Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу

«Операционные системы»

Тема работы

“Освоение принципов работы с файловыми системами. Обеспечение обмена между процессами посредством технологии File mapping”

Студент: Андреев Александр Олегович
Группа: М8О-206Б-20
Вариант: 2
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

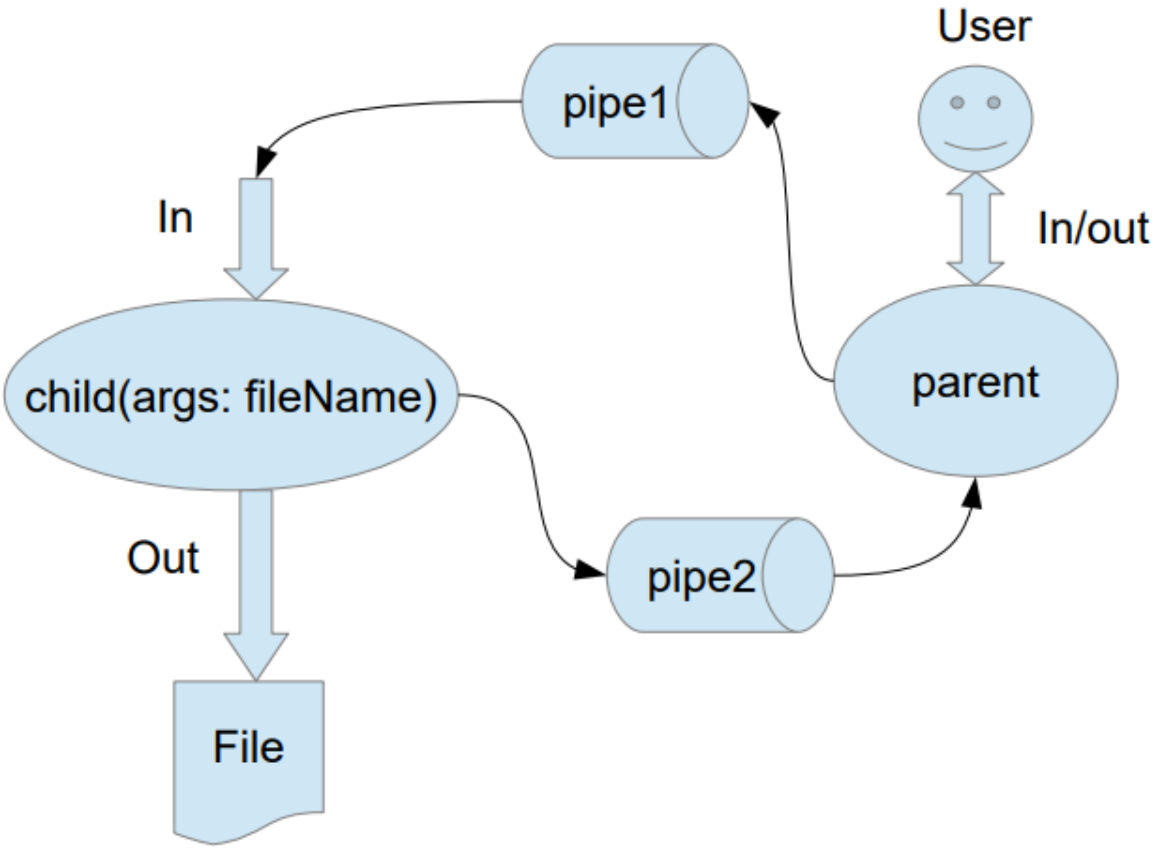
Содержание

Постановка задачи

Задача: 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.



Вариант 2: Пользователь вводит команды вида: «число число число<endline>». Далее эти числа
передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и
выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

Исходный код

parent.c

```c
#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 "myString.h"
#include "myVector.h"
/*
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.
2 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа
передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и
выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.
*/

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;
}
```

```c
r_status read_float(int fd, float* cur){
    bool dot_fnd = false;
    char c;
    *cur = 0;
    double i = 0.1;
    int res = read(fd, &c, sizeof(char));
    while(res > 0){
        if (c == '-') {
            i *= -1;
            res = read(fd, &c, sizeof(char));
            continue;
        }
        if(c == '\n') return R_EOL;
        if(c == ' ')
            break;
        if(((c < '0') || (c > '9')) && c != '.'){
            return R_ERROR;
        }
        if (!dot_fnd) {
            if(c == '.')
                dot_fnd = true;
            else {
                *cur = *cur * 10 + c - '0';
            }
        } else {
            if(c == '.')
                return R_ERROR;

            *cur = *cur + i * (c - '0');
            i /= 10;
        }
        res = read(fd, &c, sizeof(char));
    }
    if(res == 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);
        }
}

int main(){
    int fd1[2];
    int fd2[2];
    bool firstEnter = true;
    vector_dbl vec;
    string fileName;

    if (firstEnter){
        float tmp = 0;
        s_create(&fileName);
        reading_filename(&fileName);
        v_create(&vec);
        r_status st = read_float(STDIN_FILENO, &tmp);
        while(st != R_ERROR){
            v_push(&vec, tmp);
            if(st == R_EOF){
                perror("\nUSAGE: «число число число<endline>»\n");
                return -8;
            }
            else if(st == R_EOL){
                break;
            }
            tmp = 0;
            st = read_float(STDIN_FILENO, &tmp);
        }
        if (st == R_ERROR){
            perror("Wrong value\n");
            return -9;
        }
        firstEnter = false;
    }
    else{//на втором заходе в родительский процесс удаляем созданную строку и вектор чисел
        s_destroy(&fileName);
        v_destroy(&vec);
    }
```

```c
int N = v_size(&vec);
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;
}

if (pipe(fd1) < 0) {
    printf("Something is wrong with 1 pipe\n");
    return -1;
}
if (pipe(fd2) < 0) {
    printf("Something is wrong with 2 pipe\n");
    return -2;
}
int id = fork();
if (id < 0) {
    printf("Something is wrong with fork\n");
    return -3;
}
if (id == 0) {//Child process
    printf("[%d] It's child\n\n", getpid());
    fflush(stdout);

    close(fd1[1]);
    close(fd2[0]);
    if (dup2(fd1[0], STDIN_FILENO) == -1){
        perror("Cannot dup reading channel of pipe1 to stdin\n");
        return -5;
    }
    if (dup2(fd2[1], STDOUT_FILENO)== -1){
        perror("Cannot dup recording channel of pipe2 to stdout\n");
        return -6;
    }
    //заменяет текущий процесс, процессом, описанном в исп. файле
    if(execl("child", s_get(&fileName), template,  NULL) == -1){
        perror("Execl error!");
        return -7;
    }
}
```

```
    else {

        close(fd1[0]);
        write(fd1[1], &N, sizeof(int));
        close(fd1[1]);
        close(fd2[1]);

        int desc = open(template, O_RDWR);
        if(desc < 0){
            perror("Tmp file create error!\n");
            return -8;
        }

        float* fd = mmap(NULL, N*sizeof(float),
                    PROT_READ | PROT_WRITE,
                    MAP_SHARED, desc, 0);
        if (fd == MAP_FAILED){
            perror("Mmap error!\n");
            return -9;
        }
        for (int i = 0; i < N; i++){
            float x = v_getInd(&vec, i);
            fd[i] = x;
        }
        if(msync(fd, N*sizeof(float), MS_SYNC) < 0){
            perror("Msync error!");
            return -10;
        }

        int status;
        wait(&status);
        int exit_status = WEXITSTATUS(status);
        fprintf(stdout,"Exit status of the child was %d\n\n", exit_status);
        printf("\n[%d] It's parent. Child id: %d\n\n", getpid(), id);
        fflush(stdout);

        unlink(template);

        if(munmap(fd, N*sizeof(float)) < 0){
            perror("Munmap error!");
            return -12;
        }
        close(desc);
        close(fd1[1]);
        close(fd2[0]);
    }
    return 0;
}
```

child.c

```c
#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 "myVector.h"
#include "myString.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 = " ";
    if (a == -2) 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_float(float cur, int fd){
    int left = cur;
    float right = cur - left;
    int new_right = right * pow_ten(7);
    int l = length_int(left) - 1;
    if (left == 0){
        write_num(0, fd);
    }
    else{
        while (left > 0){
            int tmp = left;
            int c = tmp / pow_ten(l);

            write_num(c, fd);

            left = left % pow_ten(l);
            l--;
        }
    }
    write_num(-2, fd);
    l = length_int(new_right) - 1;
    if (new_right == 0){
        for (int i = 0; i < 7; i++){
            write_num(0, fd);
        }
    }
    else{
        while (new_right > 0){
            int tmp = new_right;
            int c = tmp / pow_ten(l);
            write_num(c, fd);
            new_right = new_right % pow_ten(l);
            l--;
       }
    }
    write_num(-1, fd);
}
```

```c
int main(int argc, char* argv[]){
    int N;
    read(STDIN_FILENO, &N, sizeof(int));
    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;
    }

    float* fd = mmap(0, N*sizeof(float),
                     PROT_WRITE,
                     MAP_SHARED, desc, 0);
    if (fd == MAP_FAILED){
        perror("Mmap error!\n");
        return -3;
    }


    int file = open(argv[0], O_WRONLY);
    if(file == -1){
        perror("file error\n");
        return -1;
    }

    float x;
    float sum = 0;
    int len;
    for (int i = 0; i < N; i++){
        printf("%f ", fd[i]);
    }
    printf("\n");
    for (int i = 0; i < N; i++){
        float x = fd[i];
        sum += x;
        printf("%f", sum);
        write(STDOUT_FILENO, &sum, sizeof(float));
    }
```

```
        writing_float(sum, file);
        close(file);
        if(msync(fd, N*sizeof(float), MS_SYNC) < 0){
            perror("Msync error!");
            return -4;
        }
        if(munmap(fd, N*sizeof(float)) < 0){
            perror("Munmap error!");
            return -5;
        }
        return 0;
    }
```

myString.h

```
#ifndef MYSTRING_H
#define MYSTRING_H

#include <stdbool.h>
#include <stdlib.h>

typedef struct {
    int size;
    int cap;
    char* buf;
} string;

void s_create(string *s);
void s_destroy(string *s);
bool s_isEmpty(string *s);
int s_cap(string *s);
int s_size(string *s);
bool s_growBuf(string *s);
bool s_push(string *s, char c);
bool s_shrinkBuf(string *s);
char s_pop(string *s);
char s_getInd(string *s, int id);
char* s_get(string *s);

#endif
```

myString.c

```c
#include <stdio.h>
#include "myString.h"

void s_create(string *s){
    s->buf = NULL;
    s->cap = 0;
    s->size = 0;
}

void s_destroy(string *s){
    s->size = 0;
    s->cap = 0;
    free(s->buf);
}

bool s_isEmpty(string *s){
    return s->size == 0;
}

int s_cap(string *s){
    return s->cap;
}

int s_size(string *s){
    return s->size;
}

bool s_growBuf(string *s){
    int tmp = s->cap * 3/2;
    if (!tmp){
        tmp = 10;
    }
    char *newBuf = realloc(s->buf, sizeof(char) * tmp);
    if (newBuf != NULL){
        s->buf = newBuf;
        s->cap = tmp;
        return true;
    }
    return false;
}
```

```c
bool s_push(string *s, char c){
    if (s_size(s) == s_cap(s)){
        if (!s_growBuf(s)){
            return false;
        }
    }
    s->buf[s_size(s)] = c;
    s->size++;
    return true;
}

bool s_shrinkBuf(string *s){
    int tmp = s->cap * 4/9;
    if (tmp < s_size(s)){
        return true;
    }
    char *newBuf = realloc(s->buf, sizeof(char) * tmp);
    if (newBuf != NULL){
        s->buf = newBuf;
        s->cap = tmp;
        return true;
    }
    return false;
}

char s_pop(string *s){
    char tmp = s->buf[s_size(s) - 1];
    s_shrinkBuf(s);
    s->size--;
    return tmp;
}

char s_getInd(string *s, int id){
    return s->buf[id];
}

char* s_get(string *s){
    return s->buf;
}
```

myVector.h

```c
#ifndef MYVECTOR_H
#define MYVECTOR_H

#include <stdbool.h>
#include <stdlib.h>

typedef struct {
    int size;
    int cap;
    double* buf;
} vector_dbl;

void v_create(vector_dbl *v);
void v_destroy(vector_dbl *v);
bool v_isEmpty(vector_dbl *v);
int v_cap(vector_dbl *v);
int v_size(vector_dbl *v);
bool v_growBuf(vector_dbl *v);
bool v_push(vector_dbl *v, double val);
bool v_shrinkBuf(vector_dbl *v);
double v_pop(vector_dbl *v);
double v_getInd(vector_dbl *v    , int id);
double* v_get(vector_dbl *s);

#endif
```

myVector.c

```c
#include <stdio.h>
#include "myVector.h"
void v_create(vector_dbl *v){
    v->buf = NULL;
    v->size = 0;
    v->cap = 0;
}

void v_destroy(vector_dbl *v){
    v->size = 0;
    v->cap = 0;
    free(v->buf);
}

bool v_isEmpty(vector_dbl *v){
    return v->size == 0;
}

int v_cap(vector_dbl *v){
    return v->cap;
}

int v_size(vector_dbl *v){
    return v->size;
}

bool v_growBuf(vector_dbl *v){
    int tmp = v->cap * 3/2;
    if (!tmp){
        tmp = 10;
    }
    double *newBuf = realloc(v->buf, sizeof(double) * tmp);
    if (newBuf != NULL){
        v->buf = newBuf;
        v->cap = tmp;
        return true;
    }
    return false;
}
```

```
bool v_push(vector_dbl *v, double val){
    if (v_cap(v) == v_size(v)){
        if (!v_growBuf(v)){
            return false;
        }
    }
    v->buf[v_size(v)] = val;
    v->size++;
    return true;
}

bool v_shrinkBuf(vector_dbl *v){
    int tmp = v->cap * 4/9;
    if (tmp < v_size(v)){
        return true;
    }
    double *newBuf = realloc(v->buf, sizeof(double) * tmp);
    if (newBuf != NULL){
        v->buf = newBuf;
        v->cap = tmp;
        return true;
    }
    return false;
}

double v_pop(vector_dbl *v){
    double tmp = v->buf[v_size(v) - 1];
    v_shrinkBuf(v);
    v->size--;
    return tmp;
}

double v_getInd(vector_dbl *v, int id){
    return v->buf[id];
}

double* v_get(vector_dbl *v){
    return v->buf;
}
```

Демонстрация работы программы

Тест1. Неправильные значения

missclick3@missclick3:~/Desktop/OSLabs/os_lab_2$ ./a.out

q

12

[43382] It's child

file error
: No such file or directory
Exit status of the child was 255

[43381] It's parent. Child id: 43382

Тест 2. Не введено чисел.
missclick3@missclick3:~/Desktop/OSLabs/os_lab_2$ ./a.out
test

[43392] It's child

Exit status of the child was 0

[43390] It's parent. Child id: 43392

missclick3@missclick3:~/Desktop/OSLabs/os_lab_4$ cat test
0.0000000

Тест 3. Корректные значения.
missclick3@missclick3:~/Desktop/OSLabs/os_lab_2$ ./a.out
test
1 2 3 1.23
[43437] It's child

Exit status of the child was 0

[43433] It's parent. Child id: 43437

missclick3@missclick3:~/Desktop/OSLabs/os_lab_4$ cat test
7.2300000

Выводы

В данной лабораторной работе я познакомился с механизмом межпроцессорного взаимодействия при помощи отображаемых файлов. Это позволяет отобразить информацию на оперативной памяти. Тем самым несколько процессов могут иметь к ней доступ.