Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Sistemas Operativos

Ano Letivo de 2022/2023

Grupo 104

Gonçalo Peres Costa (a93309) Cláudio Alexandre Freitas Bessa (a97063) José Fernando Monteiro Martins (a97903)

13 de maio de 2023

Indice

1 Introdução

O documento apresentado visa apresentar as metodologias praticadas e processos utilizados durante a o projeto que tem vindo a ser realizado no âmbito da UC de Sistemas Operativos (SO).

Processos esses lecionados durante este semestre que englobam os métodos de ***fifos***, ***forks***, ***exec***. Existe uma clara divisão entre **cliente/servidor** que separa a interação direta com o utilizador e o processamento e todo o programa pedido para executar em questão.

Esta ligação de **cliente/servidor**, é necessário efetuar 2 (dois) programas, respetivamente ***tracer*** e ***monitor***. O servidor tem como objetivo manter em memória e em ficheiros a informação relevante ao suporte das diversas funcionalidades que foram implementadas. O cliente irá utilizar o *standard output*.

2 Cliente

2.1 Inicialização e interação

O cliente, programa *tracer*, é o meio dos utilizadores fazerem os seus pedidos. Sendo inicializado através de comandos no terminal, este envia para os pedidos para o servidor, onde o mesmo responde, enviando mensagens relativas ao processamento dos ficheiros. É possível fazer dois tipos de pedidos, através dos seguintes comandos:

- ***execute*** usado para executar determinado programas que são também definidos pelo utilizador nos argumentos dados como *input*, podendo receber a *flag* **-u** ou **-p** consoante se é apenas uma execução ou uma *pipeline* delas.
  e.g. $*./tracer execute -u [nome_programa] [args]*

- ***status*** usado para verificar o estado atual de programas que ainda não tenham terminado no servidor, i.e. segundo a nossa implementação ainda estejam na *hash table* do servidor.
  e.g. $*./tracer status*

2.2 Envio de pedidos

Após a validação dos argumentos fornecedidos, o programa *tracer* abre um **FIFO**.

Enviamos então uma mensagem pelo **FIFO** que contêm um identificador, que vai permitir ao *monitor* saber o que é suposto ele fazer com o pedido, o **PID** que vai permitir mais a frente guardar na *Hashtable* temporariamente e escrever no ficheiro de *log*, o nome do programa e por fim, a hora de início do mesmo.

Se o pedido foi recebido pelo servidor com sucesso, resta ao cliente aguardar as atualizações relativas ao respetivo pedido.

3 Servidor

3.1 Inicialização e configuração

O servidor, programa *monitor*, ocupa-se de processar os pedidos que recebe dos vários clientes. A sua inicialização, feita simplemente por,
e.g. ***$ ./monitor [PIDS-dir]***.

3.2 Receção de pedidos

O servidor, recebe pedidos dos clientes através da leitura assíncrona do *fifo* usado para a comunicação **cliente -> servidor**.

Neste o servidor pode receber 3 tipos de pedidos, o pedido de começo do servidor, o pedido de término do servidor, ou o pedido de status do servidor, que irá demonstrar qual os programas que ainda se encontram em execução.

3.3 Gestão de processos

3.3.1 *Hashtable*

Tendo em vista o que seria necessário para guardarmos os diferentes pedidos que ainda não obtiveram um fim, identificamos que o perfeito seria a utilização de uma *hashtable*, onde as chaves são respetivamente os **PIDs** dos processos recebidos por parte do cliente e os seus valores os programas em questão.

3.3.2 *Ficheiro de Log*

Para termos um documento que guardasse todas as ações efetuadas entre utilizador, servidor e cliente, o servidor guarda todos os comandos que lhe chegam. Guarda por isso o PID, o nome do programa e a hora de começo de execução do programa.

4 Mecanismos de comunicação

Foi necessário o desenvolvimento de mecanismos de comunicação inter-processos de forma a que o envio de pedidos ao servidor por parte dos clientes, assim como a execução de transformações em paralelo dentro do servidor fosse possível, permitindo assim concorrência entre os pedidos. Para isso recorreu-se a *pipes* anónimos, intitulados de **FIFOs** tendo em conta a sitaução para se fazer o uso correto.

4.1 Comunicação Cliente/Servidor

No caso da comunicação entre os clientes e o servidor e vice-versa, optou-se que esta fosse feita através de ficheiros **FIFO**, criado pelo servidor, ou cliente dependendo do sentido da ligação. De relembrar que existe no total 2 (dois) ficheiros **FIFO** não permitindo assim que o mesmo possua dois sentidos de entrada/saída.

4.2 Comunicação Inter-Processos

As sequências de transformações pedidas por um dado cliente são executadas, dentro do servidor, em concorrência com a receção de pedidos de outros clientes. Sendo um dos requisitos evitar a criação de ficheiros temporários relativos aos estados intermédios dos processos, foi fulcral o uso de *pipes* anónimos para estabelecer pontes de comunicação entre os processos filhos do servidor, de forma a que fossem utilizados processos em cadeia.

5 Componentes auxiliares

De forma a manter o código mais limpo, i.e. mantendo os ficheiros do servidor e do cliente apenas com funções principais, optamos por criar um ficheiro titulado de ***utils***.

Nesse mesmo ficheiro não contém somente funções auxiliares, e.g. *print_status*, *transformList*, etc, mas também tem funções que tem comportamentos similares a funções de alto nível de C, e.g. *printf*, *fprintf* mas desta vez de baixo nível, i.e. uma vez que se trata de um projeto em torno de sistemas operativos.

Seguindo agora as funções auxiliares presentes, a maioria acaba por ter um comportamento de conversão, sendo eles conversão entre tipos, e.g. *time_t_to_str*, *int_to_str*, outras também como escrita em ficheiro uma vez que estamos inconcessos de utilizar determinadas operações. É também visível algumas funções relativas à ***GLib*** uma vez, a sitaução já descrita, da utilização das *hash tables*.

6 Makefile

Sendo um dos requisitos a criação de um ficheiro ***Makefile*** capaz de carregar e preparar para a execução do *software* como um todo, foi um dos primeiros pontos a serem tratados. Tendo como exemplo o fornecido no enunciado,[1] efetuamos algumas pequenas alterações indo, obviamente, de acordo com a nossa estruturação para a solução do trabalho.

```
CC = gcc
CFLAGS = -Wall -g  -Wextra -pedantic -lm
PKG = `pkg-config --cflags --libs glib-2.0`

all: folders server client #utils
server: bin/monitor
client: bin/tracer
utils: bin/utils

folders:
        @mkdir -p src obj bin tmp

bin/monitor: obj/monitor.o obj/utils.o
        $(CC) -o bin/monitor -g obj/monitor.o obj/utils.o $(PKG
           )

bin/tracer: obj/tracer.o obj/utils.o
        $(CC) -o bin/tracer -g obj/tracer.o obj/utils.o $(PKG)

bin/utils: obj/utils.o
        $(CC) -o bin/utils -g obj/utils.o $(PKG)

obj/%.o: src/%.c
        $(CC) $(CFLAGS) -c $< -o $@ $(PKG)

clean:
        @echo "Cleaning..."
        rm -rf bin/*
        rm -rf obj/*.o
```

1. *"Enunciado Trabalho Prático"*, https://elearning.uminho.pt/bbcswebdav/pid-1329692-dt-content-rid-6753364_1/courses/2223.J304N1_2/TP_SO.pdf, Accessed: 2023-05-11.

7 Conclusão

O serviço implementado é funcional no que toca aos comandos requeridos. No entanto, há situações que se demonstram pouco eficientes podendo assim ser submitos a otimizações. Porém, com este trabalho passamos a conhecer novos mecanismos deste nível de programação, que até outrora nunca tinhamos trabalho de forma tão extensiva em tão baixo nível que a UC proporciona.

Bibliografia

"Enunciado Trabalho Prático". https://elearning.uminho.pt/bbcswebdav/pid-1329692-dt-content-rid-6753364_1/courses/2223.J304N1_2/TP_SO.pdf. Accessed: 2023-05-11.