Leonardo Richter Korndörfer
2010-04-21

Primeiro Problema proposto:

Descreva detalhadamente o comportamento do script abaixo:

```
 1 #!/bin/bash
 2 while true; do
 3     if [ "$#" -lt 2 ]; then exit 0; fi
 4     case "$1" in
 5     -e)
 6         i=0
 7         for h in $(cat $2); do
 8             if [ -f $h -a -x $h ]; then
 9                 $h >/dev/null 2>/dev/null
10                 echo "$i:$h -> $?\n"
11             fi
12             i=$((i+1))
13         done ;;
14 #   -c) cat "$h" | grep "$USER" | sort ;;
15     *) echo "$0 [-e|-c] filename..."; exit -1 ;;
16     esac
17     shift 2
18 done
```

Resposta:

O script recebe nomes de arquivos como argumento (-e) contendo nomes de executáveis, pega o conteúdo desse(s) arquivo(s) e tenta executar em background. O nome dos arquivos deve ser precedido por "-e" como no exemplo:

```
script -e arq.txt -e arq.txt
```

O script fica em um laço eterno lendo os argumentos que foram passados. As condições de parada são:

- ter menos de dois argumentos, exit status 0
- ter 2 argumentos mas a opção "-e" estar errada, vai executar o comportamento defult, imprimir o USAGE, e sair com status -1. O seguinte exemplo demonstra o comportamento:

```
$ script -f arq.txt
$ script [-e|-c] filename...
```

Caso sejam passados 3 argumentos para o script ele irá tentar executar os dois primeiros argumentos e em seguida tem comportamento correspondente ao ponto 1. Esse comportamento vai se repetir para qualquer 2n+1 numero de argumentos.

Quando o token "-e" for encontrado o conteúdo do arquivo passado a seguir é lido e expandido dentro de outro laço, onde cada iteração corresponde a um nome de programa dentro deste arquivo. Para cada nome de programa passado é feito um teste confirmando que o arquivo existe e é um executável. Se o resultado do teste for negativo vai pular para a próxima iteração. Se for verdade então vai executar o programa passado redirecionando o stdout e o stderr para /dev/null. Depois da execução do programa vai imprimir na tela o número da iteração, o nome do programa e o seu status de retorno. Depois disso volta para o início do loop.
A linha iniciada por “#” (linha 14) é um comentário e nunca será executada.

Segundo Problema proposto:

Proponha um programa em C que implemente uma funcionalidade equivalente à do programa em shellscript abaixo

```
1  #!/bin/bash
2  if [ $# -lt 1 ]; then
3     echo -e "Parametros insuficientes. \nUse: $0 comando [comando...]"; exit -1
4  fi
5  ns=0
6  nf=0
7  for cmd in $*; do
8     $cmd &
9     pid=$!
10    sleep 2;
11    kill -KILL $pid 2>/dev/null
12    wait $pid 2>/dev/null
13    if test $? -eq 0 ; then ns=$((ns+1)); else nf=$((nf+1)); fi
14 done
15 echo "================================="
16 echo "Numero total de comandos executados: $((ns+nf))"
17 echo "Número de comandos executados com sucesso: $ns"
```

Resposta:

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <unistd.h>
 4 #include <signal.h>
 5 #include <string.h>
 6
 7 #define POINTER_SIZE 4
 8
 9 int main(int argc, char **argv)
10 {
11         if (argc < 2) {
12                 printf("Unsuficient parameters.\nUse %s command [command]\n",
13                        argv[0]);
14                 exit(-1);
15         }
16
17         /* arg_num is argc - 1 bacause in argv position 0 is the cmd name
18          * and we just want to get passed args
19          **/
20         int arg_num = argc - 1;
21
22         /* ns = return status 0
23          * nf = other return status
24          **/
25         int ns, nf;
26
27         int pid[arg_num];
```

```
28          char **commands = malloc(POINTER_SIZE * argc);
29
30          /* copy contents from argv to **commands */
31          int i;
32          for (i = 1; i < argc; i++) {
33                  int current_size = sizeof(char) * (strlen(argv[i]) + 1);
34
35                  /* now allocate the size of the current command and
36                   * store it into the commands pointer-pointer
37                   **/
38                  commands[i - 1] = malloc(current_size);
39                  strcpy(commands[i - 1], argv[i]);
40          }
41
42          /* loop through command[] and execute each */
43          for (i = 0; i < arg_num; i++) {
44
45                  /* THIS IS DANGEROUS
46                   *
47                   * get command to run and cerefull with the command pointer.
48                   * when we free the commands[i] the memory pointed by *command
49                   * is freed within.
50                   **/
51                  char *command = commands[i];
52
53                  /* do fork */
54
55                  /* if connot create child exit with error */
56                  if ((pid[i] = fork()) < 0) {
57                          printf("Could not create the child");
58                          exit(-1);
59                  }
60                  /* if pid is 0 is because we are the child */
61                  else if (pid[i] == 0) {
62 #ifdef DEBUG
63                          /* this will run when DEBUG is defined in compile time */
64                          printf("This is a child process with PID %d\n",
65                                 getpid());
66                          printf("Executing command: %s\n", command);
67 #endif
68                          /* no args are passed to the cmd */
69                          char *args[] = { command, (char *)0 };
70
71                          /* execute the command then exit */
72                          execvp(command, args);
73                          exit(0);
74                  }
75                  /* if pid is > 0 we are the parent code */
76                  else {
77                          /* if pid still not done kill it */
78                          sleep(2);
79                          kill(pid[i], SIGKILL);
80
81                          /* get the return status of the child */
```

```
 82                         int stat_loc;
 83                         waitpid(pid[i], &stat_loc, WUNTRACED);
 84
 85                         if (stat_loc == 0)
 86                                 ns++;
 87                         else
 88                                 nf++;
 89                 }
 90         }
 91         /* end for */
 92
 93         printf("\nStatus:\n");
 94         printf("Total commands executed is %d\n", ns + nf);
 95         printf("Total commands executed with success is %d\n", ns);
 96
 97         /* we can free it all now, even knowing OS will free all areas
 98          * of this process when it is done, it is nice to know we are
 99          * doing it ourselves
100          **/
101         for (i = arg_num - 1; i >= 0; i--) {
102                 int current_size = sizeof(char) * (strlen(commands[i]) + 1);
103
104                 free(commands[i]);
105                 commands[i] = NULL;
106                 printf("free ok\n");
107         }
108
109         free(commands);
110
111         return 0;
112 }
```

Todos os códigos apresentados neste podem ser encontrados em http://github.com/leokorndorfer/university/tree/master/operational_systems_lab/1_grade_test/