Sistemas Operativos

MIEI - 2º ano - 2º semestre
Universidade do Minho

Controlo e Monitorizacao de Processos e Comunicacao

Grupo 61

Ana César A86038
Angélica Cunha A84398

Braga, 14 de Junho de 2020

1. Introdução

Este projeto insere-se na UC de Sistemas Operativos, e tem como propósito desenvolver um serviço de monotorização de execução e de comunicação entre processos, utilizando a linguagem C, de forma a aplicar os conhecimentos adquiridos durante o semestre, nomeadamente nos guiões práticos.

2. Conceção e Estrutura

2.1 Abordagem

Para responder às necessidades de implementação impostas pelo enunciado, foi feita um planeamento prévio detalhado de toda a estruturação do programa e das estruturas de dados necessárias à concretização deste projeto.

Tal como referido no enunciado, foi cumprida a comunicação cliente servidor através de pipes com nome, e para isso são criados dois pipes com nome:

- pelo servidor - o servidor cria e abre este pipe em modo de leitura, enquanto que o cliente o abre em modo de escrita (é responsável pela comunicação cliente -> servidor)
- pelo cliente - o cliente cria este pipe cujo nome do caminho contêm o seu pid, e graças a uma primeira comunicação pelo pipe referido anteriormente, em que o cliente manda o seu pid ao servidor, este é então capaz de fazer a abertura deste fifo (em modo de escrita). O sentido de comunicação é então servidor -> cliente. Apesar de para este projeto, estarmos a criar os pipes do lado do cliente associados ao seu pid não ser muito útil, se quisessemos escalar o projeto a múltiplos clientes era bastante vantajoso.

As escritas e leituras nos pipes pelo cliente são feitas concorrentemente, onde dois procesos são criados explicitamente para isso (um para cada uma das funcionalidades). Já no servidor, quando é feita uma leitura de um comando enviado pelo cliente, o servidor é responsável por identificar devidamente as tarefas e executá-las da melhor maneira possível, devolvendo posteriormente uma resposta ao cliente.

2.2 Estruturas de Dados

Relativamente às estruturas de dados criadas, foi necessário criar duas estruturas que nos permitissem proceder a toda a implementação e correto funcionamento do programa.

- **struct task**: responsável por armazenar toda a informação relativa a uma tarefa enviada pelo cliente.
- **struct circularArray**: array circular de tamanho fixo, que contém a informação de todas as tarefas mandadas executar no tempo em que o servidor executa. Cada posição deste array não é nada mais do que um apontador para uma estrutura task, em cima referida.

```c
typedef struct task {
    int id;
    int state; //0->finished, 1->executing, 2->user interrupcion, 3->max inactivity, 4->max executing
    int beginning; //where output beggins in log file
    int size; //size of output
    int pid;
    char command[MAX_READER];
}Task;

typedef struct circularArray{
    Task** tasks;
    int size;
    int currentPos;
}CircularArray;
```

Figura 2.1: Estrutura de dados para gestao de tarefas.

2.3 Funcionalidade Mínima

2.3.1 Tempo de Inactividade

O tempo de inactividade foi intrepertado como sendo o intervalo máximo de tempo de execução de cada um dos comandos separados por pipes. Este tempo é definido pelo utilizador através dos respetivos comandos (-i pela linha de comando e tempo-inactividade pela shell) e alterado pelo servidor. Assim,para terminar a execução de uma tarefa caso não se verifique qualquer comuniação através de pipes anónimos ao fim de um tempo especificado, foi necessário associar um novo handler ao signal SIGALRM nos processos netos do servidor (que vão executar os diferentes comandos separados por pipes de uma tarefa). Este novo handler garante que quando o tempo do alarm instanciado neste processo neto chega ao fim, este mesmo processo sai com um exit code 1. O que acontece, é que o processo filho quando recolhe os códigos de saída dos seus respetivos filhos, se algum deles for 1, mata devidamente todos os processos netos, caso ainda estejam a executar, e o processo filho sai com um exit code correspondente ao número que equivale a uma tarefa estar sinalizada como "max inatividade", neste caso 3, que será posteriormente recolhido pelo servidor.

2.3.2 Tempo de Execução

De forma a definir o tempo máximo de execução de um comando dado pelo cliente (argus) foi associado um handler ao signal SIGALRM no processo filho do servidor, responsável por tratar de executar corretamente a tarefa (dividir todos comandos da tarefa separados por pipes e criar processos filhos para os executar concorrentemente), e foi definido um alarme com o tempo de execução (caso > 0), em vigor no momento. Este alarme é acionado no instante imediatamente anterior ao processo filho criar todos os seus filhos e os mandar executar os comandos, e caso seja desencadeado todos os processos criados para tratar desta tarefa (processos netos do servidor) são devidamente mortos e o processo filho sai com um exit code correspondente ao número que equivale a uma tarefa estar sinalizada como "max execucao", neste caso 4.

2.3.3 Executar Tarefa

A funcionalidade de executar uma tarefa foi facilmente implementada, com o auxílio do material de estudo das aulas práticas. O maior desafio foi mesmo garantir concorrência no servidor a nível dos diferentes pedidos do cliente e na execução das tarefas, isto é garantir que o servidor continuava disponível para ler e identificar corretamente novos pedidos do cliente, enquanto que as tarefas previamente validadas executavam paralelamente. Assim, o servidor inicialmente trata de atribuir o id à nova tarefa e armazenar corretamente a informação relativa no array circular, incluindo o pid do filho criado pelo servidor que fica responsável pela execução da tarefa. Assim, e sendo que estamos a guardar toda a informação relativa às diferentes tarefas do programa em memória (para evitar ao máximo o número de acessos ao disco) foi necessário associar ao sinal SIGCHLD um handler, que quando desencadeado interrompe imediatamente a execução do servidor e trata de identificar corretamente qual o pid do filho que terminou a execução e caso não tenha ocorrido nenhum erro, recolhe o seu código de saída e atribui-o à tarefa cujo campo pid é igual ao pid recolhido pelo wait feito pelo servidor. Posteriormente, e se não houver nenhuma execução de uma outra tarefa terminada, o servidor retoma à posição onde ocorreu a interrupção.

2.3.4 Listar Tarefas em Execução

Como se trata de um pedido meramente de consulta, o servidor cria um processo filho que trata de encaminhar para o cliente o número e o comando de todas as tarefas armazenadas em memória, que estão a ser executadas (estado da tarefa == 1). Este processo filho sai com um exit code de 5 para o servidor, quando o recolher, saber que não precisa de fazer nada.

2.3.5 Histórico de Tarefas Terminadas

Esta funcionalidade permite ao utilizador saber quais as tarefas que já terminaram, enquanto o servidor estiver a executar. Para isso o servidor cria um processo filho que envia ao cliente não só a informação relativa ao número de tarefa e ao comando em si, mas também o estado de terminação da tarefa. Este processo filho sai com um exit code de 5 para o servidor, quando o recolher, saber que não precisa de fazer nada.

2.3.6 Terminar Tarefa em Execução

O utilizador tem ainda a possibilidade de terminar uma tarefa em execução, através dos devidos comandos e flags. O servidor mais uma vez, atribui a responsabilidade de executar esta funcionalidade a um processo filho e abordagem para proceder à terminação de uma tarefa foi facilmente implementada, associando neste novo processo o handler criado previamente que trata de matar, devidamente, todos os processos que este processo filho cria para executar concorrentemente a tarefa, a um novo sinal SIGINT que é enviado ao pid deste processo após o processo filho, corretamente identificar qual a tarefa alvo pelo número de tarefa fornecido pelo utilizador. Este processo filho sai com um exit code de 5 para o servidor, quando o recolher, saber que não precisa de fazer nada.

2.3.7 Menu Ajuda

Este menu tem o objetivo de auxiliar o utilizador na utilização do programa em si, indicando ao cliente todas as possíveis funcionalidades implementadas. Esta funcionalidade esta implementada do lado do cliente.

2.4 Funcionalidade Adicional

2.4.1 Consultar Output

A funcionalidade adicional implementada permite ao utilizador consultar os outputs de todas as tarefas que foram executadas previamente. Para isso, tornou-se necessário garantir a persistência tanto do outpout das tarefas (num ficheiro log) como as estruturas das tarefas que contêm toda a necessária informação relativa à sua execução (log.idx). Assim, sempre que uma tarefa executa corretamente, sem qualquer tipo de interrupções, o seu output é redirecionado para um ficheiro temporário cujo caminho contêm o pid do processo filho outrora responsável pela sua execução. Deste modo é possível ao servidor atualizar a estrutura da tarefa com a posição inicial onde vai ser escrito o output e o seu tamanho (tamanho lido do ficheiro temporário), escrever o respetivo output no ficheiro alvo (log) e persistir a estrutra representativa da tarefa.

2.5 Execução

Para testar toda a funcionalidade do programa, tomou-se auxílio da system call sleep. Este sleep foi colocado no último pipe responsável por executar o último comando de um pipeline.

Na figura seguinte encontra-se uma demonstração de uma execução simples, de forma a ilustrar as funcionalidades implementadas:

```
MBP-de-Ana:src anacesar$ make all
mkdir -p tmp
mkdir -p files
touch files/log
touch files/log.idx
gcc -Wall -g   -c -o argus.o argus.c
gcc -Wall -g   -c -o utilities.o utilities.c
gcc   argus.o utilities.o   -o argus
gcc -Wall -g   -c -o argusd.o argusd.c
gcc   argusd.o utilities.o   -o argusd
MBP-de-Ana:src anacesar$ ./argusd
```

```
MBP-de-Ana:src anacesar$ ./argus -e 'ls -l'
nova tarefa #1
MBP-de-Ana:src anacesar$ ./argus
ajuda
tempo-inactividade segs
tempo-execucao segs
executar 'p1 | p2 ... | pn'
listar
terminar tarefa
historico
output tarefa
tempo-inactividade 4
executar 'echo helooo'
nova tarefa #2
tempo-execucao 2
executar 'cat Makefile'
nova tarefa #3
executar 'cut -f7 -d: /etc/passwd | uniq | wc -l'
nova tarefa #4
tempo-execucao 0
tempo-inactividade 0
executar 'ls -l | wc
nova tarefa #5
executar 'cut -f7 -d: /etc/passwd | uniq | wc -l'
nova tarefa #6
listar
#5: ls -l | wc
#6: cut -f7 -d: /etc/passwd | uniq | wc -l
terminar 5
historico
#1, concluida: ls -l
#2, max inatividade: echo helooo
#3, max execução: cat Makefile
#4, max execução: cut -f7 -d: /etc/passwd | uniq | wc -l
#5, terminada: ls -l | wc
#6, concluida: cut -f7 -d: /etc/passwd | uniq | wc -l
output 1
total 264
-rw-r--r--@ 1 anacesar  staff    315 14 Jun 18:54 Makefile
-rwxr-xr-x  1 anacesar  staff  20836 15 Jun 10:31 argus
-rw-r--r--  1 anacesar  staff   3966 15 Jun 10:23 argus.c
-rw-r--r--  1 anacesar  staff   1557 14 Jun 18:45 argus.h
-rw-r--r--  1 anacesar  staff  10456 15 Jun 10:31 argus.o
-rwxr-xr-x  1 anacesar  staff  23204 15 Jun 10:31 argusd
-rw-r--r--  1 anacesar  staff  12255 14 Jun 22:33 argusd.c
-rw-r--r--  1 anacesar  staff  20924 15 Jun 10:31 argusd.o
drwxr-xr-x  4 anacesar  staff    128 15 Jun 10:31 files
drwxr-xr-x  5 anacesar  staff    160 15 Jun 10:32 tmp
-rw-r--r--  1 anacesar  staff   4740 14 Jun 21:07 utilities.c
-rw-r--r--  1 anacesar  staff  12720 15 Jun 10:31 utilities.o

output 6
^C    17
MBP-de-Ana:src anacesar$
```

Figura 2.2: Ilustração do funcionamento do programa

3. Conclusão

Com este projecto, as necessidades propostas pelo enunciado foram respondidas, bem como possibilitou uma melhor compreensão dos contéudos lecionados na unidade curricular durante o semestre. Fundamentalmente, este projecto facultou-me uma compreensão a um nível mais profundo dos mecanismos para a criação de processos bem como o seu modo de funcionamento e de comunicação com outros, tanto por pipes anónimos como por pipes com nome, assim como as características associados aos mesmos referente a herança de memória e redireccionamento de descritores. Fundamentalmente, considero que o meu nível de conhecimento face aos conteúdos leccionados foi expandido com a realização deste projecto prático, contribuindo para o meu percurso de *developer*.