Universidade do Minho
Licenciatura em Engenharia Informática

Unidade Curricular de Sistemas Operativos

Ano Letivo de 2023/2024

Orquestrador de Tarefas

Fernando Pires, A77399

Pedro Teixeira, A103998

Sara Silva, A104608

Maio, 2024

SO

Índice

Conteúdo

1. Introdução

Este relatório é relativo ao trabalho prático proposto na unidade curricular de Sistemas Operativos em Licenciatura em Engenharia Informática da Universidade do Minho no ano de 2024.

O objetivo do trabalho consiste na implementação de um serviço de orquestração de tarefas num computador. Os utilizadores devem usar um programa cliente para submeter ao servidor a intenção de executar uma tarefa, dando uma indicação da duração em milissegundos que necessitam para a mesma, e qual a tarefa a executar.

No que se segue é possível ver as decisões tomadas pelo grupo durante a realização do projeto, e o porquê das mesmas.

2. Funcionalidades

O nosso projeto possui todas as funcionalidades propostas no enunciad, das quais:

- **Execução de tarefas do utilizador;**
- **Consulta de tarefas em execução;**
- **Execução encadeada de programas;**
- **Processamento de várias tarefas em paralelo;**
- **Duas políticas de escalonamento;**
- **Avaliação de políticas de escalonamento;**

3. Arquitetura

Como mencionado anteriormente, a aplicação segue o modelo cliente/servidor, pelo que é necessário desenhar dois programas: o servidor que é responsável por escalonar as tarefas dos múltiplos clientes de acordo com diferentes políticas, executar os programas das tarefas e armazenar a informação. E o cliente, que é responsável por executar commandos e consultar os programas em execução, os finalizados e os em espera.

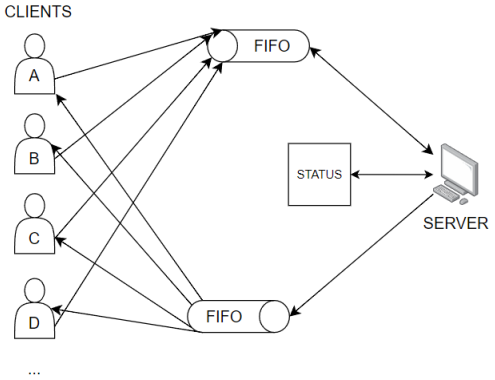

Figure 1: Arquitetura

3.1. Servidor

O servidor funciona da seguinte maneira:

O servidor, que está sempre aberto, recebe através do FIFO(main_fifo) um pedido de execução de um programa.

Ao receber o programa, o mesmo processa-o e envia-o novamente para o FIFO, onde são feitas as devidas verificações para ver se já está terminado e adiciona-o aos “Status”.

Caso o pedido de programa seja um “status”, o servidor reencaminha para o cliente todos os programas pedidos até ao momento, dividindo-os em “*Scheduled*”, “*Executing*” e “*Finished*”.

3.2. Cliente

Todos os pedidos criados pelo cliente são enviados através de um FIFO(main_fifo) para o servidor, aquando de um pedido de “status” o cliente recebe todas as informações geradas pelo servidor através de outro FIFO(sv_to_cl_fifo).

Posto isto, é dada a possibilidade ao cliente de realizar diferentes tipos de pedidos. Este pode pedir um e*xecute*, sendo possível este ser um comando único ou vários comandos encadeados(*flags -u e -p* respetivamente)(pipeline). Assim que este pedido é iniciado, o cliente recebe notificação do mesmo. Pode também ser pedido pelo cliente um *status* que, tal como já foi referido, é demonstrado ao cliente todos os programas pedidos até ao momento, dividindo-os em “*Scheduled*”, “*Executing*” e “*Finished*”.

4. Implementação

Foram testadas várias formas de implementação no decorrer do projeto, porém foi escolhida esta pois acreditamos que é a que melhor se adequa ao nosso projeto.

Inicialmente, é aberto o servidor fornecendo-se como argumentos o nome de uma pasta(que , caso não exista, é criada), o número de programas que podem ser executados em paralelo e por fim a política de escalonamento. Após isto, o mesmo fica sempre aberto e qualquer cliente já pode mandar pedidos. Estes pedidos são então processados e enviados novamente para o FIFO(main_fifo), onde são feitas as devidas verificações para ver se já estão terminados e adiciona-os aos “Status”. Em caso de um pedido de *status* o FIFO(sv_to_cl_fifo) é aberto e as informações são enviadas para o cliente.

4.1. Estruturas

Para facilitar a comunicação entre cliente e servidor as seguintes estruturas foram criadas:

4.1.1. Programa

```
typedef struct
{
    int running; // 2 for done, 1 for running, 0 for waiting
    int status;  // 1 for status, 0 for execute
    int expected_time;
    long time;
    char flag[MAX_FLAG_SIZE];
    char arguments[MAX_ARGUMENTS_SIZE];
    int processID;
} PROGRAM;
```

Figure 2: Estrutura do programa

Esta estrutura foi criada com o propósito de guardar, de forma mais efetiva, as informações contidas no pedido do cliente.

4.1.2. Queue

```
typedef struct
{
    PROGRAM values[MAX_ELEMENTS_IN_QUEUE];
    int inicio;
    int tamanho;
} QUEUE;
```

Figure 3: Estrutura da queue

Esta estrutura foi criada com o objetivo de facilitar o armazenamento dos programas à espera de serem executados.

4.1.3. Finished

```
typedef struct
{
    PROGRAM values[MAX_FINISHED_PROGRAMS];
    int tamanho;
} FINISHED;
```

Figure 4: Estrutura do finished

Esta estrutura foi criada com o objetivo de armazenar os programas já executados.

4.1.4. Status

```
typedef struct
{
    int current_executing;
    int max_executing;
    PROGRAM *executing;
    QUEUE queue;
    FINISHED finished;
} STATUS;
```

Figure 5: Estrutura dos Status

Esta estrutura foi criada com o objetivo de facilitar o envio das informações para o cliente.

4.2. Tarefas em Paralelo

Na inicialização do servidor é necessário indicar o limite de paralelização das tarefas(N), assim, o servidor consegue executar N tarefas ao mesmo tempo, independentemente de estas serem constituídas por um ou mais programas.

4.3. Políticas de Escalonamento

Para o nosso projeto escolhemos implementar duas políticas de escalonamento:

- *First come first served*: onde as tarefas são executadas por ordem de *input* do cliente;
- *Shortest job first*: onde a tarefa que é escolhida para ser executada da queue é a que tem menor *expected time* de execução;

5. Outros

5.1. Makefile

De forma a conseguirmos realizar o trabalho da melhor forma possível utilizámos o Makefile fornecido pelo professores no enunciado, porém com algumas alterações com vista a melhorar o desempenho e a obter uma maior liberdade de utilização.

5.2. Testes

Com o objetivo de facilitar os testes do programa foi criado um script em bash com vários pedidos de tarefas diferentes, sendo assim possível ver se o programa funciona em diferentes casos.

6. Conclusão

Com a realização deste projeto foi-nos possível consolidar novos conhecimentos no âmbito da linguagem de programação C e da unidade curricular de Sistemas Operativos.

Posto isto, também acreditamos que há vertentes podiam ser melhoradas, porém, dentro do que era o objetivo do projeto e o que nos foi proposto, cremos que o mesmo foi desenvolvido de uma maneira correspondente aos novos conceitos que nos foram mostrados nas aulas.