de dados para armazenar informações de configuração relacionadas ao servidor.

```
struct ServerConfig{
    char* outputfolder;
    int numParallelTasks; //representa o numero maximo de tarefas que o servidor pode executar em paralelo
    int schedPolicy; //indica a politica de escalonamento utilizada pelo servidor para gerenciar as tarefas
}serverconfig;
```

7. Conclusão

Estamos satisfeitos com o resultado do nosso trabalho, apesar dos desafios que encontramos pelo caminho. Uma peça fundamental para superar esses obstáculos foram as soluções apresentadas nos guias práticos pela equipe docente.

Esse trabalho não apenas permitiu-nos enfrentar os desafios, mas também consolidar e aplicar os conhecimentos adquiridos ao longo do semestre na disciplina de Sistemas Operativos de uma forma prática e envolvente.