Universidade do Minho

Relatório

Sistemas Operativos

MIEI

2ºAno

2ºSemestre

Realizado pelo grupo:

Rúben Correia Cerqueira A89593

Júlio Miguel de Sá Lima Magalhães Alves A89468

Alexandra de Barros Reigada A84584

Índice

1. Introdução

No âmbito da unidade curricular de Sistemas Operativos, foi-nos proposto a realização de um trabalho prático que consiste na implementação de um serviço de monitorização de execução e de comunicação entre processos. Para este efeito, foi necessária a criação de um servidor e de um cliente, em que este último efetua pedidos e o servidor processa-os. Entre estes pedidos, está a possibilidade de executar e listar tarefas, apresentar o histórico e o output de tarefas terminadas e definir o tempo máximo de inatividade de comunicação de um pipe anónimo e o tempo máximo execução de uma tarefa.

2. Estrutura do projeto

2.1. Cliente

O cliente pode receber argumentos pela shell ou pela linha de comandos. Para este efeito, criamos uma função que interpreta os comandos recebidos na shell e uma função que interpreta comandos recebidos na linha de comandos.

Sempre que o utilizador submete um comando, é criado um processo para a validação e envio do comando através de um pipe com nome e outro processo para a recolha do resultado enviado pelo servidor através de um outro pipe com nome.

2.2. Servidor

O servidor recebe os pedidos do cliente enviados através de um pipe com nome e processa-os dependendo do tipo de pedido. Se forem pedidos de tempo de execução mais rápida, o servidor processa-os sequencialmente. Caso seja necessário executar alguma tarefa, este cria um processo filho encarregue de executá-la, tornando o servidor capaz de executar tarefas concorrentemente.

Para obter o número da tarefa seguinte a ser processada, é lido o ficheiro fileTarefa.txt que guarda o último número da tarefa executada e armazena numa variável global.

3. Funcionalidades

3.1. Tempo máximo de inatividade

Definimos uma variável global que guarda o tempo máximo de inatividade definido pelo cliente. Para detetarmos o tempo de inatividade entre pipes, decidimos criar um processo que apenas se encarrega da gestão da interligação dos comandos. Ou seja, este cria os processos necessários para executar cada comando e faz a interligação entre eles. Portanto, para detetarmos a inatividade de um pipe, colocamos esse processo a esperar por cada um dos filhos e colocamos um alarme com o tempo indicado pelo cliente. Com isto, se este processo não criar um processo filho que execute o próximo comando dentro do tempo limite do alarme, será porque o pipe estará inativo.

3.2. Tempo máximo de execução

Definimos uma variável global que guarda o tempo máximo de execução definido pelo cliente. Como referido anteriormente, é sempre criado um processo para a gestão da comunicação de comandos entre pipes. Portanto, foi apenas necessário medir o tempo de execução desse processo com o alarme no fim da criação deste.

Tal como para o tempo máximo de inatividade e para o tempo máximo de execução, quando este tempo é ultrapassado, é gerado um sinal de alarme (SIGALRM) que envia este sinal para todos os seus filhos e de seguida mata-os.

3.3. Executar

Cada vez que é pedido para executar uma tarefa, é criado um processo que fica responsável por esta e que gere o seu fluxo. Quando este termina, envia um sinal SIGUSR1 ao processo pai indicando que terminou e este recolhe esse sinal, retirando o processo da lista de processos em execução, recolhe o estado devolvido por esse processo e escreve no ficheiro de tarefas terminadas o número da tarefa, o modo como terminou e a respetiva tarefa.

O output da tarefa é escrito no ficheiro logs.txt através de um redireccionamento do output para ficheiro. Simultaneamente, é escrito no ficheiro log.idx o número da tarefa e a posição onde foi escrito o primeiro byte do output.

Por fim, é adicionada a informação da tarefa aos arrays. Para a mesma posição, no array de pids guardamos o pid do primeiro filho, no array de números de tarefas guardamos o número da tarefa (que é obtido a partir de uma variável global que guarda o número da tarefa seguinte) e no array de tarefas guardamos o comando da tarefa.

3.4. Listar tarefas em execução

Para listar as tarefas em execução, é impresso o conteúdo dos arrays de números de tarefas e de comandos, indicando assim o número de cada tarefa e o que executa.

3.5. Terminar tarefa

Quando um cliente pretende terminar uma tarefa, o sistema dá kill ao pid associado a essa tarefa que se encontra no array de pids e envia um sinal (SIGUSR2) que mata esse processo e os seus respetivos filhos. Por fim, na posição da tarefa terminada no array de pids fica o número -1 de forma a indicar que nessa posição não existe qualquer tarefa em execução.

3.6. Listar histórico

Esta função apresenta ao cliente o histórico de tarefas terminadas através da leitura do ficheiro tarefasTerminadas.txt que guarda o número da tarefa terminada, o modo como terminou e o comando que executou.

3.7. Apresentar ajuda

Esta função apresenta ao cliente os comandos que pode executar na linha de comandos.

3.8. Output

Esta função apresenta ao cliente o output da tarefa requisitada. Para este efeito, lemos o ficheiro log.idx que guarda o número da tarefa e a posição onde começa o seu output. Depois de encontrar a respetiva tarefa, lemos o ficheiro logs.txt da posição onde começa a tarefa até à posição onde começa a tarefa seguinte.

4. Conclusão

Em suma, conseguimos elaborar um projeto que implementa todas as funcionalidades propostas no enunciado prático, até mesmo a funcionalidade adicional.

Porém, à medida que fomos realizando este projeto, fomos sentindo dificuldades ao nível da comunicação entre processos e da gestão dos mesmos.

Gostaríamos também de ter implementado o redireccionamento do output (opções “<”, “>”, “<<”, “>>” da linha de comandos) mas não tivemos tempo para o fazer.

Dito isto, conseguimos com este projeto colocar em prática os conhecimentos adquiridos nas aulas práticas e teóricas e ultrapassar os obstáculos com que nos fomos confrontando com o desenvolvimento do trabalho.