**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Операционные системы»

**Лабораторная работа № 2**

Тема: Управление процессами в ОС

Студент: Волков М. А.

Группа: 80-207

Преподаватель: Миронов Е. С.

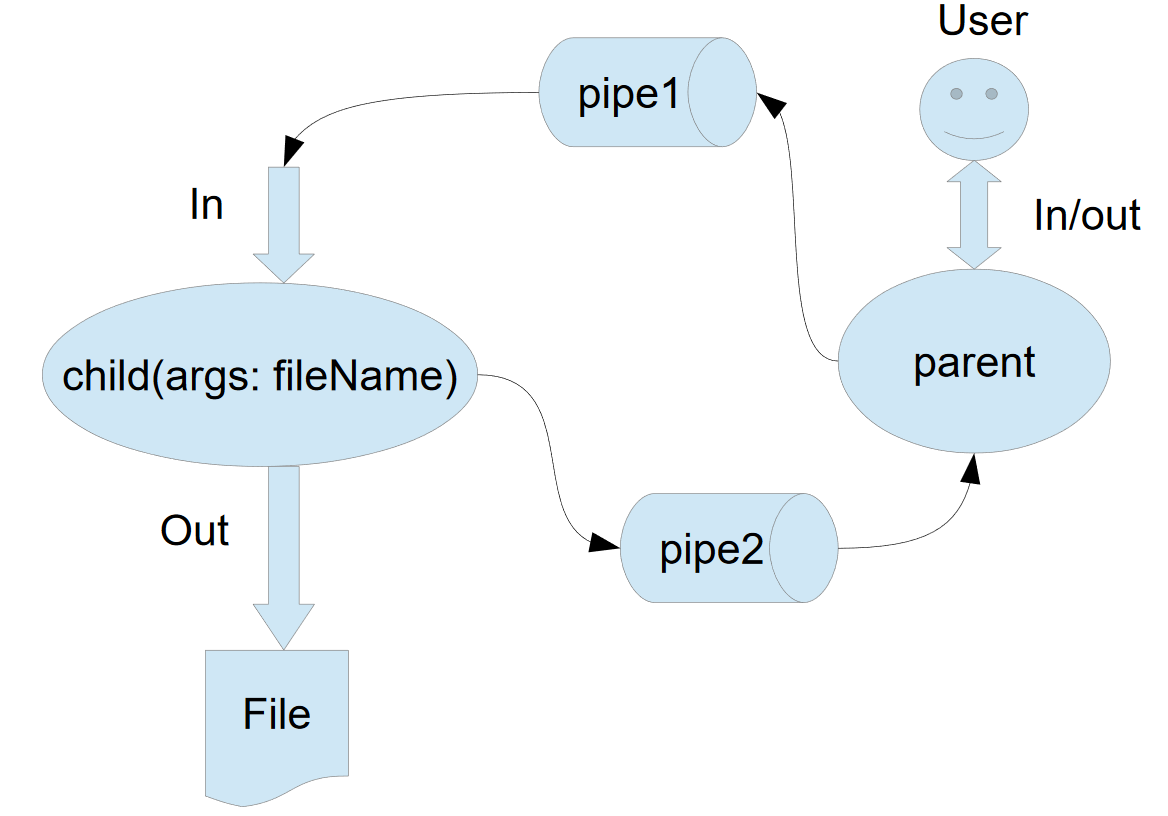
Дата:

Оценка:

**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*). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

*Вариант 1.*



Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

Пользователь вводит команды вида: «число число число <endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип *int*.

**Алгоритм решения задачи**

Программа разбита на 2 файла: *main.c* и *child.c.*

В *main.c* находится программа, которая занимается контролем всех дочерних и родительских процессов, а также создает и перенаправляет в нужные места *pipe*. Также в этом файле идет запрос у пользователя данных, которые считываются в родителе посимвольно, затем символы передаются в *child.c*, после чего парсятся там. В *main.c* происходит также и закрытие ненужных *pipe.*

В *child.c* написана программа, в которой выполняется вышеуказанное задание и отправка ответа в файл, название которого пользователь вводит при запуске программы.

**Листинг программы**

***main.c***

#include "unistd.h"  
#include "stdio.h"  
  
#define WRITE 1  
#define READ 0  
  
**int** main() {  
 printf("Enter filename\n");  
 fflush(stdout);  
 **char** arg[256];  
 scanf("%s\n", arg);  
 **int** fd2[2];  
 **int** fd1[2];  
 **if** (pipe(fd2) < 0) {  
 printf("Pipe error!");  
 **return** -1;  
 }  
 **if** (pipe(fd1) < 0) {  
 printf("Pipe error!");  
 **return** -1;  
 }  
  
 **int** ID = fork();  
 **if** (ID < 0) {  
 printf("Fork error!");  
 **return** 1;  
 }  
// ----------- parent----------  
 **else if** (ID > 0) {  
 close(fd1[READ]);  
 close(fd2[WRITE]);  
 **char** tmp;  
 **while** (scanf("%c", &tmp) > 0) {  
 write(fd1[WRITE], &tmp, **sizeof**(**char**));  
 }  
 close(fd1[WRITE]);  
 close(fd2[READ]);  
 }  
 //---------child----------  
 **else** {  
 close(fd1[WRITE]);  
 close(fd2[READ]);  
  
 dup2(fd1[READ], STDIN\_FILENO);  
 execl("child.out", arg, NULL);  
  
 close(fd1[READ]);  
 close(fd2[WRITE]);  
  
 }  
 **return** 0;  
}

***child.c***

#include "unistd.h"  
#include "stdio.h"  
#include "vector.h"  
  
**int** transform(**char** x) {  
 **return** x - '0';  
}  
  
**int** main(**int** count, **char** \*filename[]) {  
  
 FILE \*output = NULL;  
 output = fopen(filename[0], "a");  
 **if** (output == NULL) {  
 printf("file error");  
 fclose(output);  
 **return** 1;  
 }  
 vector nums;  
 create(&nums, 0);  
 **char** tmp;  
 **int** is\_minus = 1;  
 **int** digit\_result = 0;  
 **int** if\_space = 0;  
 **while** (read(STDIN\_FILENO, &tmp, **sizeof**(**char**)) > 0) {  
 **if** (tmp == '-') {  
 if\_space = 0;  
 is\_minus = -1;  
 }  
  
 **if** (tmp == ' ') {  
 digit\_result /= 10;  
 digit\_result = is\_minus \* digit\_result;  
 push\_back(&nums, digit\_result);  
 is\_minus = 1;  
 digit\_result = 0;  
 if\_space = 1;  
 }  
  
 **if** (tmp >= '0' && tmp <= '9') {  
 if\_space = 0;  
 digit\_result += transform(tmp);  
 digit\_result \*= 10;  
 }  
  
 **if** (tmp == '\n') {  
 **if** (!if\_space) {  
 digit\_result /= 10;  
 digit\_result = is\_minus \* digit\_result;  
 push\_back(&nums, digit\_result);  
 is\_minus = 1;  
 digit\_result = 0;  
 if\_space = 1;  
 }  
 **int** i;  
 **int** result = 0;  
 **for** (i = 0; i < size(&nums); ++i) {  
 result += nums.data[i];  
 fprintf(output, "%d ", nums.data[i]);  
 }  
 fprintf(output, "= %d\n", result);  
 destroy(&nums);  
 }  
 }  
 fclose(output);  
 **return** 0;  
}

**Тесты и протокол исполнения**

Тест:

Enter filename

test.txt

234234 23423 54 523 1 123

1 2 3 4 5 6 7 -23

65 -65

^D

Process finished with exit code 0

test.txt:

234234 23423 54 523 1 123 = 258358

1 2 3 4 5 6 7 -23 = 5

65 -65 = 0

strace:

execve("./OC", ["./OC"], 0x7ffc508d1c00 /\* 20 vars \*/) = 0

brk(NULL) = 0x563a1191f000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe819b96e0) = -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=34748, ...}) = 0

mmap(NULL, 34748, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f8f73b44000

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) = 0x7f8f73b42000

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) = 0x7f8f73950000

mprotect(0x7f8f73975000, 1847296, PROT\_NONE) = 0

mmap(0x7f8f73975000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7f8f73975000

mmap(0x7f8f73aed000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7f8f73aed000

mmap(0x7f8f73b38000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f8f73b38000

mmap(0x7f8f73b3e000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f8f73b3e000

close(3) = 0

arch\_prctl(ARCH\_SET\_FS, 0x7f8f73b43580) = 0

mprotect(0x7f8f73b38000, 12288, PROT\_READ) = 0

mprotect(0x563a0fbe3000, 4096, PROT\_READ) = 0

mprotect(0x7f8f73b7a000, 4096, PROT\_READ) = 0

munmap(0x7f8f73b44000, 34748) = 0

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

brk(NULL) = 0x563a1191f000

brk(0x563a11940000) = 0x563a11940000

**write(1, "Enter filename\n", 15Enter filename**

**) = 15**

**fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0**

**read(0, file.txt**

**"file.txt\n", 1024) = 9**

**read(0, 1 2 3 4 5**

**"1 2 3 4 5\n", 1024) = 10**

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

**pipe([5, 6]) = 0**

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f8f73b43850) = 2791**

**close(5) = 0**

**close(4) = 0**

**write(6, "1", 1) = 1**

**write(6, " ", 1) = 1**

**write(6, "2", 1) = 1**

**write(6, " ", 1) = 1**

**write(6, "3", 1) = 1**

**write(6, " ", 1) = 1**

**write(6, "4", 1) = 1**

**write(6, " ", 1) = 1**

**write(6, "5", 1) = 1**

**write(6, "\n", 1) = 1**

**read(0, "", 1024) = 0**

**close(6) = 0**

**close(3) = 0**

**exit\_group(0) = ?**

+++ exited with 0 +++

**Вывод**

Благодаря проделанной работе я узнал что такое *pipe* и как им пользоваться. Научился передавать работу программы в другие процессы. Во время выполнения работы я столкнулся с проблемой взаимодействия процессов, с чем мне помог системный вызов *dup2.* Также, в ходе безуспешной отладки при помощи *gdb*, осознал, что отладить при помощи данной программы подобную работу невозможно. Также узнал о существовании ошибки *stack smashed detected* и успешно разобрался в причине ее происхождения.

**Список литературы**

1. [Изучаем процессы в Linux / Хабр — Habr](https://habr.com/ru/post/423049/)
2. Таненбаум Э., Бос Х. *Современные операционные системы. — 4-е изд.*