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 do relatório de acordo com as mudanças na segunda parte do trabalho.

Índice

Índice de Figuras

1 Arquitetura do Serviço

1.1 Diagrama de Arquitetura do Serviço

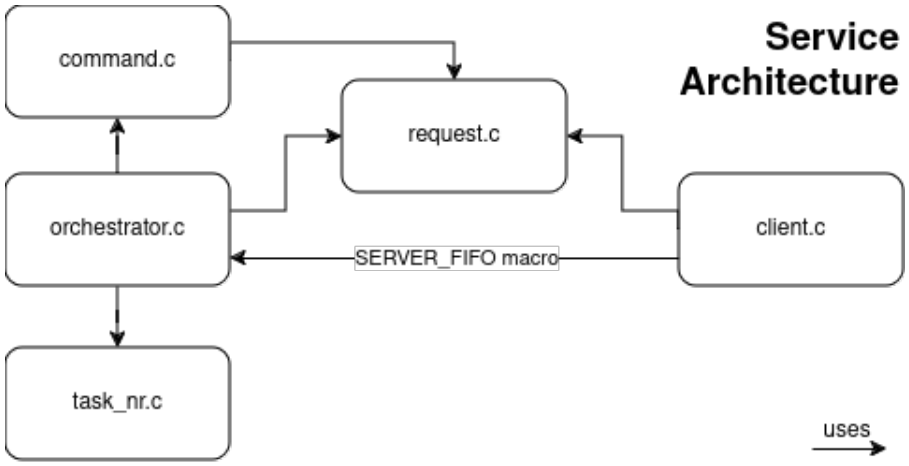

Figura 1.1: Arquitetura do Serviço

1.2 Descrição dos Módulos Desenvolvidos

1.2.1 Orquestrador (*orchestrator.c*)

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

- Fazer o *parsing* dos argumentos da linha de comandos;
- Criar o seu FIFO;
- Receber os pedidos dos clientes enviados pelo seu FIFO e lidar com estes adequadamente;
- Enviar as respostas dos pedidos dos clientes pelos seus respetivos FIFOs.

Este lida com os pedidos dos clientes tendo em conta a política de escalonamento definida previamente.

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 até que uma tarefa termine, enviando uma resposta ao cliente com o número da tarefa e o estado do pedido (em espera ou a executar).
- ***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 executará o próximo comando em espera - se existir.

- ***STATUS*** - Pedido para consultar o estado de um pedido. O orquestrador envia uma resposta ao cliente com o estado do pedido através do seu FIFO.
- ***KILL*** - Pedido para terminar o servidor. Quando o orquestrador recebe este pedido, termina o programa, porém, antes de terminar, guarda o número do último pedido de *execute* recebido num ficheiro, fecha os FIFOs 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.

1.2.2 Cliente (*client.c*)

O módulo *client.c* é responsável por:

- Executar comandos, verificar o *status* de tarefas e/ou terminar o servidor;
- Fazer *parsing* dos argumentos e enviar os pedidos para o servidor do programa através do FIFO do servidor.

As opções de *status* e *execute* requerem a criação de um FIFO para o cliente receber a resposta do servidor.

1.2.3 Pedido (*request.c*)

1.2.4 Comando (*command.c*)

O módulo *command.c* é responsável por:

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

Primeiramente, todos os ficheiros necessários são criados - nomeadamente a diretoria de *output* para os ficheiros necessários às tarefas, como o ficheiro de *output*, o ficheiro de *error* e o ficheiro de *time*. Também abre o ficheiro de histórico para escrita do número de tarefa, do comando executado e do respetivo tempo de execução.

De seguida, cria um processo-filho para executar o comando. O processo-pai aguardará que o seu processo-filho termine. Após a execução do comando pelo processo-filho, o processo-pai escreve o número da tarefa, o comando executado e o respetivo tempo de execução para o ficheiro de histórico. [CONTINUAR e falar do processo interior/exterior]

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

1.3.1 Execução de Tarefas (*execute*)

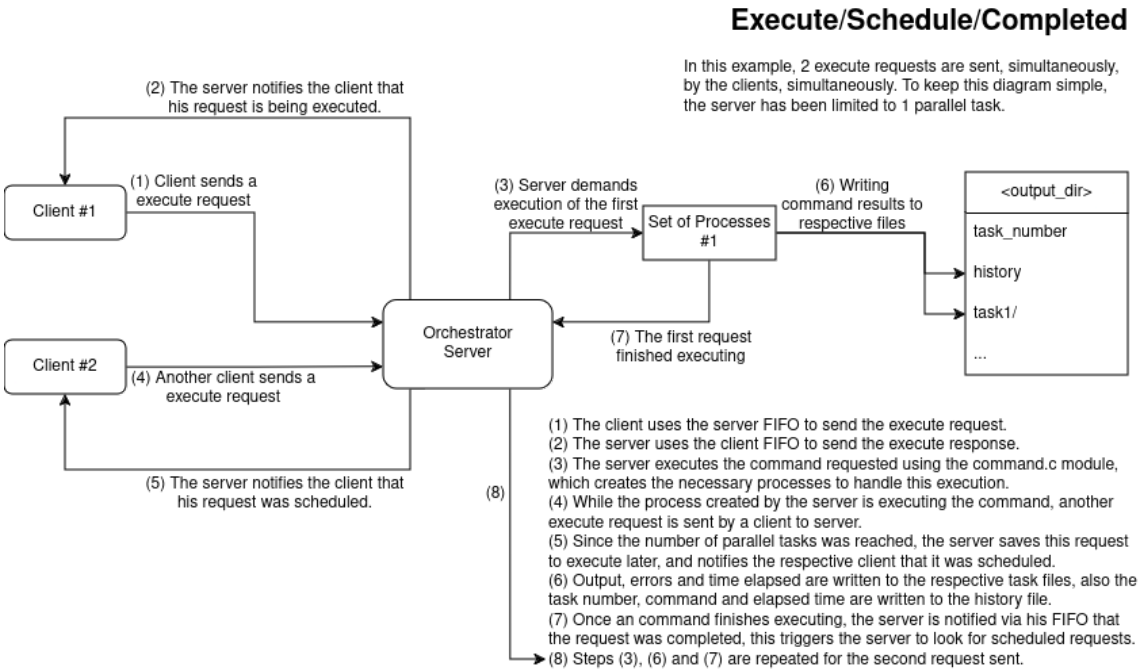

Figura 1.2: Diagrama de Comunicação para Execução de Tarefas

1.3.2 Estado de Tarefas (*status*)

1.3.3 Terminação do Servidor (*kill*)

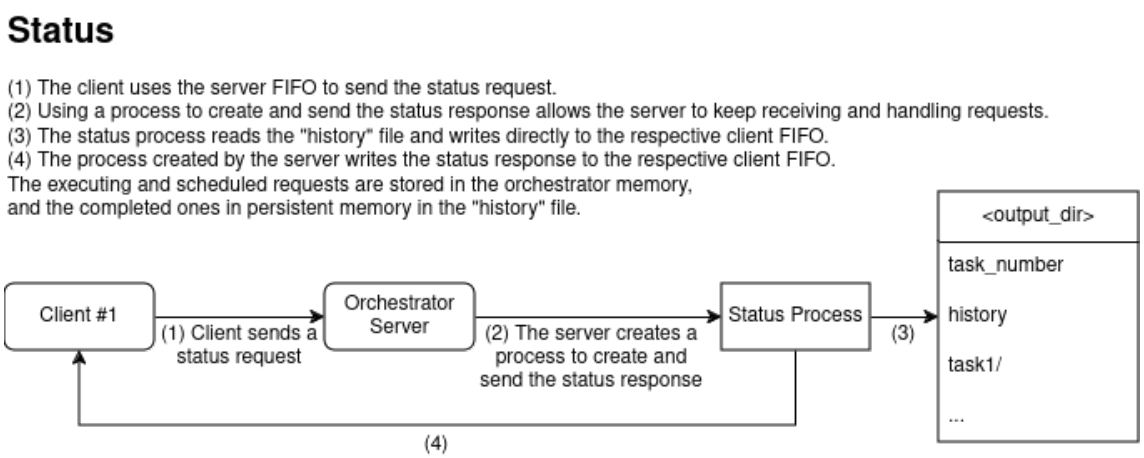

Figura 1.3: Diagrama de Comunicação para Estado de Tarefas

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