Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**Освоение принципов работы с файловыми системами. Обеспечение обмена данных между процессами посредством технологии «File mapping».**

Студент: Попов Илья Павлович

Группа: М80-206Б-20

Вариант: 3

Преподаватель: Соколов Андрей Алексеевич

Дата: 27.11.2021

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 через отображаемые файлы (memory-mapped files).

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

Варианты задания: см. лабораторная работа №2.

Варианты выбираются такие же, как и в лабораторной работе №2.

Листинг программы

**child.c**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <fcntl.h>

#include <string.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 argc, char\* argv[]){

int N;

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

//fprintf(stderr, "child: %s\n\n", argv[0]);

if(argc != 2){

perror("Execl arguments error!\n");

return -1;

}

int desc = open(argv[1], O\_RDWR);

if(desc < 0){

perror("Tmp file create error!\n");

return -2;

}

int\* fd = mmap(0, N\*sizeof(int),

PROT\_WRITE,

MAP\_SHARED, desc, 0);

if (fd == MAP\_FAILED){

perror("Mmap error!\n");

return -3;

}

/\*

FILE \*F;

F = fopen(argv[0], "w");

if (F == NULL){

perror("File can't be opened!\n");

}

\*/

int file = open(argv[0], O\_WRONLY);

if(file == -1){

perror("file error\n");

return -1;

}

int delimoe, delitel;

delimoe = fd[0];

writing\_int(delimoe, file);

//fprintf(F, "%d ", delimoe);

for (int i = 1; i < N; i++){

delitel = fd[i];

if (delitel == 0){

//exit(1);

return 1;

}

else{

int res = delimoe / delitel;

//fprintf(stderr, "child: %d / %d = %d\n", delimoe, delitel, res);

writing\_int(res, file);

//fprintf(F, "%d ", res);

fd[i] = res;

}

}//fprintf(stderr, "\n\n");

close(file);

//fclose(F);

if(msync(fd, N\*sizeof(int), MS\_SYNC) < 0){

perror("Msync error!");

return -4;

}

if(munmap(fd, N\*sizeof(int)) < 0){

perror("Munmap error!");

return -5;

}

//exit(0);

return 0;

}

**main.c**

//gcc lol\_.c string.c -o child

//gcc lol\_m.c vector.c string.c

//./a.out

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <fcntl.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(){

bool first = true;

vector v;

string file\_name;

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

if (first){

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 -1;

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

break;

}

tmp = 0;

stat = reading\_int(&tmp);

}

if (stat == R\_ERROR){

perror("Wrong value!\n");

return -2;

}

first = false;

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

}else{

s\_destroy(&file\_name);

v\_destroy(&v);

}

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

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

//создание уникального временного файла

char template[] = "/tmp/tmpXXXXXX";

int desc = mkstemp(template);

if(desc < 0){

perror("Tmp file create error!\n");

return -3;

}

if(ftruncate(desc, N\*sizeof(int)) < 0){

perror("Tmp file filling error!\n");

return -4;

}

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

//создание пайпа, в котором дочернему процессу передается кол-во чисел N, введенных пользователем, чтобы он знал, временный файл какого размера ему необходим

int fd1[2];

if(pipe(fd1) < 0){

perror("Pipe error\n");

return -5;

}

int pid = fork();

if(pid == 0){ //child

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

//fflush(stdout);

close(fd1[1]);

//перенаправить pipe1 на консольный ввод дочернему процессу

if (dup2(fd1[0], STDIN\_FILENO) == -1){

perror("Dupe error!\n");

return -6;

}

//заменяет текущий процесс, процессом, описанном в исп. файле

if(execl("child", s\_get\_all(&file\_name), template, NULL) == -1){

perror("Execl error!");

return -7;

}

} else{ //parent

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

//fflush(stdout);

//запись числа N в pipe1

close(fd1[0]);

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

close(fd1[1]);

/\*

fprintf(stdout, "Your file\_name: %s\n", s\_get\_all(&file\_name));

fprintf(stdout, "Your numbers: ");

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

fprintf(stdout, "%d ", v\_get(&v, i));

}

fprintf(stdout, "\n\n");

\*/

int desc = open(template, O\_RDWR);

if(desc < 0){

perror("Tmp file create error!\n");

return -8;

}

int\* fd = mmap(NULL, N\*sizeof(int),

PROT\_READ | PROT\_WRITE,

MAP\_SHARED, desc, 0);

if (fd == MAP\_FAILED){

perror("Mmap error!\n");

return -9;

}

//запись во временный файл

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

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

fd[i] = x;

}

if(msync(fd, N\*sizeof(int), MS\_SYNC) < 0){

perror("Msync error!");

return -10;

}

//ждем завершение работы дочернего

printf("3\n");

int status;

wait(&status);

int exit\_status = WEXITSTATUS(status);

fprintf(stdout,"Exit status of the child was %d\n\n", exit\_status);

if (exit\_status == 1){

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

}

printf("3\n");

/\*

int status;

waitpid(pid, &status, 0);

if (WIFEXITED(status)){

int exit\_status = WEXITSTATUS(status);

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

if (exit\_status == 1){

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

//fprintf(stdout, "You can't divide by zero!\n\n");

return -11;

}

}

\*/

unlink(template);

/\*

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

fflush(stdout);

fprintf(stdout, "Your processed numbers: \n");

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

fprintf(stdout, "%d ", fd[i]);

}fprintf(stdout, "\n\n");

\*/

if(munmap(fd, N\*sizeof(int)) < 0){

perror("Munmap error!");

return -12;

}

close(desc);

}

return 0;

}

**makefile**

CC=gcc

LD=gcc

CCFLAGS=-pedantic -Wall -g

LDFLAGS=

SRC=\

main.c\

string.c\

vector.c

OBJ=$(SRC:.c=.o)

OUT=prog.out

CHILD=child

SRC\_CHILD=child.c

OBJ\_CHILD=$(SRC\_CHILD:.c=.o)

.SUFFIXES: .c .o

start: $(OUT) $(CHILD)

./$(OUT)

$(OUT): $(OBJ)

${LD} ${LDFLAGS} -o $(OUT) $(OBJ) -lm

$(CHILD): $(OBJ\_CHILD)

${LD} ${LDFLAGS} -o $(CHILD) $(OBJ\_CHILD) -lm

play: $(OUT) $(CHILD)

valgrind ./$(OUT)

main.o: $(SRC)

child.o: child.c

.c.o:

${CC} ${CCFLAGS} -c $< -o $@

clear:

rm -f $(OBJ) $(OUT)

**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

В ходе выполнения данной лабораторной работы я познакомился с механизмом межпроцессорного взаимодействия при помощи отображаемых файлов (технология «File Mapping»). Этот механизм позволяет отобразить информацию на оперативную память, чтобы несколько процессов могли иметь доступ к ней.

Сравнивая этот метод межпроцессного взаимодействия с рассмотренным во 2 лабораторной работе методом каналов, у первого можно выделить следующие особенности:

1. Выигрыш в скорости, за счет отсутствия небыстрых запросов на чтение и запись. Произвольный доступ к данным файла, отображенного на ОЗУ происходит за О(1).
2. В качестве недостатка можно выделить тот факт, что дочерние процессы обязательно должны знать имя отображаемого файла и выполнить их отображение перед началом работы. Также этот метод менее эффективен по памяти, нежели каналы.