Universidade do Minho
Escola de Engenharia

Licenciatura em Engenharia Informática
Maio 2023

Relatório do Trabalho Prático

Rastreamento e Monitorização da Execução de Programas

Sistemas Operativos

Ano Letivo 2022/2023

a100066, Ricardo Miguel Queirós de Jesus
a100659, Rui Pedro Fernandes Madeira Pinto
a100758, Hugo Arantes Dias
Grupo 69

Índice

1 Introdução

No âmbito da cadeira de Sistemas Operativos, o grupo desenvolveu um programa em C que pretende implementar um serviço de monitorização de programas executados numa máquina. Para tal, foi necessário desenvolver um cliente (programa *tracer*) que oferece ao utilizador uma interface através da linha de comandos, e um servidor (programa *monitor*) para interação com o cliente. Estes programas comunicam via pipes com nome.

O servidor comunicam através de dois pipes com nome, um em cada sentido. Segundo o enunciado apresentado, o trabalho encontra-se divido em funcionalidades básicas e funcionalidades avançadas.

Como funcionalidade básicas o programa deverá suportar a execução de programas do utilizador através da opção *execute -u* com o nome do programa a executar com os respetivos argumentos.

O cliente e o servidor comunicam através de dois pipes com nome (fifo), um envia informação do cliente para o servidor (nomeadamente o pid do programa, o nome do mesmo e os tempos inicial e final), e o outro do servidor para o cliente (usado apenas no comando status). Além disso, usamos uma flag para identificar que informação o servidor vai ler o que comando vai executar.

2 Funcionalidades disponíveis

O nosso projeto suporta as seguintes funcionalidades:

2.1 Execução de programas do utilizador

Tendo em conta os argumentos passado no *standard input*, através da chamada *"execute -u"*, o cliente executa os programas e os respetivos argumentos passados pelo utilizador através da função *execvp*. O programa tem a capacidade se saber se é um comando "simples"ou com pipes. No caso de ser apenas um comando, o programa cria um procesos filho usando a função *fork()* e executa o comanod usando a função *execvp()*

Dando o argumentos *"execute -u"* e entre aspas uma função com os respetivos argumetos, o programa vai executar a função *"execvp*. Nesta, ele vai começar por efetuar um *"fork"*, de modo a executar o *"execvp"* no processo filho, mas antes disso o cliente vai comunicar com o servidor, via o pipe *"communication_fifo_TM"*, uma flag, para o servidor saber o que é que o cliente está a executar, o seu respetivo pid,o nome da função que vai executar e o tempo a que está a começar a executar. Do lado do servidor, este vai primeiro identificar a flag enviada, e depois recolher toda a informação para uma struct *"package"* e guardar a mesma numa lista ligada, vai tambem colocar na struct um *"0"* no parâmetro estado, para indicar que o processo ainda não foi terminado. No final da execução da função o Cliente comunica ao Servidor que já terminou,através de uma flag, e diz qual é o pid do processo que terminou e o tempo a que acabou, o Servidor vai procurar na lista ligada o struct *"package"* com aquele pid e vai calcular o tempo que o processo demorou, comunicando o mesmo ao Cliente.

2.2 Consulta de programas em execução

Dando como argumento o *"status"* o utilizador recebe os processos que estão em execução naquele momento. Para isto, o cliente começa por enviar a respetiva flag pelo pipe com nome(*fifo*), do outro lado o *servidor* identifica a flag e vai percorrer a lista ligada, em todos as estruturas *"package"* com o estado a *"0"* ele vai guardar o seu pid, nome de processo e calcular quanto tempo passou desde do início da execução daquele processo, e vai concatenar estes dados todos numa só string. No final de percorrer a lista, o *servidor* envia a string ao *cliente*, o qual vai imprimir ao utlizador através do *standard output*.

2.3 Execução encadeada de programas

Relativamente à simulação de pipeline, começamos por dividir os comandos a serem executados em sequência, através do delimitador | , de forma a possibilitar a nossa função *"execCommand"* lidar com a com a execução dos comandos individualmente.

Nós criamos um array de i pipes(sendo i o número de processos a executar -1), de modo a conseguir fazer uma cadeia de execução em que o output de um pipe seja o input de outro. Nesta cadeia de pipes nós categorizamos cada pipe dentro de uma de 3 possíveis categorias que são, o pipe inicial, o qual únicamente precisa de fechar a leitura do pipe *"i"* e fazer um *"dup2"* de modo a que o seu output vá para o pipe *"i"*, o pipe final, que precisa de dar *"close"* à escrita do pipe anterior e efetuar um *"dup2"* de modo a que o seu input venha do pipe *"i - 1"* e os pipes intermédios, os quais são uma junção do pipe incial e o pipe final, estes vão ter de dar *"close"* da escrita ao pipe *"i - 1"* e da leitura do pipe *"i"* e através de *"dup2* fazer com que o input seja dado pelo pipe *"i - 1"* e o output vá para o pipe *"i"*.

3 Decisões Tomadas

Decidimos no *"Servidor"* haver a função *"calculate_time"* que calcula o tempo de execução, isto para para evitar haver maior fluxo de comunicação entre o *"Servidor"* e o *"Cliente"*. Neste projeto nós optamos por guardar as informações relativas aos processos a realizaos em structs numa lista ligada, para identificar se já estão concluídas ou em execução nós temos uma parâmetro *"estado"*, no qual é *"0"* caso o processo esteja em execução e *"1"* caso já esteja concluído. Reconhecemos que a melhor opção seria retirar da lista ligada os processos terminados.

Além disso, no cliente temos optamos por criar uma função chamada *"execCommand"*, que recebe o comando a ser executado em forma de string e processa-o de forma a poder ser usado na função *"execvp"*.

4 Conclusões

Inicialmente, começa-mos

O nosso projeto, no que diz respeito às funcionalidades básicas, é capaz executar programas do utilizador, informar o programa servidor da execução dos mesmos, enviando as suas informações, nome e *timestamp*, e após essa execução o cliente tem a capacidade de infromar o servidor do mesmo, enviado o seu PID e o seu *timestamp*. No nosso trabalho prático concluimos com sucesso a consulta de programas em execução.

Durante este projeto, coloca-mos em prática os conceitos aprendidos durante as aulas ao longo deste semestre, através da realização das tarefas a nível prático, como a criação de processos, execução de programas, etc.
Além disso, adquirimos capacidades de criar soluções mais eficientes para resolver os problemas que enfrentamos ao longo da realização deste trabalho.