Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Unidade Curricular de Redes Definidas por Software

Ano Letivo de 2022/2023

Rastreamento e Monitorização da Execução de Programas

João Mendes, a93256 Cláudia Silva, a93177 Rui Alves, a93252

13 de maio de 2023

SO

Data de Receção	
Responsável	
Avalição	
Observações	

Rastreamento e Monitorização da Execução de Programas

João Mendes, a93256 Cláudia Silva, a93177 Rui Alves, a93252

13 de maio de 2023

Resumo

Área de Aplicação: Sistemas Operativos

Palavras-Chave: tracer, monitor

Índice

Lista de Figuras

Capítulo 1

Introdução

No âmbito da unidade curricular de Sistemas Operativos, desenvolvemos, em C, um serviço de monitorização de programas executados em uma máquina, visando fornecer aos usuários informações precisas sobre o tempo total de execução. Através de um cliente, os usuários têm a capacidade de executar programas e obter dados relevantes sobre o tempo gasto durante a execução. Além disso, um administrador de sistemas pode consultar todos os programas em execução e verificar o tempo de cada um por meio de um servidor.

Neste relatório, iremos apresentar o funcionamento e a estrutura específica que nos baseamos-nos para o desenvolvimento das funcionalidades básicas e avançadas, bem como uma breve explicação acerca das mesmas.

Por fim, iremos apresentar alguns testes realizados para demonstrar o funcionamento do programa.

Capítulo 2

Estruturas de Dados

De forma a armazenar todos os programas que se encontram em execução houve a necessidade de criar uma estrutura de dados denominada Program.

```
typedef struct program{
    pid_t pid;                  // ID do processo
    char program_name[50];      // Nome do programa
    struct timespec start_time;        // Timestamp de início do programa
    struct timespec end_time;          // Timestamp de fim do programa
}Program;

Program programs[MAX\_PROGRAMS];
```

Esta struct contém o pid, o nome do programa que está em execução, o tempo que o programa iniciou e o tempo a que o programa terminou. Todos os programas são guardados num array que contém um máximo de programas definido por nós como 30.

Capítulo 3

Funcionalidades Básicas

Dada a necessidade de existir comunicação entre o nosso **tracer** e o **monitor** foram utilizados pipes com nomes. Neste caso, utilizamos 2 pipes, *monitor_to_tracer* e *tracer_to_monitor*, uma vez que estes são unidirecionais.

3.1 Tracer

3.1.1 Execução de programas

O utilizador através do seguinte input:

```
./tracer execute -u <nome\_do\_programa> <arg1> <arg2> ...
```

consegue executar os comandos. Para tal, é criado um processo filho, onde será executado o comando recorrendo à função *execvp* sobre um array que armazena todos os seus argumentos. Relativamente ao processo pai, será responsável por enviar informação sobre este novo programa para o monitor através dos fifos, é enviada uma primeira mensagem do tipo 1, que possui o nome_programa, pid_programa e o tempo_inicial informando o monitor da execução de um novo programa, e uma segunda mensagem do tipo 2, que possui pid_programa e o tempo_final, informando o monitor do término do respetivo programa. Estas mensagens são distinguidas através de números sendo que 1 sinaliza a mensagem de inicio da execução de um programa e 2 o fim de um programa.

Por fim, o tracer mostra quanto tempo demorou a execução do programa, efetuando um print para o *stdout*.

3.1.2 Comando Status

O utilizador através do comando:

```
./tracer status
```

Pergunta quais são os programas que ainda se encontram em execução (status, 3). Posteriormente, os programas que se encontrarem em execução são imediatamente enviados pelo fifo para o tracer de novo, onde este escreve no ecrã o pid_programa, o nome_programa e o tempo_a_ser_executado. Caso não existam programas a executar, essa informação também é enviada através dos fifos.

3.2 Monitor

De modo a inicializar o monitor executa-se o seguinte comando :

```
./monitor <path folder dos pids>
```

No monitor, de forma a facilitar o *debugging*, são impressas diversas frases consoante a informação que é enviada e recebida.

3.2.1 Execução de programas

Uma vez que o monitor recebe a mensagem com o número 1 (nome_programa, pid_programa, tempo_inicial) proveniente do tracer essa informação é guardada num array que contem estruturas de dados anteriormente explicada. Quando o programa termina o monitor recebe uma mensagem 2 (pid_programa, tempo_final) proveniente do tracer alterando o tempo do fim da estrutura de dados para zero.

3.2.2 Comando Status

Quando o cliente efetua este comando o tracer envia, através dos fifos, uma mensagem com o número 3 e com o status, desta maneira quando o monitor recebe esta mensagem, acede ao array onde são guardados todos os programas, que foi abordado anteriormente. Desta maneira, é possível averiguar quais são os programas que se encontram em execução, uma vez que os programas que se encontram em execução contém um $End_timeigualazero$.

Capítulo 4

Funcionalidades Avançadas

Nesta secção iremos abordar as funcionalidades avançadas que decidimos elaborar e explicar o desenvolvimento das mesmas.

4.0.1 Execução encadeada de programas

Para elaboração desta funcionalidade, o utilizador através do seguinte input*:* execute -p <comando> | <comando1> | <comando2> ... | <comandoN-1> | <comandoN> *. De seguida, iremos abordar o código realizado:*

```
[...]
char **commands = malloc((argc - 3) * sizeof(char *));
int i = 0;
for (int a = 3; a < argc; a++) {
    commands[i] = strdup(argv[a]);
    i++;
}

char **exec_args = malloc((i + 1) * sizeof(char *));
i = 0;
char *command = strdup(commands[0]);
char *string;
while ((string = strsep(&command, "|")) != NULL) {
    exec_args[i] = strtrim(string);
    i++;
}
exec_args[i] = NULL;
[...]
```

No código acima apresentado, primeiramente armazenamos na variável commands *os comandos que aparecem a seguir ao "./tracer execute -p". Consequentemente, na variável exec_args*

armazenamos os diferentes comandos que serão executados. Para tratar dos programas encadeados, baseamos-nos em pipes anónimos, por isso definimos uma variável ***int pipes[number_of_numbers-1][2]****, onde a variável de number_of_numbers corresponde ao numero máximo de comandos a executar e, por isso, tem de ser o número máximo menos 1. De seguida, para o seu funcionamento baseamos-nos na imagem que se segue:*

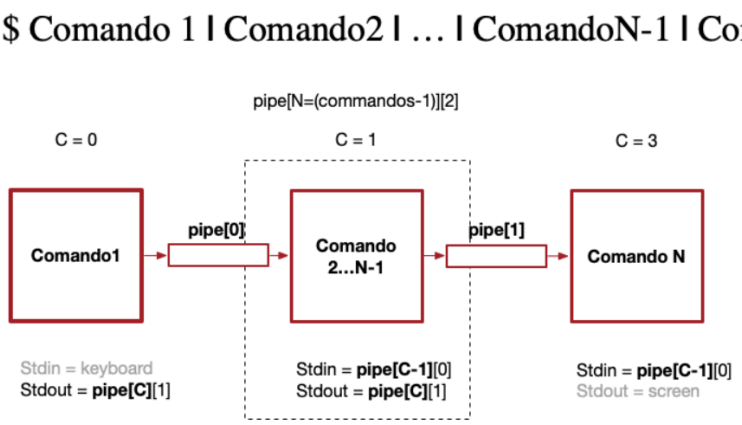

Figura 4.1: Funcionamento da execução de programas encadeados

Assim sendo, criamos um processo filho responsável por cria uma cópia de um descritor para outro descritor que segue. Para além disso, executará os comandos armazenado no array exec_args, como foi referido anteriormente, a partir da função auxiliar execute_commands. *Já o pai, enviará a informação sobre este novo programa para o monitor através dos fifos. Assim sendo, é enviada uma mensagem do tipo 7, que contem o nome_programa, o pid_programa e o tempo_inicial, onde o pid obtido é o pid do primeiro programa, que irá informar o monitor da execução desse novo programa. De seguida, envia outra mensagem do tipo 2 com o pid_programa w o tempo_final informando o monitor do fim do respetivo programa.*

4.0.2 Armazenamento de informação sobre programas terminados

Para a realização desta funcionalidade, primeiramente criamos a pasta com o nome de "PIDS_folder", onde irão ser adicionados ficheiros com o seguinte formato "PIDS_folder/pid.txt"contendo o tempo decorrido para cada programa terminado. Estes ficheiros são criados no monitor aquando da receção de uma mensagem de término de um programa.

4.0.3 Consulta de programas terminados

- ***stats-time***
  Através do comando stats-time,*tipo de mensagem 4, é impresso no tracer o tempo decorrido para um dado conjunto de programas que já foram executados, identificados por uma lista de* pids *passada como argumento do comando. Inicialmente percorre todos os programas que se encontram guardados no array de structs e quando encontra os pids que foram passados no argumento o tempo que elas demoram a executar é somado e ,posteriomente, devolvido para o tracer.*

Capítulo 5

Testes-exemplo

De seguida, iremos mostrar alguns exemplos de testes utilizados para mostrar o funcionamento nosso programa.

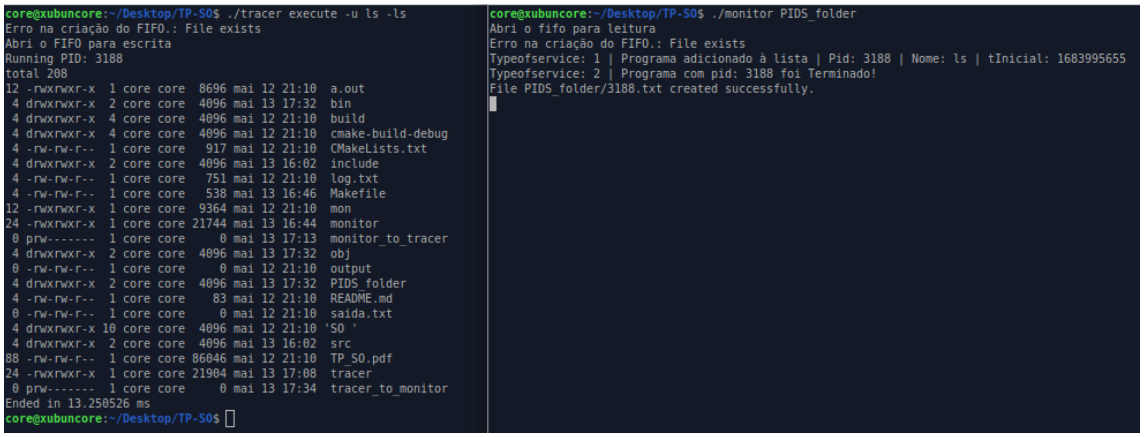

Figura 5.1: ./tracer execute -u programa arg1

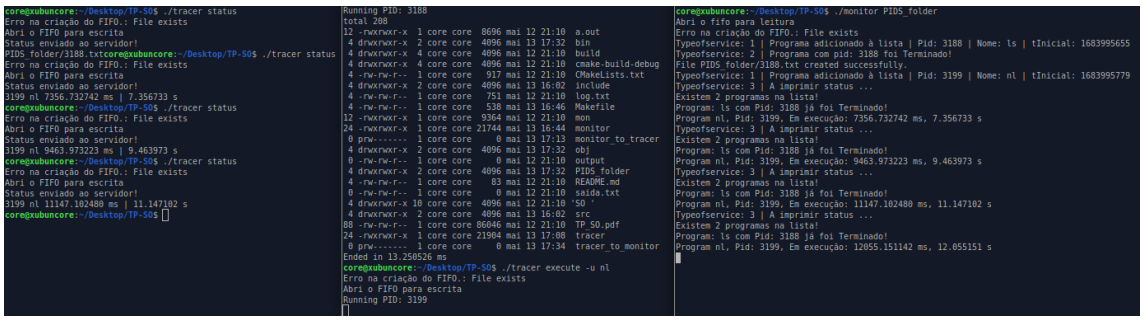

Figura 5.2: ./tracer status

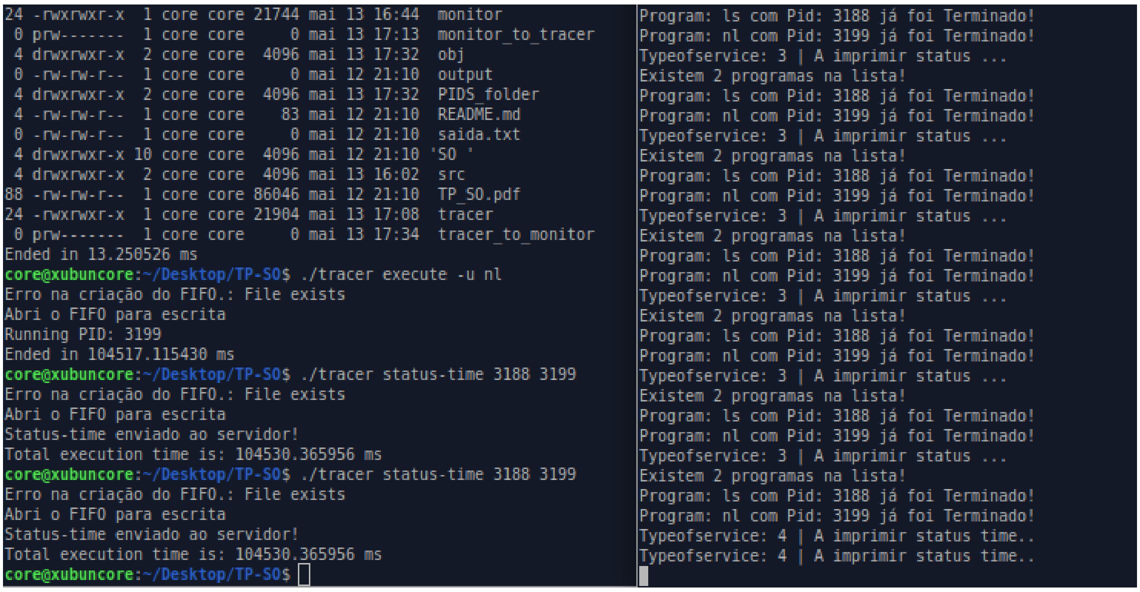

Figura 5.3: ./tracer status-time PID1 PID2

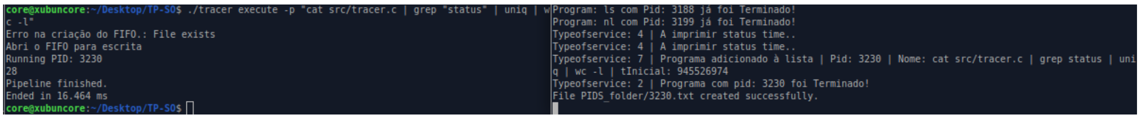

Figura 5.4: ./tracer execute -p "programa_encadeado"

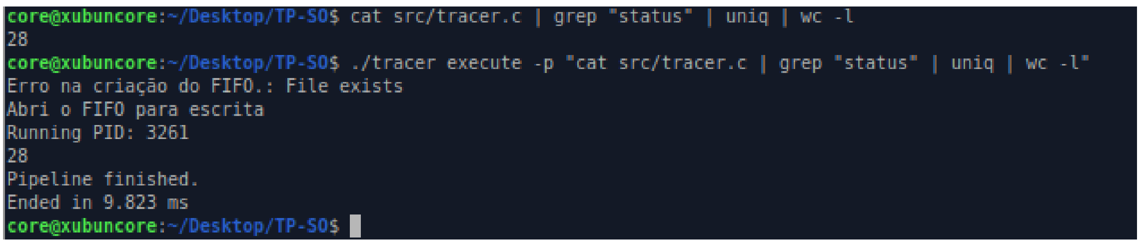

Figura 5.5: Verificação se o resultado obtido está correto

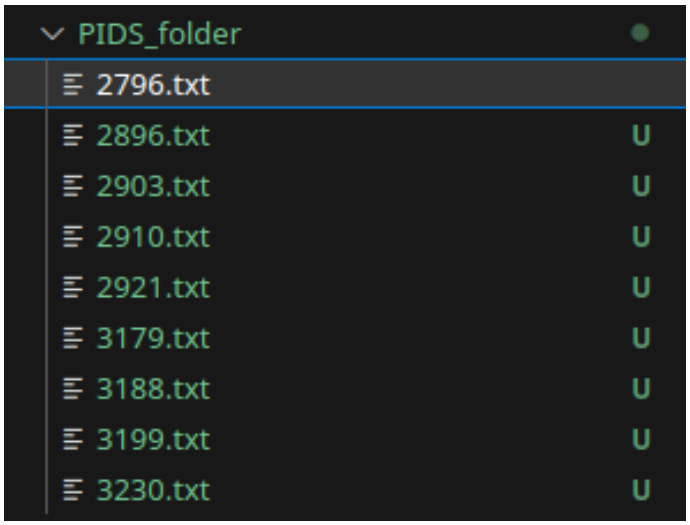

Figura 5.6: Files criados com o nome do PID do programa

Capítulo 6

Conclusão

Este trabalho prático consistiu na implementação de um serviço de monitorização dos programas executados. Através do cliente (tracer), os utilizadores conseguem executar programas e obter o seu tempo de execução e, para além disso, através do servidor (monitor) é possivel consultar os programas que se encontram em execução e as estatisticas sobre os programas que ja terminaram.

Em retroespetiva, consideramos que este projeto foi-nos útil para aprofundar e consolidar o conhecimento adquirido nas aulas.

Contudo, apresentamos algumas dificuldades, primeiramente, a perceber o funcionamento dos fifos com nome, para saber a quantidade de fifos necessários para serem criados. Depois, manifestamos algumas dificuldades na realização das funcionalidades avançadas.

Por fim, constatamos que conseguimos implementar as funcionalidades básicas presentes no enunciado, bem como, algumas funcionalidades avançadas. Para um trabalho futuro, poderíamos tentar abordar outras funcionalidades avançadas indicadas no enunciado, bem como, aprimorar o código e a estrutura desenvolvida.

Concluindo, apesar de todas as dificuldades encontradas, estamos satisfeitos com a realização deste projeto que espelha o desafio entregue por esta UC.