Sistemas Operativos

Controlo e Monitorização de Processos e Comunicação

Relatório do trabalho prático

Licenciatura em Ciências da Computação

Universidade do Minho

2019/2020

Grupo 2

André Gonçalves A87942

Anabela Pereira A87990

Márcia Cerqueira A87992

Junho de 2020

Universidade do Minho
Escola de Ciências

Índice

1 Introdução----------3
2 Modelo cliente – servidor----------3
2.1 Cliente ----------3
2.2 Servidor----------3
3 Funcionalidades pretendidas no projeto----------4
4 Explicação do código----------5
4.1 Funções do cliente----------5
4.2 Funções do servidor----------6
4.3 Funções de controlo de sinais----------6
4.4 Estrutura tarefa----------6
5 Tempos máximos----------7
5.1 Tempo máximo de inatividade e execução----------7
6 Máximos definidos ----------7
7 Conclusão----------8

1 Introdução

Foi proposto, no âmbito da unidade curricular de Sistemas Operativos, o desenvolvimento de um sistema de controlo e monotorização de processos e comunicação. Este tipo de sistema utiliza uma estrutura de aplicação distribuída que distribui tarefas e cargos de trabalho entre fornecedores de serviços, designados como servidores, e os requerentes dos serviços, designados como clientes, este projeto baseia-se, portanto, no modelo Cliente-Servidor.

Ao longo deste relatório, iremos explicar a nossa abordagem em relação ao projeto, justificar a estrutura do nosso sistema e demonstrar a utilização dos conhecimentos adquiridos na UC, nomeadamente no que toca à criação de processos, duplicação de descritores, utilização de *pipes* com nome, execução de processos, sinais e várias *system calls*.

2 Modelo cliente – servidor

O nosso trabalho baseia-se numa interface cliente – servidor. O cliente e o servidor comunicam através de dois *fifos*, o *fifo_in* e o *fifo_out*. Se o servidor não está ativo, o cliente espera que o servidor execute. Na nossa implementação deste modelo, o cliente escreve os argumentos recebidos separando-os por '\n' para o *fifo_in* sendo estes lidos pelo servidor. O servidor faz parse do que lê e baseado nisso faz algo e gera um *output*(*String*) que escreve no *fifo_out* e que, posteriormente, é lido pelo cliente e passado para o STDOUT.

2.1 Cliente

O cliente contém duas possibilidades de utilização. A primeira consiste, simplesmente, em executar em base no que recebe dos argumentos e outra quando o cliente não recebe argumentos executa uma linha textual intrepretada (shell). Na forma shell o cliente lê ciclicamente do STDIN e passa o que lê para o servidor até que o usuário escreva "quit" (opção de sair), que termina o cliente mas não termina o servidor.

2.2 Servidor

O servidor é, basicamente, um ciclo no qual o servidor espera que o cliente escreva no FIFO. Quando o servidor lê, faz parse da String lida e faz algo dependendo do que foi lido. O servidor gera uma String output que é escrita no FIFO fifo_out, para depois ser usada pelo cliente. O ciclo termina quando é dada a opção "quit".

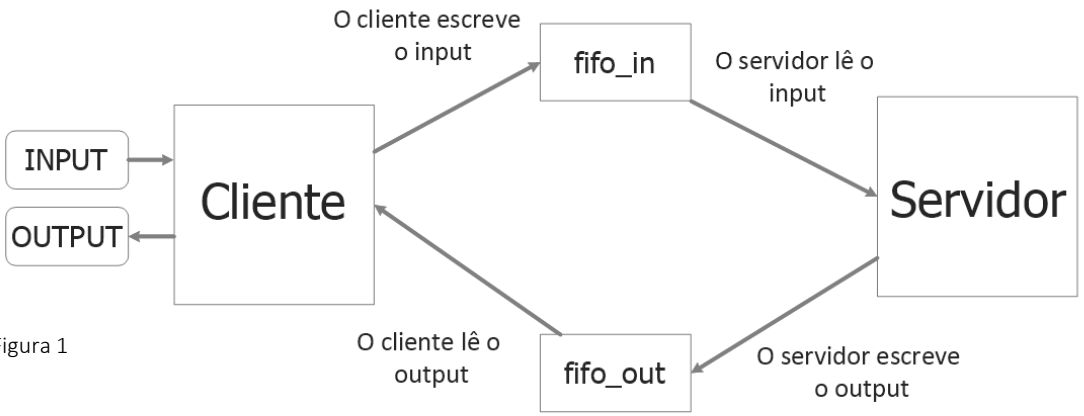

Figura 1

3 Funcionalidades pretendidas no projeto

- Tempo de inatividade: define o tempo máximo(segundos) de inatividade de comunicação num *pipe* anónimo (opção -i *n* da linha de comando).

  argus$ tempo-inactividade 10

- Tempo de execução: define o tempo máximo(segundos) de execução de uma tarefa (opção -m *n* da linha de comandos).

  argus$ tempo-execucao 20

- Executar: executa uma tarefa (opção -e "p1 | p2 …| pn" da linha de comando)

  argus$ executar 'cut -f7 -d: /etc/passwd |uniq | wc -l '
  nova tarefa #5

- Listar tarefas: lista tarefas em execuação (opção -l da linha de comandos)

  argus$ listar
  #1: ./a.out | teste
  #3: date
  #5: cut -f7 -d: /etc/passw | uniq | wc -l

- Terminar: termina uma tarefa em execução (opção -t *n*)

  argus$ terminar 1

- Histórico: lista o registo históiroc de tarefas terminadas (opção -r)

  argus$ histórico
  #2, concluída: ./a.out | ./b.out
  #3, max inactividade: prog1 | prog2 | prog3
  #4, max execução: prog2 | prog3

- Ajuda: apresenta ajuda à sua utilização (opção -h)

  argus$ ajuda
  tempo-inatividade segs
  tempo-execucao segs
  executar p1 | p2 … | pn
  listar
  terminar n
  historico
  ajuda
  quit

Funcionalidades	Realização
Tempo de inatividade	Implementada
Tempo de execução	Implementada
Executar	Implementada
Listar tarefas	Implementada
Terminar	Implementada
Histórico	Implementada
Ajuda	Implementada
Output	Não implementado

4 Explicação do código

4.1 Funções do cliente

- void parseInput(char* dest, char* src)

Função usada no cliente para fazer parse da String que é dado pelo STDIN. Esta função recebe dois apontadores onde src é a função dada para ler e dest é a String onde deve escrever (dest será depois passada para o *fifo_in*).

Exemplo: src = " -e ' ls | wc ' "

dest é um array de 256 char's

depois de executar parseInput(dest,src) :

dest = "3/n/n-e/n ls | wc\n\n"

- void paraIN(int argc,char* argv[],char* buf)

Função usada pelo cliente que cria escreve na *String buf*, que será passada para o *fifo_in*(é a string resultante de delimitar por '\n' os argumentos de argv[]).

- void escreverLerFIFO(char buf[]);

Função usada pelo cliente para que passa para o *fifo_in* o buf e espera a resposta do servidor pelo *fifo_out*, que será depois impresso no STDOUT.

4.2 Funções do servidor

- void mudarOUTandERR ();

Função usada pelo servidor que muda o STDOUT e o STDERR para um ficheiro LOGS.txt e erros.txt respetivamente.

- int parse (char* texto,char* comandos[]);

Função usada pelo servidor que faz parse do que é lido do *fifo_in* e retorna o número de argumentos dados.

- int executarTarefa(char* arg,int t_exec,int t_ina);

Função que executa uma tarefa.

- void listarTarefas(tarefa t[],int n,int x,char res[]);

Dá a lista das tarefas em execução se x == 0 se não dá a lista das tarefas terminadas.

- void atualizarHistorico(tarefa t[],int n);

Atualiza o array com n tarefas.

4.3 Funções para controlo de sinal

- void handlerMaxExec(int sig);
- void handlerKill (int sig);
- void handlerMaxInat (int sig);

Funções que matam os processos filhos que estão em execução se é dado um sinal de alarme. Funções mudam a variável global terminada, com exceção handlerMaxInat.

4.4 Estrutura *tarefa*

A cada tarefa executada pelo servidor é atribuído uma estrutura *tarefa*.

Uma tarefa é defenida por:

- argumentos - representa os argumentos da tarefa;
- terminada - é um int que representa o estado da aplicação:

    0. em execução
    1. terminou com sucesso
    2. terminou por max inativdade
    3. terminou por max execucao
    4. terminada por cliente
    5. terminou com erro

- pid - int que representa o *pid* do processo filho.

5 Tempos máximos

5.1 Máximo de inatividade e execução

Para definir máximos foram usados sinais alarm quando um dos processos recebe o sinal de alarme mata todos os processos filhos em execução através dos *pids* guardados na variável global *pids*. Cada processo relativo à tarefa termina com _exit(i) onde i depende da maneira que terminou que será depois o terminado da estrutura tarefa relativa à tarefa.

6 Máximos definidos

- MAX_COMMANDS 100 - representa o número máximo de *pipes* anónimos possíveis;
- MAX_ARGS 30 - representa o número máximo de argumentos que pode ser dado a cada função exec;
- MAX 256 -Número máximo de chars que podem ser escritos no *fifo_in*. Número máximo de tarefas que podem ser guardadas no histórico e por consequente listadas;
- MAX_OUT 2048- número máximo de chars que podem ser escritos no *fifo_out*.

7 Conclusão

Este trabalho prático foi bastante interessante visto que pudemos aplicar os conceitos adquiridos nas aulas de Sistemas Operativos. A resolução sequencial dos guiões das aulas práticas, revelou-se algo fundamental para tornar possível a realização deste projeto, em que fomos bem sucedidos na implementação das funcionalidades básicas.