

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: Горбачев Ф.М..

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 03.10.25

Москва, 2025

Постановка задачи

Вариант 4.

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 со 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<newline>». 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 `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 неименованный канал для межпроцессорного взаимодействия.
- **`ssize_t write(int fd, void *buf, size_t count)`**– записывает данные из буфера в файловый дескриптор.
- **`ssize_t read(int fd, void *buf, size_t count)`** – читает данные из файлового дескриптора в буфер.
- **`int open(const char *pathname, int flags, mode_t mode)`** – открывает\создает файл.
- **`int close(int fd)`** – закрывает файл.
- **`int dup2(int oldfd, int newfd)`** – переназначение файлового дескриптора
- **`int execl(const char *path, const char *arg0, ..., NULL)`** – запуск другой программы, замещая текущий процесс.
- **`pid_t wait(int *status)`** – ожидание завершения дочернего процесса.
- **`void exit (int status)`** – завершения выполнения процесса и возвращение статуса.

Алгоритм решения:

Создается родительский процесс, запрашивающий имя файла и открывающий его на чтение. Содержимое файла отправляет дочернему процессу, который выполняет следующие задачи: обработка строк, парсинг чисел, операция деления чисел. Если встречается деление на ноль, дочерний процесс выводит сообщение об ошибке, уведомляя родителя через пайп. В остальных случаях результат деления отправляется родителю через второй пайп, который выводит его на экран. Таким образом, родитель и дочерний процесс обмениваются данными и выводят результаты построчно, обеспечивая корректное завершение при ошибках.

Код программы

parent.c

```
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    char filename[1024];

    // Запрос имени файла у пользователя
    {
        const char msg[] = "Enter output filename: ";
        write(STDOUT_FILENO, msg, sizeof(msg) - 1);

        ssize_t n = read(STDIN_FILENO, filename, sizeof(filename) - 1);
        if (n <= 0) {
            const char msg[] = "Error: cannot read filename\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            exit(EXIT_FAILURE);
        }
        filename[n - 1] = '\0'; // Удаление символа новой строки
    }

    // Создание каналов для межпроцессного взаимодействия
    int parent_to_child[2]; // pipe1 - передача команд от родителя к ребенку
    int child_to_parent[2]; // pipe2 - передача статуса от ребенка родителю

    if (pipe(parent_to_child) == -1) {
        const char msg[] = "Error: cannot create pipe1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    const char msg[] = "Error: cannot create pipe2\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

// Создание дочернего процесса
pid_t pid = fork();

switch(pid) {
    case -1: {
        // Ошибка создания процесса
        const char msg[] = "Error: cannot create child process\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }
    case 0: {
        // Дочерний процесс
        close(parent_to_child[1]);
        close(child_to_parent[0]);

        // Перенаправляем стандартный ввод на чтение из pipe1
        dup2(parent_to_child[0], STDIN_FILENO);
        close(parent_to_child[0]);

        // Перенаправляем стандартный вывод ошибок на запись в pipe2
        dup2(child_to_parent[1], STDERR_FILENO);
        close(child_to_parent[1]);

        // Запускаем программу дочернего процесса с передачей имени файла
        execl("./child", "child", filename, NULL);

        const char msg[] = "Error: cannot execute child process\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }
    default: {
        // Родительский процесс
        close(parent_to_child[0]);
        close(child_to_parent[1]);
        const char prompt[] = "Enter numbers separated by spaces (or 'exit' to quit):\n";
        write(STDOUT_FILENO, prompt, sizeof(prompt) - 1);

        char buffer[4096];
        int child_alive = 1;

        while (child_alive) {
            // Чтение команды от пользователя
            const char input_prompt[] = "> ";
            write(STDOUT_FILENO, input_prompt, sizeof(input_prompt) - 1);

            ssize_t n = read(STDIN_FILENO, buffer, sizeof(buffer) - 1);
            if (n <= 0) break;

            buffer[n - 1] = '\0'; // Удаляем символ новой строки

            // Проверка на команду выхода
            if (strcmp(buffer, "exit") == 0) {
                break;
            }

            // Проверяем пустой вход
            if (strlen(buffer) == 0) {
                continue;
            }
        }
    }
}
```

```

// добавляем символ новой строки для правильной обработки
buffer[n - 1] = '\n';
buffer[n] = '\0';

// отправка команды дочернему процессу через pipe1
write(parent_to_child[1], buffer, n);

// проверка статуса от дочернего процесса через pipe2
char status_buf[256];
fd_set readfds;
struct timeval timeout;

FD_ZERO(&readfds);
FD_SET(child_to_parent[0], &readfds);
timeout.tv_sec = 0;
timeout.tv_usec = 100000; // 100ms

int ready = select(child_to_parent[0] + 1, &readfds, NULL, NULL, &timeout);
if (ready > 0) {
    ssize_t status_n = read(child_to_parent[0], status_buf, sizeof(status_buf) - 1);
    if (status_n > 0) {
        status_buf[status_n] = '\0';
        if (strstr(status_buf, "division by zero") != NULL) {
            const char error_msg[] = "Error: division by zero detected. Terminating...\n";
            write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
            child_alive = 0;
        }
        // Вывод других сообщений от дочернего процесса
        write(STDERR_FILENO, status_buf, status_n);
    }
}
}

// Закрытие каналов и ожидание завершения дочернего процесса
close(parent_to_child[1]);
close(child_to_parent[0]);

// Ожидание завершения дочернего процесса
int status;
wait(&status);

const char exit_msg[] = "Parent process terminated.\n";
write(STDOUT_FILENO, exit_msg, sizeof(exit_msg) - 1);
}

return 0;
}

```

child.c

```

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        const char msg[] = "Error: output filename not provided\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    const char *filename = argv[1];

    // Открытие файла для записи результатов
    int output_file = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (output_file == -1) {
        const char msg[] = "Error: cannot open output file\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    char buf[4096];
    ssize_t n;
    size_t pos = 0;

    const char header[] = "Calculation Results:\n===== \n";
    write(output_file, header, sizeof(header) - 1);

    while ((n = read(STDIN_FILENO, buf + pos, sizeof(buf) - pos - 1)) > 0) {
        pos += n;
        buf[pos] = '\0';
    }
}

```

```

char *current_line = buf;
char *line_end;

// Обработка каждой строки
while ((line_end = strchr(current_line, '\n'))) {
    *line_end = '\0';

    // Пропуск пустых строк
    if (strlen(current_line) == 0) {
        current_line = line_end + 1;
        continue;
    }

    float result = 0.0;
    int numbers_seen = 0;
    int division_by_zero = 0;
    int valid_input = 1;

    // Парсинг чисел из строки
    char *ptr = current_line;
    while (*ptr) {
        while (*ptr && isspace((unsigned char)*ptr)) {
            ptr++;
        }
        if (!*ptr) {
            break;
        }

        // Проверка на отрицательно число
        int is_negative = 0;
        if (*ptr == '-') {
            is_negative = 1;
            ptr++;
        }
    }
}

```

Select Indentation

```

// Парсинг целой части
float number = 0.0;
int digits_found = 0;
while (*ptr && isdigit((unsigned char)*ptr)) {
    number = number * 10.0 + (*ptr - '0');
    ptr++;
    digits_found = 1;
}

// Парсинг дробной части
if (*ptr == '.') {
    ptr++;
    float fraction = 0.1;
    while (*ptr && isdigit((unsigned char)*ptr)) {
        number += (*ptr - '0') * fraction;
        fraction *= 0.1;
        ptr++;
        digits_found = 1;
    }
}

if (!digits_found) {
    while (*ptr && !isspace((unsigned char)*ptr)) ptr++;
    continue;
}

if (is_negative) {
    number = -number;
}

numbers_seen++;

```

```

if (numbers_seen == 1) {
    // Первое число - делимое
    result = number;
} else {
    // Последующие числа - делители
    if (number == 0.0) {
        division_by_zero = 1;
        break;
    }
    result /= number;
}

while (*ptr && !isspace((unsigned char)*ptr)) {
    valid_input = 0;
    break;
}
if (!valid_input) {
    break;
}
}

// Обработка полученных вычислений
if (!valid_input) {
    const char msg[] = "Error: invalid input format\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);

    char error_entry[128];
    int len = snprintf(error_entry, sizeof(error_entry),
        "Input: \"%s\" -> Error: invalid format\n", current_line);
    write(output_file, error_entry, len);
} else if (division_by_zero) {
    const char msg[] = "Error: division by zero\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
}

```

```

char error_entry[128];
int len = snprintf(error_entry, sizeof(error_entry),
    "Input: \"%s\" -> Error: division by zero\n", current_line);
write(output_file, error_entry, len);

// Если деление на ноль, то завершаем работу
close(output_file);
exit(EXIT_FAILURE);
} else if (numbers_seen < 2) {
    const char msg[] = "Error: not enough numbers (need at least 2)\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);

    char error_entry[128];
    int len = snprintf(error_entry, sizeof(error_entry),
        "Input: \"%s\" -> Error: not enough numbers\n", current_line);
    write(output_file, error_entry, len);
} else {
    // Запись результата в файл при успешном вычислении
    char result_entry[128];
    int len = snprintf(result_entry, sizeof(result_entry),
        "Input: \"%s\" -> Result: %.6f\n", current_line, result);
    write(output_file, result_entry, len);

    const char success_msg[] = "Calculation completed successfully\n";
    write(STDERR_FILENO, success_msg, sizeof(success_msg) - 1);
}

current_line = line_end + 1;
}

pos = strlen(current_line);
memmove(buf, current_line, pos);
}

```

```

// Запись завершающего сообщения в файл
const char footer[] = "\nEnd of calculations.\n";
write(output_file, footer, sizeof(footer) - 1);

close(output_file);
return 0;
}

```

Протокол работы программы

Тесты:

vboxuser@Ubuntu1:~/MAI_OS/MAI_OS/lab_1/src\$./parent

Enter output filename: input.txt

Enter numbers separated by spaces (or 'exit' to quit):

> 10 5 2

Calculation completed successfully

> 450 9 10 5

Calculation completed successfully

> 347 13

Calculation completed successfully

> 1024 2 2 2 4

Calculation completed successfully

> 65 13 5

Calculation completed successfully

> exit

Parent process terminated.

```

input.txt U X
lab_1 > src > input.txt
1  Calculation Results:
2  =====
3  Input: "10 5 2" -> Result: 1.000000
4  Input: "450 9 10 5" -> Result: 1.000000
5  Input: "347 13" -> Result: 26.692308
6  Input: "1024 2 2 2 4" -> Result: 32.000000
7  Input: "65 13 5" -> Result: 1.000000
8
9  End of calculations.
10

```

vboxuser@Ubuntu1:~/MAI_OS/MAI_OS/lab_1/src\$./parent

Enter output filename: input_error.txt

Enter numbers separated by spaces (or 'exit' to quit):

> 12a 3 4

Error: invalid input format

> aaa aa b c

Error: not enough numbers (need at least 2)

> 10 a

Error: not enough numbers (need at least 2)

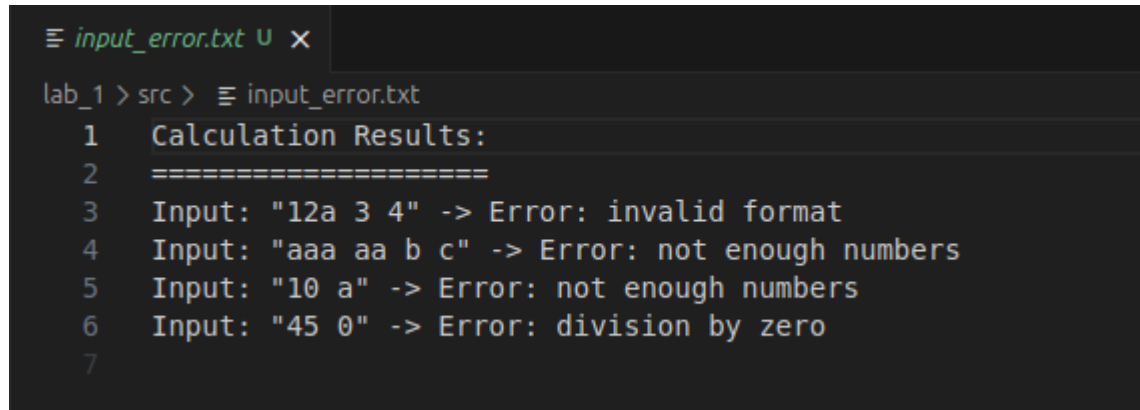
>

> 45 0

Error: division by zero detected. Terminating...

Error: division by zero

Parent process terminated.



```
lab_1 > src > input_error.txt
1 Calculation Results:
2 =====
3 Input: "12a 3 4" -> Error: invalid format
4 Input: "aaa aa b c" -> Error: not enough numbers
5 Input: "10 a" -> Error: not enough numbers
6 Input: "45 0" -> Error: division by zero
7
```

Strace:

```
vboxuser@Ubuntu1:~/MAI_OS/MAI_OS/lab_1/src$ strace ./parent
execve("./parent", ["/parent"], 0x7ffcc2074da0 /* 65 vars */) = 0
brk(NULL)                               = 0x62a1344f7000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x771a3927c000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=56079, ...}) = 0
mmap(NULL, 56079, PROT_READ, MAP_PRIVATE, 3, 0) = 0x771a3926e000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832
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"..., 784, 64) = 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
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"..., 784, 64) = 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x771a39000000
mmap(0x771a39028000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x771a39028000
mmap(0x771a391b0000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x771a391b0000
mmap(0x771a391ff000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x771a391ff000
mmap(0x771a39205000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x771a39205000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x771a3926b000
```

```

arch_prctl(ARCH_SET_FS, 0x771a3926b740) = 0
set_tid_address(0x771a3926ba10) = 7089
set_robust_list(0x771a3926ba20, 24) = 0
rseq(0x771a3926c060, 0x20, 0, 0x53053053) = 0
mprotect(0x771a391ff000, 16384, PROT_READ) = 0
mprotect(0x62a0f8e69000, 4096, PROT_READ) = 0
mprotect(0x771a392ba000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0
munmap(0x771a3926e000, 56079) = 0
write(1, "Enter output filename: ", 23Enter output filename: ) = 23
read(0, input_strace.txt
"input_strace.txt\n", 1023) = 17
pipe2([3, 4], 0) = 0
pipe2([5, 6], 0) = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x771a3926ba10) = 7131
close(3) = 0
close(6) = 0
write(1, "Enter numbers separated by space"..., 55Enter numbers separated by spaces (or 'exit' to
quit):
) = 55
write(1, "> ", 2> ) = 2
read(0, 1000 4 5 6
"1000 4 5 6\n", 4095) = 11
write(4, "1000 4 5 6\n", 11) = 11
pselect6(6, [5], NULL, NULL, {tv_sec=0, tv_nsec=100000000}, NULL) = 1 (in [5], left
{tv_sec=0, tv_nsec=99998489})
read(5, "Calculation completed successful"..., 255) = 35
write(2, "Calculation completed successful"..., 35Calculation completed successfully
) = 35
write(1, "> ", 2> ) = 2
read(0, 345 98 234
"345 98 234\n", 4095) = 11
write(4, "345 98 234\n", 11) = 11
pselect6(6, [5], NULL, NULL, {tv_sec=0, tv_nsec=100000000}, NULL) = 1 (in [5], left
{tv_sec=0, tv_nsec=99998667})
read(5, "Calculation completed successful"..., 255) = 35
write(2, "Calculation completed successful"..., 35Calculation completed successfully
) = 35
write(1, "> ", 2> ) = 2
read(0, 50a a d
"50a a d\n", 4095) = 8
write(4, "50a a d\n", 8) = 8
pselect6(6, [5], NULL, NULL, {tv_sec=0, tv_nsec=100000000}, NULL) = 1 (in [5], left
{tv_sec=0, tv_nsec=99998749})
read(5, "Error: invalid input format\n", 255) = 28
write(2, "Error: invalid input format\n", 28Error: invalid input format
) = 28
write(1, "> ", 2> ) = 2
read(0, 56 7 8 9 5 4
"56 7 8 9 5 4\n", 4095) = 13
write(4, "56 7 8 9 5 4\n", 13) = 13

```



```

    pselect6(6, [5], NULL, NULL, {tv_sec=0, tv_nsec=100000000}, NULL) = 1 (in [5], left
{tv_sec=0, tv_nsec=99689242})
    read(5, "Calculation completed successful"..., 255) = 35
    write(2, "Calculation completed successful"..., 35Calculation completed successfully
) = 35
    write(1, "> ", 2> )          = 2
    read(0, exit
"exit\n", 4095)          = 5
    close(4)              = 0
    close(5)              = 0
    wait4(-1, [{ WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 7131
    --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=7131, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
    write(1, "Parent process terminated.\n", 27Parent process terminated.
) = 27
    exit_group(0)          = ?
    +++ exited with 0 +++

```

Вывод

В ходе лабораторной работы были изучены и применены основные системные вызовы для работы с процессами и межпроцессного взаимодействия в Linux. Реализована программа, демонстрирующая создание процессов, организацию каналов связи и перенаправление стандартных потоков. Для обработки входных данных были разработаны собственные алгоритмы разбора строк и преобразования чисел, учитывающие различные пробелы и разделители. Также обеспечено корректное закрытие файловых дескрипторов для предотвращения утечек.