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-211БВ-24

Студент: Губеев Д.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 02.10.25

Москва, 2025

Постановка задачи

Вариант 4.

Пользователь вводит команды вида: «число число число<endline>». 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 анонимный канал, возвращает 0 при успехе, -1 при ошибке
- ssize_t read(int fd, void *buf, size_t count) - читает данные из файла по дескриптору fd в буфер buf. Возвращает количество прочитанных байт
- ssize_t write(int fd, const void *buf, size_t count) - записывает данные в файл по дескриптору fd. Возвращает число записанных байт
- int open(const char *pathname, int flag, mode_t mode) - Открывает файл, возвращает файловый дескриптор
- int close(int fd) - закрываает файловый дескриптор
- int execv(const char *path, char *const argv[]) - заменяет текущий процесс новым. Возвращает -1 при ошибке
- int dup2(int fd, int fd2) - замена файлового дескриптора
- pid_t wait(int *wstatus) - ожидает завершения дочернего процесса. Возвращает pid завершившегося процесса
- pid_t getpid(void) - возвращает pid текущего процесса
- ssize_t readlink(const char *pathname, char *buf, size_t bufsiz) - считывает путь, на который указывает символьная ссылка

Клиент и сервер общаются через пайпы. Клиент запускает сервер в дочернем процессе, перенаправляя ему стандартный ввод и вывод на пайпы. Из консоли клиент читает числа, отправляет их серверу. Сервер принимает строку чисел из stdin, разбивает на отдельные значения и делит первое число на каждое из следующих. Результаты сервер записывает в выходной файл. Если встречается деление на ноль, сервер и клиент завершают свою работу.

Код программы

client.c

```c
#include <stdint.h>

#include <stdbool.h>



#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>
```

```c
#include <stdio.h>
#include <ctype.h>

static char SERVER_PROGRAM_NAME[] = "server.out";

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);
    }

    // NOTE: Get full path to the directory, where program resides
    char progpath[1024];
    {
        // NOTE: Read full program path, including its name
        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);
        }

        // NOTE: Trim the path to first slash from the end
        while (progpath[len] != '/')
            --len;
```

```
    progpath[len] = '\0';
}

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

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

// NOTE: Spawn a new process
const pid_t child = fork();

switch (child) {
case -1: { // NOTE: Kernel fails to create another process
    const char msg[] = "error: failed to spawn new process\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
} break;
```

```
    case 0: { // NOTE: We're a child, child doesn't know its pid after fork

        {

            pid_t pid = getpid(); // NOTE: Get child PID


            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]);

        close(server_to_client[0]);


        dup2(client_to_server[0], STDIN_FILENO);

        close(client_to_server[0]);


        dup2(server_to_client[1], STDOUT_FILENO);

        close(server_to_client[1]);


        {

            char path[1024];

            snprintf(path, sizeof(path) - 1, "%s/%s", progpath,
SERVER_PROGRAM_NAME);


            // NOTE: args[0] must be a program name, next the actual arguments

            // NOTE: `NULL` at the end is mandatory, because `exec*`

            //       expects a NULL-terminated list of C-strings

            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: { // NOTE: We're a parent, parent knows PID of child after fork
        {
            pid_t pid = getpid(); // NOTE: Get parent PID

            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]);
        close(server_to_client[1]);

        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);
            } else if (buf[0] == '\n') {
                // NOTE: When Enter is pressed with no input, then exit client
                break;
            }

            write(client_to_server[1], buf, bytes);
            bytes = read(server_to_client[0], buf, sizeof(buf));
            if (bytes == 0) { // division by zero
                break;
            }
            write(STDOUT_FILENO, buf, bytes);
        }

        close(client_to_server[1]);
        close(server_to_client[0]);

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

server.c

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

```c
#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <fcntl.h>

#include <math.h>


void divideNumbers(char * input, int32_t file) {

    char buf[4096];

    strncpy(buf, input, sizeof(buf) - 1);

    buf[sizeof(buf) - 1] = '\0';


    char * saveptr;

    char * token = strtok_r(buf, " ", &saveptr);

    float numbers[100];

    int count = 0;


    while (token != NULL && count < 100) {

        numbers[count++] = strtof(token, NULL);

        token = strtok_r(NULL, " ", &saveptr);

    }


    if (count < 2) {

        const char msg[] = "error: need at least two numbers\n";

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

        return;

    }


    float dividend = numbers[0];
```

```
    for (int i = 1; i < count; i++) {
        if (fabs(numbers[i]) < 1e-6) {
            const char msg[] = "error: division by zero\n";
            write(STDERR_FILENO, msg, sizeof(msg));

            exit(EXIT_FAILURE);
        }

        float result = dividend / numbers[i];

        char resultStr[50];
        int len = snprintf(resultStr, sizeof(resultStr), "%f ", result);

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

int main(int argc, char **argv) {
    char buf[4096];
    ssize_t bytes;
```

```
    pid_t pid = getpid();


    // NOTE: `O_WRONLY` only enables file for writing
    // NOTE: `O_CREAT` creates the requested file if absent
    // NOTE: `O_TRUNC` empties the file prior to opening
    // NOTE: `O_APPEND` subsequent writes are being appended instead of
overwritten
    int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
    if (file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }


    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);
        }


        buf[bytes] = '\0';
        divideNumbers(buf, file);


        {
            {
                const char msg[] = "Server received: ";
                write(STDERR_FILENO, msg, sizeof(msg) - 1);
```

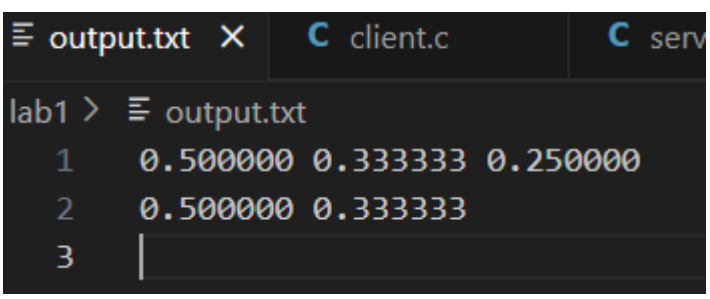
```
            }


            // NOTE: Echo back to client
            int32_t written = write(STDOUT_FILENO, buf, bytes);
            if (written != bytes) {
                const char msg[] = "error: failed to echo\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
        }
    }


    close(file);
}
```

Протокол работы программы

Входные данные:

```
15849: I'm a parent, my child has PID 15850
15850: I'm a child
1 2 3 4
Server received: 1 2 3 4
1 2 3
Server received: 1 2 3
1 0
error: division by zero
```

Выходные данные:



Вывод

При выполнении данной лабораторной работы я научился работать с процессами в ОС. На практике увидел работу fork, pipe и других системных вызовов. Научился обрабатывать ввод, вывод как поток байтов, преобразовывать его в строки.