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: Попов Илья Павлович

Группа: М80-206Б-20

Вариант: 3

Преподаватель: Соколов Андрей Алексеевич

Дата: 20.11.2021

Оценка: 5

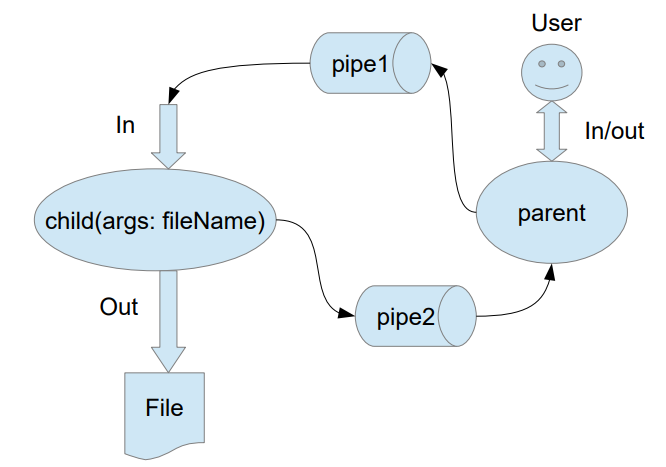
Москва, 2021

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 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.



3 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

Листинг программы

**main.c**

/\*

Попов Илья М80-206Б-20

Группа вариантов 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 принеобходимости

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.

3 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа

передаются от родительского процесса в дочерний. Дочерний процесс производит деление

первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то

тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна

осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может

быть произвольным.

\*/

//gcc child.c string.c -o child

//gcc main.c vector.c string.c

//./a.out

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <sys/wait.h>

#include "unistd.h"

#include "vector.h"

#include "string.h"

typedef enum{

R\_SUCCESS,

R\_EOL,

R\_EOF,

R\_ERROR,

} r\_status;

r\_status reading\_int(int \*cur){//чтение чисел типа int с STDIN

char c;

\*cur = 0;

int tmp = read(STDIN\_FILENO, &c, sizeof(char));

while(tmp > 0){

if(c == '\n') return R\_EOL;

if(c == ' ') break;

if((c < '0') || (c > '9')){

return R\_ERROR;

}

\*cur = \*cur \* 10 + c - '0';

tmp = read(STDIN\_FILENO, &c, sizeof(char));

}

if(tmp == 0) return R\_EOF;

return R\_SUCCESS;

}

void reading\_filename(string\* str){

char cur;

while(read(STDIN\_FILENO, &cur, sizeof(char)) > 0){

if(cur == '\n'){

break;

}

s\_push(str, cur);

}

}

void usage\_str(){

printf("Введите имя файла, в который будет записан результат работы программы.\n\n");//----------------------------------------

}

void usage\_vect(){

printf("\n\nВведите числа в виде «число число число<endline>».\nРезультат работы программы - набор чисел - часных от деления первого введеного числа на последующие.\n\n");//----------------------------------------

}

int main(){

int fd1[2];

int fd2[2];

bool first = true;

vector v;

string file\_name;

if (first){//считывание имени файла и чисел (на первом заходе в родительский процесс)

//////////////////////////////////////////////////////////////////////

usage\_str();

s\_init(&file\_name);

reading\_filename(&file\_name);

//////////////////////////////////////////////////////////////////////

usage\_vect();

int tmp = 0;

v\_init(&v);

r\_status stat = reading\_int(&tmp);

while(stat != R\_ERROR){

v\_push\_back(&v, tmp);

if(stat == R\_EOF){

perror("\nUSAGE: «число число число<endline>»\n");

return -8;

} else if(stat == R\_EOL){

break;

}

tmp = 0;

stat = reading\_int(&tmp);

}

if (stat == R\_ERROR){

perror("Wrong value\n");

return -9;

}

first = false;

//////////////////////////////////////////////////////////////////////

}else{//на втором заходе в родительский процесс удаляем созданную строку и вектор чисел

s\_destroy(&file\_name);

v\_destroy(&v);

}

// Попытаемся создать pipe'ы

if(pipe(fd1) < 0){// Если создать pipe не удалось, печатаем об этом сообщение и прекращаем работу

perror("Pipe error\n");

return -1;

}

if(pipe(fd2) < 0){// Если создать pipe не удалось, печатаем об этом сообщение и прекращаем работу

perror("Pipe error\n");

return -2;

}

// Порождаем новый процесс

int pid = fork();

if(pid < 0){ // Если создать процесс не удалось, сообщаем об этом и завершаем работу

perror("Fork error\n");

return -3;

}

//parent

else if (pid > 0){

printf("\n[%d] It's parent. Child id: %d\n\n", getpid(), pid);//----------------------------------------

fflush(stdout);

close(fd1[0]);//закрываем канал на чтение pipe1, т.к. мы в него пишем

close(fd2[1]);//закрываем канал на запись pipe2, т.к. мы из него потом читать будем(после завершения работы дочернего прочеса)

//запись в pipe1 от родителя ребенку

////////////////////////////////////////////////

//запись в pipe1 имени файла

int len\_str = s\_get\_size(&file\_name);

write(fd1[1], &len\_str, sizeof(int));

printf("Your file\_name: %s\n", s\_get\_all(&file\_name));//----------------------------------------

for(int i = 0; i < len\_str; i++){

char x = s\_get(&file\_name, i);

write(fd1[1], &x, sizeof(char));

}

///////////////////////////////////////////////

//запись в pipe1 чисел

int len = v\_get\_size(&v);

write(fd1[1], &len, sizeof(int));

printf("Your numbers: ");//----------------------------------------

for (int i = 0; i < len; i++){

int x = v\_get(&v, i);

printf("%d ", x);//----------------------------------------

write(fd1[1], &x, sizeof(int));

}printf("\n\n");

//ждем завершение работы дочернего

int status;

waitpid(pid, &status, 0);

if (WIFEXITED(status)){

int exit\_status = WEXITSTATUS(status);

printf("Exit status of the child was %d\n", exit\_status);//----------------------------------------

if (exit\_status == 1){

perror("You can't divide by zero!\n\n");

return -8;

}

else if (exit\_status != 0){

perror("Some error!\n\n");

return -9;

}

}

printf("\n[%d] It's parent. Child id: %d\n\n", getpid(), pid);//----------------------------------------

fflush(stdout);

//читаем результат работы дочернего процесса из pipe2

int res;

int length;

read(fd2[0], &length, sizeof(int));

printf("Your processed numbers: ");//----------------------------------------

while(length > 0){

read(fd2[0], &res, sizeof(int));

printf("%d ", res);

length--;

}

printf("\n\n\n");//----------------------------------------

close(fd1[1]);

close(fd2[0]);//закрываем оставшиеся каналы у pipe1 и pipe2

}

//child

else {

printf("[%d] It's child\n\n", getpid());//----------------------------------------

fflush(stdout);

close(fd1[1]);

close(fd2[0]);//закрываем эти каналы, т.к. они нам не понадобятся, а два оставшихся закрывать смысла нет, т.к. после завершения функции execl() мы сразу вернемся в процесс parent

//создем копии файловых дескрипторов

if (dup2(fd1[0], STDIN\_FILENO) == -1){//перенаправить pipe1 на консольный ввод дочернему процессу

perror("Cannot dup reading channel of pipe1 to stdin\n");

return -5;

}

if (dup2(fd2[1], STDOUT\_FILENO)== -1){//перенаправить консольный вывод дочернего процесса в pipe2

perror("Cannot dup recording channel of pipe2 to stdout\n");

return -6;

}

if (execl("child", "", NULL) == -1){//заменяет текущий процесс, процессом, описанном в исп. файле

perror("Execl child problem\n");

return -7;

}

}

return 0;

}

**child.c**

#include <stdio.h>

#include <stdbool.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include "unistd.h"

#include "vector.h"

#include "string.h"

void write\_num(int a, int fd) {

char\* num;

if (a == 0) num = "0";

if (a == 1) num = "1";

if (a == 2) num = "2";

if (a == 3) num = "3";

if (a == 4) num = "4";

if (a == 5) num = "5";

if (a == 6) num = "6";

if (a == 7) num = "7";

if (a == 8) num = "8";

if (a == 9) num = "9";

if (a == -1) num = " ";

write(fd, num, sizeof(char));

}

int pow\_ten(int l){

int res = 1;

while (l > 0){

res \*= 10;

l--;

}

return res;

}

int length\_int(int cur){

int tmp = cur;

int c = 0;

while (tmp > 0){

tmp /= 10;

c++;

}

return c;

}

void writing\_int(int cur, int fd){

int l = length\_int(cur) - 1;

while (cur > 0){

int tmp = cur;

int c = tmp / pow\_ten(l);

write\_num(c, fd);

cur = cur % pow\_ten(l);

l--;

}

write\_num(-1, fd);

}

int main(){

///////////////////////////////////////////////

int len\_str;

read(STDIN\_FILENO, &len\_str, sizeof(int));

string file\_name;

s\_init(&file\_name);

for (int i = 0; i < len\_str; i++){

char x;

read(STDIN\_FILENO, &x, sizeof(char));

s\_push(&file\_name, x);

}

s\_push(&file\_name, '\0');

int file = open(s\_get\_all(&file\_name), O\_WRONLY);

if(file == -1){

s\_destroy(&file\_name);

return -1;

}

s\_destroy(&file\_name);

///////////////////////////////////////////////

int x, delimoe, delitel;

int len;

read(STDIN\_FILENO, &len, sizeof(int));

write(STDOUT\_FILENO, &len, sizeof(int));

bool first = true;

while (len > 0){

read(STDIN\_FILENO, &x, sizeof(int));

if (first){

delimoe = x;

first = false;

writing\_int(delimoe, file);

write(STDOUT\_FILENO, &delimoe, sizeof(int));

}

else{

delitel = x;

if (delitel == 0){

exit(1);

}

int res = delimoe / delitel;

writing\_int(res, file);

write(STDOUT\_FILENO, &res, sizeof(int));

}

len--;

}

close(file);

return 0;

}

**Makefile**

start: prog.out child

rm -r \*.o

./prog.out

prog.out: main.o string.o vector.o

gcc -o prog.out main.o string.o vector.o -lm

child: child.o string.o

gcc -o child child.o string.o -lm

play: prog.out child

valgrind ./prog.out

main.o: main.c string.c vector.c

child.o: child.c string.c

.c.o:

gcc -pedantic -Wall -g -c $< -o $@

**string.c**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include "string.h"

void s\_init(string \*s){

s->buf = NULL;

s->size = 0;

s->capacity = 0;

}

void s\_destroy(string \*s){

s->size = 0;

free(s->buf);

s->capacity = 0;

s->buf = NULL;

}

int s\_get\_cap(string \*s){

return s->capacity;

}

int s\_get\_size(string \*s){

return s->size;

}

bool s\_is\_empty(string \*s){

return s->size == 0;

}

bool s\_grow\_buf(string \*s){

int tmp = s->capacity \* 3 / 2;

if(!tmp){

tmp = 10;

}

char \*newd = realloc(s->buf, sizeof(char) \* tmp);

if(newd != NULL) {

s->buf = newd;

s->capacity = tmp;

return true;

}

return false;

}

bool s\_push(string \*s, char new\_char){

if(s\_get\_size(s) == s\_get\_cap(s)){

if(!s\_grow\_buf(s))

return false;

}

s->buf[s\_get\_size(s)] = new\_char;

s->size++;

return true;

}

bool s\_shrink\_buf(string \*s){

int tmp = s->capacity \* 4 / 9;

if(tmp < s\_get\_size(s)){

return true;

}

char \*newd = realloc(s->buf, sizeof(char) \* tmp);

if(newd != NULL) {

s->buf = newd;

s->capacity = tmp;

return true;

}

return false;

}

char s\_pop(string \*s){

char tmp = s->buf[s\_get\_size(s) - 1];

s\_shrink\_buf(s);

s->size--;

return tmp;

}

char s\_get(string \*s, int i){

return s->buf[i];

}

char\* s\_get\_all(string \*s){

return s->buf;

}

**string.h**

#ifndef STRING\_H\_

#define STRING\_H\_

#include <stdbool.h>

#include <stdlib.h>

typedef struct {

int size;

int capacity;

char \*buf;

} string;

void s\_init(string \*s);

void s\_destroy(string \*s);

int s\_get\_cap(string \*s);

int s\_get\_size(string \*s);//

bool s\_is\_empty(string \*s);

bool s\_grow\_buf(string \*s);

bool s\_push(string \*s, char new\_char);

bool s\_shrink\_buf(string \*s);

char s\_pop(string \*s);

char s\_get(string \*s, int i);

char\* s\_get\_all(string \*s);

#endif

**vector.c**

#include <stdlib.h>

#include <stdbool.h>

#include <stdio.h>

const size\_t MIN\_CAP = 4;

typedef struct{

int \*buf;

size\_t size;

size\_t capacity;

} vector;

bool v\_init(vector \*v){

int \*newbuf = malloc(MIN\_CAP \* sizeof(int));

if (newbuf != NULL){

v->buf = newbuf;

v->size = 0;

v->capacity = MIN\_CAP;

return true;

}

return false;

}

void v\_destroy(vector \*v){

free(v->buf);

v->buf = NULL;

v->size = 0;

v->capacity = 0;

}

int v\_get(vector \*v, size\_t i){

return v->buf[i];

}

void v\_set(vector \*v, size\_t i, int val){

v->buf[i] = val;

}

size\_t v\_get\_size(vector \*v){

return v->size;

}

size\_t v\_get\_cap(vector \*v){

return v->capacity;

}

bool empty(vector \*v){

if (v->size == 0) {return true;}

else {return false;}

}

bool v\_set\_size(vector \*v, size\_t new\_size){

if (new\_size > v->capacity){

size\_t new\_cap = v->capacity \* 3 / 2;

if (new\_cap < new\_size){

new\_cap = new\_size;

}

if (new\_cap < MIN\_CAP){

new\_cap = MIN\_CAP;

}

int \*new\_buf = realloc(v->buf, new\_cap \* sizeof(int));

if (new\_buf == NULL){

return false;

}

v->buf = new\_buf;

v->capacity = new\_cap;

}

else if (new\_size \* 3 / 2 < v->capacity){

size\_t new\_cap = new\_size \* 3 / 2;

if (new\_cap < MIN\_CAP){

new\_cap = MIN\_CAP;

}

v->buf = realloc(v->buf, new\_cap \* sizeof(int));

v->capacity = new\_cap;

}

for (size\_t i = v->size; i < new\_size; i++){

v\_set(v, i, 0);

}

v->size = new\_size;

return true;

}

int v\_pop\_back(vector \*v){

int res = v\_get(v, v\_get\_size(v) - 1);

v\_set\_size(v, v\_get\_size(v) - 1);

return res;

}

bool v\_push\_back(vector \*v, int val){

if (v\_set\_size(v, v\_get\_size(v) + 1)){

v\_set(v, v\_get\_size(v) - 1, val);

return true;

}

return false;

}

void v\_print(vector \*v){

for (int i = 0; i < v\_get\_size(v); i++){

printf("%d ", v\_get(v, i));

}

printf("\n\n");

}

**vector.h**

#ifndef VECTOR\_H

#define VECTOR\_H

#include <stdlib.h>

#include <stdbool.h>

typedef struct{

int \*buf;

size\_t size;

size\_t capacity;

} vector;

bool v\_init(vector \*v);

void v\_destroy(vector \*v);

int v\_get(vector \*v, size\_t i);

void v\_set(vector \*v, size\_t i, int val);

size\_t v\_get\_size(vector \*v);

size\_t v\_get\_cap(vector \*v);

bool empty(vector \*v);

bool v\_set\_size(vector \*v, size\_t size);

int v\_pop\_back(vector \*v);

bool v\_push\_back(vector \*v, int val);

void v\_print(vector \*v);

#endif

Примеры работы

**Тест № 1**

lunidep@lunidep-VirtualBox:~/Desktop/OS\_lab1$ gcc child.c string.c -o child

lunidep@lunidep-VirtualBox:~/Desktop/OS\_lab1$ gcc main.c vector.c string.c

lunidep@lunidep-VirtualBox:~/Desktop/OS\_lab1$ ./a.out

Введите имя файла, в который будет записан результат работы программы.

Если файла с таким именем нет, он будет создан.

t.txt

Введите числа в виде «число число число<endline>».

Результат работы программы - набор чисел - часных от деления первого введеного числа на последующие.

100 2 5

[10235] It's parent. Child id: 10236

Your file\_name: t.txt

Your numbers: 100 2 5

[10236] It's child

The file you want to open - t.txt

100 / 2 = 50

100 / 5 = 20

[10235] It's parent. Child id: 10236

Your processed numbers: 100 50 20

**Тест №2**

lunidep@lunidep-VirtualBox:~/Desktop/OS\_lab1$ gcc child.c string.c -o child

lunidep@lunidep-VirtualBox:~/Desktop/OS\_lab1$ gcc main.c vector.c string.c

lunidep@lunidep-VirtualBox:~/Desktop/OS\_lab1$ ./a.out

Введите имя файла, в который будет записан результат работы программы.

Если файла с таким именем нет, он будет создан.

t.txt

Введите числа в виде «число число число<endline>».

Результат работы программы - набор чисел - часных от деления первого введеного числа на последующие.

100 0 5

[10443] It's parent. Child id: 10449

Your file\_name: t.txt

Your numbers: 100 0 5

[10449] It's child

The file you want to open - t.txt

You can't divide by zero

Вывод

Данная лабораторная работа была посвящена межпроцессовому взаимодействию - обмену данными между потоками одного или разных процессов. Это взаимодействие реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. В ходе данной лабораторной работы я познакомился с такими понятиями, как «процесс» и методы работы с процессами, а также способ обмена данными между процессами, называемый «пайпами» (pipe). Разбиение программы на несколько процессов существенно ускоряет ее работу, потому что процессы могут выполняться одновременно. В настоящее время подавляющие большинство программ и приложений написаны на безе идей разделения на процессы и мне, как будущему программисту, крайне полезно было с этим познакомиться.