Escola de Engenharia
Universidade do Minho

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
**Mestrado Integrado em Engenharia Informática**
***Sistemas Operativos***

# *Controlo e Monitorização de Processos e Comunicação*

## Grupo 52

**Figura 1:** Lucas Pereira a68547

Braga, 16 de Junho de 2020

# Conteúdo

**Resumo**

O documento descreve o desenvolvimento do projeto onde foi pedida a implementação de um serviço de **Controlo e Monitorização de Processos e Comunicação**. Através do paradigma **Cliente-Servidor** a ideia é imitar a bash do sistema Unix, tirando partido da biblioteca de chamadas ao sistema.

# 1. Introdução

O presente relatório documenta o trabalho prático referente a Unidade Curricular de Sistemas Operativos pertencente ao plano de estudos do 2º ano do Mestrado Integrado em Engenharia Informática. Neste projeto é pretendida a construção de 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 da 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.

Este documento está divido em 6 capítulos de forma a organizar a informação. No presente capítulo fazemos uma breve introdução ao projeto.

No segundo faremos uma descrição geral do projeto.

No terceiro explicaremos os ficheiros que compõem e definem a arquitetura do nosso projeto.

Relativamente ao quarto capítulo descrevemos tanto as funcionalidades básicas como as avançadas que efetivamente estão implementadas.

De forma a visualizar o funcionamento do nosso projeto, criamos o quinto capítulo onde exibimos os exemplos de teste executados.

Por fim, teremos o capítulo de conclusões e trabalho futuro que como o próprio nome indica explicamos o que fizemos e o que poderíamos ter feito, assim como os aspetos a melhorar.

# 2. Descrição geral do projeto

Neste trabalho foi pedido a construção de um serviço de controlo e monitorização de processos e comunicação à semelhança de uma shell de um sistema Unix. Através do modo de linha de comandos ou de um interpretador que recebe comandos específicos, é possível executá-los através deste serviço. O cliente envia o respetivo comando para o servidor e este devolve uma mensagem, em que esta é o output do comando desejado.

As funcionalidades que este serviço disponibiliza são:

- definir o tempo máximo (segundos) de inactividade de comunicação num pipe anónimo (opção *-i n* da linha de comando)
- definir o tempo máximo (segundos) de máximo de uma tarefa (opção *-m n* da linha de comandos)
- executar uma tarefa (opção *-e "p1 | p2 ... | pn"* da linha de comando)
- listar tarefas em execução (opção *-l* da linha de comando)
- terminar uma tarefa em execução (opção *-t n*)
- apresentar ajuda à sua utilização (opção *-h*)

# 3. Arquitetura do projeto

## 3.1 Implementação

### 3.1.1 argusd.c

Neste ficheiro está a implementação do servidor do serviço Argus (Daemon). Após a execução, este fica sempre a correr até que surja um sinal de interrupção. Tem um ponto de entrada que é a **Main** e após a receção de uma mensagem este faz o parse da mensagem e executa o comando, devolvendo a mensagem de resposta ao cliente.

### 3.1.2 argus.c

Neste ficheiro está a implementação do cliente do serviço Argus. Este cliente tem um modo de funcionamento sobre a linha de comandos e outro em modo CLI(command line interpreter). Ambos os modos chamam as mesmas funções de interação com o servidor. Após receção do comando por parte do utilizador, o cliente envia a mensagem ao servidor com o comando e fica à espera de uma resposta.

### 3.1.3 lib.c

Neste ficheiro tem implementações auxiliares tanto ao servidor do serviço como ao cliente.

## 3.2 Makefile

Com o objetivo de automatizar tarefas para o nosso projeto definimos a seguinte *Makefile*:

```
CXX      = gcc
BIN_SERVER = argusd
BIN_CLIENT = argus
CXXFLAGS  = -Wall -Wextra -g
#####################################################################
BIN_DIR  = ../bin
SRC_SERVER = argusd.c
SRC_CLIENT = argus.c
SRC_LIB  = lib.c
#####################################################################

all: argusd argus

argusd: $(SRC_SERVER) $(SRC_LIB)
$(CXX) $(CXXFLAGS) -o $(BIN_DIR)/$(BIN_SERVER) $(SRC_SERVER) $(SRC_LIB)

argus: $(SRC_CLIENT) $(SRC_LIB)
$(CXX) $(CXXFLAGS) -o $(BIN_DIR)/$(BIN_CLIENT) $(SRC_CLIENT) $(SRC_LIB)

clean:
rm -f $(BIN_DIR)/*
```

# 4. *Argus*

## 4.1 Executar comandos

O cliente do serviço recebe o comando que o utilizador pretende executar e faz parse ao mesmo para chamar a respetiva função que tem que executar. Apoś o reconhecimento da função, o comando é enviado para o servidor com uma etiqueta para ser mais fácil o seu reconhecimento. Após o envio, o cliente fica à espera de uma resposta.

- Etiqueta executar - 900
- Etiqueta tempo-inatividade - 901
- Etiqueta tempo-execução 902
- Etiqueta listar 903
- Etiqueta terminar 904
- Etiqueta histórico 905
- Etiqueta ajuda 906

Do lado do servidor, é feito novamente um parse para identificação do comando a executar. O parse é feito através do reconhecimento se a tarefa tem pipes anónimos ou não. A sua execução passa por um ciclo em que é manipulada a chamada ao sistema **dup2** para ser feito o redirecionamento de um executável para o outro.

## 4.2 Definir o tempo de inativade de comunicação entre pipes anónimos

Mais uma vez, o cliente do serviço faz um parse deste comando e envia uma mensagem ao servidor com a respetiva etiqueta. O servidor por sua vez, ativa este modo e define o tempo de timeout. Quando o comando está a ser executado, o alarme é posto com o tempo definido e quando este dispara, todos os processos que estavam a executar comandos são terminados e o servidor resume o seu programa.

## 4.3 Definir o tempo de execução de uma tarefa

Do mesmo modo que se definiu o tempo de inativade é definido o tempo de execução de uma tarefa. Caso um programa esteja a executar e o alarme disparar por este ter ultrapassado o limite de tempo, os processos que estão a executar os comandos são terminados e o servidor resume o seu programa.

## 4.4 Listar

Não implementado.

## 4.5 Terminar

Não implementado.

## 4.6 Histórico

Não implementado.

## 4.7 Ajuda

Este comando imprime um pequeno conjunto de frases para ajuda ao utilizador.

# 5. Funcionamento do *Argus*

## 5.1 Input

A título de exemplo, apresento de seguida alguns comandos possíveis e o seu respetivo resultado. Foram feitos alguns scripts de teste para automatizar este processo sendo estes:

### 5.1.1 Script 1

```
./argus -e "cut -f7 -d: /etc/passwd | uniq | wc -l"
./argus -e "cut -f7 -d: /etc/passwd | uniq"
./argus -e "cut -f7 -d: /etc/passwd"
```

### 5.1.2 Script 2

```
./argus -e "cat /etc/passwd | head -10 | tail -5"
./argus -e "cat /etc/passwd | head -10"
```

### 5.1.3 Script 3

```
./argus -m 3
./argus -e "cat"
./argus -e "cat | cat"
```

## 5.2 Output

### 5.2.1 Script 1

```
13

/bin/bash
/usr/sbin/nologin
/bin/sync
(...)
/usr/sbin/nologin
/bin/bash
/usr/sbin/nologin
/bin/false

/bin/bash
```

```
/usr/sbin/nologin
/usr/sbin/nologin
/usr/sbin/nologin
/usr/sbin/nologin
/usr/sbin/nologin
/usr/sbin/nologin
(...)
/usr/sbin/nologin
/usr/sbin/nologin
/usr/sbin/nologin
/bin/bash
/usr/sbin/nologin
/bin/false
/bin/false
```

### 5.2.2 Script 2

```
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

### 5.2.3 Script 3

```
1-Execution timeout set
```

# 6. Conclusões

Este trabalho não ficou completo faltando as funcionalidades **Listar, Terminar e Histórico**. No entanto, o trabalho desenvolveu bem a parte de controlo de comunicação e execução de processos e com mais um bocado de trabalho as funcionalidades que faltam seriam implementadas. Trabalhar a baixo nível é complicado pois quando surge um erro nem sempre é claro qual é a sua origem.

## 6.1 Trabalho Futuro

Como trabalho futuro, este passaria por implementar as 3 funcionalidades que faltam. Também seria interessante tornar este serviço num verdadeiro paradigma Cliente-Servidor em que este aceita várias conexões e consegue dar resposta.