E-fólio A | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Sistemas Operativos

CÓDIGO: 21111

DOCENTE: Paulo Shirley e Paulo Quaresma

A preencher pelo estudante

NOME: Hélio Emanuel Soares de Sousa

N.º DE ESTUDANTE: 2000027

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 05 de Abril de 2021

TRABALHO / RESOLUÇÃO:

Este trabalho foi desenvolvido no sistema operativo Ubuntu 20.04, através do IDE NetBeans 12.3, testado no terminal do Ubuntu com o compilador gcc Ubuntu 9.3.0-17ubuntu1~20.04 9.3.0, o relatório em LibreOffice Version: 6.4.6.2, Build ID: 1:6.4.6-0ubuntu0.20.04.1 no formato Word 2007-365(docx). Para a documentação anotada utilizei doxygen 1.9.1, a mesma pode ser consultada diretamente no código ou em formato html em *documentation/index.html.*

O programa teve como bibliografia os conteúdos teóricos da Unidade Curricular (UC), AF1 e AF2, o Modern Operating Systems (Taneubaum & Bos, 2014), as man pages dos programas ls, echo, wc, cat, stdout, stdin, waitpid.

Como complemento para resolver questões especificas consultei as seguintes fontes web: (GeeksforGeeksCat, 2021), (GeeksforGeeksFork, 2017), (Thompson, 2014), (Thompson, 2014), (akanshgupta, 2018), (pevik, 2019), (pts, 2009), (tutorialspoint-freopen, 2021).

A estrutura do código na main foi definida para ter na medida do possível apenas chamadas de funções. Os comentários são feitos quase em exclusivo nas funções, e o código da main tem apenas referencias a que processo está a ocorrer numa dada condição If. Os nomes das funções são descritivos do que executam e as funções foram desenvolvidas para executarem o mínimo de tarefas possível, numa lógica de uma função uma tarefa. Penso que torna a leitura do código mais simples, a partir do momento em que nos inteiramos do que faz cada função.

Analise geral do código:

Criei a função valida_args que faz um teste ao número de argumentos passados na linha de comandos usando para isso a variável argc da própria main. O valor é igual a 2 porque o argc conta também a chamada do comando em si. Garante que existe apenas um único argumento na linha de comandos.

Feito o teste anterior para o número de argumentos, fiz duas funções uma com fopen (valida_file) outra com DIR (valida_dir). Optei apenas pela DIR pois pretendemos apenas diretórios, mas ambas servem o propósito requerido. A

base para esta função valida_dir foi obtida em (pevik, 2019), embora com algumas modificações no código em termos de indentação, mensagem de output e o return.

Após as validações iniciais temos de criar 5 processos, o programa dit é o primeiro (processo A). A partir do mesmo serão gerados todos os outros.

A base para a geração dos processos foi obtida em (akanshgupta, 2018), e posteriormente adaptada com conceitos aprendidos na AF2, (GeeksforGeeksFork, 2017) e (pts, 2009).

Ao iniciar o comando `./dit` começamos o processo A e na main começamos com as variáveis `pid_t pidB,pidC,pidD,pidE;` para identificar os filhos, e a variável `int status;` para poder fazer uso da função `waitpid()`.

De seguida, validamos o numero de argumentos com `valida_args()` e o diretório com `valida_dir();`

Se tudo OK, seguimos para criar o primeiro filho com `pidB=fork();` A seguir a cada geração temos sempre uma verificação da geração do processo, por exemplo, `fork_check(pidB)`

O processo filho foi gerado, e ao contrário da AF2, invertemos os processos nos if´s, primeiro corremos o processo filho, com `if(pidB==0)`, pois o `fork()` retorna dois inteiros, o zero para o processo filho e o PID do processo filho para o pai. Assim sabemos, que sempre que igual a zero é o filho e >0 é o pai. Com base nisso, aninha-se vários if´s e else´s.

Cada if/else, inicia com a função `msg_processo()` para dar como output os dados requeridos no enunciado, "Processo X: PID=xxx PPID=xxx".

Depois, no if (ou seja, no filho) faz-se a chamada da função especifica do processo, no caso do B teremos a função `processo_Bls(argv)` para o C `processo_Cecho()` para o D `processo_Dwc()` e para o E `processo_Ecat()`. Estas funções executam os comandos requeridos para cada processo.

Em cada else, a seguir á `msg_processo()` temos um `waitpid()` para que o pai espere pelo filho, por exemplo `waitpid(pidB,&status,0)`, pai(A) vai esperar pelo processo filho B, e assim sucessivamente para todos os outros processos.

E terminamos a estrutura da função main, criando os 5 processos necessários, em que o processo A cria os processos B, C, D e E de modo semelhante, em que cada processo inicia após o anterior terminar.

Agora, analisando especificamente cada uma das 4 funções dos processos:

Excerto do código do Processo_Bls:

```
96  void processo_Bls(char *argv[]) {
97  freopen("tmp1.txt", "w+", stdout);
98  execl("/bin/ls","ls","-1",argv[1], NULL);
```

Primeiro redireciona-se o stdout para o tmp1.txt e depois substitui-se a imagem com um execl, tendo como argumento o diretório passado na linha de comandos, para executar o comando ls.

Excerto do código do Processo_Cecho:

```
void processo_Cecho() {
107  freopen("tmp1.txt", "a+", stdout);
108  execlp("echo","echo","-n", "Itens encontrados: ", NULL);
```

Neste caso direciona-se o stdout para acrescentar (append) ao ficheiro tmp1.txt “itens encontrados”. Depois, utiliza-se a troca de imagem através do execlp, apenas é passado o nome do ficheiro echo porque como indicado na Lab da AF2, nas funções com argumento file, se file não contém o carater "/" (ou seja, não é uma pathname, absoluta ou relativa), o ficheiro executável é procurado nas diretorias indicadas por PATH. Nas funções que não têm envp como argumento, as variáveis de ambiente são as definidas na variável global environ(…), funciona porque acrescentei esta variável global no código.

Excerto do código do Processo_Dwc:

```
void processo_Dwc() {
120  freopen("tmp1.txt", "r", stdin);
121  freopen("tmp2.txt", "w+", stdout);
122  char *arguments[2];
123  arguments[0] = "wc";
124  arguments[1] = "-l";
125  arguments[2] = NULL;
126  execv( "/bin/wc", arguments);
```

Aqui utiliza-se o freopen para colocar no tmp1.txt o standard input e no tmp2.txt o stdout. Com base nas man pages, a função wc pode ler o stdin e apresenta

para o stdout. Já o freopen, permite abrir um ficheiro para o stdin e redirecionar o conteúdo do stdout para um ficheiro cumprindo dessa forma o pretendido. Para este caso, foi utilizado o execv, para isso fiz um array que foi passado como argumento para o execv, na essência é o mesmo que o execl.

Excerto do código do Processo_Ecat:

```
void processo_Ecat() {
 134  char *arguments[3];
 135  arguments[0] = "cat";
 136  arguments[1] = "tmp1.txt";
 137  arguments[2] = "tmp2.txt";
 138  arguments[3] = NULL;
 139  execvp("cat", arguments);
```

Aqui recorri á função cat, que concatena dois ficheiros e envia para o stdout o seu resultado. A opção recaiu pelo execvp, tendo assim usado 4 funções exec diferentes, o comando cat está disponível na variável global, e os argumentos são passados através de um vetor, que tem de terminar com NULL.

Por fim como nota, na primeira passagem do programa não conta o tmp2.txt, porque o ls corre antes da criação desse ficheiro, mas se o programa for testado uma segunda vez o ls já o verifica e aparece na saída do cat, ficando assim com semelhanças ao output do enunciado.

Refiro também que como extra apresento a documentação anotada através do doxygen, que penso ser uma mais-valia para a melhor compreensão do trabalho realizado.

Em suma, o trabalho apresentado executa o requerido pelo enunciado e possui explicação passo a passo para cada uma das opções tomadas.

BIBLIOGRAFIA

akanshgupta. (13 de 05 de 2018). *Using fork() to produce 1 parent and its 3 child processes.* Obtido de https://www.geeksforgeeks.org/: https://www.geeksforgeeks.org/using-fork-produce-1-parent-3-child-processes/

GeeksforGeeks. (03 de 11 de 2017). *Create n-child process from same parent process using fork() in C.* Obtido de https://www.geeksforgeeks.org/: https://www.geeksforgeeks.org/create-n-child-process-parent-process-using-fork-c/

GeeksforGeeks. (18 de 02 de 2021). *Cat command in Linux with examples.* Obtido de https://www.geeksforgeeks.org/: https://www.geeksforgeeks.org/cat-command-in-linux-with-examples/

pevik. (27 de 05 de 2019). *How can I check if a directory exists?* Obtido de https://stackoverflow.com/: https://stackoverflow.com/questions/12510874/how-can-i-check-if-a-directory-exists

pts. (09 de 05 de 2009). *Multiple child process.* Obtido de https://stackoverflow.com: https://stackoverflow.com/questions/876605/multiple-child-process

Taneubaum, A. S., & Bos, H. (2014). *Modern Operating Systems, 4th Ed.* Boston: Pearson.

Thompson, K. (18 de 11 de 2014). *EXIT_FAILURE vs exit(1)?* Obtido de https://stackoverflow.com/: https://stackoverflow.com/questions/13667364/exit-failure-vs-exit1

tutorialspoint-freopen. (28 de 03 de 2021). *C library function - freopen().* Obtido de https://www.tutorialspoint.com/index.htm: https://www.tutorialspoint.com/c_standard_library/c_function_freopen.htm