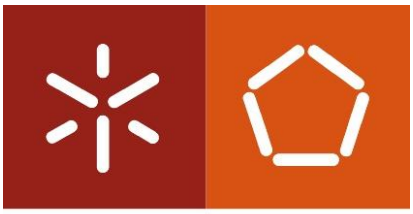

Universidade do Minho

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

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 um comando, 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

Comandos 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 um comando irá precisar antes de terminar o *parsing* total do input, por uma questão de eficiência, decidimos dar a cada execução do comando 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 um comando.

2.3 Histórico

O servidor contém duas listas, uma que contém todos os comandos executados até ao momento, e outra que contém o estado da execução de cada comando. Este estado pode ser: em execução; concluído; terminado manualmente; terminado por tempo de execução; terminado 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 os comandos do histórico que ainda estejam em execução.

2.5 Output

Todos os comandos que terminam normalmente escrevem o seu output no ficheiro *log*, e é guardado no ficheiro *log.idx*, associado ao número do comando, a posição do byte final do output no *log*. O comando *output* lê o ficheiro *log* e envia para o cliente o output do comando especificado, começando a ler a partir do fim do output do comando anterior (posição 0 caso seja o primeiro comando) até à posição indicada no ficheiro *log.idx*. Com este sistema apenas temos que armazenar o número do comando e uma posição num ficheiro auxiliar

2.6 Terminar

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

Capítulo 3

Conclusão