

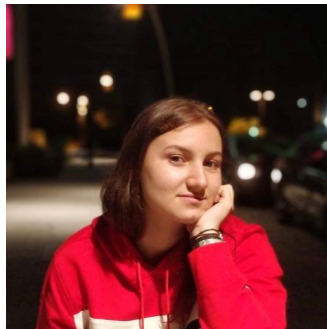
UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

Sistemas Operativos - Trabalho Prático  
Grupo 12

Sofia Santos (A89615)      Ana Filipa Pereira (A89589)  
Carolina Santejo (A89500)

Ano Letivo 2019/2020



# Conteúdo

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Introdução</b>                              | <b>3</b> |
| <b>2</b> | <b>Funcionalidades disponíveis no servidor</b> | <b>4</b> |
| 2.1      | Ajuda . . . . .                                | 4        |
| 2.2      | Executar . . . . .                             | 4        |
| 2.3      | Histórico . . . . .                            | 4        |
| 2.4      | Listar . . . . .                               | 5        |
| 2.5      | Output . . . . .                               | 5        |
| 2.6      | Terminar . . . . .                             | 5        |
| 2.7      | Tempo de execução . . . . .                    | 5        |
| 2.8      | Tempo de inatividade . . . . .                 | 5        |
| <b>3</b> | <b>Decisões tomadas</b>                        | <b>6</b> |
| <b>4</b> | <b>Conclusão</b>                               | <b>7</b> |

# Capítulo 1

## Introdução

Este projeto consistiu na criação de um serviço de execução de tarefas, no qual um cliente é capaz de enviar sucessivas tarefas a um servidor, para este as executar. Para além de executar tarefas, o servidor permite consultar tarefas em execução ou executadas previamente, consultar o output de cada tarefa, terminar manualmente tarefas, e ainda definir um tempo máximo de execução de cada tarefa ou tempo máximo de comunicação entre pipes.

Numa fase inicial, o maior desafio foi encontrar uma forma de poder encadear um número arbitrário de comandos sucessivos. Mais tarde, também tivemos alguma dificuldade em conseguir obter o estado de uma tarefa, mas acabámos por conseguir superar estes obstáculos.

O cliente e o servidor comunicam através de dois pipes com nome, um envia comandos do cliente para o servidor, e o outro envia o output dos comandos do servidor para o cliente.

## Capítulo 2

# Funcionalidades disponíveis no servidor

### 2.1 Ajuda

Este é o comando mais simples, que apenas envia para o cliente uma lista de todos os comandos existentes e de como os deve usar.

### 2.2 Executar

Comando central do servidor, permite executar uma ou mais funções encadeadas, enviando para um ficheiro *log* o seu output. Neste comando usamos um ou mais *execvp*, após fazer *parsing* do input, para executar cada função fornecida. Estas funções comunicam entre si através de pipes anónimos. Como não sabemos de quantos pipes uma tarefa irá precisar antes de terminar o *parsing* total do input, por uma questão de eficiência, decidimos dar a cada tarefa 32 pipes anónimos. Podíamos também ter usado alocação dinâmica de memória, para por um lado evitar usar memória desnecessariamente e por outro permitir a execução de comandos com mais de 32 pipes, mas acreditamos que, para além de que o número atual de pipes que temos não ter um grande peso em termos de memória, 32 já é um número mais que suficiente de pipes para uma tarefa.

### 2.3 Histórico

O servidor contém duas listas, uma que contém todas as tarefas executadas até ao momento, e outra que contém o estado da execução de cada tarefa. Este estado pode ser: em execução; concluída; terminada manualmente; terminada por tempo de execução; terminada por inatividade. O comando *historico* envia para o cliente uma lista contendo a informação destas duas listas.

## 2.4 Listar

Este comando é uma versão mais simples do comando *historico*, apenas envia para o cliente as tarefas do histórico que ainda estejam em execução.

## 2.5 Output

Todas as tarefas que terminam normalmente escrevem o seu output no ficheiro *log*, e é guardado no ficheiro *log.idx*, associada ao número da tarefa, a posição do byte final do output no *log*. O comando *output* lê o ficheiro *log* e envia para o cliente o output da tarefa especificada, começando a ler a partir do fim do output da tarefa anterior (posição 0 caso seja a primeira tarefa) até à posição indicada no ficheiro *log.idx*. Com este sistema apenas temos que armazenar o número da tarefa e uma posição num ficheiro auxiliar.

## 2.6 Terminar

Como uma tarefa pode gerar vários processos filhos, é necessário guardar os pids de todos eles, para os podermos terminar. Para isso usamos uma lista, que o comando *terminar* percorre, e para cada pid de um processo filho de uma dada tarefa usa a função *kill* para o terminar, usando o sinal *SIGTERM*.

## 2.7 Tempo de execução

Este comando permite-nos definir um tempo máximo durante o qual cada tarefa pode correr antes de ser terminada à força. O programa guarda numa lista o tempo de execução de cada tarefa, e a cada segundo verifica, caso a tarefa ainda esteja em execução, se esse tempo é superior ao tempo máximo de execução. Em caso afirmativo, essa tarefa é terminado pelo mesmo método do comando *terminar* e o seu estado atualizado. Caso não seja, o tempo de execução dessa tarefa é incrementado em um segundo. Esta sistema funciona usando a função *alarm*, que é chamada uma vez por segundo, e faz essa verificação.

## 2.8 Tempo de inatividade

Ao dar a flag *O\_ASYNC* aos pipes de comunicação das tarefas, estes enviam um sinal (*SIGIO*) quando passa informação por eles. Desta forma, somos capazes de saber quando é que houve comunicação num pipe anónimo. Este comando usa o mesmo método de medição de tempo que o comando *tempo-execucao*, mas este apenas interrompe uma tarefa se não tiver havido comunicação nos pipes dessa tarefa. O comando sabe isso através de uma variável que passa a ser zero sempre que há comunicação num dos pipes da tarefa, e que é incrementada a cada segundo. Desta forma, se chegar ao valor do tempo de inatividade, é porque não houve atividade durante aquele tempo.

## Capítulo 3

# Decisões tomadas

O nosso ficheiro de log, onde são armazenados os outputs das tarefas, é apagado sempre que o servidor é reiniciado. Decidimos fazer desta forma, mas seria muito simples fazer de outra forma, apenas fizemos assim por escolha e porque o enunciado não referia que método tínhamos que usar. Desta forma, cada execução do servidor irá produzir um novo ficheiro de log.

Decidimos que o nosso servidor suporta a execução de até 2048 tarefas. Podíamos ter usado alocação dinâmica de memória para suportar a execução de mais tarefas, mas sentimos que não seria necessário, já que, tendo em conta a simplicidade do programa, seria quase inconcebível executar mais que algumas centenas de tarefas, o histórico seria demasiado grande. Num projeto de mais larga escala, com um histórico navegável por páginas, por exemplo, isso já seria algo que implementaríamos.

## Capítulo 4

# Conclusão

Como é possível constatar, o nosso trabalho possui todas as funcionalidades pedidas a funcionar perfeitamente, para além da funcionalidade adicional. Desta forma, acreditamos que o nosso trabalho está bem conseguido. Tal como referimos no capítulo anterior, podíamos ter feito algumas coisas de forma diferente, mas da forma que fizemos funciona tudo bem.