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 O(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.