

**Московский авиационный институт (национальный  
исследовательский университет)**

Факультет информационных технологий и прикладной  
математики Кафедра вычислительной математики и  
программирования

Лабораторная работа по предмету "операционные  
системы" №2

Студент: Мокеева С.А.

Преподаватель: Соколов А.А.

Группа: М8О-206Б-20

Дата: 12.04.2022

Оценка:

Подпись:

**Москва 2022г.**

## Вариант 2.

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.

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.

Определения:

Канал

1. Секция общей памяти, которую процессы могут использовать для коммуникации  
(Способ передачи данных между процессами)
2. Процесс, кто создал `pipe` является сервером
3. Тот, кто использует называется клиентом

## Реализация

### Файл `main.cpp`

```
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

char* read_filename() { //функция считывания имени файла
    char* line = NULL;
    size_t bufsize = 0;
    int nread = getline(&line, &bufsize, stdin); // getline(100, 200, stdin)
    if (nread == -1) {
        return NULL;
    }
}
```

```

    size_t len = strlen(line); //количество символов строки(т е имя файла) до
'\0'
    line[len - 1] = '\0';
    return line; // 456
}

char write_number(int fd, float number) { //
    int n_written_bytes = write(fd, &number, sizeof(number));
    if (n_written_bytes <= 0) {
        printf("write error\n");
        return 0;
    }
    assert(n_written_bytes == sizeof(number));
    return 1;
}

int main(int argc, char *argv[]) { //int main(сколько аргументов, имя файла),
имя файла - тоже аргумент
    if (argc != 1) { // arg count, если количество аргументов не равно 1,
выводим ошибку
        printf("Usage: %s\n", argv[0]);
        return 1;
    }

    char* filename_result = read_filename(); //считали имя файла
    if (filename_result == NULL) { //если его нет, выводим ошибку и сообщение
об ожидании имени файла
        printf("expected filename\n");
        return 1;
    }

    //открываем два конца трубы
    int pipe_fds[2];
    if (pipe(pipe_fds) == -1) { //проверяем трубу на ошибки(EMFILE, EFAULT),
если есть, выводим
        printf("pipe error\n");
        return 1;
    }
    // pipe_fds = {10, 20}
    // write(20, "abc") -> (kernel buffer) -> read(10) = "abc"

    //создаём дочерний процесс
    int pid_child = fork();
    if (pid_child == -1) { //если не вышло создать, выходит с ошибкой
        printf("fork error\n");
        return 1;
    }
    // pid_child=0 -- у дочернего процесса
    // pid_child=номер_дочернего -- у родительского процесса

    if (pid_child == 0) {
        // Мы в дочернем процессе.
        if (close(pipe_fds[1]) != 0) {

```

```

    printf("close error\n");
    return 1;
}
// Закрыли тот конец трубы, что на запись
// Теперь из трубы можем только читать
// Говорим, что дальше при обращении к stdin нужно в действительности
обращаться к тому концу трубы, что на чтение
if (dup2(pipe_fds[0], 0) == -1) {
    printf("dup2 error\n");
    return 1;
}

//вызываем дочерний процесс
char* argv_child[3] = {"/child", filename_result, (char *)NULL};
//предполагаем, что дочерний процесс находится в текущем каталоге
//как аргумент мы передаём имя файла, тк дочерняя программа должна быть
отдельной программой
if (execv("child", argv_child) == -1) {
    printf("exec error\n");
    return 1;
}

return 0; // Если вдруг execv вернёт не -1 (такого не должно быть)
}

assert(pid_child > 0); //условие pid_child == 0 не сработало, мы в
родительском процессе

if (close(pipe_fds[0]) != 0) {
    printf("close error\n");
    return 1;
}
// Закрыли тот конец трубы, что на чтение.

while (1) { //читаем со входа числа
    // 10 20\n
    //   ^
    //   sep=' '
    //   ^
    //   sep='\n'
    //

    // Считываем очередное число из входного файла и преобразуем его из
    // текста во float.
    float number;
    int result_scanf = scanf("%f", &number); //считали число
    if (result_scanf == EOF) { // -1, если встретили конец файла, вышли
        break;
    }
    if (result_scanf == 0) { // 0, если встретили ерунду
        printf("expected a number\n");
        return 1;
    }
}

```

```

    assert(result_scanf == 1); //благополучно считали число

    if (! write_number(pipe_fds[1], number)) { //отправляем число нашему
дочернему процессу
        return 1;
    }

    //мы пытаемся понять, кончились цифры в строке
    char sep = getchar();
    if (sep == '\n') { //если разделитель перенос строки,
        // записываем в дочерний процесс, что данные кончились через
бесконечность
        if (! write_number(pipe_fds[1], 1.0 / 0.0)) { //если не удалось
напечатать
            return 1;
        }
    }
}
//вышли из цикла

if (close(pipe_fds[1]) != 0) {
    printf("close error\n");
    return 1;
}
// Закрыли тот конец трубы, что на запись
// Даём понять дочернему процессу, что данных больше не будет

//надо дождаться конца работы дочернего процесса
int wstatus;
if (wait(&wstatus) == -1) { //вернуть -1 в случае ошибки
    printf("wait error\n");
    return 1;
}
// Ребёнок завершился.

if (! (WIFEXITED(wstatus) && WEXITSTATUS(wstatus) == 0)) { //Мы не можем
вернуть 0, если с ребёнком что-то не так
    printf("error in child\n");
    return 1;
}

return 0;
}
/*
user -> main (stdin): result.txt
main: creates child, passes "result.txt"

user -> main (stdin): 1 2 3
main -> child (pipe1): 1 2 3
child -> result.txt: 1+2+3=6

user -> main (stdin): 1 2
main -> child (pipe1): 1 2

```

```
child -> result.txt: 1+2=3
*/
```

## Файл child.cpp

```
#include <assert.h>
#include <float.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char* argv[]){
    printf("child: started\n");

    FILE *file = fopen(argv[1], "w"); //открываем файл
    if (file == NULL){
        printf("fopen error\n");
        return 1;
    }

    float sum = 0;
    while (1) {
        // File descriptors:
        // 0 -- stdin
        // 1 -- stdout
        // 2 -- stderr
        float number;
        int n_read = read(0, &number, sizeof(number)); //читаем со
        стандартного потока ввода (нам переброшен один конец трубы)
        if (n_read == -1) {
            printf("read error\n");
            return 1;
        }
        if (n_read == 0) { // конец файла
            // Другой конец трубы закрыт в родительском процессе
            break;
        }
        // n_read > 0
        assert(n_read == sizeof(number));
        printf("number=%f\n", number);

        if (number <= FLT_MAX) { //если число не бесконечность, добавляем к
        сумме число
            sum += number;
            continue;
        }

        // если бесконечность
        printf("sum=%f\n", sum); //выводим сумму
        if (fprintf(file, "%f\n", sum) < 0) {
            printf("fprintf error\n");
            return 1;
        }
        sum = 0;
    }
}
```

```
    if (fclose(file) != 0) { //если fclose работает неверно(ех, место на
жестком диске кончилось)
        printf("fclose error\n"); // ошибка
        return 1;
    }

    return 0;
}
```

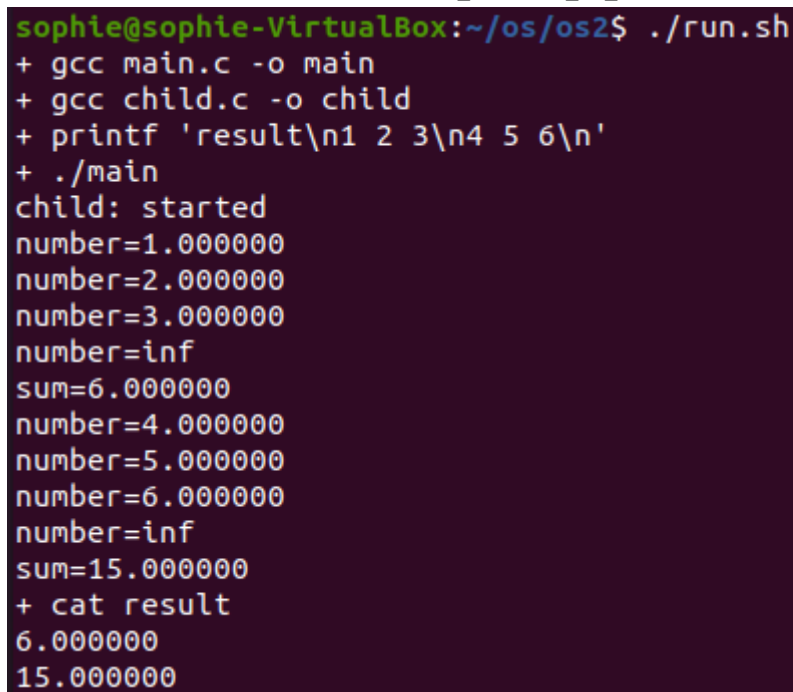
## Файл run.sh

```
#!/bin/bash

set -e # exit on error
set -x # trace (print commands)

gcc main.c -o main
gcc child.c -o child
printf 'result\n1 2 3\n4 5 6\n' | ./main
cat result
```

## Пример работы



```
sophie@sophie-VirtualBox:~/os/os2$ ./run.sh
+ gcc main.c -o main
+ gcc child.c -o child
+ printf 'result\n1 2 3\n4 5 6\n'
+ ./main
child: started
number=1.000000
number=2.000000
number=3.000000
number=inf
sum=6.000000
number=4.000000
number=5.000000
number=6.000000
number=inf
sum=15.000000
+ cat result
6.000000
15.000000
```

## Вывод

В ходе лабораторной я работала с процессами в ОС UNIX, научилась передавать данные между процессами с помощью pipe, а также перенаправлять поток ввода/вывода.