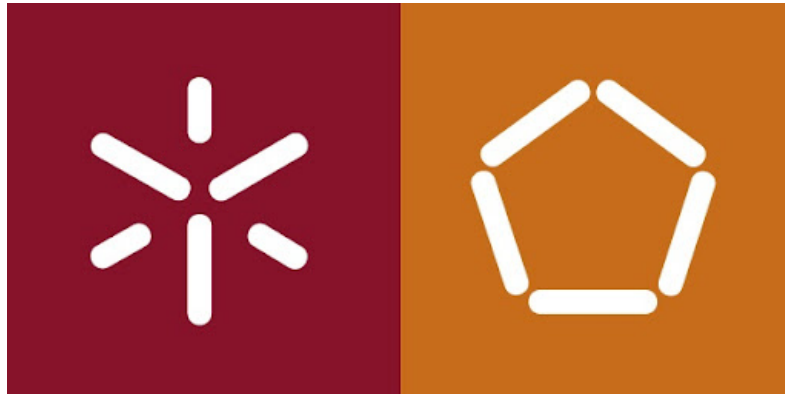


Universidade do Minho

Escola de Engenharia



Rastreamento e Monitorização da Execução de Programas

Sistemas Operativos

Relatório do Trabalho Prático

Grupo nº50

Guilherme Rodrigues do Outeiro Cunha Marques A94984

13 de maio de 2023

Índice

Introdução	3
Estratégias de Implementação	3
Servidor (monitor)	3
Cliente(tracer)	4
Funcionalidades Básicas	5
Execução de programas do utilizador	5
Consulta de programas em execução	6
Servidor	6
Funcionalidades Avançadas	6
Execução encadeada de programas	6
Armazenamento de informação sobre programas terminados	6
Consulta de programas terminados	7
Melhorias Futuras	7
Considerações Finais	7
Anexos	8

Introdução

O presente relatório descreve o projeto de implementação de um serviço de monitorização de programas em execução numa máquina, utilizando a linguagem de programação C e pipes com nome para comunicação entre processos. O objetivo deste projeto é permitir que os utilizadores possam executar programas através de um cliente com interface de linha de comando, obtendo o tempo total de execução dos mesmos.

Além disso, deve-se ser capaz de consultar todos os programas em execução e seus respetivos tempos de execução, bem como obter estatísticas sobre programas já terminados.

Este relatório descreve a implementação detalhada do cliente e do servidor, bem como os desafios encontrados e as soluções adotadas. Também são apresentados exemplos de uso do serviço e sugestões para trabalhos futuros.

Estratégias de Implementação

Servidor (monitor)

O servidor recebe pedidos dos clientes usando pipes com nome e cria um novo processo para cada pedido recebido usando a função *fork()*.

A função *fork()* cria um novo processo para cada pedido recebido. Dessa forma, cada pedido pode ser processado independentemente e em paralelo com outros pedidos, evitando que clientes que realizam pedidos que exigem mais tempo de processamento possam bloquear a interação de outros clientes com o servidor.

A função *process_request(char *request)* processa todos os pedidos do cliente que são recebidos, fazendo a distinção entre pedidos através de *if/else* e um *strcmp* do *request* recebido.

Quando o request é “*start*” o servidor processa as mensagens de início de execução de um programa do cliente. Quando o servidor recebe uma mensagem de início de execução de um programa do cliente (*start*), este extrai as informações necessárias da mensagem (PID do processo, nome do programa e timestamp atual) e guarda essas informações num arquivo cujo nome corresponde ao PID do processo, mostra como o servidor armazena em arquivos o estado de programas terminados. É também introduzido num array de uma struct todas as informações do processo, incluído o facto de este ainda se encontrar a ser corrido.

Quando o servidor recebe uma mensagem de fim de execução de um programa do cliente (*end*), ele carrega as informações de início de execução do programa a partir do arquivo correspondente e calcula o tempo de execução total do programa. Em seguida, o servidor guarda as informações de fim de execução do programa no arquivo correspondente. É também introduzido num array de uma struct todas as informações do processo, incluído o facto de este já ter terminado a sua execução.

Quando o request é *“status”* é feita uma iteração sobre o array de todos os processos onde é feita uma análise daqueles que se encontram a correr que são por sua vez enviados para o fifo do cliente a partir do qual é imprimido no standard output do tracer.

Quando o request é *“stats-time”* é feita uma iteração sobre o array de todos os processos a partir do qual é calculado o tempo total dos processos mencionados no request como argumentos. Este é então enviado para o fifo do cliente a partir do qual é imprimido no standard output do tracer.

Quando o request é *“stats-command”* é feita uma iteração sobre o array de todos os processos a partir do qual é calculado o número de vezes que foi executado um certo programa num dado conjunto de processos. Este é então enviado para o fifo do cliente a partir do qual é imprimido no standard output do tracer.

Quando o request é *“stats-uniq”* é feita uma iteração sobre o array de todos os processos a partir do qual é escrita no fifo do cliente a lista de nomes de programas diferentes, contidos na lista de PIDs passada como argumento.

Cliente(tracer)

O cliente processa todos os pedidos do utilizador, realizados pelo terminal, que são recebidos, fazendo a distinção entre pedidos através de *if/else* e um *strcmp* dos argumentos, após o *./bin/tracer*, recebidos.

Ao receber o argumento *“execute”* o tracer faz por sua vez a distinção se a flag que acompanha esse argumento é *“-u”*, execução de um só programa, ou *“-p”*, execução de uma pipeline. No caso da flag ser *“-u”* no cliente são guardadas em variáveis locais o nome do programa assim como dos seus argumentos, caso os tenha, e executa a função *execute_program(char *program, char **args)*. Nesta função realiza-se um *fork()* no qual o processo filho é responsável por executar o programa passado como argumento através da função *“execvp”*, por sua vez o processo pai envia as informações do pid para o *Standard Output*. O pai é também responsável por enviar a informação *“start”* (pid, program, início do programa) do programa a ser corrido para o pipe do servidor. O pai espera (*wait*) que o filho acabe a execução do programa e após isso envia a mensagem de fim com o tempo que a execução demorou em milissegundos assim como envia a informação de *“end”* para o pipe do servidor de maneira a que o monitor possa escrever as informações do processo no seu ficheiro (denominada pelo pid deste mesmo processo). Por fim, este procede ao fecho dos ficheiros e pipes abertos.

No caso da flag que acompanha o argumento ser *“-p”*, seguida pela pipeline entre aspas tal como demonstrado no enunciado, o tracer guarda numa variável local a pipeline e procede à execução da função *execute_pipeline*. A pipeline é passada como um argumento para a função. O programa usa a função *“strtok”* de maneira a dividir a string da pipeline numa matriz de programas. De seguida, este cria um conjunto de pipes, um para cada par de programas consecutivos, a partir daí usa a função *fork* para criar um processo filho para cada programa na pipeline. Para cada processo filho, o programa redireciona a entrada padrão do processo filho para o pipe do processo pai anterior (exceto para o primeiro processo filho), e redireciona o standard output do processo filho para o pipe do próximo processo filho (exceto para o último processo filho), com recurso à função *“dup2”*.

Depois de todos os processos filhos serem criados, o programa espera que todos os processos filhos terminem antes de enviar uma mensagem de término (*end*) para o servidor.

O tracer pode também receber o argumento "*status*" que, comunica com um servidor por meio de dois fifos: *SERVER_PIPE_NAME* e *CLIENT_PIPE_NAME*. O cliente envia uma solicitação de "*status*" ao servidor abrindo o pipe do servidor, formatando uma mensagem no buffer e gravando-a no pipe do servidor. De seguida, ele abre o canal do cliente para leitura e aguarda uma resposta do servidor. Quando os dados estão disponíveis para leitura no canal do cliente, este lê-os no *buffer* e envia-os para o standard output .

O cliente pode também receber o argumento "*stats-time*" que, comunica com um servidor por meio de dois fifos: *SERVER_PIPE_NAME* e *CLIENT_PIPE_NAME*. O cliente envia uma solicitação de "*stats-time*" assim como os seus argumentos ao servidor abre o pipe do servidor, formata uma mensagem no *buffer* e grava-a no pipe do servidor. De seguida, este abre o canal do cliente para leitura e aguarda uma resposta do servidor. Quando os dados estão disponíveis para leitura no canal do cliente, este lê-os no *buffer* e envia-os para o standard output.

O cliente pode também receber o argumento "*stats-command*" que comunica com um servidor por meio de dois *fifos*. O cliente envia uma solicitação de "*stats-command*" assim como os seus argumentos ao servidor abrindo o pipe do servidor, formatando uma mensagem no *buffer* e grava-a no pipe do servidor. De seguida, este abre o canal do cliente para leitura e aguarda uma resposta do servidor. Quando os dados estão disponíveis para leitura no canal do cliente, este lê-os no *buffer* e envia-os para o standard output.

Por fim, o cliente pode também receber o argumento "*stats-uniq*" que comunica com um servidor por meio de dois pipes com nome. O cliente envia uma solicitação de "*stats-uniq*" assim como os seus argumentos ao servidor abre o pipe do servidor, formata-o numa mensagem no *buffer* e grava-a no pipe do servidor. De seguida, este abre o canal do cliente para leitura e aguarda uma resposta do servidor. Quando os dados estão disponíveis para leitura no canal do cliente, este lê-os no *buffer* e envia-os para o standard output.

Funcionalidades Básicas

Execução de programas do utilizador

O cliente é responsável por executar programas dos utilizadores através da opção "*execute -u program args(caso se aplique)*" e comunicar ao servidor o estado dessa execução.

Enquanto é realizada a execução de um novo programa do utilizador, o cliente informa o servidor do mesmo, enviando o PID do processo, o nome do programa e o timestamp atual.

O cliente notifica o utilizador, via standard output, do PID do programa que será executado. Após notificar o utilizador e o servidor, o cliente executa o programa.

Quando o programa termina, o cliente informa o servidor do mesmo, enviando o PID do processo que terminou e o timestamp atual, que representa o tempo imediatamente após

o término da execução do programa. O cliente também notifica o utilizador, via standard output, do tempo de execução (em milissegundos) utilizado pelo programa.

Após vários testes foi possível averiguar que esta funcionalidade se encontra bem implementada e completamente funcional.

Consulta de programas em execução

Através da opção status são listados, um por linha, os programas em execução no momento. A informação de cada programa contém o seu PID, nome, e tempo de execução até ao momento (em milissegundos).

O processamento para dar resposta à interrogação anterior é realizado pelo servidor com o cliente apenas sendo responsável por enviar o pedido e receber a resposta, apresentado-a ao utilizador.

Devido a um erro na atualização do array criado para armazenar a informação relativa a todos os processos esta funcionalidade encontra-se implementada porém não funcional.

Servidor

O servidor suporta, sempre que possível, o processamento concorrente de pedidos, evitando que clientes, ao realizar pedidos que obriguem a um maior tempo de processamento, possam bloquear a interação de outros clientes com o servidor.

Após vários testes foi possível averiguar que esta funcionalidade se encontra bem implementada e funcional.

Funcionalidades Avançadas

Execução encadeada de programas

Através da opção “*execute -p*”, seguida por um argumento que representa uma pipeline entre aspas, o cliente suporta a execução encadeada de programas do utilizador (pipelines) tal como acontece na linha de comandos quando usado o operador |.

Após vários testes foi possível averiguar que esta funcionalidade se encontra bem implementada e completamente funcional.

Armazenamento de informação sobre programas terminados

O servidor armazena em ficheiros o estado de programas terminados. Existe um ficheiro diferente por programa executado. O nome deste ficheiro corresponde ao PID do programa em questão. O ficheiro contém o nome e tempo de execução total (em milissegundos) do programa.

Após vários testes foi possível averiguar que esta funcionalidade se encontra bem implementada e funcional.

Consulta de programas terminados

Através do comando `stats-time` é impresso no standard output o tempo total (em milissegundos) utilizado por um dado conjunto de programas identificados por uma lista de PIDs passada como argumento.

Recorrendo ao comando `stats-command` é impresso no standard output o número de vezes que foi executado um certo programa, cujo nome é passado como argumento, para um dado conjunto de PIDs, também, passados como argumento.

Usando o comando `stats-uniq` é impresso no standard output a lista de nomes de programas diferentes contidos na lista de PIDs passada como argumento.

Tal como aconteceu com a funcionalidade básica `status`, devido a um erro na atualização do array criado para armazenar a informação relativa a todos os processos, estas funcionalidades encontram-se implementadas porém não funcionais.

Melhorias Futuras

Caso houvesse mais tempo disponível, poderiam ser executadas algumas melhorias no programa. Tais como:

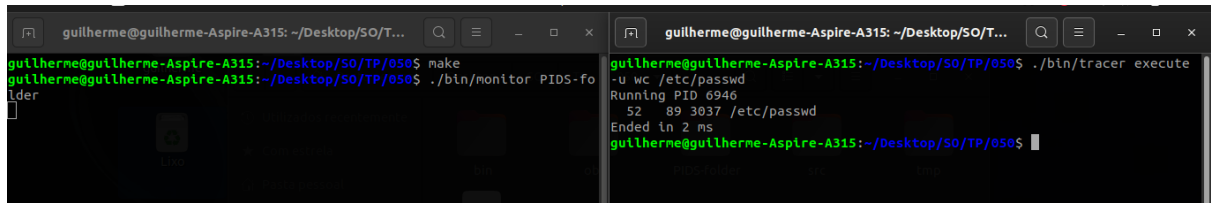
- Resolver o problema, que existe com o array da struct definida, de maneira a que fosse possível realizar as funcionalidades corretamente;
- Remover o loop infinito que existe no servidor de maneira a que este seja mais eficiente.

Considerações Finais

Neste projeto apresentam-se os conceitos de sistemas operativos lecionados durante o semestre, as suas funções, os tipos e exemplos. A realização deste trabalho tornou possível melhorar os conhecimentos relativos à cadeira de Sistemas Operativos, assim como da linguagem de programação C.

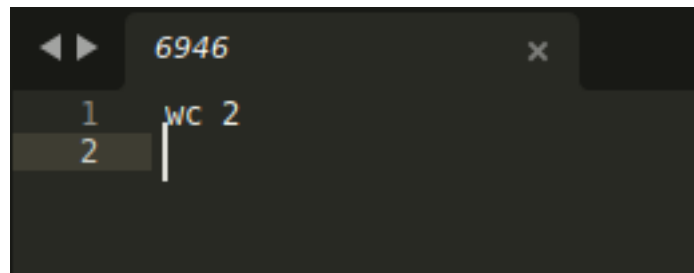
O resultado final não é perfeito e tem alguns erros e limitações que poderiam ser melhorados em trabalhos futuros. No entanto, considero que foi um bom desafio e uma boa oportunidade de aprendizagem, tendo em conta que foi feito por um grupo de apenas um elemento.

Anexos



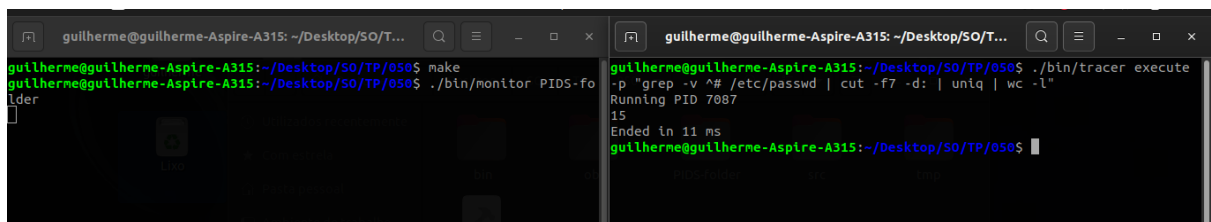
```
guilherme@guilherme-Aspire-A315: ~/Desktop/SO/TP/050$ make
guilherme@guilherme-Aspire-A315: ~/Desktop/SO/TP/050$ ./bin/monitor PIDS-fo
guilherme@guilherme-Aspire-A315: ~/Desktop/SO/TP/050$ ./bin/tracer execute
-u wc /etc/passwd
Running PID 6946
52 89 3037 /etc/passwd
Ended in 2 ms
guilherme@guilherme-Aspire-A315: ~/Desktop/SO/TP/050$
```

Figura 1 - Execução do comando “./bin/tracer execute -u wc /etc/passwd”



```
6946
1 wc 2
2
```

Figura 2 - Informação armazenada da execução do programa da Figura 1.



```
guilherme@guilherme-Aspire-A315: ~/Desktop/SO/TP/050$ make
guilherme@guilherme-Aspire-A315: ~/Desktop/SO/TP/050$ ./bin/monitor PIDS-fo
guilherme@guilherme-Aspire-A315: ~/Desktop/SO/TP/050$ ./bin/tracer execute
-p "grep -v ^# /etc/passwd | cut -f7 -d: | uniq | wc -l"
Running PID 7087
15
Ended in 11 ms
guilherme@guilherme-Aspire-A315: ~/Desktop/SO/TP/050$
```

Figura 3 - Execução do comando “./bin/tracer execute -p “grep -v ^# /etc/passwd | cut -f7 -d: | uniq | wc -l””