

Universidade do Minho

Departamento de Informatica

Lcc

**Relatório trabalho prático Sistemas
Operativos 19/20**

Aluno: a71532 Ricardo Jose Magalhaes Silva

Aluno: a65432 João Manuel Martins Cerqueira

Aluno:

Junho
2020

1 Introdução

O trabalho pede para desenvolver 2 programas (cliente e servidor) que comunicam entre si através de pipes com nome. O cliente (“argus”) é o que é usado pelo utilizador para executar as funcionalidades mínimas. Os comandos inseridos pelo utilizador são tratados pelo cliente e enviados para o servidor (“argus-server”), o qual executa-os.

Para executar os programas, após correr o comando “make” é preciso manter 2 terminais abertos: no 1º terminal executa-se “./argus-server” para iniciar o servidor e deixa-se esse terminal aberto a correr. No 2º terminal é onde executamos o argus. Este pode ser invocado de 2 formas (tal como é pedido no enunciado): ou executamos apenas “./argus” e este abre a sua consola para escrever os comandos (ex.: “ajuda”) ou executamos “./argus [ARGUMENTOS]” e este executa apenas o comando e não abre a consola (ex.: “./argus -h”).

Os comandos escritos no argus são enviados pelos pipes para o servidor, o qual executa e devolve a resposta ao cliente. Os processos criados para a execução dos comandos vão comunicar entre si através de pipes anónimos. Neste trabalho vão ser aplicadas chamadas ao sistema, mecanismos e conhecimentos recolhidos durante as aulas de Sistemas Operativos.

Conteúdo

1	Introdução	1
2	Resumo	1
3	Hierarquia de processos	3
4	Descrição das funcionalidades	4
4.1	Funcionalidades minimas	4
4.1.1	Tempo-inatividade	4
4.1.2	Tempo-execução	5
4.1.3	Ajuda	6
4.1.4	Terminar	7
4.1.5	Listar	8
4.1.6	Historico	9
4.1.7	Executar	10
5	Análise de resultados	12
6	Conclusão	13
	Bibliografia	14
	Anexo	15

2 Resumo

Pretende-se que implemente um serviço de monitorização de execução e de comunicação entre processos. O serviço deverá permitir a um utilizador a submissão de sucessivas tarefas, cada uma delas sendo uma sequência de comandos encadeados por pipes anónimos. Além de iniciar a execução das tarefas, o serviço deverá ser capaz de identificar as tarefas em execução, bem como a conclusão da sua execução. Deverá terminar tarefas em execução, caso não se verifique qualquer comunicação através de pipes anónimos ao fim de um tempo especificado. O serviço deverá ainda terminar as tarefas em execução caso seja especificado um tempo máximo de execução. A interface com o utilizador deverá contemplar duas possibilidades: uma será através de linha de comando, indicando opções apropriadas; a outra será uma interface textual interpretada (shell), e nesse caso o comando irá aceitar instruções do utilizador através do standard input.

O nosso trabalho contém vários módulos que serão resumidos abaixo.

`mySOLib.c` - biblioteca que criei para as aulas práticas para colocar as funções que precisava de usar nas várias fichas (ex.: `my-printf`, `tokenize`, `my-readln`, etc). Daqui uso a “`my-printf`” e “`myprintf2`” para imprimir e a “`tokenize`” para partir strings numa lista de strings (divide as strings pelos espaços). Tem apenas 1 include, o `mySOLib.h`.

`mySOLib.h` - como a `mySOLib` é uma biblioteca, para a importar para outros projectos é preciso definir este ficheiro header. Este ficheiro tem todos os includes que a `mySOLib.c` precisa e a assinatura de cada função criada. Apenas é preciso alterar este ficheiro se for adicionada alguma função à lib.

`main-cliente.c` - este é o ficheiro que após compilado transforma-se no `argus`. A única função que coloco aqui é a `main`. Todas as outras funções executadas pelo `argus` ou variáveis globais e includes estão nos ficheiros `cliente.c` e `cliente.h`. Não é necessário dividir em 3 ficheiros, mas assim fica mais fácil de perceber quais os passos que o `argus` executa. Tem apenas 1 include, o `cliente.h`.

`cliente.c` - todas as funções que o `argus` executa estão neste ficheiro. Tem apenas 1 include, `cliente.h`.

`cliente.h` - este header tem todos os includes que os 2 ficheiros anteriores precisam e tem ainda as assinaturas de todas as funções do ficheiro `cliente.c` para que possam ser usadas pela `main-cliente.c`. Tem também várias

variáveis e estruturas globais. Coloquei comentários na maioria, mas para exemplificar: os fifos (fifo-fd[2]) estão neste ficheiro para que a main os possa inicializar e depois quando uma função precisar de usar algum destes não é preciso passar como argumento a essa função, porque estes estão visíveis para todas as funções na main-cliente.c e cliente.c. As strings mais usadas também estão aqui, de forma a evitar erros ao escrever os nomes repetidamente. Por exemplo, se nos enganamos e escrevemos “tempo-inctividade”, não ocorre nenhum erro ao compilar e podemos perder tempo à procura deste erro, enquanto que TEMPO-INCTIVDADE irá lançar erro a informar que esta não existe (“undeclared identifier”).

Main-servidor.c, servidor.c e servidor.h - Para o servidor a estrutura é praticamente igual à do cliente: 3 ficheiros, apenas a main no principal a fazer include do servidor.h, servidor.c com todas as funções usadas pela main-servidor.c e com o 1 único include servidor.h, e o servidor.h com as assinaturas de todas as funções do servidor.c para que estas fiquem visíveis para o main-servidor.c e ainda as variáveis globais e os includes que o main-servidor.c e o servidor.c precisam. O ficheiro main-servidor.c após compilado passa a ser o argus-server.

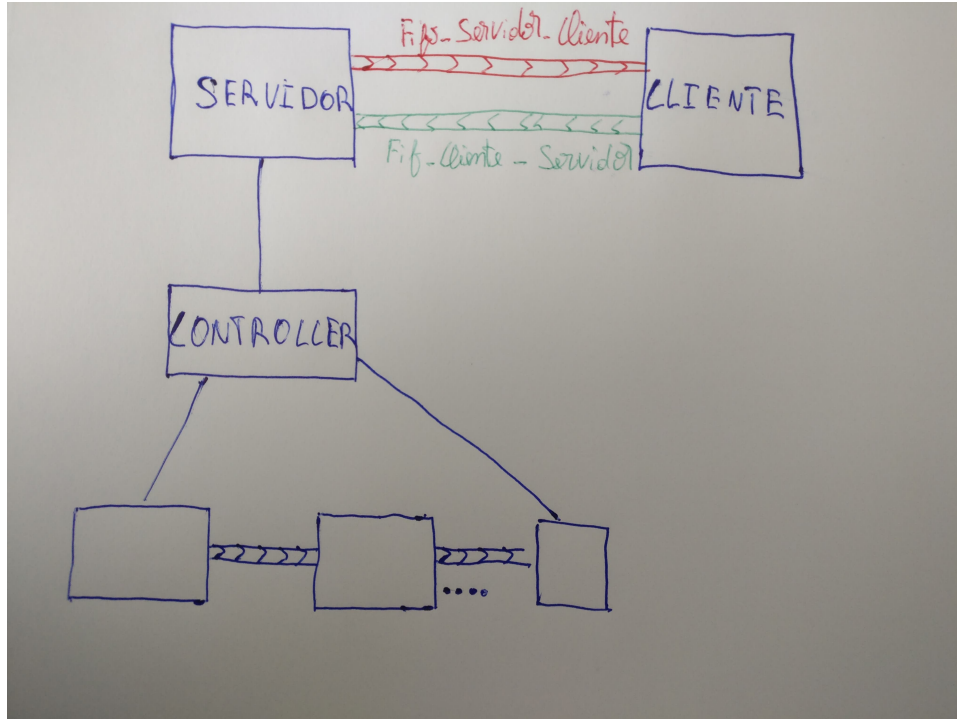
HELP - quando o comando “ajuda” ou “./argus -h” é executado, este ficheiro é lido pelo servidor e escrito para o cliente.

- apenas aparece após a execução do “./argus-server”. Ainda não está a ser usado, mas será o ficheiro para o qual são escritos os comandos já terminados (penso eu).

Fifo-client-server e fifo-server-client - estes são os fifos usados para comunicar entre o cliente e o servidor. No primeiro escreve o cliente e lê o servidor e no segundo escreve o servidor e lê o cliente. Apenas aparecem depois de iniciar o servidor.

argus e argusd - são os 2 executáveis do cliente e servidor, respectivamente.

3 Hierarquia de processos



O servidor faz fork e cria um processo filho que esse vai ser o responsável por criar os processos para executar uma determinada tarefa

4 Descrição das funcionalidades

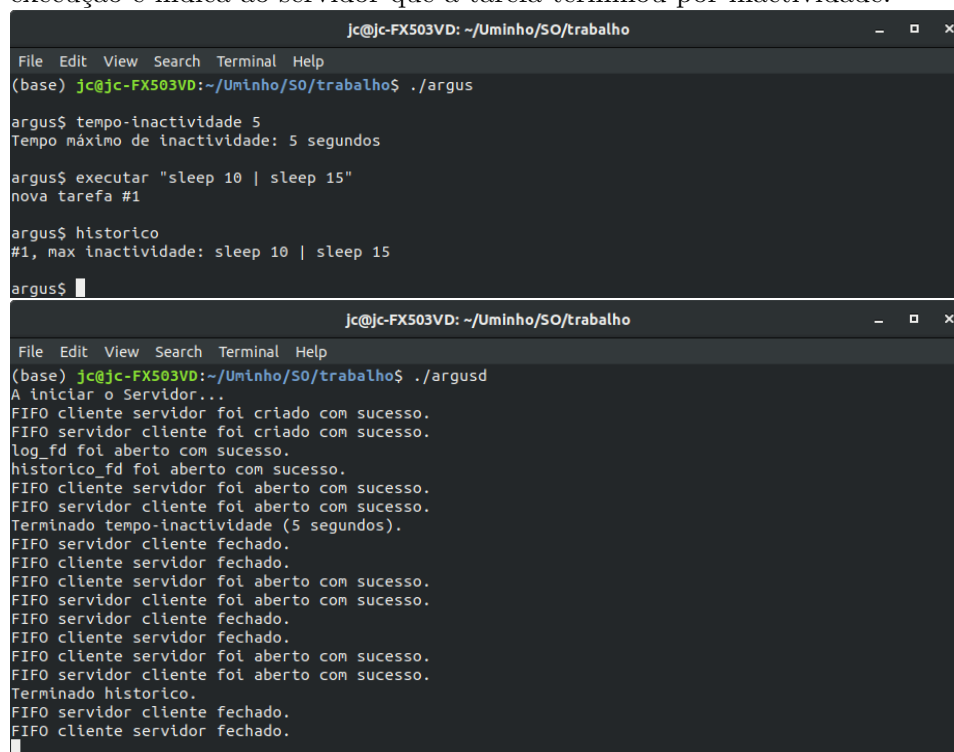
Foram pedidas uma lista de funcionalidades mínimas para o nosso serviço de monitorização que serão apresentadas a seguir

4.1 Funcionalidades mínimas

4.1.1 Tempo-inatividade

Definir o tempo máximo (em segundos) de inatividade de comunicação num pipe anónimo (opção -i na linha de comandos).

Quando esta funcionalidade é invocada pelo utilizador, o servidor interpreta o valor máximo de inatividade indicado e, se o valor for positivo, este é guardado numa variável global. Quando o servidor receber uma tarefa para executar, é feita uma verificação por esta variável global. Se estiver definido um valor válido, então por cada comando da tarefa a executar é definido um tempo máximo de inatividade. Se este tempo máximo for atingido por algum comando, o alarme mata todos os outros comandos da tarefa em execução e indica ao servidor que a tarefa terminou por inatividade.



```
jc@jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
(base) jc@jc-FX503VD:~/Uminho/SO/trabalho$ ./argus

argus$ tempo-inatividade 5
Tempo máximo de inatividade: 5 segundos

argus$ executar "sleep 10 | sleep 15"
nova tarefa #1

argus$ historico
#1, max inatividade: sleep 10 | sleep 15

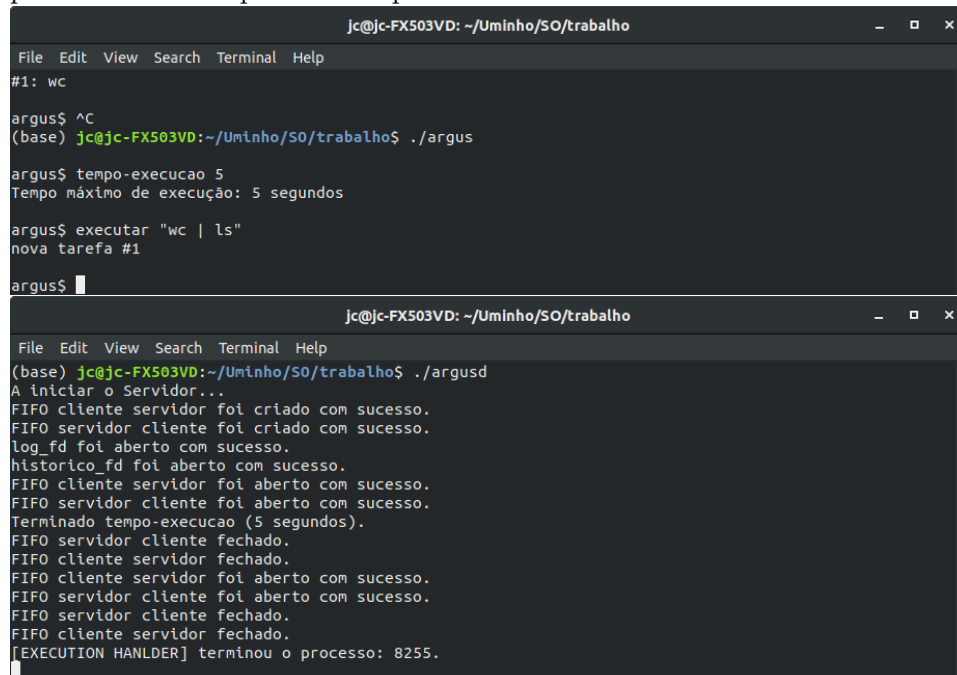
argus$

jc@jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
(base) jc@jc-FX503VD:~/Uminho/SO/trabalho$ ./argusd
A iniciar o Servidor...
FIFO cliente servidor foi criado com sucesso.
FIFO servidor cliente foi criado com sucesso.
log_fd foi aberto com sucesso.
historico_fd foi aberto com sucesso.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
Terminado tempo-inatividade (5 segundos).
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
Terminado historico.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
```

4.1.2 Tempo-execução

Definir o tempo máximo (em segundos) de execução de uma tarefa (opção -m n na linha de comandos)

Esta funcionalidade é semelhante à anterior. Após ser invocada pelo cliente, guarda o valor máximo de execução numa variável global. Quando o servidor começa a executar uma tarefa, verifica a variável global e se encontrar um valor positivo, define um alarme com esse valor. Se a tarefa exceder o tempo de execução, o alarme mata a tarefa e informa o servidor que esta terminou por exceder o tempo máximo permitido.



```
jc@jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
#1: wc
argus$ ^C
(base) jc@jc-FX503VD:~/Uminho/SO/trabalho$ ./argus

argus$ tempo-execucao 5
Tempo máximo de execução: 5 segundos

argus$ executar "wc | ls"
nova tarefa #1

argus$
```

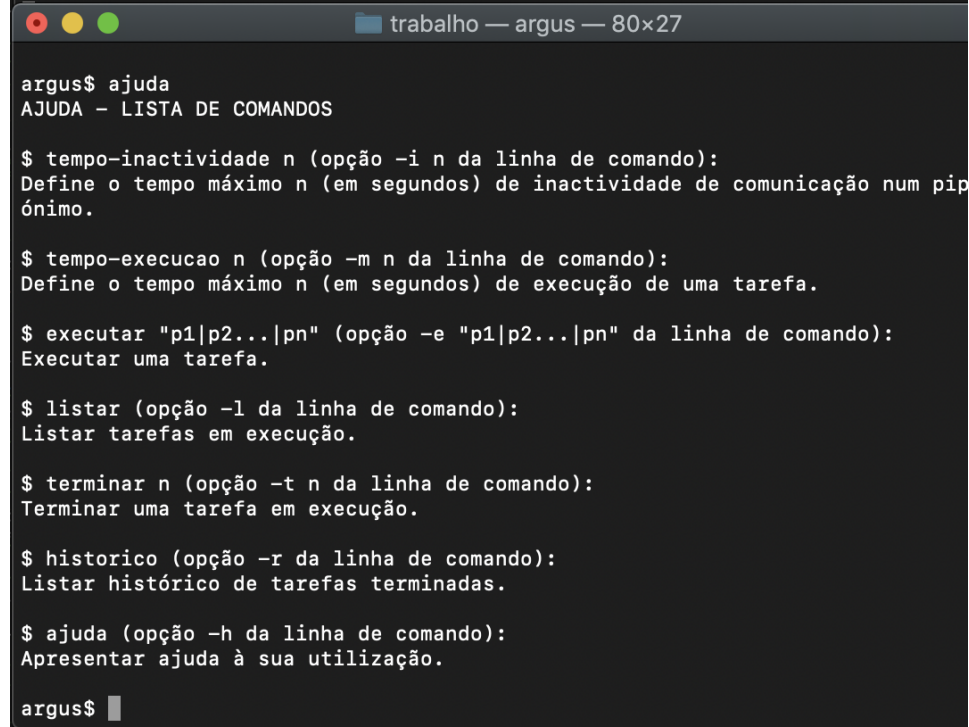
```
jc@jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
(base) jc@jc-FX503VD:~/Uminho/SO/trabalho$ ./argusd
A iniciar o Servidor...
FIFO cliente servidor foi criado com sucesso.
FIFO servidor cliente foi criado com sucesso.
log_fd foi aberto com sucesso.
historico_fd foi aberto com sucesso.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
Terminado tempo-execucao (5 segundos).
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
[EXECUTION HANDLER] terminou o processo: 8255.
```


4.1.3 Ajuda

Apresentar ajuda á sua utilização.

Esta funcionalidade lê um ficheiro auxiliar para um buffer, onde estão descritas todas as funcionalidades e como as usar, e escreve a partir desse buffer para o pipe do cliente.

```
A iniciar o Servidor...
log_fd foi aberto com sucesso.
historico_fd foi aberto com sucesso.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
Terminado ajuda.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
```



```
argus$ ajuda
AJUDA - LISTA DE COMANDOS

$ tempo-inactividade n (opção -i n da linha de comando):
Define o tempo máximo n (em segundos) de inactividade de comunicação num pipe
ónimo.

$ tempo-execucao n (opção -m n da linha de comando):
Define o tempo máximo n (em segundos) de execução de uma tarefa.

$ executar "p1|p2...|pn" (opção -e "p1|p2...|pn" da linha de comando):
Executar uma tarefa.

$ listar (opção -l da linha de comando):
Listar tarefas em execução.

$ terminar n (opção -t n da linha de comando):
Terminar uma tarefa em execução.

$ historico (opção -r da linha de comando):
Listar histórico de tarefas terminadas.

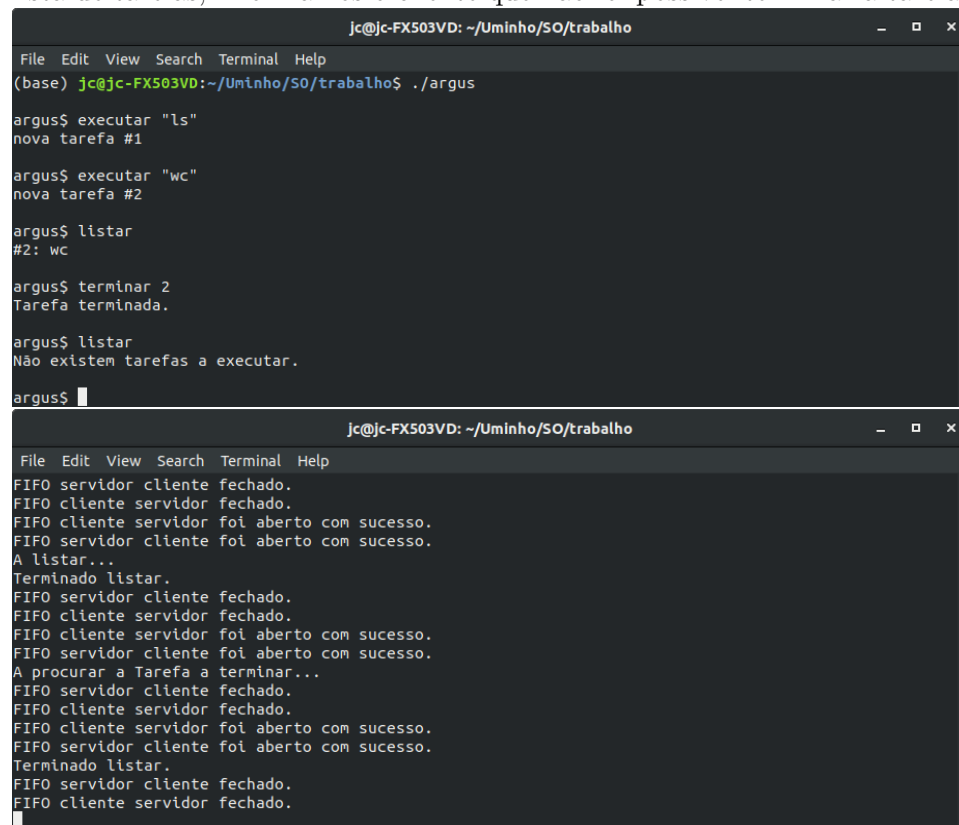
$ ajuda (opção -h da linha de comando):
Apresentar ajuda à sua utilização.

argus$
```

4.1.4 Terminar

Termina a execução de uma tarefa.

Para esta funcionalidade, recebemos o número da tarefa a terminar e, se for um número válido, procuramos por esta na lista de tarefas a executar. Se encontrarmos a tarefa, obtemos o seu pid, e se o pid for válido executamos o comando kill sobre este. Como a tarefa foi terminada, escrevemos essa informação para o histórico. Se a tarefa indicada não for encontrada na lista de tarefas, informamos o cliente que não foi possível terminar a tarefa.



```
jc@jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
(base) jc@jc-FX503VD:~/Uminho/SO/trabalho$ ./argus

argus$ executar "ls"
nova tarefa #1

argus$ executar "wc"
nova tarefa #2

argus$ listar
#2: wc

argus$ terminar 2
Tarefa terminada.

argus$ listar
Não existem tarefas a executar.

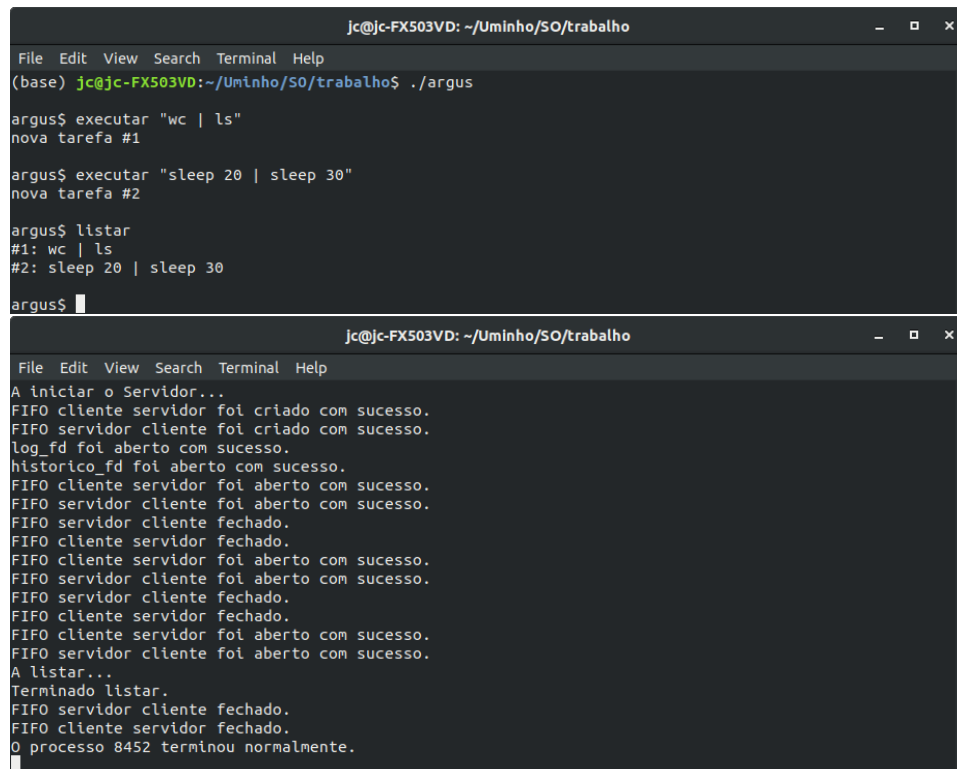
argus$
```

```
jc@jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
A listar...
Terminado listar.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
A procurar a Tarefa a terminar...
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
Terminado listar.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
```

4.1.5 Listar

Listar as tarefas em execução (opção -l na linha de comandos)

Esta funcionalidade percorre a lista de tarefas que estão a executar e escreve cada uma delas para o cliente através de um fifo. Como fazemos várias operações sobre cada tarefa (adicionar tarefa à lista, verificar estado final da tarefa, escrever tarefa para o histórico quando terminar), decidimos usar uma lista ligada para implementar a lista de tarefas em que cada elemento contem o id da tarefa, o seu pid e a tarefa a executar. Cada vez que o comando executar é iniciado, a tarefa a executar é colocada neste lista, e quando uma tarefa é terminada esta é removida e o seu estado final é escrito para o histórico.



```
jc@jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
(base) jc@jc-FX503VD:~/Uminho/SO/trabalho$ ./argus

argus$ executar "wc | ls"
nova tarefa #1

argus$ executar "sleep 20 | sleep 30"
nova tarefa #2

argus$ listar
#1: wc | ls
#2: sleep 20 | sleep 30

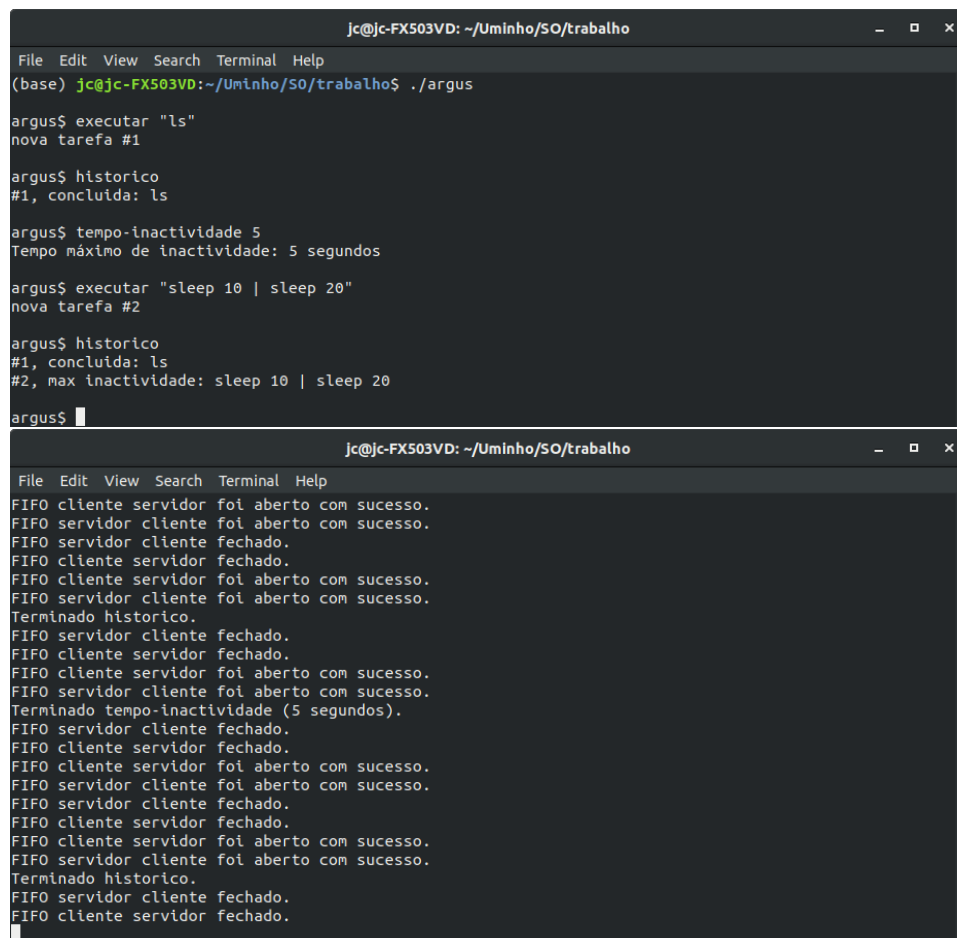
argus$
```

```
jc@jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
A iniciar o Servidor...
FIFO cliente servidor foi criado com sucesso.
FIFO servidor cliente foi criado com sucesso.
log_fd foi aberto com sucesso.
historico_fd foi aberto com sucesso.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
A listar...
Terminado listar.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
O processo 8452 terminou normalmente.
```

4.1.6 Historico

Listar o histórico de tarefas terminadas (opção -r).

Abrimos o ficheiro com a informação a imprimir, depois lemos do ficheiro para o buffer e escrevemos para o fifo-client-server. Como o histórico regista todas as tarefas executadas e terminadas até ao momento, e como apenas escrevemos para este ficheiro 1 vez por cada tarefa, decidimos manter este registo num ficheiro e não em memória.



```
Jc@Jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
(base) Jc@Jc-FX503VD:~/Uminho/SO/trabalho$ ./argus

argus$ executar "ls"
nova tarefa #1

argus$ historico
#1, concluida: ls

argus$ tempo-inactividade 5
Tempo máximo de inactividade: 5 segundos

argus$ executar "sleep 10 | sleep 20"
nova tarefa #2

argus$ historico
#1, concluida: ls
#2, max inactividade: sleep 10 | sleep 20

argus$

Jc@Jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
Terminado historico.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
Terminado tempo-inactividade (5 segundos).
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
Terminado historico.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
```

4.1.7 Executar

Executar uma tarefa (opção -e na linha de comandos).

Para esta funcionalidade, começamos por criar 1 processo filho no servidor para executar os comandos recebidos. Este processo filho começa por verificar se existe um tempo máximo de execução, e se existir então inicia um alarme com esse tempo. Após a definição do alarme, o processo verifica se a tarefa recebida contem pipes. Em caso negativo, invoca 1 processo filho e executa a tarefa sem definir o tempo máximo de inactividade, uma vez que não vai haver comunicação através de pipes. Em caso afirmativo, este processo vai dividir a tarefa nos múltiplos comandos a executar e vai inicializar os pipes a usar pelos comandos. Por cada comando a executar, partimos o comando para uma lista de argumentos (`args[]`) e é invocado um novo processo filho. Este novo processo filho começa por definir qual o pipe de onde recebe os dados e qual o pipe para onde escreve o resultado (recorrendo a redireccionamento), fechando todas as extremidades dos outros pipes que não irá usar. De seguida, se verificar que foi definido um tempo máximo de inactividade, lança 1 alarme com esse valor. Após definir o alarme, invoca o `execvp`, para o qual passa a lista de argumentos a executar. O resultado do `execvp` é escrito para a extremidade de escrita do pipe do próximo comando, excepto quando este é o último comando a ser executado. Neste último caso, o resultado do `execvp` é escrito para o ficheiro de "log", no qual escrevemos o output de todos os comandos.

Se o tempo de inactividade máximo for atingido por algum comando da lista de comandos a executar, o processo filho invoca o comando `kill` para todos os comandos a executar. Se o processo pai verificar que o processo filho foi terminado por um sinal `SIGALRM`, este assume que o processo terminou por inactividade. O pai do processo faz então `_exit(INACTIVIDADE)`, de forma a informar o servidor o motivo pelo qual o processo terminou.

Se o tempo de execução máximo for atingido, o processo filho invoca o comando `kill` sobre ele próprio com o sinal `SIGKILL`. Se o processo pai verificar que o processo filho terminou com o sinal `SIGKILL`, este assume que o processo filho atingiu o tempo máximo de execução.

Como o servidor deve aceitar e executar várias tarefas ao mesmo tempo, este não espera que cada um dos seus processos filho termine. Mas como também precisamos de actualizar a lista de tarefas de forma a remover as tarefas terminadas, tivemos de implementar um handler para o sinal `SIGCHLD` para podermos verificar qual o estado final de execução de cada filho. Este handler começa por verificar se o processo terminou com o sinal `SIGKILL`. Em caso afirmativo, remove o item da lista de tarefas e escreve para o histórico que este terminou por exceder o tempo de execução. Caso contrário, vai verificar qual o valor de retorno do filho. Se o valor for 0, indica que a tarefa terminou com sucesso, se for 2 indica que terminou por inactividade, e se for -1 indica que terminou em erro.

```
jc@jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
argus$ listar
Não existem tarefas a executar.

argus$ ^C
(base) jc@jc-FX503VD:~/Uminho/SO/trabalho$ ./argus

argus$ executar "ls"
nova tarefa #1

argus$
```

```
jc@jc-FX503VD: ~/Uminho/SO/trabalho
File Edit View Search Terminal Help
A iniciar o Servidor...
FIFO cliente servidor foi criado com sucesso.
FIFO servidor cliente foi criado com sucesso.
log_fd foi aberto com sucesso.
historico_fd foi aberto com sucesso.
FIFO cliente servidor foi aberto com sucesso.
FIFO servidor cliente foi aberto com sucesso.
FIFO servidor cliente fechado.
FIFO cliente servidor fechado.
```

```
historico
~/Uminho/SO/trabalho
Open Save
```

```
#1, concluida: ls
```

```
Plain Text Tab Width: 8 Ln 1, Col 18 INS
```

```
log
~/Uminho/SO/trabalho
Open Save
```

```
##### OUTPUT TAREFA 1
argus
argus.c
argusd
argusd.c
argusd.o
argus.o
cliente.c
cliente.h
cliente.o
enunciado.pdf
fifo_client_server
fifo_server_client
HELP
historico
log
Makefile
mysolib.c
mysolib.h
mysolib.o
servidor.c
servidor.h
servidor.o
TP - informações adicionais.pdf
```

```
Plain Text Tab Width: 8 Ln 25, Col 32 INS
```

5 Análise de resultados

Atendendo ao que nos foi proposto no enunciado do trabalho pratico, pensamos que as funcionalidades mínimas estão dentro do que era pedido por isso achamos que tivemos sucesso nessa parte. Infelizmente, e apesar de termos tentado, não conseguimos realizar a funcionalidade adicional. Apenas estamos a enviar o output de cada tarefa para o ficheiro log.

6 Conclusão

Apesar de termos trabalhado para tentar implementar todas as funcionalidades mínimas, acabamos por ter alguns problemas que não conseguimos resolver facilmente. Por exemplo, a implementação de 2 funcionalidades que controlam o tempo que cada tarefa tem para executar levou-nos a implementar 2 alarmes que aparentavam ser semelhantes, o que acabou por nos tirar mais tempo do que pretendíamos. Acabamos este trabalho com a ideia de que talvez fosse melhor implementar menos funcionalidades, dedicando mais tempo a implementar e testar as restantes.

Contudo, tentamos aplicar as várias soluções que usamos nas aulas. Usamos redirecionamento para encadear o output de cada comando dentro de cada tarefa juntamente com pipes anónimos, implementamos FIFOs para a comunicação entre cliente e servidor (como pedido no enunciado), sinais para controlar os tempos de execução e inactividade e para verificar o estado final de cada processo filho do servidor, entre outros.

Bibliografia

AGUIRRE, L. A. Introdução à Identificação de Sistemas, Técnicas Lineares e Não lineares Aplicadas a Sistemas Reais. Belo Horizonte, Brasil, EDUFMG. 2004.

Anexo