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

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <stdbool.h>

#include "vector.h"
#include "vector.cpp"

char read_command() {
    printf("Write a command(<h> for help):\n");
    char command;
    scanf("%c", &command);
    return command;
}

void help_function() {
    printf("command:\n");
    printf("<h>\thelp\n");
    printf("<q>\texit\n");
    printf("<a>\tappend element into array\n");
    printf("<p>\tprint array\n");
    printf("<d> <index>\tdelete element in array by index\n");
}

int main( int argc, char* argv[] ) {
    int i, id, size;
    char app_element[101], *element;
    vector v;
    vector_init(&v);

    while ( true ) {

        int wstatus;

        // create a pipe
        int my_pipe[2];
        if ( pipe( my_pipe ) == -1 ) {
            perror("Error creating pipe\n");
        }

        // fork
        pid_t child_id;
        child_id = fork();
        if ( child_id == -1 ) {
```

```
48              perror("Fork error\n");
49          }
50
51
52          if ( child_id == 0 ) // child process
53          {
54              if ( close( my_pipe[0] ) ) { // child doesn't read
55                  perror("Failed to close pipe descriptors");
56              }
57
58              char command;
59              command = read_command();
60
61              switch (command) {
62                  case 'a':
63                      printf("write element:\n");
64                      scanf("%s", app_element);
65                      size = strlen(app_element);
66                      write( my_pipe[1], &command, 1 );
67                      write( my_pipe[1], &size, sizeof(int) );
68                      write( my_pipe[1], app_element, strlen(app_element) );
69                      break;
70                  case 'd':
71                      scanf("%d", &id);
72                      write( my_pipe[1], &command, 1 );
73                      write( my_pipe[1], &id, sizeof(int) );
74                      break;
75                  case 'p':
76                  case 'q':
77                  case 'h':
78                      write( my_pipe[1], &command, 1 );
79                      break;
80                  default:
81                      write( my_pipe[1], "c", 1 );
82                      break;
83
84              }
85
86
87
88
89              if ( close( my_pipe[1] ) ) {
90                  perror("Failed to close pipe descriptors");
91              }
92              break;
93
94          } else // parent process
95          {
96              if ( wait ( &wstatus ) ) {
```

```
97              perror("Failed in wait function");
98          }
99
100         if ( close( my_pipe[1] ) ) { // parent doesn't write
101             perror("Failed to close pipe descriptors");
102         }
103
104         char reading_buf;
105         do {
106             read( my_pipe[0], &reading_buf, 1 );
107         }
108         while(reading_buf == '\n' || reading_buf == ' ');
109
110         switch (reading_buf) {
111             case 'a':
112                 read( my_pipe[0], &size, sizeof(int) );
113                 element = (char*)malloc(sizeof(char) * size);
114                 read( my_pipe[0], element, size );
115                 vector_add(&v, element);
116                 break;
117             case 'd':
118                 read( my_pipe[0], &id, sizeof(int) );
119                 vector_delete(&v, id);
120                 break;
121             case 'p':
122                 printf("vector:\n");
123                 for (i = 0; i < vector_count(&v); i++) {
124                     element = vector_get(&v, i);
125                     printf("%s\n", element);
126                 }
127                 break;
128             case 'q':
129                 vector_free(&v);
130                 if ( close( my_pipe[0] ) ) {
131                     perror("Failed to close pipe descriptors");
132                 }
133                 return 0;
134             case 'h':
135                 help_function();
136                 break;
137             default:
138                 printf("wrong command\n");
139                 break;
140
141         }
142
143         if ( close( my_pipe[0] ) ) {
144             perror("Failed to close pipe descriptors");
145         }
```

```
146 ||
147 ||          }
148 ||      }
149 ||      return 0;
150 || }
```

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:
sad
Write a command(<h>for help):
p
vector:
sad
Write a command(<h>for help):
a
write element:
hello
Write a command(<h>for help):
p
vector:
sad
hello
Write a command(<h>for help):
d 1
Write a command(<h>for help):
p
vector:
sad
Write a command(<h>for help):
d 5
Write a command(<h>for help):
p
vector:
sad
Write a command(<h>for help):
```

```
d 0
Write a command(<h>for help):
p
vector:
Write a command(<h>for help):
d 0
Write a command(<h>for help):
p
vector:
Write a command(<h>for help):
```

4 Выводы

Выполнив первую лабораторную работу по курсу «Операционные системы», я приобрёл практические навыки в управлении процессами в ОС и обеспечении обмена данных между процессами посредством каналов. Мною были изучены и применены системные вызовы, необходимые для создания процессов и для работы с ними.

Список литературы

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