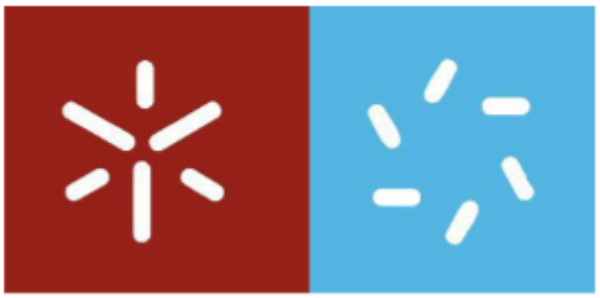

Universidade do Minho
Licenciatura em Ciências da Computação

Sistemas Operativos-Trabalho Prático
Grupo 12

Rastreamento e Monitorização da Execução de Programas

Francisco José Pereira Teófilo
A93741

Maria Filipa Veiga Luso Rodrigues
A97536

Tiago Emanuel Lemos Teixeira
A97666

Maio, 2023

Índice

1. Introdução

Ao longo do 2º semestre foi proposto, no âmbito da unidade curricular de Sistemas Operativos, o desenvolvimento de um sistema que seja capaz de monitorizar programas que são executados numa máquina. Esta monitorização tem como objetivo um levantamento de dados estatísticos sobre os programas já terminados, como por exemplo, o tempo de execução de um conjunto de programas, ou mesmo sobre os programas que se encontram em execução.

No decorrer deste relatório, vamos fundamentar a nossa abordagem ao problema sugerido, justificando a estrutura do nosso programa e demonstrar os conhecimentos anteriormente adquiridos nas aulas, tais como a criação e utilização de pipes com e sem nome, duplicação de descritores, criação, utilização e execução de processos e system calls.

Em seguida, iremos descrever as funcionalidades que o programa suporta e depois faremos uma caracterização do nosso código, referindo a arquitetura de processos e os mecanismos de comunicação que utilizamos.

No final do desenvolvimento do programa, o nosso grupo almeja conseguir que todas as funcionalidades (básicas e avançadas) estejam bem implementadas e totalmente operacionais.

2. Funcionalidades Básicas

Como nos foi pedido no enunciado, foram desenvolvidos dois programas - *monitor.c* e *tracer.c* - que funcionarão para ler do terminal os repetitivos pedidos.

Destes, o programa mais rico em código será o *tracer.c* que simula o comportamento de um cliente, enquanto que o *monitor.c* simula o comportamento de um servidor.
Entendamos então, quais são as funcionalidades presentes neste projeto:

- *./tracer execute -u "prog-a arg-1 … arg-n"* - executa o programa que se encontra entre aspas.

- *./monitor* - executável que suportará os pedidos vindos dos clientes e enviará informação para os tais, para obter o resultado esperado.

- *./tracer status* - mostrará no terminal do cliente os programas que, até ao momento, estão a correr.

2.1 Programa *tracer.c*

Aqui explicaremos em detalhe as decisões tomadas acerca do programa *tracer.c,* no que toca a funcionalidades básicas.

Funções auxiliares presentes neste programa:

- long int gettime() : retorna o tempo em milissegundos desde a inicialização do programa;

- int count_param(char* string): retorna o número de parâmetros (divididos por espaços) na string;

- void listar_arguments(char* token,char** l_strings): retorna uma lista de strings (cujo último elemento é "NULL"), usando o token como divisor da string em questão;

No decorrer do programa, colocamos numa variável "*pipe1*", o descritor do pipe *"monitor_ler",* de onde o servidor vai receber informações mandadas pelo cliente.

Execução de *execute -u*

Após a seleção do modo "*execute -u*", temos como variáveis importantes à disposição:

- ***prog*** - string com o comando e respectivos argumentos, que posteriormente é reutilizada para guardar o nome do comando;
- ***token*** - string onde será guardada a repartição por espaços do argv[3];
- ***cmd*** - lista de string que será o resultado da função auxiliar "*listar_arguments*";
- ***fd*** - array de descritores que constituem um *pipe* anónimo;
- ***pipe2*** - descritor do pipe "*monitor_escrever*".

Sendo assim, cria-se dois processos (pai e filho), que terão características distintas entre si

Processo filho

O processo filho é responsável pelo envio do seu pid via ***fd[1]*** para poder ser futuramente utilizado pelo processo pai - no início do processo o descritor de leitura é fechado.

Envia-se também, através de ***pipe1***, uma mensagem que será interpretada pelo servidor. Essa mensagem é constituída pelo caracter 'f', que denota que a mensagem é proveniente do processo filho do cliente, assim como o respetivo pid, nome do comando e o tempo inicial da execução do programa (***tempo_i***).

Destaca-se ainda que, o valor do tempo inicial de execução do programa é calculado o mais breve possível em relação à sua execução, tendo em conta que foi feito mesmo antes do envio da mensagem.

A seguir ao envio da mensagem, é finalmente executado, recorrendo há função "*execvp*", dando uso às variáveis ***prog*** e ***cmd*** como argumentos, e fecha-se também o descritor de escrita de ***fd***.

Processo pai

Este processo começa por esperar pelo seu descendente e fechar o descritor de escrita. Assim que essas instruções acabam, é calculado o valor do tempo final da execução do programa (***tempo_f***) dado no argumento ***argv[3]***.

Depois, lê do descritor de leitura ***fd[0]*** o valor do pid do processo filho. Esse valor, em conjunto com elementos como, caracter 'p' e ***tempo_f***, formam uma mensagem que será enviada novamente para o servidor.

Realizado o envio, é declarada a variável **pipe2**, de onde se vai ler informação acerca do tempo de execução do programa em milissegundos, que é posteriormente escrito para o standard output. Por fim. tanto o descritor de leitura de ***fd*** como o ***pipe2***, são fechados.

Execução de *status*

Após a seleção do modo "*status*", temos como variáveis importantes à disposição:

- ***agora*** - tempo em milissegundos até àquele instante;
- ***pipe2*** - conforme explicado anteriormente;
- ***bufstatus*** - buffer que guarda informação vinda do servidor.

Após o cálculo da variável ***agora***, é enviado ao servidor a mensagem cujos constituintes são, o caracter 's' para denotar que é proveniente do status e do valor do ***agora***.

No ***pipe2*** é devolvido informação escrita pelo servidor que é por sua vez escrita para o standard output.

Execução de *kill*

Aqui simplesmente envia pro servidor um caracter 'k'.

2.2 Servidor

No nosso servidor, com nome de monitor, encontramos a necessidade de criar uma estrutura, de modo a este armazenar informação sobre os diferentes programadas que cada cliente irá executar e, posteriormente e a pedido de um cliente, a disponibilizar para consulta. Como a informação que é necessária armazenar sobre cada programa é o PID do processo que executa o program, o nome do programa que irá ser executado pelo cliente e a sua duração (em ms), criámos uma estrutura com estes três elementos. De forma a guardar esta informação de forma organizada decidimos que faríamos desta estrutura uma lista ligada, pois desta forma é mais fácil o seu acesso.
Numa tentativa de tornar o código presente na main menos pesado e mais legível escrevemos três funções auxiliares, sendo elas as seguintes:

1. A write_struct que vai alocar espaço na lista ligada que é dada como argumento de modo a acrescentar mais um elemento da estrutura programas à lista com os respetivos valores que estão presentes no argumento dado e denominado de linha. Como essa linha vai ser da forma "f PID nome tempo_i", é necessária a sua desconstrução pelos espaços de forma a ser possível o acesso à respetiva informação correspondente aos diferentes valores da estrutura.

2. A read_pid que vai ter como argumentos uma linha e uma lista ligada denominda de temp. A linha vai ser da forma "p PID tempo_f" e esta vai ser desconstruída de modo a ser possível saber qual é o número do PID e, em seguida, vai ser procurada na lista temp. Esta procura tem como objetivo a atualização da variável duracao, do programa com o PID dado em linha, de forma a esta corresponder à duração no programa e não o tempo inicial, como inicialmente.

3. A remove_terminados que tem como argumentos duas listas ligadas da estrutura programas, nomeadas de terminados e running, e um PID. Esta função tem como objetivo retirar o elemento da lista runing cujo PID é o mesmo que o dado como argumento e adicioná-lo no final da lista terminados.

Entrando agora na explicação da main, temos a criação de dois FIFOS, onde o primeiro que é criado é o modo de comunicação entre o cliente (tracer) e o servidor (monitor), ou seja, neste FIFO a escritura é função do tracer e o monitor vai, somente, ler a informação compartilhada pelo tracer, tal como o seu nome indica (monitor_ler). O segundo, monitor_escreve, faz a comunicação de forma contrária, ou seja, neste FIFO a responsabilidade de escrita é do monitor e a de leitura é do tracer. De seguida procede-se à

abertura do FIFO monitor_ler em modo read only, para ser possível o monitor ler a informação por lá passada, e em modo write only, esta abertura realiza-se para ser possível a permanência do estado ativo do monitor. O descritor da primeira abertura é denomida por leitura e o descritor da segunda é designada por manter.

Posteriormente são criadas variáveis que serão úteis no decorrer do monitor. Sendo estas:

1. Um buffer para onde vai ser lida toda a informação passada pelo tracer.
2. Um inteiro onde vai ser colocada a quantidade de bytes que vão ser lidas do buffer anteriormente mancionado.
3. Uma lista ligada da estrutura programas denominada de running e inicializada a NULL que vai ser onde vão ser armazanadas as informações dos programas que se encontram a decorrer.
4. Uma lista ligada da estrutura programas denominada de terminados e inicializada a NULL que vai ser onde vão ser armazanadas as informações dos programas que já terminaram.

De forma a uma leitura continuada da informação fornecida pelo tracer realiza-se um ciclo onde a condição de paragem é a não leitura de bytes ou caso dessa mesma leitura ocorra um erro. Como a escrita no FIFO monitor_ler vai ser uma instrução por linha (cada instrução termina com \n) então vai ser necessária a desconstrução do que for lido por \n. Esta desconstrução é feita através da função strtok, sendo então implementado outro ciclo de modo a percorrer todas as instruções que forem lidas de uma só vez.

De forma a ser possível o monitor distinguir essas instruções, pois cada instrução diferente requer respostas do monitor diferentes, é utilizado o primeiro caractere de cada linha escrita pelo tracer. Caso esse caractere for:

1. f, a resposta do monitor é a atualização da lista running de forma a adicionar a informação de um novo programa que se encontra a ser executado pelo tracer (informação fornecida no resto da instrução), utilizando a função write_struct. Nota-se que esta inicial vai ter como variável correspondente à duracao o tempo em que este programa começou a ser executado pelo tracer.

2. p, a resposta do monitor é a chamada da função read_pid para atualizar o valor da duração do respetivo PID dado pelo tracer, sendo de seguida enviado esse novo valor ao tracer pelo FIFO monitor_escrever, tendo sido aberto em modo de write only, para o tracer poder informar o utilizador da duração da execução do programa. De seguida invoca a função remove_terminados de forma a atualizar a lista de terminados e de running. Como já não ser preciso escrever mais nenhuma informação para o tracer, este fecha o descritor de escrita para o FIFO.

3. s, o monitor abre um desscritor de escrita para o FIFO monitor_escrever, pois este vai precisar de passar a informação por ele guardada ao tracer sobre todos os programas que estão a decorrer. Desta forma, o monitor percorre a lista running e realiza a diferença entre o tempo fornecido pelo tracer e o tempo inicial dos vários programas. De seguida envia essa informação, tal como o respetivo PID e nome, para o FIFO monitor_escrever de forma a esta ser posteriormente lida pelo tracer e fornecida, por este, ao utilizador, sendo

que cada linha enviada corresponde a um tracer diferente. Após ter fornecido toda a informação de todos os programas que ainda estão a decorrer o monitor fecha o descritor de escrita, pois já não vai ser mais usado.

4. k, significa que o cliente pretende fazer com que o servidor deixe de estar disponível para receber novos pedidos de clientes (tracers). Deste modo, o monitor sai de ambos os ciclos e fecha os descritores que tem abertos e os respetivos FIFOS que tem abertos. Esta funcionalidade foi adicionada por nós por uma questão de praticalidade, pois se não houvesse esta opção o monitor ficaria em constante funcionamento, sendo necessário forçar a sua terminação, algo que não é a melhor prática.

3. Conclusão

No decorrer da realização do trabalho prático, embora a equipa docente considerasse que o mesmo era simples, sentimos uma grande necessidade de organização para que conseguíssemos que o trabalho fosse realizado no tempo correspondido. A aplicação dos conceitos teóricos mostrou-se extremamente desafiante.

A resolução dos guiões práticos mostrou-se extremamente vantajosa, pois foi um apoio extremamente valioso para o esclarecimento de dúvidas e dificuldades que tivemos.

Sendo assim, sentimos que com este trabalho conseguimos enriquecer o nosso conhecimento acerca da matéria lecionada e ainda desafiar-nos a nós mesmos.