Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М80 – 201Б-19
Студент: Цыкин И.А.
Преподаватель: Миронов Е.С.
Оценка: _____
Дата: _____

Москва, 2020.

Содержание

Постановка задачи

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).

Вариант 6: В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int.

Общие сведения о программе

Программа компилируется из двух файлов parent.c и child.c.

В программе используются следующие системные вызовы:

1. **pipe** – для создания однонаправленного канала, через который могут общаться два процесса. Системный вызов возвращает два дескриптора файлов. Один для чтения из канала, другой для записи в канал.
2. **fork** – для создания дочернего процесса.
3. **int execve(const char *filename, char *const argv[], char *const envp[])** (и другие вариации exec) - замена образа памяти процесса
4. **int dup2(int oldfd, int newfd)** - переназначение файлового дескриптора
5. **freopen** - используется для замены файла связанного со стандартными потоками ввода-вывода

Общий метод и алгоритм решения.

Для реализации, поставленной задачи необходимо:

1. Создать программу, которую будет принимать данные, по этим данным открывать файл и прочитать команды, а затем вывести результат этих команд
2. Прочитать название файла
3. Открыть файл и направить его в стандартный поток ввода дочернего процесса
4. Используя системный вызов pipe создать канал, по которому будут обмениваться данными два процесса
5. Используя системный вызов fork создать дочерний процесс
6. В дочернем процессе происходит замена параметров вывода
7. В дочернем процессе происходит замена образа памяти процесса
8. Нужно посимвольно прочитать команды из стандартного ввода
9. Вывести результат используя write в стандартный поток вывода
10. Родительский процесс прочитывает данные и выводит результат

Листинг программы

parent.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define N 255

int main(int argc, char* argv[]){
	int fd[2], id;
	char name[N];
	printf("Enter name of file: ");
	scanf("%s", name);

	if(pipe(fd) == -1){
		printf("Error with opening pipe\n");
		return -1;
	}

	if(fork() > 0) {
		close(fd[1]);
		int n = 1;
		int res;
		while(read(fd[0], &res, sizeof(int))){
			printf("%d Result from child:%d\n", n, res);
			n++;
		}
		close(fd[0]);
	}else {
		FILE* file = freopen(name, "r", stdin);
		if(file == NULL){
			printf("Error with opening file\n");
			return -2;
		}
		if (dup2(fd[1],1) == -1){
			printf("Error with opening dup2\n");
			return -3;
		}
		char** arg = NULL;
		execve("./child", arg, NULL);
	}
	return 0;
}
```

child.c:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char* argv[]){
	char c;
	int a, b, res, flag;
	a = 0; res =0; flag = 0;
	while (scanf("%c", &c) > 0) {
		if (c >= '0' && c <= '9') {
		b = atoi(&c);
		a = a*10 + b;
		}else if (c == '-') {
			flag = 1;
		}else if(c == ' ') {
			if(flag){
				res-=a;
				flag = 0;
			}else{
				res+=a;
			}
			a = 0;
		}else if(c == '\n') {
			if(flag){
				res-=a;
				flag = 0;
			}else{
				res+=a;
			}
			write(1, &res, sizeof(res));
			res = 0;
			a = 0;
		}
	}
	return 0;
}
```

text.txt:

```
3 2 5
1 2 3
0 0 3
-3 -4 -5 -1 -2  15
10 101 1000
```

Результаты работы программы

```
vaney@V-box:~$ cd */os_lab2
vaney@V-box:~/Examples/os_lab2$ gcc -o a.out parent.c
vaney@V-box:~/Examples/os_lab2$ gcc -o child child.c
vaney@V-box:~/Examples/os_lab2$ ./a.out
Enter name of file: test.txt
1 Result from child:10
2 Result from child:6
3 Result from child:3
4 Result from child:0
5 Result from child:1111
```

Strace

```
vaney@V-box:~/Examples/os_lab2$ strace ./main
execve("./main", ["./main"], 0x7ffd33080780 /* 58 vars */) = 0
brk(NULL)                               = 0x55f645a3b000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffeae063930) = -1 EINVAL (Invalid
argument)
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=67999, ...}) = 0
mmap(NULL, 67999, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f8511ec1000
close(3)                                = 0
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\360q\2\0\0\0\0\0"...,
832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\
0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\
0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\
355Y\377\t\334"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8511ebf000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\
0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\
0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\
355Y\377\t\334"..., 68, 880) = 68
mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f8511ccd000
mprotect(0x7f8511cf2000, 1847296, PROT_NONE) = 0
mmap(0x7f8511cf2000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x25000) = 0x7f8511cf2000
mmap(0x7f8511e6a000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x19d000) = 0x7f8511e6a000
mmap(0x7f8511eb5000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1e7000) = 0x7f8511eb5000
mmap(0x7f8511ebb000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7f8511ebb000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f8511ec0540) = 0
mprotect(0x7f8511eb5000, 12288, PROT_READ) = 0
mprotect(0x55f644f65000, 4096, PROT_READ) = 0
mprotect(0x7f8511eff000, 4096, PROT_READ) = 0
munmap(0x7f8511ec1000, 67999)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
brk(NULL)                               = 0x55f645a3b000
brk(0x55f645a5c000)                     = 0x55f645a5c000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "Enter name of file: ", 20Enter name of file: )    = 20
read(0, test.txt
"test.txt\n", 1024)             = 9
```

```
pipe([3, 4])                            = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLD, child_tidptr=0x7f8511ec0810) = 4620
close(4)                                = 0
read(3, "\n\0\0\0", 4)                  = 4
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4620, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
write(1, "1 Result from child:10\n", 231 Result from child:10
) = 23
read(3, "\6\0\0\0", 4)                  = 4
write(1, "2 Result from child:6\n", 222 Result from child:6
) = 22
read(3, "\3\0\0\0", 4)                  = 4
write(1, "3 Result from child:3\n", 223 Result from child:3
) = 22
read(3, "\0\0\0\0", 4)                  = 4
write(1, "4 Result from child:0\n", 224 Result from child:0
) = 22
read(3, "W\4\0\0", 4)                   = 4
write(1, "5 Result from child:1111\n", 255 Result from child:1111
) = 25
read(3, "", 4)                          = 0
close(3)                                = 0
lseek(0, -1, SEEK_CUR)                  = -1 ESPIPE (Illegal seek)
exit_group(0)                           = ?
+++ exited with 0 +++
vaney@V-box:~/Examples/os_lab2$
```

Вывод

Выполнив лабораторную работу, я познакомился с такими вещами, как процессы и каналы, а также с системные вызовы. Связывание процессов может помочь повысить эффективность программы, решающей задачу, в которой есть множество действий. Каналы позволяют передавать информацию между процессами, что позволяет процессам синхронизировать работу, решая общую задачу, с их помощью можно управлять работой процессов, передавая им различные данные. Эти знания необходимы для эффективного написания программ, использующих идею разделенния процессов.