Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Операционные системы»

Студент: А. О. Дубинин
Преподаватель: Е. С. Миронов
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2018

Лабораторная работа №2

Цель работы: Приобретение практических навыков в управлении процессами в ОС, обеспечении обмена данных между процессами посредством каналов.

Задание: Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант 22: Родительский процесс представляет собой сервер по работе с массивами и принимает команды со стороны дочернего процесса.

1 Описание

Инфраструктура программы: создается один дочерний процесс, который через pipe передает входные данные для управления массивом в родительском процессе. Для получения и передачи данных будем использовать read, write.

Так же необходимо предусмотреть обрабатывание возвращаемых значений системных вызовов, это обуславливает большое количество вызовов функции perror в коде.

2 Исходный код

```
1  //main.c
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <sys/types.h>
6  #include <unistd.h>
7  #include <string.h>
8  #include <sys/wait.h>
9  #include <stdbool.h>
10
11 #include "vector.h"
12 #include "vector.cpp"
13
14 char read_command() {
15     char command;
16     do {
17         command = getchar();
18     } while ( command == '\n' || command == ' ' );
19     return command;
20 }
21
22 void help_function() {
23     printf( "command:\n" );
24     printf( "<h>\thelp\n" );
25     printf( "<q>\texit\n" );
26     printf( "<a>\tappend element into array\n" );
27     printf( "<p>\tprint array\n" );
28     printf( "<d> <index>\tdelete element in array by index\n" );
29 }
30
31 int main( int argc, char* argv[] ) {
32     int i, id, size;
33     char* app_element, * element;
34
35     // create a pipe
36     int my_pipe[2];
37     if ( pipe( my_pipe ) == -1 ) {
38         perror( "Error creating pipe\n" );
39     }
40
41     // fork
42     pid_t child_id;
43     child_id = fork();
44     if ( child_id == -1 ) {
45         perror( "Fork error\n" );
46     }
47
```

```
48      if ( child_id == 0 ) // child process
49      {
50          if ( close( my_pipe[0] ) < 0 ) { // child doesn't read
51              perror( "Failed to close pipe descriptors" );
52          }
53
54          char command;
55          printf( "Write a command(<h> for help):\n" );
56          while ( true ) {
57              command = read_command();
58
59              if ( command == EOF) {
60                  write( my_pipe[1], &command, 1 );
61                  break;
62              }
63
64              switch ( command ) {
65                  case 'a':
66                      printf( "write element:\n" );
67                      getchar();
68                      int capacity_app_elem = 3;
69                      int size_app_elem = 0;
70                      app_element = ( char* ) malloc( sizeof( char ) * capacity_app_elem )
                            ;
71                      char c;
72                      while ( true ) {
73                          if ( size_app_elem == capacity_app_elem ) {
74                              capacity_app_elem *= 2;
75                              app_element = ( char* ) realloc( app_element, sizeof( char )
                                    * capacity_app_elem );
76                          }
77                          c = getchar();
78                          if ( c == '\n' || c == ' ' ) break;
79                          app_element[size_app_elem] = c;
80                          size_app_elem++;
81                      }
82
83                      size = strlen( app_element );
84                      write( my_pipe[1], &command, 1 );
85                      write( my_pipe[1], &size, sizeof( int ));
86                      write( my_pipe[1], app_element, strlen( app_element ));
87                      free(app_element);
88                      break;
89                  case 'd':
90                      scanf( "%d", &id );
91                      write( my_pipe[1], &command, 1 );
92                      write( my_pipe[1], &id, sizeof( int ));
93                      break;
94                  case 'q':
```

```
95                  write( my_pipe[1], &command, 1 );
96                  if ( close( my_pipe[1] ) < 0 ) {
97                      perror( "Failed to close pipe descriptors" );
98                  }
99                  return 0;
100             case 'p':
101             case 'h':
102                 write( my_pipe[1], &command, 1 );
103                 break;
104             default:
105                 printf( "wrong command\n" );
106                 break;
107
108         }
109     }
110
111
112 } else // parent process
113 {
114     vector v;
115     vector_init( &v );
116
117     if ( close( my_pipe[1] ) < 0 ) { // parent doesn't write
118         perror( "Failed to close pipe descriptors" );
119     }
120
121     while ( true ) {
122         char reading_buf;
123         do {
124             read( my_pipe[0], &reading_buf, 1 );
125         } while ( reading_buf == '\n' || reading_buf == ' ' );
126
127         if ( reading_buf == EOF) {
128             break;
129         }
130
131         switch ( reading_buf ) {
132             case 'a':
133                 read( my_pipe[0], &size, sizeof( int ));
134                 element = ( char* ) malloc( sizeof( char ) * size );
135                 read( my_pipe[0], element, size );
136                 vector_add( &v, element );
137                 break;
138             case 'd':
139                 read( my_pipe[0], &id, sizeof( int ));
140                 vector_delete( &v, id );
141                 break;
142             case 'p':
143                 printf( "vector:\n" );
```

```
144                     for ( i = 0; i < vector_count( &v ); i++ ) {
145                         element = vector_get( &v, i );
146                         printf( "%s\n", element );
147                     }
148                     break;
149                 case 'q':
150                     vector_free( &v );
151                     if ( close( my_pipe[0] ) < 0 ) {
152                         perror( "Failed to close pipe descriptors" );
153                     }
154                     return 0;
155                 case 'h':
156                     help_function();
157                     break;
158                 default:
159                     printf( "wrong command\n" );
160                     break;
161 
162             }
163             printf( "Write a command(<h> for help):\n" );
164         }
165 
166         if ( close( my_pipe[0] ) < 0 ) {
167             perror( "Failed to close pipe descriptors" );
168         }
169 
170     }
171 
172     return 0;
173 }
```

3 Тесты

```
art@mars:~/study/semester_3/OS/lab_2/$ ./main
Write a command(<h>for help):
h
command:
<h>help
<q>exit
<a>append element into array
<p>print array
<d><index>delete element in array by index
Write a command(<h>for help):
a
write element:
hello
Write a command(<h>for help):
p
vector:
hello
Write a command(<h>for help):
a
write element:
world
Write a command(<h>for help):
p
vector:
hello
world
Write a command(<h>for help):
d 0
Write a command(<h>for help):
p
vector:
world
Write a command(<h>for help):
q
```

4 Диагностика strace

```
art@mars:~/study/semester_3/OS/lab_2/variant_22$ strace ./main
execve("./main",["./main"],0x7ffc5c6b3c50 /* 52 vars */) = 0
brk(NULL)                                = 0x55ef4cfdc000
access("/etc/ld.so.nohwcap",F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload",R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD,"/etc/ld.so.cache",O_RDONLY|O_CLOEXEC) = 3
fstat(3,st_mode=S_IFREG|0644,st_size=90822,...) = 0
mmap(NULL,90822,PROT_READ,MAP_PRIVATE,3,0) = 0x7f0225e46000
close(3)                                 = 0
access("/etc/ld.so.nohwcap",F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libc.so.6",O_RDONLY|O_CLOEXEC) = 3
read(3,"77ELF>604"...,832) = 832
fstat(3,st_mode=S_IFREG|0755,st_size=2030544,...) = 0
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7f0225e44000
mmap(NULL,4131552,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7f0225845000
mprotect(0x7f0225a2c000,2097152,PROT_NONE) = 0
mmap(0x7f0225c2c000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x7f0225c2c000
mmap(0x7f0225c32000,15072,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1,
= 0x7f0225c32000
close(3)                                 = 0
arch_prctl(ARCH_SET_FS,0x7f0225e45500) = 0
mprotect(0x7f0225c2c000,16384,PROT_READ) = 0
mprotect(0x55ef4c356000,4096,PROT_READ) = 0
mprotect(0x7f0225e5d000,4096,PROT_READ) = 0
munmap(0x7f0225e46000,90822)           = 0

//pipe
pipe([3,4])                              = 0

//fork
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_tid
= 11818

close(4)                                 = 0
read(3,Write a command(<h>for help):
a
write element:
abcd
```

```
"a",1)                          = 1
read(3,"",4)                   = 4
brk(NULL)                                = 0x55ef4cfdc000
brk(0x55ef4cffd000)                      = 0x55ef4cffd000
read(3,"abcd",4)                        = 4
fstat(1,st_mode=S_IFCHR|0620,st_rdev=makedev(136,0),...) = 0
write(1,"Write a command(<h>for help):",31Write a command(<h>for help):
) = 31
read(3,p
"p",1)                          = 1
write(1,"vector:",8vector:
)                 = 8
write(1,"abcd",5abcd
)                    = 5
write(1,"Write a command(<h>for help):",31Write a command(<h>for help):
) = 31
read(3,q
"q",1)                          = 1
close(3)                                 = 0
//выход группы процессов
exit_group(0)                            = ?
+++ exited with 0 +++
art@mars:~/study/semester_3/OS/lab_2/variant_22$
```

5 Выводы

Выполнив первую лабораторную работу по курсу «Операционные системы», я понял, как можно работать сразу с несколькими процессами, понял что дочерний процесс это копия родителя. Мною были изучены и применены системные вызовы, необходимые для создания процессов и для работы с ними. Так же я поработал с утилитой strace, осознав, что с помощью этой утилиты можно посмотреть какие системные функции вызывает уже готовая программа, не имея доступа к коду. На практике я увидел, что родительский процесс, завершает группу процессов, которую он создал.

Список литературы

[1] *Проект OpenNet MAN() FreeBSD и Linux*
URL: `https://www.opennet.ru/man`$_2$*.shtml*