

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: М80-211Бв-24

Студент: Скворцов Ю.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 22.12.25

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 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». 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 однонаправленный канал межпроцессного взаимодействия.
- `int dup2(int oldfd, int newfd)`; – дублирует файловый дескриптор
- `int execv(const char *path, char *const argv[])`; – заменяет текущий образ процесса новым образом из исполняемого файла.
- `pid_t wait(int *status)`; – приостанавливает выполнение вызывающего процесса до завершения одного из его дочерних процессов.
- `pid_t getpid(void)`; – возвращает PID текущего процесса.

Код программы

parent.c

```
#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

static char CHILD_PROGRAM_NAME[] = "child";
```

```
int main(int argc, char **argv) {  
    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);  
    }  
    char progbath[1024];  
    {  
        ssize_t len = readlink("/proc/self/exe", progbath, sizeof(progbath) - 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 (progbath[len] != '/')  
            --len;  
  
        progbath[len] = '\\0';  
    }  
  
    int parent_to_child[2];  
    if (pipe(parent_to_child) == -1) {  
        const char msg[] = "error: failed to create pipe\n";  
        write(STDERR_FILENO, msg, sizeof(msg));  
        exit(EXIT_FAILURE);  
    }  
}
```

```
}

int child_to_parent[2];

if (pipe(child_to_parent) == -1) {

    const char msg[] = "error: failed to create pipe\n";

    write(STDERR_FILENO, msg, sizeof(msg));

    exit(EXIT_FAILURE);

}


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: {

    close(parent_to_child[1]);

    close(child_to_parent[0]);


    dup2(parent_to_child[0], STDIN_FILENO);

    close(parent_to_child[0]);


    dup2(child_to_parent[1], STDOUT_FILENO);

    close(child_to_parent[1]);


    {
```

```

        char path[1024];

        size_t path_len = snprintf(path, sizeof(path) - 1, "%s/%s", progbpath,
CHILD_PROGRAM_NAME);

        if (path_len >= sizeof(path)) {

            const char msg[] = "error: path too long\n";

            write(STDERR_FILENO, msg, sizeof(msg));

            exit(EXIT_FAILURE);

        }

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

        int32_t status = execv(path, args);

        if (status == -1) {

            const char msg[] = "error: failed to exec into new executable
image\n";

            write(STDERR_FILENO, msg, sizeof(msg));

            exit(EXIT_FAILURE);

        }

    }

} break;

default: {

    close(parent_to_child[0]);

    close(child_to_parent[1]);

    {

        pid_t pid = getpid();

        char msg[128];

        const int32_t length = snprintf(msg, sizeof(msg),

```

```
        "%d: Родительский процесс, дочерний PID: %d\n", pid, child);

    write(STDOUT_FILENO, msg, length);

}

printf("Введите команды в формате: число число число ...\n");

printf("Для выхода введите 'exit'\n");

char buf[4096];

ssize_t bytes;

while (bytes = read(STDIN_FILENO, buf, sizeof(buf))) {

    if (bytes < 0) {

        const char msg[] = "error: failed to read from stdin\n";

        write(STDERR_FILENO, msg, sizeof(msg));

        exit(EXIT_FAILURE);

    }

    if (bytes >= 4 && strncmp(buf, "exit", 4) == 0) {

        break;

    }

    write(parent_to_child[1], buf, bytes);

    bytes = read(child_to_parent[0], buf, sizeof(buf));

    if (bytes > 0) {

        if (strstr(buf, "DIVISION_BY_ZERO") != NULL) {

            printf("Обнаружено деление на ноль! Завершение работы.\n");

            break;

        } else if (strstr(buf, "OK") != NULL) {

        }

    } else if (bytes == 0) {
```

```

        printf("Дочерний процесс завершился.\n");

        break;

    }

}

close(parent_to_child[1]);

close(child_to_parent[0]);

wait(NULL);

printf("Родительский процесс завершен.\n");

} break;

}

return 0;

}

```

child.c

```

#include <stdint.h>

#include <stdbool.h>

#include <ctype.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdio.h>

#include <string.h>

#define MAX_NUMBERS 100

int main(int argc, char **argv) {

```

```
int 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;

float numbers[MAX_NUMBERS];


while ((bytes = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {

    if (bytes < 0) {

        const char msg[] = "error: failed to read from stdin\n";

        write(STDERR_FILENO, msg, sizeof(msg));

        exit(EXIT_FAILURE);

    }

    buf[bytes] = '\0';


    int count = 0;

    char *token = strtok(buf, " \t\n");


    while (token != NULL && count < MAX_NUMBERS) {

        int valid = 1;

        int dot_count = 0;

        for (int i = 0; token[i] != '\0'; i++) {

            if (!isdigit(token[i]) && token[i] != '.' &&

                !(i == 0 && token[i] == '-')) {

                valid = 0;

            }

        }

    }

}
```



```
        break;

    }

    if (token[i] == '.') dot_count++;

}

if (valid && dot_count <= 1) {

    numbers[count++] = atof(token);

}

token = strtok(NULL, " \t\n");

}

if (count < 2) {

    const char error_msg[] = "ERROR: Need at least 2 numbers\n";

    write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);

    const char file_error[] = "Ошибка: необходимо минимум 2 числа\n";

    write(file, file_error, sizeof(file_error) - 1);

    continue;

}

float first = numbers[0];

char calculation[512];

int calc_len = snprintf(calculation, sizeof(calculation),

                        "Вычисление: %.2f / ", first);

write(file, calculation, calc_len);

for (int i = 1; i < count; i++) {

    calc_len = snprintf(calculation, sizeof(calculation),
```

```
        "%.2f", numbers[i]);

    write(file, calculation, calc_len);

    if (i < count - 1) {

        const char divider[] = " / ";

        write(file, divider, sizeof(divider) - 1);

    }

}

const char newline[] = "\n";

write(file, newline, sizeof(newline) - 1);


float result = first;

for (int i = 1; i < count; i++) {

    if (numbers[i] == 0.0f) {

        const char error_msg[] = "DIVISION_BY_ZERO\n";

        write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);

        const char file_error[] = "ОШИБКА: деление на ноль!\n";

        write(file, file_error, sizeof(file_error) - 1);

        close(file);

        exit(EXIT_FAILURE);

    }

    result /= numbers[i];

}

char result_str[128];

int result_len = snprintf(result_str, sizeof(result_str),

        "Результат: %.6f\n\n", result);

write(file, result_str, result_len);
```

```

    const char success_msg[] = "OK\n";

    write(STDOUT_FILENO, success_msg, sizeof(success_msg) - 1);

}

close(file);

return 0;

}

```

Протокол работы программы

```

53847: Родительский процесс, дочерний PID: 53848
Введите команды в формате: число число число ...
Для выхода введите 'exit'
10 5 2
50 2 3
1 1 1
10 10 10
exit
Родительский процесс завершен.

```

```

≡ out.txt
1  Вычисление: 10.00 / 5.00 / 2.00
2  Результат: 1.000000
3
4  Вычисление: 50.00 / 2.00 / 3.00
5  Результат: 8.333333
6
7  Вычисление: 1.00 / 1.00 / 1.00
8  Результат: 1.000000
9
10 Вычисление: 10.00 / 10.00 / 10.00
11 Результат: 0.100000
12

```

```
54630: Родительский процесс, дочерний PID: 54631
Введите команды в формате: число число число ...
Для выхода введите 'exit'
3 4 2
0 1 1
1 0 0
Обнаружено деление на ноль! Завершение работы.
Родительский процесс завершен.
54630: Родительский процесс, дочерний PID: 54631
Введите команды в формате: число число число ...
Для выхода введите 'exit'
3 4 2
0 1 1
1 0 0
Обнаружено деление на ноль! Завершение работы.
Родительский процесс завершен.
```

```
≡ out.txt
1  Вычисление: 3.00 / 4.00 / 2.00
2  Результат: 0.375000
3
4  Вычисление: 0.00 / 1.00 / 1.00
5  Результат: 0.000000
6
7  Вычисление: 1.00 / 0.00 / 0.00
8  ОШИБКА: деление на ноль!
9
```

Вывод

В ходе выполнения лабораторной работы были успешно освоены практические навыки управления процессами в ОС Linux. Была реализована программа, использующая системные вызовы `fork()`, `pipe()` и `dup2()` для создания дочернего процесса и организации межпроцессного взаимодействия через каналы. Программа корректно обрабатывает пользовательский ввод, выполняет арифметические операции с проверкой деления на ноль и сохраняет результаты в файл. Работа продемонстрировала эффективность использования механизмов межпроцессного взаимодействия для распределения задач между родительским и дочерним процессами.