



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	
Avaliação	
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

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Estruturas de Dados</b>	<b>2</b>
<b>3</b>	<b>Funcionalidades Básicas</b>	<b>3</b>
3.1	Tracer . . . . .	3
3.1.1	Execução de programas . . . . .	3
3.1.2	Comando Status . . . . .	4
3.2	Monitor . . . . .	4
3.2.1	Execução de programas . . . . .	4
3.2.2	Comando Status . . . . .	4
<b>4</b>	<b>Funcionalidades Avançadas</b>	<b>5</b>
4.0.1	Execução encadeada de programas . . . . .	5
4.0.2	Armazenamento de informação sobre programas terminados . . . . .	6
4.0.3	Consulta de programas terminados . . . . .	7
<b>5</b>	<b>Testes-exemplo</b>	<b>8</b>
<b>6</b>	<b>Conclusão</b>	<b>11</b>

# Lista de Figuras

4.1	Funcionamento da execução de programas encadeados . . . . .	6
5.1	./tracer execute -u programa arg1 . . . . .	8
5.2	./tracer status . . . . .	8
5.3	./tracer status-time PID1 PID2 . . . . .	9
5.4	./tracer execute -p "programa_encadeado" . . . . .	9
5.5	Verificação se o resultado obtido está correto . . . . .	9
5.6	Files criados com o nome do PID do programa . . . . .	10

# 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 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 início 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\_time igual a zero*.

# 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 número 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:

\$ Comando 1 | Comando2 | ... | ComandoN-1 | Comando N

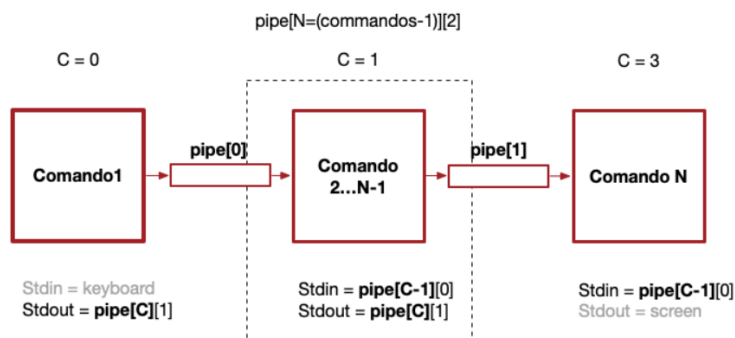


Figura 4.1: Funcionamento da execução de programas encadeados

Assim sendo, criamos um processo filho responsável por criar 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` e 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 ,posteriormente, devolvido para o tracer.*

# Capítulo 5

## Testes-exemplo

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

```
core@xubuncore:~/Desktop/TP-S05$ ./tracer execute -u ls -ls
Erro na criação do FIFO.: File exists
Abriu o FIFO para escrita
Running PID: 3188
total 208
12 -rw-rw-r-- 1 core core 8696 mai 12 21:10 a.out
4 drwxrwxr-x 2 core core 4096 mai 13 17:32 bin
4 drwxrwxr-x 4 core core 4096 mai 12 21:10 build
4 drwxrwxr-x 4 core core 4096 mai 12 21:10 cmake-build-debug
4 -rw-rw-r-- 1 core core 917 mai 12 21:10 CMakeLists.txt
4 drwxrwxr-x 2 core core 4096 mai 13 16:02 include
4 -rw-rw-r-- 1 core core 751 mai 12 21:10 log.txt
4 -rw-rw-r-- 1 core core 530 mai 13 16:46 Makefile
12 -rw-rw-r-- 1 core core 9364 mai 12 21:10 mon
24 -rw-rw-r-- 1 core core 21744 mai 13 16:44 monitor
0 prw----- 1 core core 0 mai 13 17:13 monitor_to_tracer
4 drwxrwxr-x 2 core core 4096 mai 13 17:32 obj
0 -rw-rw-r-- 1 core core 0 mai 12 21:10 output
4 drwxrwxr-x 2 core core 4096 mai 13 17:32 PIDS_folder
4 -rw-rw-r-- 1 core core 83 mai 12 21:10 README.md
0 -rw-rw-r-- 1 core core 0 mai 12 21:10 saida.txt
4 drwxrwxr-x 10 core core 4096 mai 12 21:10 'S0 '
4 drwxrwxr-x 2 core core 4096 mai 13 16:02 src
88 -rw-rw-r-- 1 core core 86046 mai 12 21:10 TP_S0.pdf
24 -rw-rw-r-- 1 core core 21904 mai 13 17:08 tracer
0 prw----- 1 core core 0 mai 13 17:34 tracer_to_monitor
Ended in 13.250526 ms
core@xubuncore:~/Desktop/TP-S05$
```

```
core@xubuncore:~/Desktop/TP-S05$ ./monitor PIDS_folder
Abriu o fifo para leitura
Erro na criação do FIFO.: File exists
Typeofservice: 1 | Programa adicionado à lista | Pid: 3188 | Nome: ls | tInicial: 1683995655
Typeofservice: 2 | Programa com pid: 3188 foi Terminado!
File PIDS_folder/3188.txt created successfully.
```

Figura 5.1: ./tracer execute -u programa arg1

```
core@xubuncore:~/Desktop/TP-S05$ ./tracer status
Erro na criação do FIFO.: File exists
Abriu o FIFO para escrita
Status enviado ao servidor!
PIDS_folder/3188.txt
core@xubuncore:~/Desktop/TP-S05$ ./tracer status
Erro na criação do FIFO.: File exists
Abriu o FIFO para escrita
Status enviado ao servidor!
3199 nl 7356.732742 ms | 7.356733 s
core@xubuncore:~/Desktop/TP-S05$ ./tracer status
Erro na criação do FIFO.: File exists
Abriu o FIFO para escrita
Status enviado ao servidor!
3199 nl 9463.973223 ms | 9.463973 s
core@xubuncore:~/Desktop/TP-S05$ ./tracer status
Erro na criação do FIFO.: File exists
Abriu o FIFO para escrita
Status enviado ao servidor!
3199 nl 11147.102400 ms | 11.147102 s
core@xubuncore:~/Desktop/TP-S05$
```

```
Running PID: 3188
total 208
12 -rw-rw-r-- 1 core core 8696 mai 12 21:10 a.out
4 drwxrwxr-x 2 core core 4096 mai 13 17:32 bin
4 drwxrwxr-x 4 core core 4096 mai 12 21:10 build
4 drwxrwxr-x 4 core core 4096 mai 12 21:10 cmake-build-debug
4 drwxrwxr-x 2 core core 4096 mai 13 16:02 include
4 -rw-rw-r-- 1 core core 751 mai 12 21:10 log.txt
4 -rw-rw-r-- 1 core core 530 mai 13 16:46 Makefile
12 -rw-rw-r-- 1 core core 9364 mai 12 21:10 mon
24 -rw-rw-r-- 1 core core 21744 mai 13 16:44 monitor
0 prw----- 1 core core 0 mai 13 17:13 monitor_to_tracer
4 drwxrwxr-x 2 core core 4096 mai 13 17:32 obj
0 -rw-rw-r-- 1 core core 0 mai 12 21:10 output
4 drwxrwxr-x 2 core core 4096 mai 13 17:32 PIDS_folder
4 -rw-rw-r-- 1 core core 83 mai 12 21:10 README.md
0 -rw-rw-r-- 1 core core 0 mai 12 21:10 saida.txt
4 drwxrwxr-x 10 core core 4096 mai 12 21:10 'S0 '
4 drwxrwxr-x 2 core core 4096 mai 13 16:02 src
88 -rw-rw-r-- 1 core core 86046 mai 12 21:10 TP_S0.pdf
24 -rw-rw-r-- 1 core core 21904 mai 13 17:08 tracer
0 prw----- 1 core core 0 mai 13 17:34 tracer_to_monitor
Ended in 13.250526 ms
core@xubuncore:~/Desktop/TP-S05$ ./tracer execute -u nl
Erro na criação do FIFO.: File exists
Abriu o FIFO para escrita
Running PID: 3199
```

```
core@xubuncore:~/Desktop/TP-S05$ ./monitor PIDS_folder
Abriu o fifo para leitura
Erro na criação do FIFO.: File exists
Typeofservice: 1 | Programa adicionado à lista | Pid: 3188 | Nome: ls | tInicial: 1683995655
Typeofservice: 2 | Programa com pid: 3188 foi Terminado!
File PIDS_folder/3188.txt created successfully.
Typeofservice: 1 | Programa adicionado à lista | Pid: 3199 | Nome: nl | tInicial: 1683995779
Typeofservice: 3 | A imprimir status ...
Existem 2 programas na lista!
Program ls com Pid: 3188 já foi Terminado!
Program nl, Pid: 3199, Em execução: 7356.732742 ms, 7.356733 s
Typeofservice: 3 | A imprimir status ...
Existem 2 programas na lista!
Program ls com Pid: 3188 já foi Terminado!
Program nl, Pid: 3199, Em execução: 9463.973223 ms, 9.463973 s
Typeofservice: 3 | A imprimir status ...
Existem 2 programas na lista!
Program ls com Pid: 3188 já foi Terminado!
Program nl, Pid: 3199, Em execução: 11147.102400 ms, 11.147102 s
Typeofservice: 3 | A imprimir status ...
Existem 2 programas na lista!
Program ls com Pid: 3188 já foi Terminado!
Program nl, Pid: 3199, Em execução: 12055.151142 ms, 12.055151 s
```

Figura 5.2: ./tracer status

```

24 -rwxrwxr-x 1 core core 21744 mai 13 16:44 monitor
0 prw----- 1 core core 0 mai 13 17:13 monitor_to_tracer
4 drwxrwxr-x 2 core core 4096 mai 13 17:32 obj
0 -rw-rw-r-- 1 core core 0 mai 12 21:10 output
4 drwxrwxr-x 2 core core 4096 mai 13 17:32 PIDS folder
4 -rw-rw-r-- 1 core core 83 mai 12 21:10 README.md
0 -rw-rw-r-- 1 core core 0 mai 12 21:10 saida.txt
4 drwxrwxr-x 10 core core 4096 mai 12 21:10 'S0 '
4 drwxrwxr-x 2 core core 4096 mai 13 16:02 src
88 -rw-rw-r-- 1 core core 86046 mai 12 21:10 TP_S0.pdf
24 -rwxrwxr-x 1 core core 21904 mai 13 17:08 tracer
0 prw----- 1 core core 0 mai 13 17:34 tracer_to_monitor
Ended in 13.250526 ms
core@xubuncore:~/Desktop/TP-S0$ ./tracer execute -u nl
Erro na criação do FIFO.: File exists
Abri o FIFO para escrita
Running PID: 3199
Ended in 104517.115430 ms
core@xubuncore:~/Desktop/TP-S0$ ./tracer status-time 3188 3199
Erro na criação do FIFO.: File exists
Abri o FIFO para escrita
Status-time enviado ao servidor!
Total execution time is: 104530.365956 ms
core@xubuncore:~/Desktop/TP-S0$ ./tracer status-time 3188 3199
Erro na criação do FIFO.: File exists
Abri o FIFO para escrita
Status-time enviado ao servidor!
Total execution time is: 104530.365956 ms
core@xubuncore:~/Desktop/TP-S0$

```

```

Program: ls com Pid: 3188 já foi Terminado!
Program: nl com Pid: 3199 já foi Terminado!
Typeofservice: 3 | A imprimir status ...
Existem 2 programas na lista!
Program: ls com Pid: 3188 já foi Terminado!
Program: nl com Pid: 3199 já foi Terminado!
Typeofservice: 3 | A imprimir status ...
Existem 2 programas na lista!
Program: ls com Pid: 3188 já foi Terminado!
Program: nl com Pid: 3199 já foi Terminado!
Typeofservice: 3 | A imprimir status ...
Existem 2 programas na lista!
Program: ls com Pid: 3188 já foi Terminado!
Program: nl com Pid: 3199 já foi Terminado!
Typeofservice: 3 | A imprimir status ...
Existem 2 programas na lista!
Program: ls com Pid: 3188 já foi Terminado!
Program: nl com Pid: 3199 já foi Terminado!
Typeofservice: 4 | A imprimir status time..
Typeofservice: 4 | A imprimir status time..

```

Figura 5.3: ./tracer status-time PID1 PID2

```

core@xubuncore:~/Desktop/TP-S0$ ./tracer execute -p "cat src/tracer.c | grep 'status' | uniq | wc -l"
c -l
Erro na criação do FIFO.: File exists
Abri o FIFO para escrita
Running PID: 3230
Pipeline finished.
Ended in 16.464 ms
core@xubuncore:~/Desktop/TP-S0$

```

```

Program: ls com Pid: 3188 já foi Terminado!
Program: nl com Pid: 3199 já foi Terminado!
Typeofservice: 4 | A imprimir status time..
Typeofservice: 4 | A imprimir status time..
Typeofservice: 7 | Programa adicionado à lista | Pid: 3230 | Nome: cat src/tracer.c | grep status | uni
q | wc -l | Inicial: 945526974
Typeofservice: 2 | Programa com pid: 3230 foi Terminado!
File PIDS_folder/3230.txt created successfully.

```

Figura 5.4: ./tracer execute -p "programa\_encadeado"

```

core@xubuncore:~/Desktop/TP-S0$ cat src/tracer.c | grep "status" | uniq | wc -l
28
core@xubuncore:~/Desktop/TP-S0$ ./tracer execute -p "cat src/tracer.c | grep 'status' | uniq | wc -l"
Erro na criação do FIFO.: File exists
Abri o FIFO para escrita
Running PID: 3261
Pipeline finished.
Ended in 9.823 ms
core@xubuncore:~/Desktop/TP-S0$

```

Figura 5.5: Verificação se o resultado obtido está correto

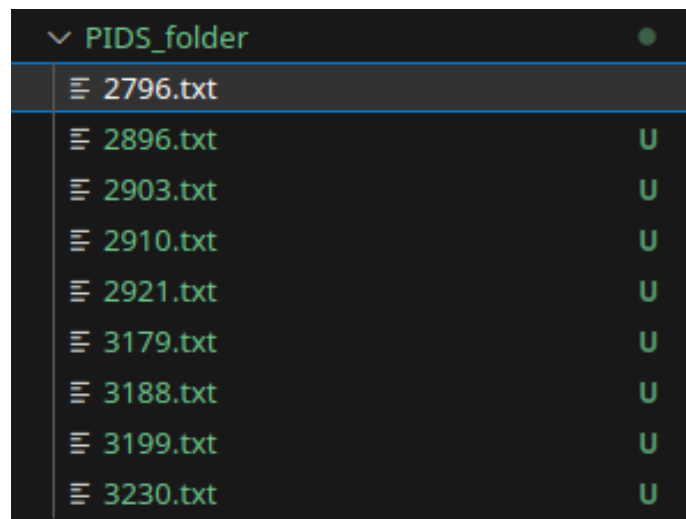


Figura 5.6: Files criados com o nome do PID do programa

# 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 possível consultar os programas que se encontram em execução e as estatísticas sobre os programas que já terminaram.*

*Em retrospetiva, 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 fífos com nome, para saber a quantidade de fífos 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.*