Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа**

**по курсу «Операционные системы»**

**III Семестр**

**Задание 2**

**Вариант 16**

|  |  |
| --- | --- |
| Студент: | Жерлыгин М.А. |
| Группа: | М80-208Б-18 |
| Преподаватель: | Миронов Е.С |
|  |  |
| Оценка: |  |
| Дата: |  |

Москва 2019

**1. Описание задания**

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 16: На вход программе подается команда интерпретатора команд и имя файла. Программа должна перенаправить стандартный ввод команды с этого файла и вывести результат команды в стандартный выходной поток. Использование операций write и printf запрещено.

**2. Код программы:**

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/wait.h>

int main(int argc, char\* argv[]){

// Если программе не переданы параметы

if (argc != 3) {

fprintf(stderr, "invalid input\n");

exit(2);

}

// идентификатор потока

pid\_t pid;

int status;

// создание нового потока

pid = fork();

// поток не создан

if (pid < 0) {

fprintf(stderr, "prosses not created\n");

exit(1);

} else if (pid == 0){ // дочерний процесс

int fd = open(argv[2], O\_RDWR); // откыть для чтения

if (fd == -1){

fprintf(stderr, "can`t open file\n");

} else fprintf(stdout, ">fd: correct\n");

if(dup2(fd, 0) == -1){ // заменить файловый дескриптор на поток стандартного ввода

fprintf(stderr, "dup eror\n");

} else fprintf(stdout, ">dup2: correct\n");

if (execlp(argv[1], argv[2], NULL) == -1){ // выполнить программу с аргументами agrv[1] и argv[2]

fprintf(stderr, "exec eror\n");

}

if(close(fd) == -1){

if(close(fd) == -1){

fprintf(stderr, "can`t close file\n"); // закрытие файла связанного с дескриптором

}

} else if(pid > 0){ // родительский процесс

if (waitpid(pid, &status, 0) == -1){ // ожидание дочернего процесса

fprintf(stderr, "smth wrong with parent\n"); // нужно ждать любого дочернего процесса, чей иден\

тификатор группы равен абсолютному значению pid.

} else fprintf(stdout, ">waitpid: correct\n"); // корректно

}

return 0;

}

**3. Протокол**

mmaxim2710@DESKTOP-RDPBU3D:~/OC$ ls

lab2.c test

mmaxim2710@DESKTOP-RDPBU3D:~/OC$ gcc -g lab2.c -o main

mmaxim2710@DESKTOP-RDPBU3D:~/OC$ ./main ls test

>fd: correct

>dup2: correct

lab2.c main test

>waitpid: correct

mmaxim2710@DESKTOP-RDPBU3D:~/OC$

**4. Объяснение результата работы программы**

Программа принимает в argv[1] и argv[2] параметры «команда» и «файл» соответственно. Если количество аргументов не равно 3 (переменная argc != 3), программа завершает работу.  
  
Далее создаётся дочерний процесс, который открывает файл для чтения/записи, заменяются файловые дескрипторы на поток стандартного ввода, выполняется команда интерпретатора команд с параметрами argv[1] и argv[2], закрывается файл.

Родительский процесс ожидает завершение дочернего процесса.

**5. Набор тестов**

1) Входные данные:

./main cat

Выходные данные:

Invalid input

2) Входные данные:

./main test

Выходные данные:

Invalid input

3) Входные данные:

./main ls test

Выходные данные:

>fd: correct

>dup2: correct

lab2.c main test

>waitpid: correct

**6. Strace**

execve("./main", ["./main", "cat", "test"], 0x7fffd3190e70 /\* 19 vars \*/) = 0

brk(NULL) = 0x7fffc4988000

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=54618, ...}) = 0

mmap(NULL, 54618, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f3b97967000

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, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"..., 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) = 0x7f3b97960000

mmap(NULL, 4131552, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f3b97200000

mprotect(0x7f3b973e7000, 2097152, PROT\_NONE) = 0

mmap(0x7f3b975e7000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f3b975e7000

mmap(0x7f3b975ed000, 15072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f3b975ed000

close(3) = 0

arch\_prctl(ARCH\_SET\_FS, 0x7f3b979614c0) = 0

mprotect(0x7f3b975e7000, 16384, PROT\_READ) = 0

mprotect(0x7f3b97c01000, 4096, PROT\_READ) = 0

mprotect(0x7f3b97827000, 4096, PROT\_READ) = 0

munmap(0x7f3b97967000, 54618) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f3b97961790) = 275

wait4(275, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 275

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=275, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

fstat(1, {st\_mode=S\_IFCHR|0660, st\_rdev=makedev(4, 1), ...}) = 0

**7. Вывод**

Проделав данную работу я научился пользоваться командами fork, dup2, wait, exec, waitpid, создавать дочерние и родительские процессы.