Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Sistemas Operativos

Ano Letivo de 2023/2024

Orquestrador de Tarefas

Flávia Alexandra Silva Araújo (A96587)
Miguel Torres Carvalho (A95485)

5 de maio de 2024

SO

Resumo

» Atualizar o resumo

Índice

Índice de Figuras

1 Arquitetura do Serviço

1.1 Diagrama de Arquitetura do Serviço

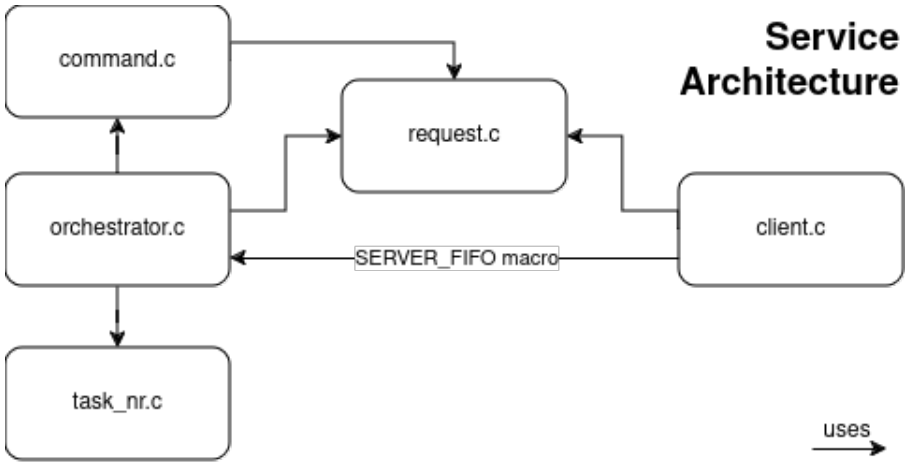

Figura 1.1: Arquitetura do Serviço

1.2 Descrição dos Módulos Desenvolvidos

1.2.1 Orquestrador (*orchestrator.c*)

O *orchestrator.c* é o módulo principal do serviço, sendo responsável por:

- Leitura e validação dos argumentos da linha de comandos;
- Criar o seu *FIFO* para receber pedidos dos clientes;
- Lidar adequadamente com os pedidos recebidos dos clientes;
- Caso o número máximo de tarefas em paralelo seja atingido, colocar os pedidos de execução em espera;
- Caso necessário, enviar as respostas dos pedidos dos clientes pelos seus respetivos *FIFOs*.

O orquestrador encarrega-se de quatro tipos de pedidos:

- ***EXECUTE*** - Pedido para executar um comando. Se o número máximo de tarefas em paralelo for atingido, o orquestrador coloca o pedido em espera, de seguida envia uma resposta ao cliente com o número da tarefa e o estado do pedido ("a executar"ou "agendada").
- ***COMPLETED*** - Assim que um comando terminar a sua execução, o orquestrador recebe um pedido deste tipo enviado pelo processo-pai do processo-filho que executou o comando, e execu-

tará, com base na política de escalonamento utilizada, o próximo comando em espera, se algum existir.

- ***STATUS*** - Pedido para consultar o estado das tarefas. O orquestrador utiliza um processo para gerar e enviar a resposta ao cliente. Deste modo, o servidor pode continuar a receber e lidar com pedidos de outros clientes, sem interrupções.

- ***KILL*** - Pedido para terminar o servidor. Quando o orquestrador recebe este pedido, termina a sua execução, porém, antes de terminar, guarda o número de tarefa atual, fecha os descritores de *FIFOs* abertos, e liberta a memória alocada para as tarefas em execução e em espera.

O orquestrador continua a correr até receber o pedido *kill* de um cliente, ou até receber um sinal do sistema operativo para interromper/terminar.

Para uma ilustração mais detalhada da comunicação entre o servidor e o cliente, por favor consulte os seguinte diagramas de comunicação nas figuras *1.2* e *1.3*, do subcapítulo *Diagramas de Comunicação entre Servidor e Cliente*.

1.2.2 Cliente (*client.c*)

O módulo *client.c* é responsável por, após a leitura e validação dos argumentos da linha de comandos, enviar pedidos - *execute*, *status* e *kill* - para o *FIFO* do servidor *orchestrator*, e por receber as respostas do servidor num *FIFO* criado para o efeito, no caso do pedido enviado ser do tipo *execute* ou *status*. O nome do *FIFO* do cliente é gerado através do seu PID, e é enviado no próprio pedido para o servidor.

1.2.3 Pedido (*request.c*)

Neste módulo, é definida a estrutura de dados que representa um pedido, constituída por:

- ***int type*** - Tipo do pedido (*EXECUTE*, *COMPLETED*, *STATUS* ou *KILL*);

- ***int est_time*** - Tempo estimado de execução da tarefa ou a prioridade da tarefa, dependendo da política de escalonamento;

- ***char command[MAX_CMD_SIZE]*** - Comando a executar, caso seja um pedido de execução;

- ***bool is_piped*** - Indica se o comando é encadeado, caso seja um pedido de execução;

- ***unsigned int task_nr*** - Número da tarefa, atribuído pelo orquestrador, caso seja um pedido de execução;

- ***char client_fifo[CLIENT_FIFO_SIZE]*** - Nome do *FIFO* do cliente, para onde o orquestrador enviará a resposta, caso seja um pedido de execução ou de estado.

Este módulo exibe garante o encapsulamento de dados, este é garantido através da definição da *struct request* dentro do ficheiro *request.c*, e a sua manipulação é feita através de funções definidas neste módulo, como por exemplo a função *create_request*, e as funções de *getters* e de *setters*.

Também a oferece outra funções de utlidade, como a função de clonagem de pedidos, e impressão de pedidos, utlizadas no orquestrador.

1.2.4 Comando (*command.c*)

O módulo *command.c* é chamado pelo orquestrador para executar os comandos enviados pelos clientes.

Este módulo é responsável por:

- Executar comandos, *single* ou *piped*, utilizando *process forking*, fazendo o respetivo *parsing* dos comandos;
- Lida com o redirecionamento do *stdout* e *stderr* para os respetivos ficheiros;
- Encarrega-se da a escrita do tempo de espera mais o de execução das tarefas para ficheiro;
- Escreve o número da tarefa, o comando executado e o tempo total para o ficheiro de histórico.

Primeiramente, todos os ficheiros necessários são criados - nomeadamente a diretoria de *output* para os ficheiros relativos a tarefa, como o ficheiro de *output*, o de erros e o do tempo total. Também prepara a abertura do ficheiro de histórico, onde serão escritos os dados como o número da tarefa, o comando executado e o tempo total, após a execução do comando.

De seguida, é criado um processo utilizado para criar outro processo-filho que executará o comando. O processo-pai aguardará que o seu processo-filho termine a execução do comando, e, após a sua execução, escreverá o número da tarefa, o comando executado e o tempo total, em milissegundos, para o ficheiro de histórico, e notifica o orquestrador que a tarefa foi concluída.

São criados dois processos para que o processo-pai interior possa esperar pela execução do comando, sem bloquear a execução do orquestrador, permitindo a este continuar a lidar com outros pedidos de clientes.

O redirecionamento do *stdout* e *stderr* para os respetivos ficheiros é feito através da utilização da função *dup2*.

No encadeamento de comandos, o redirecionamento é feito de forma a que o *stdout* do comando anterior seja redirecionado para o *stdin* do comando seguinte, e assim sucessivamente. Ao longo da cadeia de comandos é feito o redirecionamento do *stderr* para o ficheiro de erros, garantindo a similaridade com o comportamento nativo da *shell*. No último comando, o redirecionamento do *stdout* é feito para o ficheiro de *output*.

1.2.5 Número de Tarefa (*task_nr.c*)

O módulo *task_nr.c* proporciona funções para salvar e carregar o número de tarefa do respetiivo ficheiro com a designação definida pela *macro TASK_NR_FILENAME*, neste caso, "task number".

Se, ao carregar o ficheiro do número da tarefa, este não existir, o número da tarefa é inicializado a 1, caso contrário, o número da tarefa é incrementado em 1.

Este módulo é utilizado pelo orquestrador na sua inicialização para carregar o número da tarefa, e no seu término para o salvar.

1.3 Diagramas de Comunicação entre Servidor e Cliente

Nos seguintes três subcapítulos, são apresentados diagramas que ilustram a comunicação entre o servidor e o cliente, consuante diferentes tipos de pedidos.

1.3.1 Execução de Tarefas (*execute*)

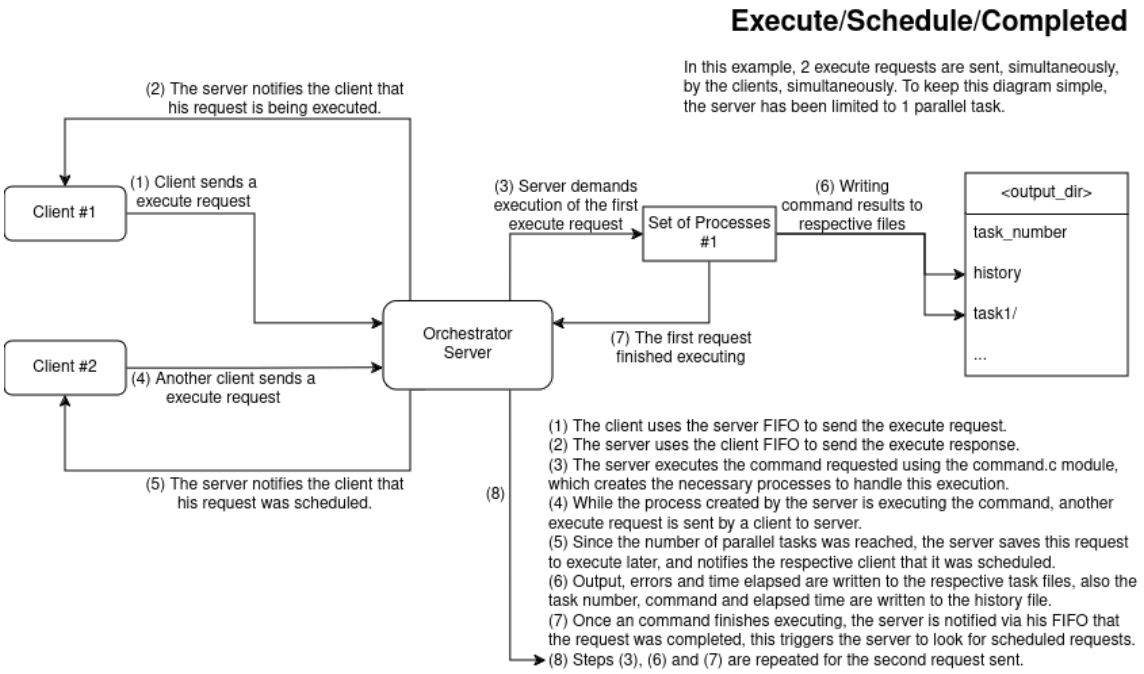

Figura 1.2: Diagrama de Comunicação para Execução de Tarefas

1.3.2 Estado de Tarefas (*status*)

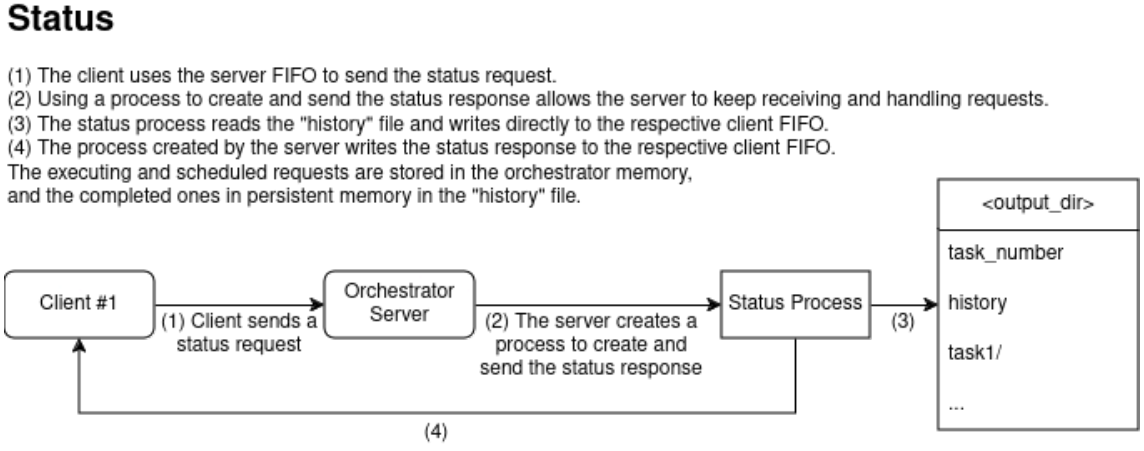

Figura 1.3: Diagrama de Comunicação para Estado de Tarefas

1.3.3 Terminação do Servidor (*kill*)

Na terminação do servidor, o cliente envia um pedido *kill* para o *FIFO* do servidor. De seguida o servidor guarda o número atual da tarefa de forma persistente, fecha os descritores de *FIFOs* abertos, liberta a memória alocada para as tarefas em execução e em espera, e, por fim, termina a sua execução.

2 Argumentos da Interface de Linha de Comandos

2.1 *Orchestrator*

2.2 *Client*

3 Avaliação de Políticas de Escalonamento

No trabalho desenvolvido, foram implementadas três políticas de escalonamento de tarefas. Estas serão aprofundadas nas secções seguintes, assim como as suas avaliações práticas. Por favor consulte o anexo [II] Implementação das Políticas de Escalonamento para mais detalhes de como foi realizada a implementação destas políticas.

» Atualizar a introdução da avaliação de políticas de escalonamento.

3.1 FCFS - *First Come First Served*

Nesta política, as tarefas são executadas por ordem de chegada, o que não é eficiente em termos de tempo de espera e execução. No entanto, é uma política simples e fácil de implementar.

» Escrever sobre a avaliação prática desta política + Imagem

3.2 SJF - *Shortest Job First*

A política *Shortest Job First* é uma política de escalonamento não preemptiva que seleciona a tarefa com o menor tempo de execução. Este tempo é passado como argumento na opção *execute* do cliente e repesenta uma estimativa do tempo que a tarefa demorará a ser executada.

Esta política é eficiente em termos de tempo de espera e execução, mas pode levar a situações de *starvation*, que ocorrem quando tarefas com tempos de execução maiores são sempre adiadas, e, num caso extremo, quando surgem constantemente tarefas com tempos de execução menores, estas de maior duração nunca serão executadas.

Existem várias maneiras de lidar com situações de *starvation*, como, por exemplo, a criação de uma especificação de tempo máximo de espera para cada tarefa, ou a definição de um número máximo de tarefas que podem ser executadas antes de uma tarefa com tempo de execução maior. Outra solução seria a implementação de uma política de escalonamento preemptiva, permitindo esta a interrupção de tarefas em execução para dar lugar a tarefas com tempos de execução menores.

» Escrever sobre a avaliação prática desta política + Imagem

3.3 PES - *Priority Escalonation Scheduling*

De forma a evitar situações de *starvation*, foi implementada a política *Priority Escalonation Scheduling*. Nesta, as tarefas são executadas de acordo com a sua prioridade, que é definida pelo cliente na opção *execute*, simliarmente como o tempo estimado que é passado como argumento na política *SJF*.

As tarefas com maior prioridade são executadas primeiro, evitando situações de *starvation*. As pri-

oridades ficam a critério do cliente, possibilitando uma maior versatilidade na execução de tarefas, permitindo ao cliente definir a prioridade adequada para cada tarefa que deseja executar. No entanto, esta flexibilidade não garante a eficiência da execução das tarefas, caso estas não sejam corretamente priorizadas.

» Escrever sobre a avaliação prática desta política + Imagem

4 Testes Desenvolvidos

5 Conclusão

Após o desenvolvimento deste serviço, foram cumpridos todos os objetivos propostos no enunciado do trabalho, desde a implementação de uma interface de linha de comandos tanto para o servidor como para o cliente - que permite a execução de tarefas do utilizador de forma assíncrona com a possibilidade de encadeamento de programas com *pipes* e a consulta do estado das tarefas -, assim como a implementação de um ficheiro *log* para guardar as tarefas executadas, bem como a implementação de um sistema com várias políticas de escalonamento e a avaliação prática destas. Também foi desenvolvido um conjunto de testes que permitem verificar o correto funcionamento do serviço.

Após a afirmação da possibilidade da terminação ou interrupção do servidor pelo professor regente, foi despertada a curiosidade de como o servidor poderia lidar com estas situações.

Num estudo apronfundado desta unidade curricular, uma solução seria a implementação de um sistema de interrupções no servidor, o que permitiria a este lidar com situações de terminação ou interrupção de forma mais eficiente através da utlização dos sinais do sistema operativo para lidar com estas situações, evitando a perda de tarefas agendadas ou em execução.

Para tal, seria necessário lidar com tarefas em execução, guardando o estado destas em memória, e permitindo a sua correta terminação ou interrupção, assim como para as tarefas em espera. Estas poderiam ser guardadas numa fila de espera, que seria consultada sempre que uma tarefa terminasse, permitindo a execução da próxima tarefa na fila.

Em suma, o desenvolvimento deste serviço permitiu a aplicação prática dos conhecimentos adquiridos ao longo do semestre, assim como a exploração de novas funcionalidades e conceitos, que proporcionaram a implementação de um serviço robusto e versátil - serviço este que propicia a execução de tarefas de forma assíncrona, com a possibilidade de encadeamento de programas e a consulta do estado das tarefas.

Anexos

[I] Documentação do Código Desenvolvido

Para uma versão mais interativa da documentação do código desenvolvido, por favor consulte o ficheiro ***doc/html/index.html*** localizado na diretoria raiz do projeto com o seu navegador de internet.

[II] Implementação das Políticas de Escalonamento