****

Aplicação de servidor

**Servidor**

Sistemas Operativos

Grupo 1 – Turma 3

Mestrado Integrado em Engenharia Informática e Computação

Angela Cruz up201806781

Eduardo

Filipe Pinto up201907747

Luísa Mesquita up201704805

Pedro Moreira up201904642

**Índice**

[**1. Introdução**](#_kjb3vmod2i3a)2

[**2. Estruturas de dados e variáveis**](#_lsawsgdwkpt4)3

[2.1. Log](#_10tx5lbdffo4) 3

[2.2. Message](#_xugz79wl4o5z) 3

[2.3. Queue](#_xsc61o3sy0hw) 4

[2.4. Arguments](#_75ui4xua9cb4) 4

[2.5. ArgsThreadsProducer](#_2lo96ajf5gh) 4

[2.6. ArgsThreadsConsumer](#_2dofque7jq9s) 5

[2.7. Variáveis globais](#_rjhvz9g6kp3r) 5

[**3. Testes**](#_c8clmfwrx6lv)6

[3.1. VALGRIND](#_gh6ufdm2cykf) 6

[3.2. CPPLINT](#_v3vmrphlk6vj) 6

[3.3. INFER](#_158ajrsqxuv4) 6

[3.4. SCRIPT](#_p7qgz8ge0dy4) 7

[**4. Compilação e execução**](#_gm5xm4rkarlx)9

[4.1. Compilação](#_5788f2q1wmrp) 9

[4.2. Execução](#_b1ncwsd572fc) 9

[**5. Conclusão**](#_7whmbs4zfimo)10

# 

# 1. Introdução

O objetivo do trabalho consiste em desenvolver uma aplicação do tipo cliente-servidor, sendo que nesta segunda parte o foco foi na parte do servidor. Os seguintes pontos definem o desenvolvimento do trabalho:

* Processamento dos argumentos introduzidos na linha de comandos.
* Tratamento do sinal SIGALRM.
* Abertura e fecho do FIFO público para leitura (O\_RDONLY) e em modo *nonblocking* (O\_NONBLOCK).
* Uso da função *task()* fornecida para obter a carga da tarefa pedida.
* Construção e uso de várias estruturas de dados (ver próxima seção)
* Escrita no *stdout* as mensagens: TSKEX, RECVD, FAILD, TSKDN
* Criação de um armazém (*cloud*) onde serão guardados os pedidos ao servidor.
* Criação de uma fila (*queue*) onde se guarda o primeiro e último pedido do armazém.
* Criação de várias funções auxiliares à *queue* que permitem colocar lá elementos, retirar elementos, verificar se está cheia ou se está vazia e verificar qual o elemento no topo da fila.
* Criação e tratamento das threads do tipo Consumidor e Produtor (é criado uma *thread* do tipo Consumidor e várias do tipo Produtor).
* Uso da função *alarm()* como medição do tempo de execução, sendo que um SIGALRM é gerado após decorrido o número de segundos introduzido na linha de comandos.
* Aplicação de *mutexes* como mecanismo de sincronização de forma a evitar conflitos entre *threads* concorrentes.
* Criação dos FIFOS privados nas *threads* consumidoras.

# 

# 2. Estruturas de dados e variáveis

As estruturas Message e Log são usadas tanto na parte do cliente como na parte do servidor, logo são guardadas num ficheiro em comum (common.h).

Foi criada uma estrutura de dados - Queue - como auxiliar ao trabalho, logo esta está colocada também num ficheiro à parte (queue.h).

As restantes estruturas e variáveis foram colocadas no ficheiro relacionado com o servidor (server.h).

## 2.1. Log

Esta estrutura guarda a informação a ser imprimida na consola. Sempre que ocorre uma determinada operação *oper*, atribui-se o respetivo valor a estes campos e chama-se a função *WriteLog()*, que imprime essa informação para o stdout.

struct Log

{

time\_t inst; //return value of time() function

int i; //unique request number

int t; //task load

pid\_t pid; //process ID

pthread\_t tid; //thread ID

int res; //task result

char \*oper; //operation made

};

## 2.2. Message

Estrutura solicitada no enunciado que guarda as informações necessárias para mandar ao servidor.

struct Message

{

int rid; // request id

pid\_t pid; // process id

pthread\_t tid; // thread id

int tskload; // task load

int tskres; // task result

};

## 

## 

## 

## 2.3. Queue

Esta estrutura guarda o primeiro elemento e último elemento a guardar na fila.

struct Queue

{

int first; //index of the first element of the queue (-1 if queue is empty)

int last; //index of the last element of the queue (-1 if queue is empty)

};

## 

## 

## 2.4. Arguments

Esta estrutura guarda todas as informações introduzidas na linha de comandos e que são depois usadas ao longo do programa. A inicialização dos campos da estrutura é feita na função *ParseArguments()*.

struct Arguments

{

size\_t nsecs; //number of seconds

int buffer\_size; //buffer size

char public\_fifo[100]; //public FIFO

};

## 

## 

## 2.5. ArgsThreadsProducer

Esta estrutura guarda variáveis a ser usadas na função *ThreadHandlerProd()*, ou seja, na *thread* produtora.

struct ArgsThreadsProducer

{ int rid; // request id

pid\_t pid; // process id

pthread\_t tid; // thread id

int tskload; // task load

int tskres; // task result

struct Message \*cloud ;

int nmax; //maximum number of elements in cloud

};

## 

## 

## 2.6. ArgsThreadsConsumer

Esta estrutura guarda variáveis a ser usadas na função *ThreadHandlerCons()*, ou seja, nas *threads* consumidoras.

struct ArgsThreadsConsumer

{

struct Message \*cloud;

int nmax;

}

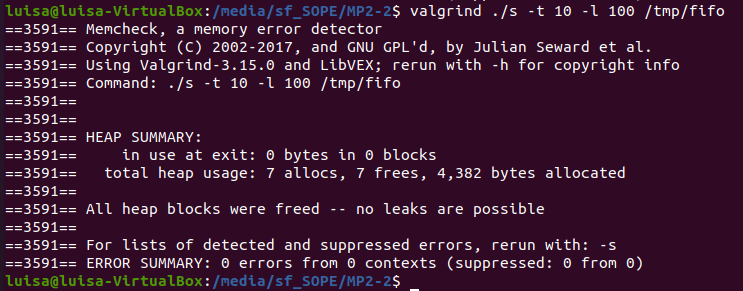
## 2.7. Variáveis globais

* **errno:** guarda o número do último erro ocorrido.
* **finish:** variável sempre a falso até o *handler* do sinal SIGALRM colocá-la a verdadeiro, altura em que a *thread* principal (*main*) parará de criar novas threads produtoras.
* **finishCons:** variável sempre a falso até a *thread* principal (*main*) parar de criar *threads* produtoras.

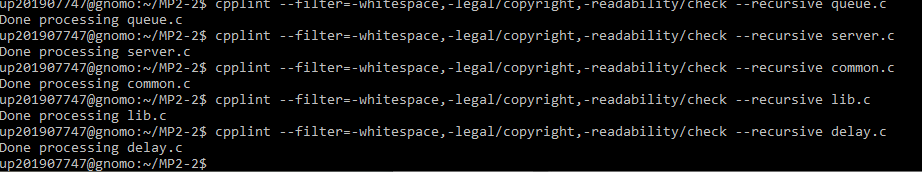
# 

# 3. Testes

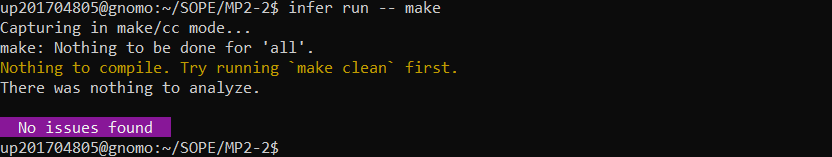
## 3.1. VALGRIND



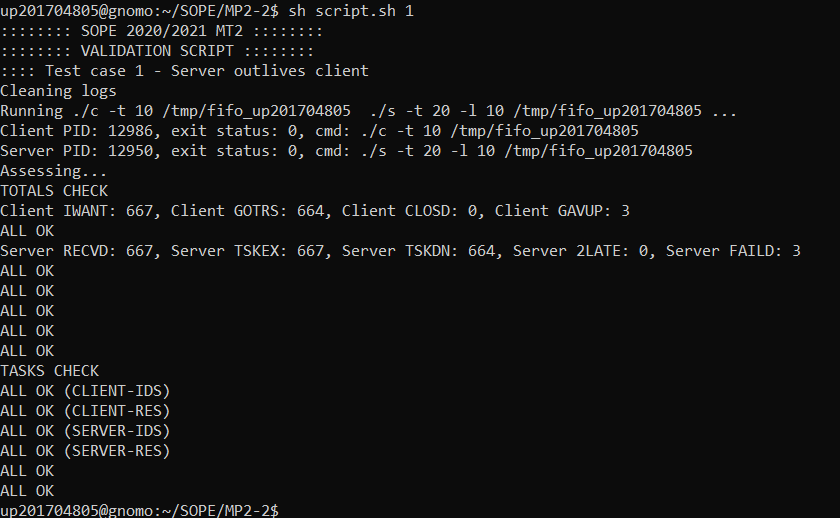
## 3.2. CPPLINT

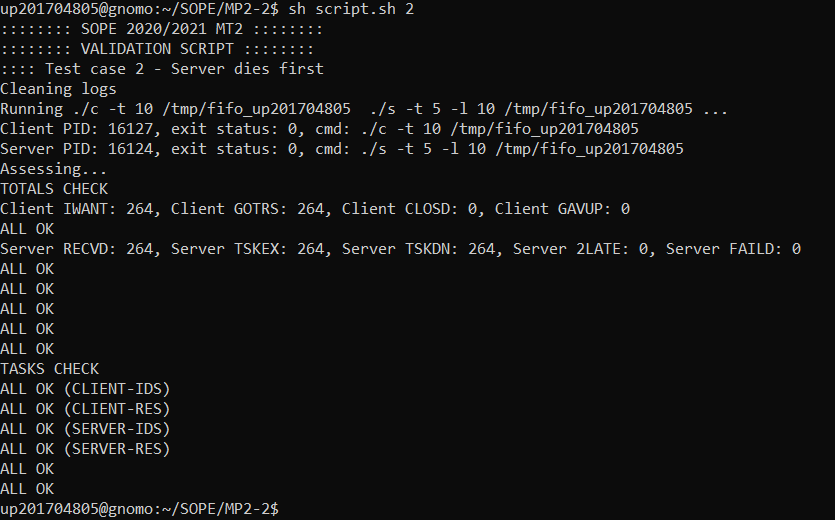


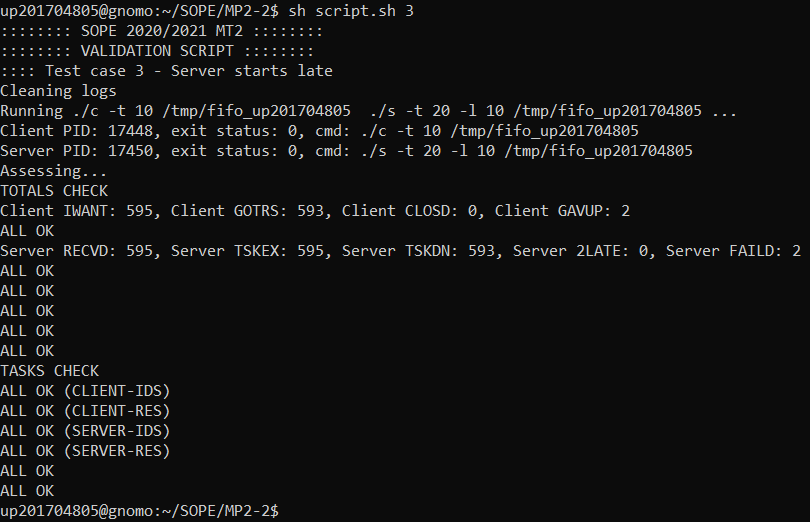
## 3.3. INFER



## 3.4. SCRIPT



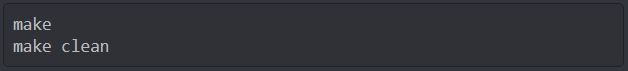




# 

# 4. Compilação e execução

## 4.1. Compilação



ou





## 4.2. Execução





# 

# 

# 5. Conclusão

As seguintes percentagens correspondem à dedicação e consistência ao longo do tempo por parte de cada um a esta parte do trabalho.

|  |  |
| --- | --- |
| Angela Cruz | 50% |
| Eduardo | 0% |
| Filipe Pinto | 25% |
| Luísa Maria Mesquita | 25% |
| Pedro Moreira | 0% |