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

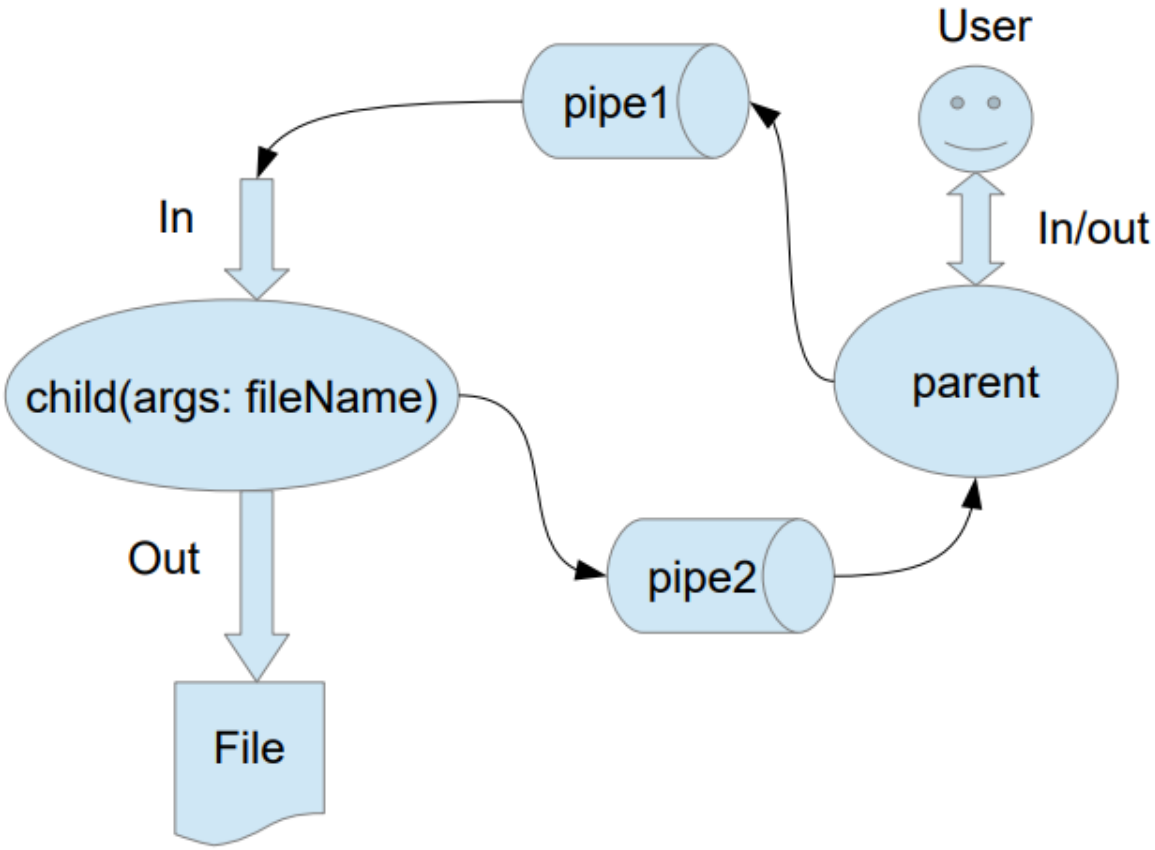
Москва, 2021

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.