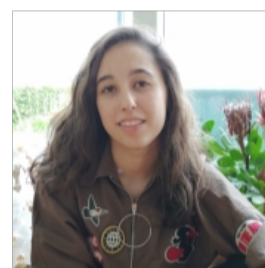




**Universidade do Minho**  
Escola de Engenharia

Sistemas Operativos  
Relatório do Trabalho Prático - Grupo 100  
2022/2023

Afonso Xavier Cardoso Marques 94940  
José Diogo Lopes Faria 95255  
Ana Filipa Cruz Pinto 96862



# Índice

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Os três programas</b>	<b>4</b>
2.1	Fifo . . . . .	4
2.2	Tracer . . . . .	4
2.3	Monitor . . . . .	6
<b>3</b>	<b>Testes realizados</b>	<b>7</b>
3.1	Teste 1 . . . . .	7
3.2	Teste 2 . . . . .	8
3.3	Teste 3 . . . . .	8
<b>4</b>	<b>Conclusão</b>	<b>10</b>

# 1 Introdução

O presente relatório expõe o desenvolvimento de um projeto que consistiu na criação de um serviço de monitorização dos programas executados numa máquina, no qual um cliente, denominado tracer, é capaz de executar programas e enviar informação sobre o estado de execução dos mesmos a um servidor, denominado monitor, que monitoriza todos os programas que estão em execução.

O monitor permite também consultar os programas atualmente em execução através de um pedido de status por parte de um tracer. Numa fase inicial, o maior desafio foi encontrar uma forma de garantir que a concorrência entre pedidos de múltiplos tracers era controlada de forma a não ocorrerem deadlocks. A comunicação entre os múltiplos tracers e o monitor é implementada usando pipes com nome (FIFO).

## 2 Os três programas

Nesta secção do relatório vamos expor os três programas que foram criados para realizar o objetivo do trabalho prático e a forma como eles interagem uns com os outros. Os programas em questão são: o Fifo, o Tracer e o Monitor.

### 2.1 Fifo

O programa de Fifo é o mais simples e admitimos que podia ter sido implementado, por exemplo, dentro do Monitor. O seu único propósito é de criar um pipe com nome, denominado FIFO, que vai servir como ponte de comunicação entre os vários Tracers e o Monitor. A comunicação é unilateral, ou seja, o Tracer escreve conteúdo no FIFO e o Monitor lê esse conteúdo. O conteúdo em questão é uma estrutura criada pelo grupo que serve para guardar a informação do programa que o Tracer está a executar.

Tal como dito anteriormente, não havia necessidade de implementar o Fifo num programa à parte, mas tendo feito essa escolha, conseguimos evitar estar sempre a tentar criar o mesmo ficheiro. Assim, o pipe com nome só é criado quando é mesmo necessário.

De forma a não provocar confusões, no decorrer do relatório iremos referir-nos ao pipe criado pelo programa Fifo como sendo o FIFO "geral".

### 2.2 Tracer

O programa de Tracer é aquele que permite a execução dos programas e dos seus argumentos. Um Tracer apenas executa um programa de cada vez. Logo se quisermos ver múltiplos programas a correr em concorrência, temos de executar múltiplos Tracers.

O Tracer possui três modos de execução que são ativados conforme os argumentos que lhe são passados. Temos o modo *execute* que, dependendo da flag passada como argumento, permite executar um programa ou então um pipeline de programas. A flag *-u* permite executar apenas um programa, criando para isso duas estruturas:

```
typedef struct prog{
    int pid;
    char* prog_name;
    char** arguments;
    int num_args;
    int time;
} Prog;
```

```
typedef struct msg{
    int pid;
    int type;           //1 inicio do programa ; 2 fim do programa ; 3 status
    char prog_name[20];
    int time;
} Msg;
```

- Struct Prog - estrutura que armazena o nome, argumentos e número de argumentos de um programa. É a estrutura que usamos para executar o programa através da função *execvp* onde passamos como parâmetros *prog\_name* e *arguments*. Inicialmente a nossa intenção era que esta estrutura fosse enviada para o Monitor, daí possuir um campo para o PID do processo e para o tempo. Estes valores, apesar de serem preenchidos, não são usados. O campo de *num\_args* serve apenas para facilitar o processo de preenchimento do array *arguments*. Esta estrutura é preenchida na função de *parse\_single*.
- Struct Msg - estrutura semelhante ao Prog mas não tem informação sobre os argumentos do programa, o que torna a estrutura mais leve e flexível para ser enviada para o Monitor através do FIFO. Guarda-se apenas o nome do programa, o PID e o tempo de execução. O campo de *type* serve para indicar o tipo de mensagem que estamos a enviar. Se for para indicar o início de execução de um programa, o tipo será 1, se for para indicar que o programa terminou, o tipo será 2 e se for para indicar que queremos fazer um pedido de status, o tipo será 3.

O segundo modo de operação do Tracer é a execução de um pipeline de programas. Para ativar este modo é preciso colocar a flag *-p* a seguir ao argumento *execute*. Para esse tipo de execução, é necessário recorrer a pipes anónimos e fazer redirecionamentos. A implementação deste modo é extremamente semelhante ao *modus operandi* da flag *-u*. A diferença está na forma como executamos os programas. Para isto foram criadas três funções:

- *parse\_pipeline* - Uma função que faz parse da string que representa o pipeline e devolve um array de strings com os comandos e respetivos argumentos em cada posição. Por exemplo a string "ls -l | ps aux | wc -l" recebida como parâmetro irá devolver o array { "ls -l", "ps aux", "wc -l" } .
- *execute\_pipeline* - Função que serve para correr cada programa do array criado em *parse\_pipeline* recorrendo à função auxiliar *exec\_command*. Aqui criamos uma matriz de pipes tendo em conta o número de programas que vamos correr e fazemos os redirecionamentos necessários tendo em conta a posição do array onde nos encontramos. O output final é impresso no ecrã e o tempo de execução é revelado.
- *exec\_command* - Uma pequena função que executa um programa com base na string que recebe como argumento. Na sua essência é uma versão simplificada da função *parse\_single* que usamos no modo *-u*.

O terceiro modo de operação do Tracer é o modo *status*. Para fazer um pedido de status apenas é necessário criar um estrutura Msg que será preenchida da seguinte forma: o campo *pid* contém o pid do processo do Tracer (chamemos-lhe XXXX), o campo *type* terá o valor 3, o *time* não é preenchido e o *prog\_name[20]* terá como valor uma string "FIFO-XXXX", que é o ficheiro de pipe temporário criado pelo Tracer especificamente para ler mensagens de status do Monitor. O Monitor recebe esta estrutura através do FIFO "geral", identifica que é do tipo 3 e percebe que deve escrever as mensagens de status num ficheiro de pipe específico, o tal "FIFO-XXXX".

Esta estratégia permite garantir que as mensagens chegam aos Tracers corretos, especialmente se tivermos múltiplos pedidos de status a decorrer. Vale a pena mencionar que o Monitor cria um processo filho para cada pedido de status que lhe é feito. Isto será explorado em maior detalhe na secção seguinte do relatório.

## 2.3 Monitor

O programa Monitor é o responsável por guardar a informação recebida do Tracer sobre a execução dos programas. O programa recebe uma estrutura Msg do Tracer através do FIFO "geral" e guarda-a numa nova estrutura:

- Struct Map - armazena a apenas a informação relevante para o monitor, isto é, o pid do programa, o nome do programa e o instante em que o programa foi iniciado.

```
typedef struct map{
    int pid;
    int time;
    char nome[20];
} Map;

struct map arr_map[1000000];
```

De seguida esta struct é armazenada num array onde a sua posição corresponde ao valor do PID. Este método de armazenamento não é o mais eficaz e tem várias falhas como por exemplo, este ser um armazenamento estático e ter uma capacidade reduzida.

Quando um programa é encerrado, o monitor remove a estrutura do array e cria um ficheiro, nomeado de acordo com o PID do programa e colocado numa pasta chamada PIDS-folder, onde escreve a informação relativa ao programa para poder ser consultado posteriormente.

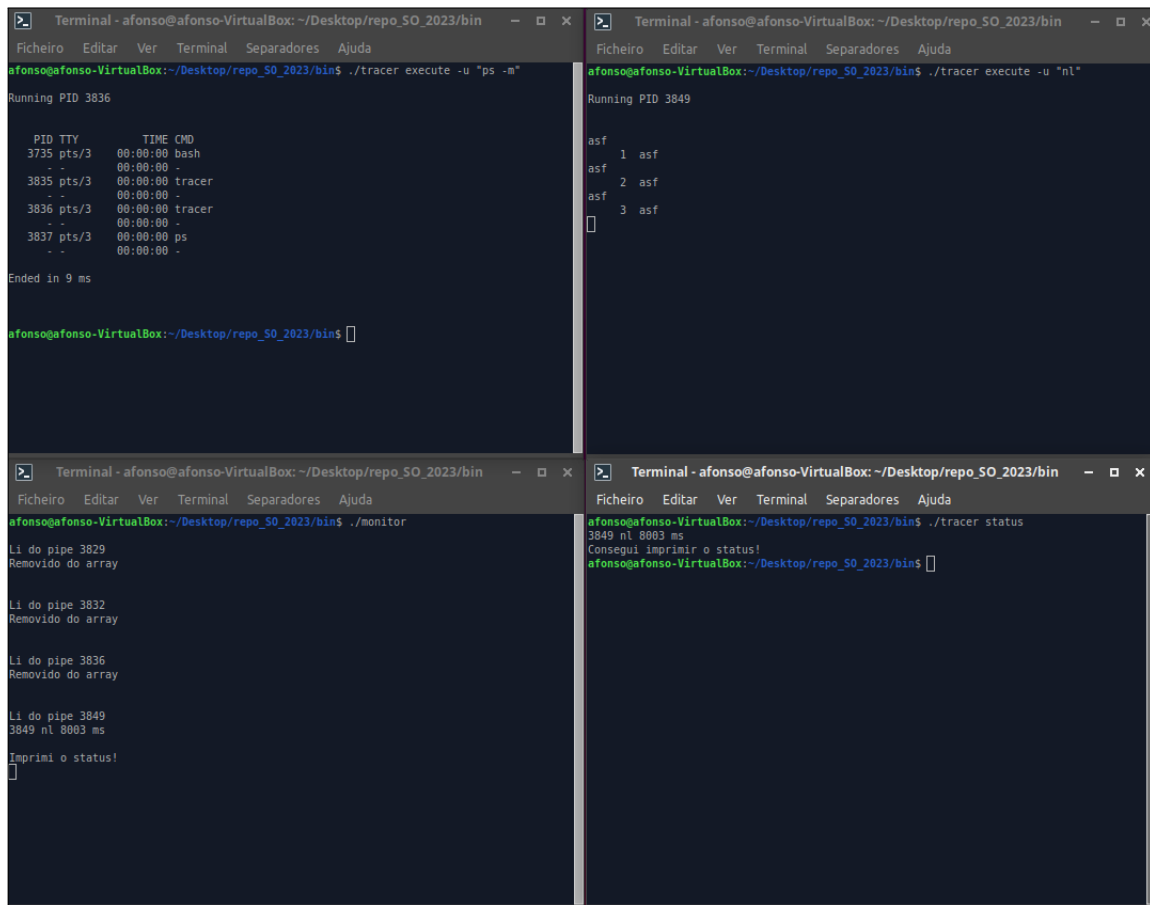
Para realizar uma observação do status o programa junta a informação da estrutura map numa string e envia para o Tracer através de um pipe com nome FIFO-XXXX (onde XXXX é o PID do processo Tracer que fez o pedido de status). Aqui houve a necessidade de criar um processo filho de forma a garantir que o Monitor consegue lidar com s pedidos de status provenientes dos vários Tracers sem causar deadlocks.

### 3 Testes realizados

Nesta secção vamos apresentar alguns testes executados pelo grupo acompanhados de imagens.

#### 3.1 Teste 1

Neste teste corremos dois programas com `./tracer execute -u`, nomeadamente, "`ps -l`" e "`nl`". O Monitor corre no background e é possível notar que os dois programas executam sem problemas, com "`ps -l`" a terminar quase de imediato e "`nl`" ainda a correr porque não se fez `Ctrl-D`. O pedido de status devolve apenas a informação de um processo, neste caso do "`nl`" que ainda está a correr e por isso não foi removido do Monitor.



```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/repo_SO_2023/bin
Ficheiro Editar Ver Terminal Separadores Ajuda
afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$ ./tracer execute -u "ps -l"
Running PID 3836

  PID TTY          TIME CMD
 3735 pts/3    00:00:00 bash
    -  -         00:00:00 .
 3835 pts/3    00:00:00 tracer
    -  -         00:00:00 .
 3836 pts/3    00:00:00 tracer
    -  -         00:00:00 .
 3837 pts/3    00:00:00 ps
    -  -         00:00:00 .

Ended in 9 ms

afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$

Terminal - afonso@afonso-VirtualBox: ~/Desktop/repo_SO_2023/bin
Ficheiro Editar Ver Terminal Separadores Ajuda
afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$ ./tracer execute -u "nl"
Running PID 3849

  asf 1 asf
  asf 2 asf
  asf 3 asf
  []

Terminal - afonso@afonso-VirtualBox: ~/Desktop/repo_SO_2023/bin
Ficheiro Editar Ver Terminal Separadores Ajuda
afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$ ./monitor
LI do pipe 3829
Removido do array

LI do pipe 3832
Removido do array

LI do pipe 3836
Removido do array

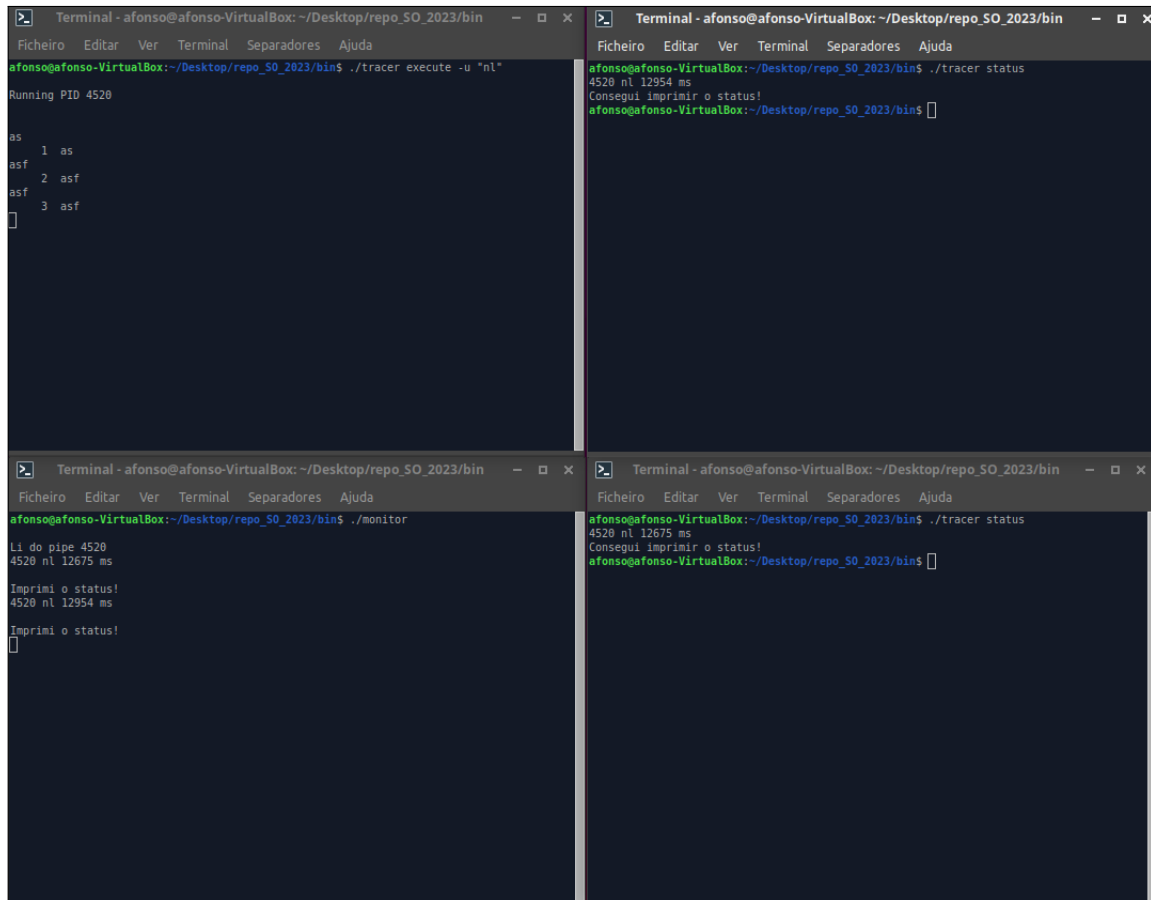
LI do pipe 3849
3849 nl 8003 ms
Imprimi o status!
[]

Terminal - afonso@afonso-VirtualBox: ~/Desktop/repo_SO_2023/bin
Ficheiro Editar Ver Terminal Separadores Ajuda
afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$ ./tracer status
3849 nl 8003 ms
Conseguir imprimir o status!
afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$
```

Figure 1: Terminais - Teste 1

### 3.2 Teste 2

Neste teste executamos apenas um programa com `./tracer execute -u "nl"`. O Monitor está a correr em background e é possível ver dois pedidos de status que foram feitos quase em simultâneo (isto porque transitar de um terminal para o outro gasta sempre uns milissegundos). A resposta é correta nos dois terminais e não houve atropelamentos. Isto acontece porque cada Tracer possui um PID XXXX e a leitura da mensagem é feita recorrendo a um FIFO-XXXX temporário. Até é possível ver que em cada resposta, tudo é igual menos o tempo de execução, o que mostra que o Monitor respondeu primeiro a um e depois ao outro.



The image displays four terminal windows arranged in a 2x2 grid, all running on a system named 'afonso@afonso-VirtualBox' with the directory path '~/Desktop/repo\_SO\_2023/bin'. Each window has a menu bar with 'Ficheiro', 'Editar', 'Ver', 'Terminal', 'Separadores', and 'Ajuda'.

- Top-left terminal:** Shows the command `./tracer execute -u "nl"` being executed. The output indicates the process is running with PID 4520. Below this, there is a list of system calls: `as` (1), `asf` (2), and `asf` (3).
- Top-right terminal:** Shows the command `./tracer status`. The output displays `4520 nl 12954 ms` and a message `Conseguí imprimir o status!`. The prompt is `afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$`.
- Bottom-left terminal:** Shows the command `./monitor`. The output includes `Li do pipe 4520`, `4520 nl 12675 ms`, and two messages `Imprimi o status!` corresponding to the two status requests.
- Bottom-right terminal:** Shows the command `./tracer status`. The output displays `4520 nl 12675 ms` and a message `Conseguí imprimir o status!`. The prompt is `afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$`.

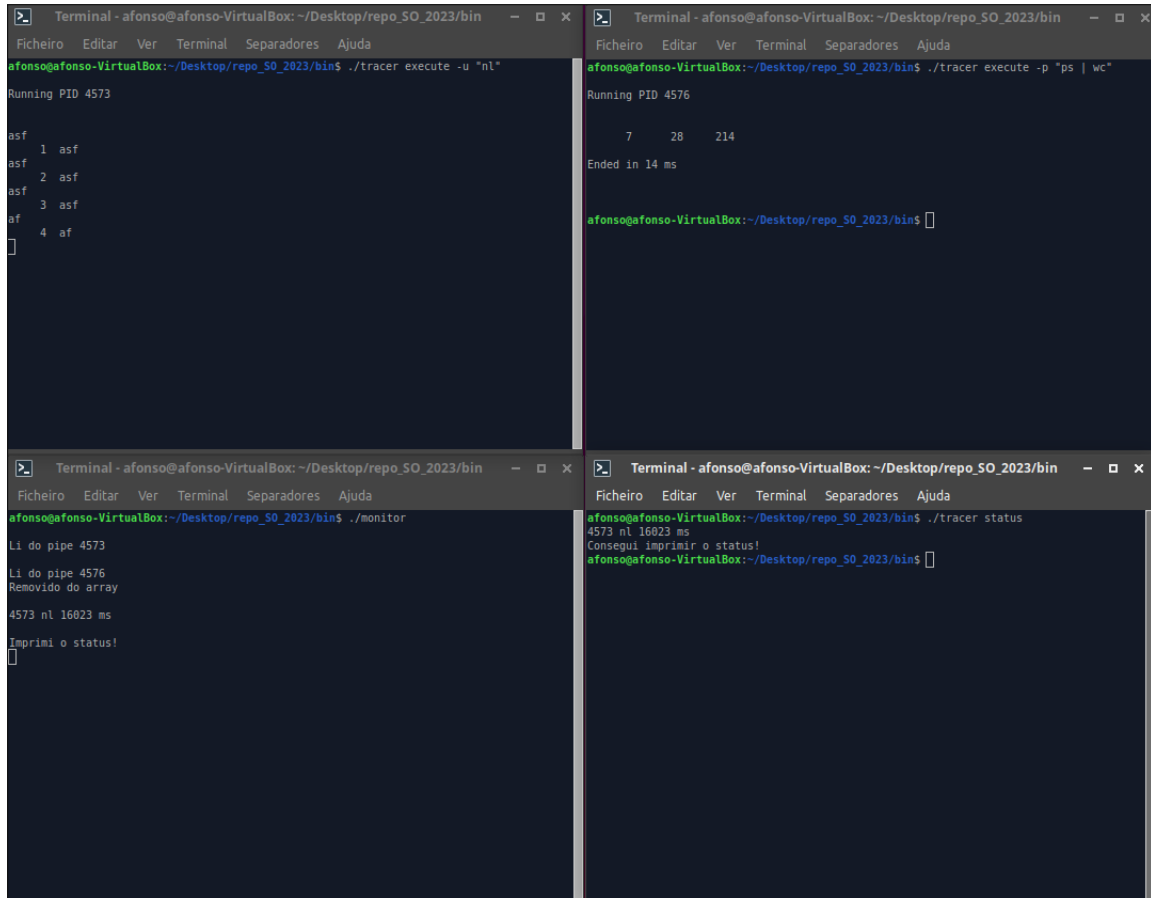
Figure 2: Terminais - Teste 2

### 3.3 Teste 3

Neste teste corremos um programa e um pipeline com dois programas com `./tracer execute -u "nl"` e `./tracer execute -p "ps | wc"` respetivamente. O Monitor corre no background e é possível notar que am-



os Tracer executam normalmente com o pipeline a terminar quase de imediato e logo de seguida a ser removido do Monitor. Foi feito um pedido de status que devolveu a informação correta.



```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/repo_SO_2023/bin
Ficheiro Editar Ver Terminal Separadores Ajuda
afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$ ./tracer execute -u "nl"
Running PID 4573

asf 1 asf
asf 2 asf
asf 3 asf
af 4 af

```

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/repo_SO_2023/bin
Ficheiro Editar Ver Terminal Separadores Ajuda
afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$ ./tracer execute -p "ps | wc"
Running PID 4576

7 28 214
Ended in 14 ms

afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$

```

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/repo_SO_2023/bin
Ficheiro Editar Ver Terminal Separadores Ajuda
afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$ ./monitor
Li do pipe 4573
Li do pipe 4576
Removido do array
4573 nl 16023 ms
Imprimi o status!

```

```
Terminal - afonso@afonso-VirtualBox: ~/Desktop/repo_SO_2023/bin
Ficheiro Editar Ver Terminal Separadores Ajuda
afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$ ./tracer status
4573 nl 16023 ms
Conseguir imprimir o status!
afonso@afonso-VirtualBox:~/Desktop/repo_SO_2023/bin$

```

Figure 3: Terminais - Teste 3

Obviamente há muitos outros testes que se podem fazer e o nosso grupo garante que fez várias combinações interessantes para observar o comportamento dos programas desenvolvidos. Vale a pena mencionar que apesar das imagens serem retiradas do mesmo computador, os testes foram realizados nas máquinas de todos os elementos do grupo e verificou-se que o comportamento era igual.

## 4 Conclusão

Este projeto permitiu-nos consolidar os conhecimentos que fomos adquirindo ao longo do semestre nas aulas de Sistemas Operativos. Conseguimos implementar a totalidade das funcionalidades básicas bem como algumas das funcionalidades mais avançadas pelo que fazemos um balanço positivo do trabalho realizado pelo grupo.

No entanto, refletindo sobre o nosso trabalho, reconhecemos que alguns aspetos das funcionalidades avançadas estão menos bons. Por exemplo a execução dos pipelines termina sempre com o resultado correto mas, às vezes, o output é devolvido numa ordem que não corresponde ao pedido no enunciado. Achamos que possa estar relacionado com a gestão de processos filhos na função *execute\_pipeline*. Também existe a questão de não termos conseguido implementar os pedidos de status mais complexos que são um requisito das funcionalidades avançadas. Ainda assim estamos confiantes que temos um trabalho sólido.