Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-214БВ-24

Студент: Александров М.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 02.10.25

Москва, 2025

Постановка задачи

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.

Общий метод и алгоритм решения

Использованные системные вызовы:

- *pid_t fork(void)*; – создает дочерний процесс.
- *int pipe(int *fd)*; – создает однонаправленный канал для межпроцессорного взаимодействия;
- *int dup2(int oldfd, int newfd)*; – создает копию файлового дескриптора *oldfd* в указанном дескрипторе *newfd*.
- *int execv(const char *path, char *const argv[])*; — заменяет текущий образ процесса на новый исполняемый файл.
- *int open(const char* pathname, int flags, mode_t mode)*; – открывает файл по указанному пути с заданными флагами и правами доступа.
- *ssize_t read(int fd, void* buf, size_t count)*; – читает данные из файлового дескриптора в буфер.
- *ssize_t write(int fd, const void* buf, size_t count)*; – записывает данные из буфера в файловый дескриптор.
- *int close(int fd)*; – закрывает файловый дескриптор.
- *pid_t wait(int* status)*; – ожидает изменения состояния указанного дочернего процесса.
- *pid_t getpid(void)*; — возвращает PID текущего процесса. Используется для отладочного вывода.

В рамках лабораторной работы была реализована программа, состоящая из двух исполняемых файлов: родительского (*parent*) и дочернего (*child*). Родительский процесс создаёт дочерний с помощью системного вызова *fork*(), после чего заменяет его образ на отдельную программу с помощью *execv*(). Для обмена данными между процессами используется анонимный канал, созданный системным вызовом *pipe*().

Родительский процесс запрашивает у пользователя имя выходного файла, а затем читает из стандартного ввода строку, содержащую произвольное количество чисел типа *float*. Эта строка передаётся дочернему процессу через канал. С помощью системного вызова *dup2*() стандартный ввод дочернего процесса перенаправляется на чтение из канала, что позволяет ему получать данные, как будто они поступают из терминала.

Дочерний процесс (*child*) получает имя выходного файла как аргумент командной строки, читает строку с числами из своего стандартного ввода, парсит их, вычисляет сумму и записывает результат в указанный файл с помощью системных вызовов *open*() и *write*().

Программа демонстрирует кооперацию процессов через каналы (*pipe*) в соответствии с Unix-

философией: данные передаются байтами, процессы связаны иерархией, а стандартные потоки перенаправляются для организации межпроцессного взаимодействия.

Код программы

parent.c

```
#include <stdint.h>
#include <stdbool.h>

#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>

static char SERVER_PROGRAM_NAME[] = "child";



int main(int argc, char *argv[]) {
    // проверка аргументов
    if (argc == 1) {
        char msg[1024];
        uint32_t len = snprintf(msg, sizeof(msg) - 1, "usage: %s filename\n", argv[0]);
        write(STDERR_FILENO, msg, len);
        exit(EXIT_SUCCESS);
    }

    // путь до директории
    char progpath[1024];
        {
            ssize_t len = readlink("/proc/self/exe", progpath,
                                   sizeof(progpath) - 1);
            if (len == -1) {
                const char msg[] = "error: failed to read full program path\n";
```

```
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        while (progpath[len] != '/')
            --len;

        progpath[len] = '\0';
    }

// Parent => Child
int client_to_server[2];
    if (pipe(client_to_server) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

// НЕОБЯЗАТЕЛЬНО
// // Child => Parent
// int server_to_client[2];
    // if (pipe(server_to_client) == -1) {
    //     const char msg[] = "error: failed to create pipe\n";
    //     write(STDERR_FILENO, msg, sizeof(msg));
    //     exit(EXIT_FAILURE);
    // }

// дочерний процесс
const pid_t child = fork();

switch (child) {
    case -1: {
```

```
            const char msg[] = "error: failed to spawn new process\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        } break;

    case 0: { // дочерний
        {
            pid_t pid = getpid(); // получение PID дочернего

                    char msg[64];
                    const int32_t length = snprintf(msg, sizeof(msg),
                            "%d: I'm a child\n", pid);
                    write(STDOUT_FILENO, msg, length);
        }

        close(client_to_server[1]);

        dup2(client_to_server[0], STDIN_FILENO);
            close(client_to_server[0]);

        {
                    char path[2048];
                    snprintf(path, sizeof(path) - 1, "%s/%s", progpath, SERVER_PROGRAM_NAME);

                    char *const args[] = {SERVER_PROGRAM_NAME, argv[1], NULL};

                    int32_t status = execv(path, args);

                    if (status == -1) {
                        const char msg[] = "error: failed to exec into new exectuable image\n";
                        write(STDERR_FILENO, msg, sizeof(msg));
                        exit(EXIT_FAILURE);
```

```
            }
    }
} break;

default: { // PARENT
    {
        pid_t pid = getpid();

        char msg[64];
            const int32_t length = snprintf(msg, sizeof(msg),
                "%d: I'm a parent, my child has PID %d\n", pid, child);
            write(STDOUT_FILENO, msg, length);
    }

    close(client_to_server[0]);

    char buf[4096];
    ssize_t bytes;
    while (bytes = read(STDIN_FILENO, buf, sizeof(buf) - 1)) {
        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        else if (bytes == 1 && buf[0] == '\n') {
            break;
        }
        buf[bytes] = '\0';
        write(client_to_server[1], buf, bytes);
    }
    close(client_to_server[1]);
```

```
        wait(NULL);
    } break;
    }
}
```

child.c

```
#include <stdint.h>
#include <stdbool.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        char msg[1024];
        uint32_t len = snprintf(msg, sizeof(msg) - 1, "error: need filename and string of floats\n");
        write(STDERR_FILENO, msg, len);
        exit(EXIT_FAILURE);
    }

    pid_t pid = getpid();

    int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0600);
    if (file == -1) {
            const char msg[] = "error: failed to open requested file\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE)
     }

    char buf[4096];
```

```
ssize_t bytes;
while (bytes = read(STDIN_FILENO, buf, sizeof(buf) - 1)) {
    if (bytes < 0) {
        const char msg[] = "error: failed to read from stdin\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }
    buf[bytes] = '\0';

    if (bytes > 0 && buf[bytes - 1] == '\n') {
        buf[bytes - 1] = '\0';
    }

    // парсинг
    float sum = 0;
    char *token = strtok(buf, " \t");
    while (token) {
        char *endptr;
        float f = strtof(token, &endptr);
        if (endptr != token && *endptr == '\0') { sum += f; }
        token = strtok(NULL, " \t");
    }

    char output[100];
    uint32_t len = snprintf(output, sizeof(output) - 1, "%.3f\n", sum);
    if (len < 0 || len >= sizeof(output)) {
        const char msg[] = "error: snprintf failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }
```

```
        if (write(file, output, len) != len) {
            const char msg[] = "error: failed to write to file\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            exit(EXIT_FAILURE);
        }
    }

    close(file);
    exit(EXIT_SUCCESS);
}
```

Протокол работы программы

Тестирование:

```
maks-alex@DESKTOP-QFPFVP1:~/MAI_OS/lab1/src$ cc -o parent parent.c && cc -o child child.c
maks-alex@DESKTOP-QFPFVP1:~/MAI_OS/lab1/src$ ./parent test1.txt
4157: I'm a parent, my child has PID 4158
4158: I'm a child
1.5 2.5
-1.0 3.0 2.0
10.0

maks-alex@DESKTOP-QFPFVP1:~/MAI_OS/lab1/src$ cat test1.txt
4.000
4.000
10.000
maks-alex@DESKTOP-QFPFVP1:~/MAI_OS/lab1/src$ ./parent test2.txt
4209: I'm a parent, my child has PID 4210
4210: I'm a child

maks-alex@DESKTOP-QFPFVP1:~/MAI_OS/lab1/src$ cat test2.txt
maks-alex@DESKTOP-QFPFVP1:~/MAI_OS/lab1/src$ ./parent test3.txt
4243: I'm a parent, my child has PID 4244
4244: I'm a child
abc xyz
-1.543 1.543
maks-alex@DESKTOP-QFPFVP1:~/MAI_OS/lab1/src$ cat test3.txt
0.000
0.000
maks-alex@DESKTOP-QFPFVP1:~/MAI_OS/lab1/src$ ./parent test1.txt
4345: I'm a parent, my child has PID 4346
4346: I'm a child
124 4234 2325 654 32
1.1 2.1 3.1 4.7

maks-alex@DESKTOP-QFPFVP1:~/MAI_OS/lab1/src$ cat test1.txt
7369.000
11.000
```

Strace:

4296 execve("./parent", ["./parent", "result.txt"], 0x7ffc8cb240f0 /* 27 vars */) = 0

4296 brk(NULL) = 0x55ba5fd28000

4296 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb489d11000

```
4296  access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
4296  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
4296  fstat(3, {st_mode=S_IFREG|0644, st_size=19375, ...}) = 0
4296  mmap(NULL, 19375, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb489d0c000
4296  close(3)                          = 0
4296  openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
4296  read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0"..., 832) = 832
4296  pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) =
784
4296  fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
4296  pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) =
784
4296  mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7fb489afa000
4296  mmap(0x7fb489b22000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fb489b22000
4296  mmap(0x7fb489caa000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7fb489caa000
4296  mmap(0x7fb489cf9000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7fb489cf9000
4296  mmap(0x7fb489cff000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fb489cff000
4296  close(3)                          = 0
4296  mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb489af7000
4296  arch_prctl(ARCH_SET_FS, 0x7fb489af7740) = 0
4296  set_tid_address(0x7fb489af7a10)   = 4296
4296  set_robust_list(0x7fb489af7a20, 24) = 0
4296  rseq(0x7fb489af8060, 0x20, 0, 0x53053053) = 0
4296  mprotect(0x7fb489cf9000, 16384, PROT_READ) = 0
4296  mprotect(0x55ba5caae000, 4096, PROT_READ) = 0
4296  mprotect(0x7fb489d49000, 8192, PROT_READ) = 0
4296  prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
4296  munmap(0x7fb489d0c000, 19375)     = 0
4296  readlink("/proc/self/exe", "/home/maks-alex/MAI_OS/lab1/src/"..., 1023) = 38
```

4296 pipe2([3, 4], 0) = 0

4296 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fb489af7a10) = 4297

4297 set_robust_list(0x7fb489af7a20, 24 <unfinished ...>

4296 getpid(<unfinished ...>

4297 <... set_robust_list resumed>) = 0

4296 <... getpid resumed>) = 4296

4297 getpid(<unfinished ...>

4296 write(1, "4296: I'm a parent, my child has"..., 42 <unfinished ...>

4297 <... getpid resumed>) = 4297

4296 <... write resumed>) = 42

4297 write(1, "4297: I'm a child\n", 18 <unfinished ...>

4296 close(3) = 0

4297 <... write resumed>) = 18

4296 read(0, <unfinished ...>

4297 close(4) = 0

4297 dup2(3, 0) = 0

4297 close(3) = 0

4297 execve("/home/maks-alex/MAI_OS/lab1/src/child", ["child", "result.txt"], 0x7ffe99824ef0 /* 27 vars */) = 0

4297 brk(NULL) = 0x55b9df7f3000

4297 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f989b4ab000

4297 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

4297 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

4297 fstat(3, {st_mode=S_IFREG|0644, st_size=19375, ...}) = 0

4297 mmap(NULL, 19375, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f989b4a6000

4297 close(3) = 0

4297 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

4297 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

4297 pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

4297 fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

4297 pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

4297 mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f989b294000

4297 mmap(0x7f989b2bc000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f989b2bc000

4297 mmap(0x7f989b444000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f989b444000

4297 mmap(0x7f989b493000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f989b493000

4297 mmap(0x7f989b499000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f989b499000

4297 close(3) = 0

4297 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f989b291000

4297 arch_prctl(ARCH_SET_FS, 0x7f989b291740) = 0

4297 set_tid_address(0x7f989b291a10) = 4297

4297 set_robust_list(0x7f989b291a20, 24) = 0

4297 rseq(0x7f989b292060, 0x20, 0, 0x53053053) = 0

4297 mprotect(0x7f989b493000, 16384, PROT_READ) = 0

4297 mprotect(0x55b9cea54000, 4096, PROT_READ) = 0

4297 mprotect(0x7f989b4e3000, 8192, PROT_READ) = 0

4297 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

4297 munmap(0x7f989b4a6000, 19375) = 0

4297 getpid() = 4297

4297 openat(AT_FDCWD, "result.txt", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3

4297 read(0, <unfinished ...>

4296 <... read resumed>"1.5 2.5\n", 4095) = 8

4296 write(4, "1.5 2.5\n", 8) = 8

4297 <... read resumed>"1.5 2.5\n", 4095) = 8

4296 read(0, <unfinished ...>

4297 write(3, "4.000\n", 6) = 6

4297 read(0, <unfinished ...>

4296 <... read resumed>"-1.0 3.0 2.0\n", 4095) = 13

4296 write(4, "-1.0 3.0 2.0\n", 13) = 13

4297 <... read resumed>"-1.0 3.0 2.0\n", 4095) = 13

4296 read(0, <unfinished ...>

4297 write(3, "4.000\n", 6) = 6

4297 read(0, <unfinished ...>

4296 <... read resumed>"10.0\n", 4095) = 5

4296 write(4, "10.0\n", 5) = 5

4297 <... read resumed>"10.0\n", 4095) = 5

4296 read(0, <unfinished ...>

4297 write(3, "10.000\n", 7) = 7

4297 read(0, <unfinished ...>

4296 <... read resumed>"\n", 4095) = 1

4296 close(4) = 0

4297 <... read resumed>"", 4095) = 0

4296 wait4(-1, <unfinished ...>

4297 close(3) = 0

4297 exit_group(0) = ?

4297 +++ exited with 0 +++

4296 <... wait4 resumed>NULL, 0, NULL) = 4297

4296 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4297, si_uid=1000, si_status=0, si_utime=0, si_stime=2 /* 0.02 s */} ---

4296 exit_group(0) = ?

4296 +++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы были успешно изучены и применены основные системные вызовы для работы с процессами и межпроцессным взаимодействием в ОС Linux. Была реализована программа, демонстрирующая создание процессов, организацию каналов связи между ними и перенаправление стандартных потоков ввода-вывода. Основными сложностями стали корректная обработка системных вызовов с проверкой ошибок, безопасное управление файловыми дескрипторами (особенно закрытие ненужных концов канала) и парсинг вещественных чисел.