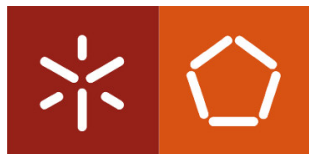


UNIVERSIDADE DO MINHO

ESCOLA DE ENGENHARIA



Sistemas Operativos

Licenciatura em Engenharia Informática

Orquestrador de Tarefas

Grupo 78

Ano letivo 2023/24

Flávio Silva - [A97352]
Tomás Melo - [A104529]
José Vasconcelos - [A100763]

Maio, 2024

Conteúdo

1	Introdução	2
2	Funcionalidades disponíveis no orquestrador	3
2.1	Status	3
2.2	Flags <i>-u</i> e <i>-p</i>	3
2.3	<i>SJF</i> e <i>time</i>	4
2.4	Execução de comandos e Ficheiro de <i>Output</i>	4
2.5	Gestão de tarefas	5
2.6	Comunicação <i>cliente-orquestrador</i>	6
3	Demonstração	7
4	Testes e Reflexão dos Testes Efetuados	9
5	Makefile	10
6	Conclusão	11

1. Introdução

No âmbito da unidade curricular de Sistemas Operativos, foi-nos proposto a realização de um trabalho prático de rastreamento e monitorização da execução de programas. Neste programa, os utilizadores devem conseguir executar programas através do cliente, e obter o respetivo tempo de execução. Foi desenvolvido para tal, um serviço de orquestração de tarefas em ambiente *Linux* com o objetivo de implementar um sistema *cliente-servidor* em linguagem *C* que permita utilizadores enviarem solicitações de execução de tarefas para o servidor, o qual é responsável por escalonar e executar essas tarefas de acordo com a política de escalonamento *SJF*, que permite a redução do tempo médio de execução das tarefas.

Um dos requisitos é que seja possível consultar, através do servidor, todos os programas que se encontram atualmente em execução, juntamente com o tempo dispendido pelos mesmos. Para além disto, o servidor deve também permitir a consulta de estatísticas sobre programas já terminados.

2. Funcionalidades disponíveis no orquestrador

2.1 Status

Esta funcionalidade permite listar todas as tarefas enviadas pelo cliente até ao momento. Ao longo do tempo em que o cliente envia tarefas para o orquestrador, estas podem ser consultados pelo comando `./client status`. A lista das tarefas é composta por:

- Program ID (ID da tarefa)
- Command (comando(s) recebidos)
- Status (*estado da tarefa* -> Completed / Executing / Scheduled)
- Time (duração da tarefa) referente a cada uma delas

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/projetoSOFINAL/SOTP-2024$ ./client status
Pipe_name: /tmp/my_pipe_181565
Program ID: 2
Command: pwd | wc -l queue.c | man ascii
Status: Completed
Time: 113ms

Program ID: 1
Command: ls -la
Status: Completed
Time: 13ms
```

Figura 2.1: Resultado produzido após `./client status`

2.2 Flags `-u` e `-p`

Para execução/envio de uma tarefa para o orquestrador, pelo comando `./client execute time flag "prog-a [args]"`, em que `time` refere-se ao tempo estimado para a execução da tarefa em *milissegundos*), pode tanto ser passado no comando a *flag* `-u`, assim como a *flag* `-p`. A primeira, permite a execução de um programa individual, enquanto a segunda suporta a execução encadeada de programas do utilizador, ao que chamamos *pipelines*. A *flag* `-p` suporta também a execução de um programa individual (envio de um único comando).

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/projetoSOFINAL/SOTP-2024$ ./client execute 100 -u "ls -la"
1 Task Received
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/projetoSOFINAL/SOTP-2024$
```

Figura 2.2: Envio de comando com a *flag* `-u` no comando `execute`

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/projetoSOFINAL/SOTP-2024$ ./client execute 3000 -p "pwd | wc -l queue.c | man ascii"
3 Tasks Received
```

Figura 2.3: Envio de comando com a *flag* `-p` no comando `execute`

2.3 SJF e *time*

Reconhecendo que o orquestrador deve escalonar tarefas de múltiplos clientes, decidimos adotar a política de escalonamento SJF. Esta política, através da indicação da duração da tarefa dada pelo '*time*', dá prioridade às tarefas cuja estimativa de tempo de duração seja menor, reduzindo assim o tempo médio de espera. Por exemplo, dadas 3 tarefas (Tarefa1 -> duração 50ms, Tarefa2 -> duração 35ms, Tarefa3 -> duração 20ms) com a política implementada, a ordem de execução seria: Tarefa3 -> Tarefa2 -> Tarefa1, resultante num tempo médio de espera de 25ms, (em vez de 45ms, valor obtido na política FCFS).

2.4 Execução de comandos e Ficheiro de *Output*

Ao enviar uma tarefa para o orquestrador utilizando o comando `./cliente execute`, como já mencionado, o orquestrador executa os comandos solicitados pelo cliente quer sejam comandos individuais ou *pipelines* de comandos.

```
./client execute 1000 -u "pwd"
```

O orquestrador executará o comando *pwd* e imprime o conteúdo no ficheiro de *output*.

Ficheiro de *output* criado para o comando acima:

Ficheiro: *output_1.txt*

```
/home/tomas/projetoS0/S0TP-2024
```

Para além do ficheiro de *output* é ainda gerado um ficheiro *status.txt* com os *status* de cada tarefa enviada pelo cliente.

Ficheiro: *program_status_1.txt*

Program ID: 3

Command: pwd

Status: Completed

Time: 1ms

Estas funcionalidades proporcionam ao orquestrador a capacidade de gerenciar eficientemente as tarefas enviadas pelos clientes. A política de escalonamento SJF, juntamente com o registo detalhado das tarefas concluídas, contribui para uma melhor utilização dos recursos do sistema e uma experiência de utilizador mais satisfatória. Além disso, as opções de consulta de *status* e o suporte para diferentes tipos de comandos proporcionam flexibilidade e conveniência aos utilizadores do sistema.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <fcntl.h>
5 #include <string.h>
6 #include <sys/stat.h>
7
8 typedef struct Node {
9     char* command;
10    int pid;
11    int time;
12    struct Node* next;
13 } Node;
14
15 typedef struct Queue {
16     Node *front, *rear;
17 } Queue;
18
19 Node* newNode(char* command, int pid, int time) {
```

Figura 2.4: Amostra do resultado produzido em output_1.txt

```
1 Program ID: 1
2 Command: ls -la
3 Status: Completed
4 Time: 13ms
5
```

Figura 2.5: Informações produzidas em program_status_1.txt

2.5 Gestão de tarefas

Cada tarefa é executada em um processo filho separado, garantindo assim isolamento e segurança. Tal permite que o orquestrador continue a receber e processar novas tarefas enquanto outras estão em execução. O registo das tarefas que entram em execução, incluindo a estimativa de tempo de duração do comando, comando em si e o PID do processo filho responsável pela execução, facilita o acompanhamento do estado das tarefas e identificação de eventuais problemas que possa ocorrer no momento de testagem. Após a conclusão de uma tarefa, o orquestrador remove a tarefa da fila e do registo de tarefas em execução, garantindo uma gestão eficiente dos recursos do sistema e evitando possíveis *memory leaks*. A comunicação entre o **cliente** e o **orquestrador** pela troca de mensagens de forma assíncrona e bidirecional garante uma integração suave entre os componentes **cliente** e **orquestrador**, facilitando a interação do utilizador com o sistema.

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/projetoSOFINAL/SOTP-2024$ ./orchestrator
Received PID: 180782
Received Time: 100
Received Command: ls -la
Child exited whit status 0
Received PID: 181451
Received Time: 3000
Received Command: pwd | wc -l queue.c | man ascii
Child exited whit status 0
```

Figura 2.6: Informações produzidas pelo servidor após envio de tarefas

2.6 Comunicação *cliente-orquestrador*

O cliente envia tarefas para execução no orquestrador através do pipe com nome. Ao executar o comando `./cliente execute`, o cliente especifica o comando a ser executado, seguido de quaisquer argumentos necessários (formato do comando `execute` já descrito acima). Estas informações são encapsuladas em mensagens e enviadas ao servidor para 'futuro' processamento. O servidor recebe as tarefas enviadas pelo cliente através do pipe. É feita a leitura das mensagens recebidas, extração das informações relevantes, seguida do enfileiramento das tarefas para execução. O cliente pode consultar o estado das tarefas em curso no servidor através do comando `./cliente status`. Ao executar este comando, o cliente envia uma mensagem ao servidor solicitando informações sobre o estado das tarefas. O servidor responde, enviando os detalhes das tarefas em curso de volta ao cliente através do pipe. São então enviadas informações detalhadas sobre o estado das tarefas em andamento (informações referidas acima em **Status**). O projeto fornece uma interface intuitiva que permite estas comunicações.

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/projetoSOFINAL/SOTP-2024$ ./client execute 1000 -u "cat queue.c"
1 Task Received
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/projetoSOFINAL/SOTP-2024$ ./client status
Pipe_name :/tmp/my_pipe_183481
Program ID: 2
Command: pwd | wc -l queue.c | man ascii
Status: Completed
Time: 113ms

Program ID: 1
Command: ls -la
Status: Completed
Time: 13ms

Program ID: 4
Command: cat queue.c
Status: Completed
Time: 1ms
```

Figura 2.7: Verificação da comunicação pelo comando status

3. Demonstração

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/FINALSO/SOTP-2024/src$ ./client execute 1000 -u "ls -la"
1 Task Received
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/FINALSO/SOTP-2024/src$
```

Figura 3.1: Envio da tarefa com o comando `ls -la` com a flag `-u` com o servidor já a correr em FCFS

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/FINALSO/SOTP-2024/src$ ./orchestrator "FCFS"
Received PID: 8269
Received Time: 1000
Received Command: ls -la
Child exited whit status 0
```

Figura 3.2: Output no servidor após envio da tarefa

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/FINALSO/SOTP-2024/src$ ./client status
Pipe_name :/tmp/my_pipe_8432
Program ID: 1
Command: ls -la
Status: Completed
Time: 6ms
```

Figura 3.3: Consulta do status

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/FINALSO/SOTP-2024/src$ ./orchestrator "FCFS"
Received PID: 8269
Received Time: 1000
Received Command: ls -la
Child exited whit status 0
Received PID: 8432
Received Time: 0
Received Command: status
pipe_name:/tmp/my_pipe_8432.
```

Figura 3.4: Output visto no servidor após execução do status

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/FINALSO/SOTP-2024/src$ ./client execute 3000 -p "pwd | wc -l queue.c | man ascii"
3 Tasks Received
```

Figura 3.5: Envio de novas tarefas, desta vez com a flag `-p`


```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/FINALSO/SOTP-2024/src$ ./orchestrator "FCFS"
Received PID: 8269
Received Time: 1000
Received Command: ls -la
Child exited whit status 0
Received PID: 8432
Received Time: 0
Received Command: status
pipe_name:/tmp/my_pipe_8432.
Received PID: 8484
Received Time: 3000
Received Command: pwd | wc -l queue.c | man ascii
Child exited whit status 0
```

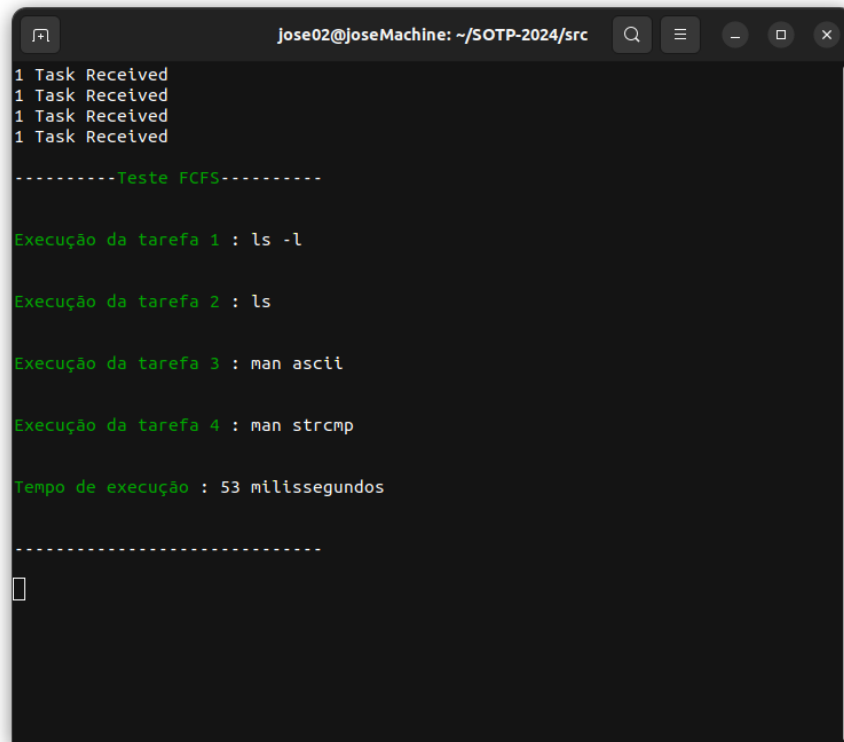
Figura 3.6: Output no servidor

```
tomas@tomas-ASUS-TUF-Dash-F15-FX517ZE-FX517ZE:~/FINALSO/SOTP-2024/src$ ./client status
Pipe_name :/tmp/my_pipe_8549
Program ID: 3
Command: pwd | wc -l queue.c | man ascii
Status: Completed
Time: 100ms

Program ID: 1
Command: ls -la
Status: Completed
Time: 6ms
```

Figura 3.7: Consulta dos status

4. Testes e Reflexão dos Testes Efetuados



```
jose02@JoseMachine: ~/SOTP-2024/src
1 Task Received
1 Task Received
1 Task Received
1 Task Received

-----Teste FCFS-----

Execução da tarefa 1 : ls -l

Execução da tarefa 2 : ls

Execução da tarefa 3 : man ascii

Execução da tarefa 4 : man strcmp

Tempo de execução : 53 milissegundos

-----
█
```

Figura 4.1: Teste de tempo de execução para FCFS

Algoritmo	FCFS	SJF
Tempo execução script 1 (ms)	18	52
Tempo execução script 2 (ms)	53	57
Tempo execução script 3 (ms)	40	74
Tempo execução script 4 (ms)	37	50
Tempo execução script 5 (ms)	40	72
Tempo execução script 6 (ms)	12	70
Tempo execução script 7 (ms)	63	62
Média do tempo de execução (ms)	37.57	62.43

5. Makefile

A compilação do código fonte é simplificada e automatizada através de um Makefile. **all:** Alvo padrão que é executado quando nenhum alvo específico é fornecido. O comando **make** ou **make all** fará a compilação quer do servidor quer do cliente.

Compilação do Orchestrator e Cliente: A compilação dos assuntos de orchestrator é baseada nos ficheiros fonte **orchestrator.c** e **queue.c**. `gcc orchestrator.c queue.c queue2.c -o orchestrator` é utilizado para compilar os ficheiros fonte num executável chamado **servidor**. Para o cliente, baseia-se no ficheiro fonte **client.c**. O comando `gcc client.c -o client` compila este ficheiro fonte num executável denominado **client**.

clean: Definida para remover os executáveis **servidor** e **cliente**, bem como os diretórios **Resultados** e **Status** criados durante a execução do programa.

6. Conclusão

O projeto foi concluído com sucesso, atendendo a todos os requisitos especificados no enunciado. Implementámos um sistema que acreditámos que seja robusto de cliente-servidor capaz de lidar com solicitações de execução de tarefas, pipelines de programas e consultas de status das mensagens enviadas. O uso de pipes com nome para a comunicação entre cliente e servidor e a política de escalonamento *SJF* contribuíram para uma melhor organização e performance do projeto. A introdução da política de escalonamento *FCFS* foi útil para verificar que, de facto, com *SJF* o tempo de espera médio é menor. No geral, o projeto proporcionou uma aprendizagem extensa e valiosa da aplicação dos conceitos aprendidos nas aulas e dos nossos conhecimentos em linguagem C e gerenciamento de sistemas em ambiente *Linux*.