Universidade do Minho

Orquestrador de Tarefas

Eduardo Vilaça · Nuno André Ni · Rafael Cunha Costa · Grupo 34

Abstrato

Este documento serve de complemento informativo para o trabalho, orquestrador de tarefas, que foi realizado para assistir como elemento de avaliação para a disciplina de Sistemas Operativos do segundo ano da licenciatura em Ciências da Computação. Têm como objetivo detalhar o processo de criação do serviço de orquestração estilo cliente/servidor, mostrando quais as decisões tomadas para o bom funcionamento deste serviço e quais as funcionalidades que conseguimos implementar.

Comunicação Cliente/Servidor

Deste inicio tivemos a intenção de utilizar de *FIFO's* para podemos comunicar entre os dois programas, *client.c* e *orquestrator.c* respetivamente, aquando o orquestrador está a correr este entra num ciclo de procura de um FIFO especial, que é criado pelo cliente ao utilizar o comando *execute*, este tem uma *string* que contêm conteúdo relevante para a execução da tarefa, como por exemplo o seu PID, se se trata de um *execute* ou de um *status*, qual o comando a correr entre outros. Quando o servidor consegue ler algo deste FIFO significa que algum pedido chegou para processar. Logo em seguida é aberto um FIFO com um identificador para enviar uma mensagem, caso seja um *execute* é enviado um identificador de tarefa e o cliente trata de mostrar essa mensagem no *stdoutput* , se o pedido for um *status* é escrito no descritor um resumo sobre o estado das tarefas no momento e o cliente imprime para o *stdoutput* do utilizador.

Escalonamento e execução de Tarefas

1. Politicas de escalonamento

   A opção *execute* tem como objetivo dar ao utilizador a possibilidade de executar tarefas, dando a opção de processamento paralelo complementada com 3 algoritmos de escalonamento *first come first served, shortest job first, longest job first*. O método de escalonamento é definido no arranque do orquestrador, e vai alterar a forma de como as tarefas são inseridas numa *queue* de execução.

   Ao receber um pedido *execute* é criado uma estrutura tarefa que dependendo do escalonador selecionado no arranque é utilizada um método *enqueue* diferente de acordo com o algoritmo de cada um.

2. Execução de Tarefas (*Child Process*)

   Para a execução de tarefas, recorremos á utilização de *fork's* para incumbir a um *child process*, a tarefa de executar o comando pretendido, deixando o *parent process* livre para outras coisas.

   O cliente pode pedir dois tipos de tarefas, as tarefas únicas e as tarefas em pipeline. As tarefas em pipeline requereram a criação de vários redirecionamentos de descritores, que foram implementados na função execute_task2.

   Outra funcionalidade deste programa presente no enunciado era a cronometragem destas tarefas, logo optamos pela utilização de outro *fork*, foi utilizado como base a função *execute_task* fornecida nas aulas práticas da disciplina para executar o comando.

   O *child process* está encarregado de fazer o tratamento da string comando fornecida para servir como parâmetro de entrada num *execvp* que trata de executar o comando. Este *child process* faz também a criação do ficheiro de output da tarefa TASK_(Identificador), que funciona como *stdoutput* e *stderror* devido ao redirecionamento feito antes pela função *dup2*. O *parent process* é chamado mal o execvp é executado, e este guarda o momento em que é executada a tarefa, depois é feito um *wait* para esperar que o exec termine e é guardado o momento em que o *wait* termina. Ao utilizar a função *gettimeofday*, podemos depois obter um *long* em milissegundos através de uma formula e chegar ao tempo de execução do comando, retornando assim *fork* incial.

3. Tratamento da informação das tarefas (*Child Process*)

   Neste momento a estrutura de tarefa, encontra-se numa zona de memória diferente do que o processo pai, por isso é necessário comunicar ao pai que esta tarefa já foi executada e cronometrada. Para isso, abrimos um *fifo* com o identificador da tarefa com a cronometragem escrita lá dentro para avisarmos o orquestrador de que é preciso atualizar o estado da tarefa pois esta já se encontra concluída e a respetiva cronometragem pode ser mostrada ao utilizador no próximo status.

Gestão de Estados de Tarefas

Para solucionar a parte do *status* o programa está sempre á procura destes *fifo's* abertos anteriormente pelo *child process*, quando estes são encontrados o tarefa presente no *array* de executados, passa para o estado de completada e é lhe atribuída o tempo respetivo, podendo assim este *array* ser utilizado nos pedidos de *status.*
A função *display_status* agrupa a *queue* de execução e um *array* de executados que deixaram a *queue*, são utilizados vários *buffers* para organizar a informação conforme os diferentes estados das tarefas, depois é enviado para o descritor aberto com o identificador do cliente, de maneira semelhante ao *execute.*

Conclusões

Este trabalho foi no geral para todos os elementos do grupo interessante e enriquecedor, pois pudemos por em prática os conhecimentos que obtemos maioritariamente das aulas práticas. Como não encontramos muito conteúdo à cerca deste tema pelo menos neste contexto, dependemos muito dos *scripts* das aulas. Percebemos também que trabalhar com FIFO's não é propriamente fácil pois existem muitos fatores que podem influenciar no funcionamento expectável destes. Muitas vezes foi utilizada a ferramenta gdb para *debug* do código que consideramos algo que nos ajudou muito, mas a escassez de informação online sobre este tema trouxe uma dificuldade extra ao trabalho e percebemos que podíamos ter explorado melhor as soluções implementadas de forma a que estas fossem mais eficientes.