



Universidade do Minho

Escola de Engenharia - Departamento de Informática

# **Relatório do Trabalho Prático**

Mestrado Integrado em Engenharia Informática

Sistemas Operativos

Grupo 20

A76936 – Luís Ferreira

A84610 – Gonçalo Almeida

A89575 – Joel Martins

Braga, Junho 2020

# 1. Introdução

Este trabalho prático, realizado no âmbito da unidade curricular de Sistemas Operativos, tem como objetivo o desenvolvimento de um serviço de monitorização de execução e de comunicação entre processos. Este serviço permite que um utilizador (cliente) submeta sucessivas tarefas a um servidor que posteriormente as irá tratar. As tarefas poderão ser desde simples comandos até a comandos encadeados. O cliente poderá ainda definir um tempo máximo de execução de cada tarefa e um tempo máximo de inatividade de comunicação com o servidor, bem como terminar uma tarefa em execução e listar, tanto as tarefas em execução, como as tarefas terminadas. A implementação deste trabalho foi feita com base em system calls e criação de novos processos, e a comunicação cliente – servidor foi realizada via pipes com nome. Ao longo deste relatório falaremos dos módulos que criamos para a construção da aplicação assim como o seu funcionamento.

## 2. Arquitetura da aplicação

### 2.1. Configuração

No módulo config estão implementadas funções que irão servir de suporte para a interação do cliente com o servidor.

Este módulo consiste na implementação de uma estrutura de dados chamada Config, que irá conter os dados essenciais a serem transmitidos ao servidor, dados esses que consistem em apenas dois inteiros, um que guardará o “id” do comando, ou seja, para facilitar a comunicação com o servidor decidimos estabelecer uma numeração fixa que servirá como identificador do comando, outro guardará o argumento do comando, comandos estes que simbolizam as funcionalidades da aplicação.

```
typedef struct _CONFIG_ {  
    int cmd;  
    int option;  
} Config;
```

Figura 1 - Estrutura Config

```
#define CONFIG_INAC_TIME 1  
#define CONFIG_EXEC_TIME 2  
#define CONFIG_EXEC 3  
#define CONFIG_LIST 4  
#define CONFIG_KILL 5  
#define CONFIG_HIST 6  
#define CONFIG_HELP 7
```

Figura 2 - Lista de tipos de config

### 2.2. Tarefa

Este módulo implementa funções que irão servir de suporte para a execução de comandos pelo servidor, assim como uma estrutura de dados, chamada Tarefa, essencial na aplicação, uma vez que conseguimos garantidamente saber que todas as tarefas terão igual tamanho em memória. A estrutura Tarefa guarda um inteiro que servirá de identificador de tarefa, uma matriz de char que guardará os comandos para serem executados (cada linha guarda apenas um comando e os seus possíveis argumentos) que deverão ser encadeados, uma matriz de inteiros para guardar os descritores dos pipes anônimos, um array de inteiros para guardar os PIDs dos processos, um inteiro com o número de processos terminados, um inteiro que irá conter o tempo em segundos que uma tarefa tem para ser executada (tempo de vida) e por fim um estado que dirá se a tarefa está em execução ou em espera ou terminada.

```
typedef enum {WAITING, RUNNING, TERMINATED} Estado;  
  
typedef struct _TAREFA_ {  
    int id;  
    char comandos[MAX_COMANDOS][MAX_COMANDO_SIZE];  
    int ncomandos;  
    int fds[MAX_COMANDOS - 1][2];  
    pid_t pids[MAX_COMANDOS];  
    int terminated_pids;  
    int ttl;  
    Estado estado;  
} Tarefa;
```

Figura 3 - Estrutura Tarefa

## 2.3. Interpretador

Este é apenas um módulo usado para simplificar a arquitetura da aplicação, uma vez que apenas implementa a shell utilizada pelo cliente.

## 2.4. Cliente

O módulo cliente é onde está implementada a base das funcionalidades do cliente. Este tem dois modos de interação com o servidor, uma será diretamente através do terminal, outra será através uma shell que irá interpretar os comandos. Ambas as implementações usam como base para o envio de dados para o servidor uma estrutura Config e Tarefa.

Será sempre criada uma Config com os dados do comando usado pelo cliente, tanto na interpretação direta como na interpretação pela shell, exceto para o comando ajuda, uma vez que esse não interage com o servidor.

A Tarefa será criada apenas quando o cliente quiser usar o comando para execução, nela irão ser guardados os dados relativamente aos comandos a serem executados.

## 2.5. Servidor

Este é o módulo principal da aplicação, é aqui que se faz a leitura dos dados enviados pelo cliente e seu processamento. Este tem um array global onde guarda as tarefas enviadas pelo cliente que estão em espera ou em execução, assim como dois inteiros com um valor default para o tempo de inatividade entre pipes anônimos e outro com o tempo de execução. Ambos estes valores serão aplicados na receção das futuras tarefas e poderão vir a ser alterados pelo cliente com os comandos apropriados para isso.

O servidor começa por criar o pipe com nome, que servirá de intermediário para a comunicação cliente-servidor, alterar as rotinas de tratamento do sinal SIGCHLD e SIGALRM, e então inicia a sua função geral. Este estará constantemente (em espera ocupada) a tentar ler do pipe estruturas Config, de modo a saber como proceder. Após a deteção de uma, verifica o tipo de comando recebido e procede conforme este.

```
static int tempo_inatividade = 10;
static int tempo_execucao = 2;

static Tarefa tarefas[MAX_TAREFAS];
static int ntarefas = 0;
```

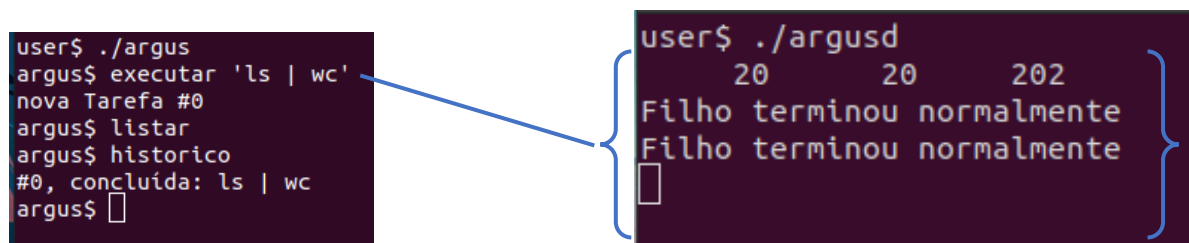
*Figura 4 - Variáveis globais*

### 3. Detalhes

Começamos por implementar a execução de uma tarefa submetida ao servidor. Isto acabou por nos dar já uma arquitetura bastante completa para a aplicação uma vez que foi necessário criar praticamente tudo que depois veio a ser reutilizado pelas outras funcionalidades, desde a criação do pipe com nome para comunicação, criação da estrutura para conter os dados a ser transferidos, criação da estrutura do servidor para armazenar as tarefas, até a implementação da própria shell a ser usada no cliente assim como a função para execução de qualquer tipo de comandos encadeados.

### 4. Demonstrações

Em seguida está apresentado um exemplo da interação do servidor ao comando executar, este apresenta os resultados da execução assim como o estado de saída de cada processo criado para executar cada comando.



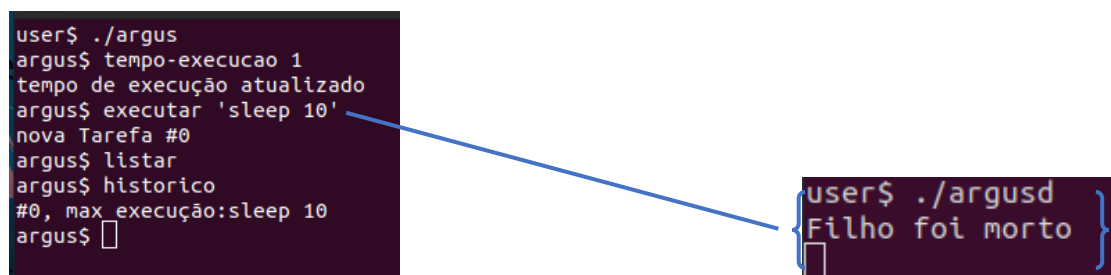
```
user$ ./argus
argus$ executar 'ls | wc'
nova Tarefa #0
argus$ listar
argus$ historico
#0, concluída: ls | wc
argus$
```

```
user$ ./argusd
      20      20      202
Filho terminou normalmente
Filho terminou normalmente
█
```

A blue arrow points from the command 'ls | wc' in the first terminal to the output in the second terminal.

*Figura 5 – Demonstração do comando executar*

Como podemos ver ao se pedir ao servidor a listagem de tarefas a executar, não nos é apresentado nada, pelo contrário, se pedirmos o histórico, é nos apresentada a tarefa acompanhada de uma mensagem que informa como o motivo de ter terminado.



```
user$ ./argus
argus$ tempo-execucao 1
tempo de execução atualizado
argus$ executar 'sleep 10'
nova Tarefa #0
argus$ listar
argus$ historico
#0, max execução:sleep 10
argus$
```

```
user$ ./argusd
Filho foi morto
█
```

A blue arrow points from the command 'sleep 10' in the first terminal to the output in the second terminal.

*Figura 6 – Demonstração de uma tarefa que excedeu o tempo máximo de execução*

```

user$ ./argus
argus$ tempo-execucao 5
tempo de execução atualizado
argus$ executar 'ls | wc'
nova Tarefa #0
argus$ executar 'cut -f7 -d: /etc/passwd | uniq | wc -l'
nova Tarefa #1
argus$ executar 'pwd'
nova Tarefa #2
argus$ executar 'ls -l -la'
nova Tarefa #3
argus$ executar 'sleep 10'
nova Tarefa #4
argus$ listar
argus$ historico
#0, concluida: ls | wc
#1, concluida: cut -f7 -d: /etc/passwd | uniq | wc -l
#2, concluida: pwd
#3, concluida: ls -l -la
#4, max execução:sleep 10
argus$ 

```

*Figura 7 - Funcionamento geral do modo shell*

```

user$ ./argus -e 'cut -f7 -d: /etc/passwd | uniq | wc -l'
user$ ./argus -m 1
user$ ./argus -e 'sleep 10'
user$ 

```

```

user$ ./argusd
{
Filho terminou normalmente
Filho terminou normalmente
13
Filho terminou normalmente
}
{
A matar tarefa
Tarefa terminada
Filho foi morto
}

```

*Figura 8 -Funcionamento geral do modo direto*

## 5. Conclusões

Em suma, o grupo considera que o trabalho está bem estruturado e que atinge bem grande parte dos requisitos pretendidos. O requisito do tempo de inatividade foi implementado, mas devido a umas incongruências na sua execução, este não se encontra operacional.

Um requisito que poderia ter sido implementado, mas que devido ao tempo não foi, seria: consultar o standard output produzido por uma tarefa já executada.

Temos também a destacar que a resolução e acompanhamento dos guiões nas aulas ajudou imenso a construção do projeto, uma vez que houve bastantes funções que já estavam implementadas dos guiões, como por exemplo, a função para execução de qualquer tipo de comandos encadeados assim como o próprio interpretador de comandos.

A nossa maior dificuldade esteve na principalmente na implementação do tempo de inatividade e tempo de execução, mas felizmente conseguimos implementar o segundo.

Contudo, achamos que o trabalho, como método de aprendizagem cumpriu bem o seu objetivo, além de ter sido um dos, ou talvez o projeto mais interessante realizado por nós.