Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Управление процессами в ОС. Обеспечение обмена данных между процессами посредством каналов.**

Студент: Бутырев Даниил Вячеславович

Группа: М80-206Б-20

Вариант: 5

Преподаватель: Соколов Андрей Алексеевич

Дата: 30.10.2021

Оценка: 5

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**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.

Родительский процесс создаёт дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процессы должны быть представлены разными программами. Родительский процесс передаёт 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.

5 вариант) Пользователь вводит команды вида: «число<endline>». Далее это число передается от родительского процесса в дочерний. Дочерний процесс производит проверку на простоту. Если число составное, то в это число записывается в файл. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

**Листинг программы**

#include <unistd.h>

#include <iostream>

#include <sys/types.h>

#include <cstdlib>

#include <charconv>

#include <string>

enum {

READ\_END=0,

WRITE\_END=1,

};

enum {

STDIN = 0,

STDOUT = 1,

STDERR = 2,

};

enum {

ALL\_GOOD=0,

ERROR\_CREATING\_PIPE,

ERROR\_CHILD\_ARGS,

ERROR\_CREATING\_PROCESS,

ERROR\_EXECUTING\_PROGRAM,

ERROR\_READING\_INPUT,

ERROR\_WRITING\_PIPE,

ERROR\_READING\_PIPE,

ERROR\_WRITING\_ERR,

};

const char error\_creating\_pipe[] = "can't create pipe\n";

const char error\_child\_args[] = "num of descriptor is too large\n";

const char error\_creating\_process[] = "can't create another process\n";

const char error\_executing\_program[] = "can't exec prog\n";

const char error\_reading\_input[] = "error reading from input\n";

const char error\_writing\_pipe[] = "error writing to pipe\n";

const char error\_reading\_pipe[] = "error reading from pipe\n";

//char buffer[200] = { 0 };

int read\_num(int& num);

int main() {

const int filename\_length = 40;

char filename[filename\_length];

//int last\_func\_error\_code;

for(int i = 0; i < filename\_length; i++) {

if(read(STDIN, filename+i, sizeof(char)) == -1) {

if(write(STDERR, error\_reading\_input, sizeof(error\_reading\_input)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_READING\_INPUT;

}

if(((filename[i] < 'a') || (filename[i] > 'z')) && ((filename[i] < 'A') || (filename[i] > 'Z')) && (filename[i]!='/') && \

(filename[i] != '.') && (filename[i] != '\_')) {

filename[i] = '\0';

break;

}

}

//std::string&& error\_opening\_file = static\_cast<std::string>("can't execute prog ") + static\_cast<std::string>(filename);

int number = 1; //>0

bool all\_ok = true;

int pipe\_parent\_to\_son[2] = { 0 };//

char pipe\_son\_reading\_pchar\_arg[20] = { 0 }; //

int pipe\_son\_to\_parent[2] = { 0 }; //

char pipe\_son\_writing\_pchar\_arg[20] = { 0 }; //

if(pipe(pipe\_parent\_to\_son)==-1) {//

if(write(STDERR, error\_creating\_pipe, sizeof(error\_creating\_pipe)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_CREATING\_PIPE;

}

if(pipe(pipe\_son\_to\_parent)==-1) {//

if(write(STDERR, error\_creating\_pipe, sizeof(error\_creating\_pipe)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_CREATING\_PIPE;//

}

if(std::to\_chars(pipe\_son\_reading\_pchar\_arg, pipe\_son\_reading\_pchar\_arg+20, \

pipe\_parent\_to\_son[READ\_END]).ec == std::errc::value\_too\_large) {

if(write(STDERR, error\_child\_args, sizeof(error\_child\_args)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_CHILD\_ARGS;

}

//locale-independent - what does it mean?

if(std::to\_chars(pipe\_son\_writing\_pchar\_arg, pipe\_son\_writing\_pchar\_arg+20, pipe\_son\_to\_parent[WRITE\_END]).ec == std::errc::value\_too\_large) {

if(write(STDERR, error\_child\_args, sizeof(error\_child\_args)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_CHILD\_ARGS;

}

//std::cout <<pipe\_parent\_to\_son[READ\_END] << " = " << pipe\_son\_reading\_pchar\_arg << ' ' << pipe\_son\_to\_parent[WRITE\_END] << " = " << pipe\_son\_writing\_pchar\_arg << std::endl;

pid\_t pid = fork();

if(pid < 0) { //

if(write(STDERR, error\_creating\_process, sizeof(error\_creating\_process)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_CREATING\_PROCESS;//

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

//std::cout << "from child: child pid:" << getpid() << std::endl;//

close(pipe\_parent\_to\_son[WRITE\_END]); //errors?

close(pipe\_son\_to\_parent[READ\_END]); // why doesn't pipe close everywhere?

if(execl(filename, pipe\_son\_reading\_pchar\_arg, pipe\_son\_writing\_pchar\_arg, static\_cast<char\*>(NULL)) == -1) {// if I give pointer, would it work?

if(write(STDERR, error\_executing\_program, sizeof(error\_executing\_program)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_EXECUTING\_PROGRAM;

}

} else {

close(pipe\_parent\_to\_son[READ\_END]);

close(pipe\_son\_to\_parent[WRITE\_END]);

//std::cout << "from parent: child pid" << pid << std::endl;

}

//FILE write\_nums\_here = fdopen(pipe\_parent\_to\_son[1], "w"); - it's wrong but why?

while(all\_ok) {

read\_num(number);

if(write(pipe\_parent\_to\_son[WRITE\_END], &number, sizeof(number)) == -1) {

if(write(STDERR, error\_writing\_pipe, sizeof(error\_writing\_pipe)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_WRITING\_PIPE;

} else if(read(pipe\_son\_to\_parent[READ\_END], &all\_ok, sizeof(all\_ok)) == -1) {

if(write(STDERR, error\_reading\_pipe, sizeof(error\_reading\_pipe)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_READING\_PIPE;

}

}

return ALL\_GOOD;

}

int read\_num(int& num) {

int sign = 1;

num = 0;

char num\_ch = 0;

bool go\_on = true;

while(true) {

if(read(STDIN, &num\_ch, sizeof(num\_ch)) == -1) {

if(write(STDERR, error\_reading\_input, sizeof(error\_reading\_input)) == -1) {

return -1;//

}

return -1;//

}

if(num\_ch == '-') {

sign = -1;

continue;

}

if((num\_ch >= '0') && (num\_ch <= '9')) break;

sign = 1;

}

while(go\_on) {

num=num\*10+(num\_ch-'0');

if(read(STDIN, &num\_ch, sizeof(num\_ch)) == -1) {

if(write(STDERR, error\_reading\_input, sizeof(error\_reading\_input)) == -1) {

return -1;//

}

return -1;//

}

if((num\_ch < '0') || (num\_ch > '9')) break;

}

num\*=sign;

return 0;

}

#include <unistd.h>

#include <iostream>

#include <sys/types.h>

#include <string>

#include <fstream>

#include <algorithm>

#include <set>

#include <cmath>

#include <charconv>

#include <sys/stat.h>

#include <fcntl.h>

enum {

READ\_END=0,

WRITE\_END=1,

};

enum {

STDIN = 0,

STDOUT = 1,

STDERR = 2,

};

enum {

ALL\_GOOD=0,

ERROR\_CREATING\_PIPE,

ERROR\_CHILD\_ARGS,

ERROR\_CREATING\_PROCESS,

ERROR\_EXECUTING\_PROGRAM,

ERROR\_READING\_INPUT,

ERROR\_WRITING\_PIPE,

ERROR\_READING\_PIPE,

ERROR\_WRITING\_ERR,

};

bool is\_prime(const int& number);

void put\_all\_primes\_until(std::set<int>& primes, int num);

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

const char space = ' ';

const char error\_not\_enough\_arguments[] = "not enough arguments";

const char error\_reading\_pipe[] = "child: error with reading";

const char error\_writing\_pipe[] = "child: error with writing";

if(argc < 2) {

if(write(STDERR, error\_not\_enough\_arguments, sizeof(error\_not\_enough\_arguments)) == -1) {

return ERROR\_WRITING\_ERR;

}

for(int i = 0; i < argc; i++)

if(write(STDERR, argv[i], sizeof(argv[i])) == -1) {

return ERROR\_WRITING\_ERR;

}

return 1; //hz pochemy ne 3 argumenta kak pri vizove iz kommandnoi stroki

}

int&& pipe\_parent\_to\_son\_read = std::stoi(static\_cast<std::string>(argv[0]));

int&& pipe\_son\_to\_parent\_write = std::stoi(static\_cast<std::string>(argv[1]));

int file\_descriptor = open("numbers.ans", O\_CREAT | O\_TRUNC | O\_WRONLY);

int number = 1;// kst kak vishlo v proshlii raz, esli negative opredelalsa vne while

char number\_string[10] = { 0 };

int num\_size\_to\_write = 0;

while(true) {

if(read(pipe\_parent\_to\_son\_read, &number, sizeof(number)) == -1) {

if(write(STDERR, error\_reading\_pipe, sizeof(error\_reading\_pipe)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_READING\_PIPE;

}

//std::cerr << number << ' ';

bool negative = number < 0;

bool prime = false;

if(!negative) {

prime = is\_prime(number);

//magic \_\_ prime=

}

bool num\_is\_correct = !(negative || prime);

if(write(pipe\_son\_to\_parent\_write, &num\_is\_correct, sizeof(num\_is\_correct)) == -1) {

if(write(STDERR, error\_writing\_pipe, sizeof(error\_writing\_pipe)) == -1) {

return ERROR\_WRITING\_ERR;

}

return ERROR\_WRITING\_PIPE;

}

if(num\_is\_correct) {

for(long unsigned int i = 0; i < sizeof(number\_string); i++) number\_string[i] = '\0';

std::to\_chars(number\_string, number\_string + sizeof(number\_string), number);

//std::cerr << number\_string << ' ';

for(long unsigned int i = 0; i < sizeof(number\_string); ++i)

if(number\_string[i] == '\0') { num\_size\_to\_write = i; break;} //I can div 10

write(file\_descriptor, number\_string, num\_size\_to\_write);

write(file\_descriptor, &space, sizeof(' '));

}

else return ALL\_GOOD;

}

return ALL\_GOOD;

}

bool is\_prime(const int& number) {

static std::set<int> primes{2, 3, 5, 7, 11, 13};

if(number < 2) return false;

if(primes.count(number)) return true;

//int min\_n = min(std::trunc(std::sqrt(number)), \*primes.rbegin());

if(std::trunc(std::sqrt(number)) > \*primes.rbegin())

put\_all\_primes\_until(primes, std::trunc(std::sqrt(number)));

for(auto another\_prime = primes.begin(); another\_prime != primes.end(); another\_prime++)

if(number % (\*another\_prime )== 0) return false;

primes.insert(number); //

return true;

} //make declarations of functions in the top of the prog

void put\_all\_primes\_until(std::set<int>& primes, int num) {

int max\_exist = \*(primes.rbegin());

if(max\_exist > num) return;

for(int i = (max\_exist - max\_exist%6)+6; i <= num; i+=6) {

is\_prime(i-1);

is\_prime(i+1); //it pushes them for u

}

}

**Примеры работы**

steep@gg:~/educat/instit/os/2lab/src/parent/build$ ./os\_lab\_02

../../child/build/child

200

300

400

500

3

steep@gg:~/educat/instit/os/2lab/src/parent/build$ sudo cat numbers.ans

500 200 300 400 steep@gg:~/educat/instit/os/2lab/src/parent/build$

**Вывод**

Я научился пользоваться pipe-ом, понял, как работают системные вызовы read и write.