U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Application Server

Relatório de projeto

Sistemas Operativos (SOPE) – MP2a

Mestrado Integrado em Engenharia Informática e Computação (MIEIC)
2ºAno 2ºSemestre - 2020/2021

Turma 7 - Grupo 4:

Ana Matilde Barra up201904795
Ângela Coelho up201907549
Marta Mariz up201907020
Nuno Castro up202003324
Patrícia Oliveira up201905427

Índice

1 – Introdução

Este projeto surge no âmbito da unidade curricular de **Sistemas Operativos,** do curso Mestrado Integrado em Engenharia Informática e Computação, em que nos foi proposta a elaboração de um "*application server*", com o objetivo de pôr em prática a criação de programas ***multithread*** em Unix/Linux. Para além disso, também tem como finalidade promover a intercomunicação entre processos através de canais com nome (*named pipes* ou FIFOs) e coordenação no acesso à memória partilhada pelas *threads*, de forma a evitar conflitos, colisões e *deadlocks* entre entidades concorrentes.

De notar que esta primeira parte foca-se apenas no desenvolvimento da comunicação por parte do cliente.

2 – Detalhes de implementação

2.1 – Informações gerais

2.1.1 – Invocação

A nossa ferramenta “Cliente” suporta apenas a seguinte invocação:

– c <-t **nsecs**> <**fifoname**>;

Sendo os argumentos:

– **nsecs** – o tempo aproximado em segundos no qual o programa **c** deve correr;

– **fifoname** – o nome (absoluto ou relativo) do canal público de comunicação (FIFO) entre o Cliente (programa **c**) e o Servidor (programa **s**).

De reforçar que a ordem dos argumentos de entrada é respeitada escrupulosamente.

2.1.2 – Funcionamento

Existe uma *thread* principal, criada na função *main* do cliente (*client.c*), que trata de criar *threads* de pedido (entre intervalos de milissegundos “pseudo-aleatórios” – de 50 a 100 – durante *nsecs*), distinguidas por números identificadores únicos, que gerem a comunicação com o servidor. São tratadas igualmente algumas situações especiais:

– o cliente espera no máximo 5 segundos que o *server* abra a FIFO pública;

– a *thread* principal só cria *threads* de pedido após confirmar que a FIFO pública se encontra aberta. No entanto, só dará *timeout* após o número de segundos determinado na invocação do programa.

2.2 – Registo de Operações

O registo de operações é realizado na saída padrão, *stdout*, através da função *writeLog()*, que se encontra no ficheiro ***src/utils.c***, e recebe como argumentos a *struct* de mensagem entre o *server* e o *client* e também a operação a ser registada.

```
void writeLog(const struct message *msg, const char* oper){
    // inst ; i ; t ; pid ; tid ; res ; oper
    fprintf(stdout, "%ld ; %d ; %d ; %d ; %ld ; %d ; %s\n", time(0), msg->rid, msg->tskload, msg->pid, msg->tid, msg->tskres, oper);
}
```

As operações a registar por parte do cliente são as seguintes:

– sempre que o que cliente realiza um pedido regista **IWANT**;

– quando se recebe com sucesso o resultado de um pedido regista-se **GOTRS**;

– quando o cliente ultrapassa o seu tempo de funcionamento, todas as *threads* de pedido em espera de resposta por parte do servidor, que não consigam ser atendidas a tempo, são terminadas registando-se **GAVUP**;

– quando um pedido não consegue ser atendido a tempo pelo servidor, receberá um resultado de -1 como resposta e dará sinal de **CLOSD**.

2.3 – Considerações acerca dos testes realizados

2.3.1 – *Cpplint*

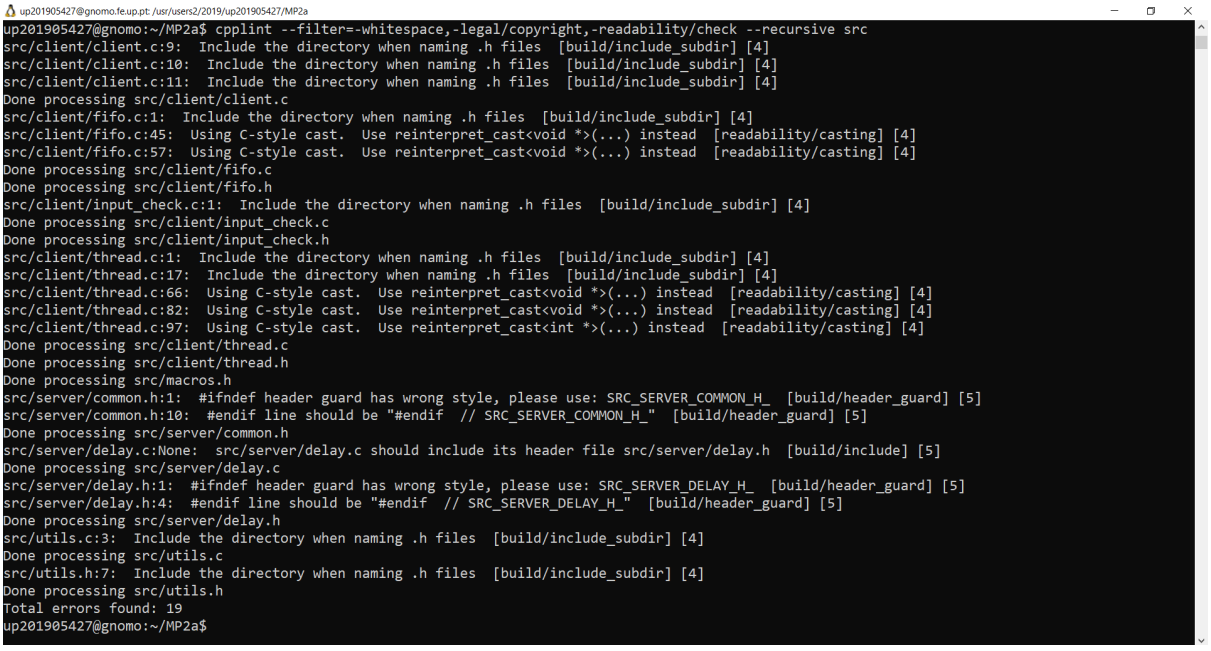
Os nomes dos nossos *defines* em cada *header file* foram definidos em concordância com o recomendado nos testes da ferramenta *cpplint*.

Achamos também importante referir que não foram considerados os avisos, referentes aos formatos de *includes,* gerados pela ferramenta de teste *cpplint*, uma vez que estes resultados se contradiziam no *gnomo* e nos nossos computadores pessoais.

Assim, o formato que utilizámos para os nossos *includes* foi o seguinte:

- **#include "../nomeficheiro.h"** – para ficheiros da pasta **src** a incluir em ficheiros da pasta **client;**
- **#include "nomeficheiro.h"** – para ficheiros pertencentes à mesma pasta (ex.: *includes* no ficheiro **src/client/client.h**).

Foi ainda obtido um aviso relativo à forma como foi realizado o cast na função *pthr_request()* no ficheiro *thread.c*. Isto acontece porque não há nenhuma configuração para estabelecer que a *target language* é na verdade C e não C++, acabando por ser ignorado este aviso.



2.3.2 – *Valgrind*

Com esta ferramenta, foi testado o uso de memória por parte do nosso programa. Consideramos este teste importante, dado que nos permite verificar se existe alguma memória alocada que não esteja a ser libertada no final da execução do programa. De forma a sintetizar a informação pretendida foi selecionada apenas a parte inicial e fim deste teste.

```
mbarra@mbarraVB:~/Desktop/SOPE_git/FEUP-SOPE-PROJ2$ valgrind --leak-check=yes ./c -t 10 /tmp/fifo_sope & ./s -t 20 -l
10 /tmp/fifo_sope
[1] 12696
[server] initial time: 1619697822, expected final time: 1619697842
[server] buffer size: 10
[server] got: nsecs=20, bufsize=10, fifoname=/tmp/fifo_sope
==12696== Memcheck, a memory error detector
==12696== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==12696== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==12696== Command: ./c -t 10 /tmp/fifo_sope
==12696==
```

```
==12696==
==12696== HEAP SUMMARY:
==12696==     in use at exit: 0 bytes in 0 blocks
==12696==   total heap usage: 72 allocs, 72 frees, 20,952 bytes allocated
==12696==
==12696== All heap blocks were freed -- no leaks are possible
==12696==
==12696== For lists of detected and suppressed errors, rerun with: -s
==12696== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[server] all clients closed, will wait for new ones
[server] timeout reached: 1619697842
[server] server, open serverfifo: Interrupted system call
[server] unlinked server fifo
[server] no longer accepting requests
[server] no. running: 0
[1]+  Done                    valgrind --leak-check=yes ./c -t 10 /tmp/fifo_sope
```

2.3.3 – *Infer*

Através desta ferramenta de teste, foi realizado apenas o teste apresentado na imagem seguinte, sem qualquer mensagem de erro a relatar.

```
up201905427@gnomo.fe.up.pt: /usr/users2/2019/up201905427/MP2a
up201905427@gnomo:~/MP2a$ infer run -- make
Capturing in make/cc mode...
gcc -Wall -DDELAY=100 -o s src/server/delay.c src/server/lib.o src/server/server.o -pthread
gcc -Wall -o c src/client/client.c src/utils.c src/client/input_check.c src/client/thread.c src/client/fifo.c -pthread
Found 6 source files to analyze in /usr/users2/2019/up201905427/MP2a/infer-out
6/6 [########################################################################] 100% 1.045s

  No issues found
up201905427@gnomo:~/MP2a$
```

2.3.3 – Considerações finais

Realçamos que tivemos sempre prioridade testar o nosso código no ***gnomo.fe.up.pt***. No entanto, ao testar com ***valgrind*** foram obtidos resultados diferentes para diferentes elementos do grupo, pelo que acabamos por não usar no ***gnomo*** esta ferramenta.

Além disso, é importante referir que relativamente ao *script* de teste disponibilizado, foram, mais uma vez, obtidos resultados diferentes para diferentes elementos do grupo ao utilizar o ***gnomo***. Contudo, fora do ***gnomo***, todos os testes passam sem qualquer problema para todos os membros do grupo.

2.4 – Informações adicionais

2.4.1 – Utilização de Semáforos

Recorremos a um *mutex* (semáforo binário) para controlar os acessos a um contador global (*counter*), partilhada por todas as threads de pedido, que determinará o seu número único. Desta forma evitam-se situações de colisão e acesso concomitante a esta variável.

2.4.2 – Permissões da FIFO "privada"

Na criação da FIFO privada por parte de cada *thread* de pedido foram usadas as permissões 0622, uma vez que o servidor apenas precisa de permissões de escrita nesta FIFO. Isto também dado que não se sabe se este servidor foi lançado pelo dono do cliente ou se pertence ao mesmo grupo, nem se pretende restringir esse tipo de situação.

2.4.3 – Notação para erros

De forma a identificar com maior facilidade a causa de possíveis erros, foi criada uma convenção de valores que aqui se segue.

```
#define ERROR -1
#define INPUT_ERROR 1
#define FIFO_ERROR 2
#define THREAD_ERROR 3
#define SEMAPHORE_ERROR 4
```

Esta atribuição encontra-se no ficheiro ***macros.h***.

3 – Estruturação do código

O código foi organizado por 2 pastas (***client*** e ***server***), subpastas do nosso *source folder*, **src**. A pasta ***server***, nesta fase do projeto, inclui os ficheiros fornecidos pelos professores da unidade curricular.

Na pasta ***client***, está contido todo o código que produzimos para esta entrega, dividido por vários módulos. Cada módulo tem a sua responsabilidade para o funcionamento do programa.

Através desta organização, tornou-se mais fácil a manutenção e gestão de todo o projeto, bem como a distribuição de tarefas pelos diversos elementos do grupo.

Todos os módulos definidos na pasta ***client*** e suas funções estão descritos na seguinte tabela:

client	Este módulo contém a ***main*** do programa **c**, sendo responsável pela criação do *thread* principal que irá posteriormente criar as *threads* de pedido.
inputcheck	Neste módulo é realizada a verificação do *input* proveniente da linha de comandos para posterior utilização pelos restantes módulos.
fifo	Este módulo contém funções que manipulam as FIFOs utilizadas na comunicação entre *client* e *server*.
threads	Módulo responsável pela criação de *threads* de pedido e comunicação destas com o servidor.

Adicionalmente, no nosso *source folder* temos também os seguintes ficheiros:

macros	Definição de variáveis globais úteis ao desenvolvimento de código.
utils	Este ficheiro inclui funções gerais de auxílio aos módulos da pasta ***client***.

4 – Autoavaliação

Abaixo apresentamos a divisão percentual (aproximada) da contribuição de cada membro do nosso grupo para o resultado final deste projeto:

- Ana Matilde Barra (up201904795) - 20%
- Ângela Coelho (up201907549) - 20%
- Marta Mariz (up201907020) - 20%
- Nuno Castro (up202003324) - 20%
- Patrícia Oliveira (up201905427) - 20%

Consideramos que todos os elementos do grupo trabalharam igualmente no trabalho e contribuíram ativamente para o seu desenvolvimento.