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-211Б-23

Студент: Соболин Т.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 12.01.25

Москва, 2025

Постановка задачи

Вариант 2.

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:

- fork() — создает новый процесс. ○ pipe() — создает анонимный канал для межпроцессного взаимодействия. ○ read() — читает данные из файлового дескриптора. ○ write() — записывает данные в файловый дескриптор.
- strtok() — разбивает строку на лексемы. ○ atof() — преобразует строку в число с плавающей точкой.
- fopen() — открывает файл для чтения или записи. ○ fprintf() — записывает форматированные данные в файл. ○ fclose() — закрывает открытый файл.
- perror() — выводит сообщение об ошибке. ○ strcpy() — копирует строку в буфер. ○ strcmp() — сравнивает две строки. ○ strlen() — возвращает длину строки. ○ strcspn() — возвращает длину части строки до первого появления любого символа из заданного набора.

Программа создает два процесса: родительский и дочерний. Родительский процесс принимает от пользователя числа, разделенные пробелами, и отправляет их дочернему процессу через анонимный канал (пайп). Дочерний процесс получает эти числа, суммирует их и записывает результат в конец указанного файла. Если пользователь вводит команду "exit", программа завершает работу.

Код программы

parent.c

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

#define BUFFER_SIZE 1024

// Функция для преобразования числа с плавающей запятой в строку
void float_to_string(float value, char* buffer, int precision) {
    int int_part = (int)value;
    float fractional_part = value - int_part;

    // Сначала добавляем целую часть
    int len = 0;
    if (int_part == 0) {
        buffer[len++] = '0';
    } else {
        int temp = int_part;
        if (temp < 0) {
            buffer[len++] = '-';
            temp = -temp;
        }
        int digits[10]; // Храним цифры целой части
        int digit_count = 0;
        while (temp > 0) {
            digits[digit_count++] = temp % 10;
            temp /= 10;
        }
        // Заполняем буфер целыми цифрами в правильном порядке
        for (int i = digit_count - 1; i >= 0; i--) {
            buffer[len++] = digits[i] + '0';
        }
    }

    // Добавляем дробную часть
    if (precision > 0) {
        buffer[len++] = '.'; // Добавляем десятичную точку
        for (int i = 0; i < precision; i++) {
            fractional_part *= 10;
            int fractional_digit = (int)fractional_part;
            buffer[len++] = fractional_digit + '0';
            fractional_part -= fractional_digit;
        }
    }

    buffer[len] = '\0'; // Завершаем строку
}

void write_to_file(const char *filename, const char *data, size_t len) {
    int fd = open(filename, O_WRONLY | O_APPEND | O_CREAT, 0666);
    if (fd != -1) {
        write(fd, data, len); // Пишем данные в файл
        close(fd);
    } else {
        const char *error_msg = "Ошибка при открытии файла\n";
        write(2, error_msg, strlen(error_msg)); // Записываем сообщение об ошибке в stderr
    }
}
```

```
int main(int argc, char *argv[]) {
    int pipe1[2]; // pipe для передачи данных от родителя к дочернему
    pid_t pid;

    if (argc != 2) {
        const char *error_msg = "Использование: <имя_файла>\n";
        write(2, error_msg, strlen(error_msg)); // Печатаем сообщение об ошибке
        exit(EXIT_FAILURE);
    }

    // Создаем pipes
    if (pipe(pipe1) == -1) {
        const char *error_msg = "Ошибка при создании pipe\n";
        write(2, error_msg, strlen(error_msg));
        exit(EXIT_FAILURE);
    }

    // Создаем дочерний процесс
    pid = fork();
    if (pid < 0) {
        const char *error_msg = "Ошибка при fork\n";
        write(2, error_msg, strlen(error_msg));
        exit(EXIT_FAILURE);
    }

    if (pid == 0) { // Дочерний процесс
        close(pipe1[1]); // Закрываем запись в pipe1
        dup2(pipe1[0], STDIN_FILENO); // Перенаправляем stdin на pipe1[0]
        close(pipe1[0]);

        char *args[] = {"./child", argv[1], NULL};
        execv(args[0], args);

        // Если execv не выполнился
        perror("Ошибка при execv");
        exit(EXIT_FAILURE);

        close(pipe1[0]);
        exit(EXIT_SUCCESS);
    } else { // Родительский процесс
        close(pipe1[0]); // Закрываем чтение из pipe1

        char input[BUFFER_SIZE];
        while (1) {
            const char *prompt = "Введите числа (или 'exit' для выхода): ";
            write(1, prompt, strlen(prompt)); // 1 - файловый дескриптор для stdout

            ssize_t bytesRead = read(0, input, BUFFER_SIZE); // 0 - файловый дескриптор для stdin
            if (bytesRead <= 0) {
                break; // Ошибка или конец ввода
            }

            input[bytesRead - 1] = '\0'; // Удаляем символ новой строки в конце (если есть)

            // Отправляем данные дочернему процессу
            write(pipe1[1], input, strlen(input) + 1);

            if (strcmp(input, "exit") == 0) {
                break;
            }
        }

        close(pipe1[1]);
        wait(NULL); // Ожидаем завершения дочернего процесса
        exit(EXIT_SUCCESS);
    }

}
```

Child.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUFFER_SIZE 1024

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Ошибка: отсутствует имя
выходного файла\n");
        exit(EXIT_FAILURE);
    }

    char *filename = argv[1]; // Имя выходного файла из
аргументов
    char buffer[BUFFER_SIZE];

    while (1) {
        ssize_t bytesRead = read(STDIN_FILENO, buffer,
BUFFER_SIZE);
        if (bytesRead <= 0) {
            break; // Ошибка или конец потока
        }

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

        if (strcmp(buffer, "exit") == 0) {
            break;
        }

        // Обработка чисел
        float sum = 0.0;
        char *token = strtok(buffer, " ");
        while (token != NULL) {
            sum += atof(token);
            token = strtok(NULL, " ");
        }

        // Записываем результат в файл
        FILE *file = fopen(filename, "a");
        if (file != NULL) {
            fprintf(file, "Сумма: %.2f\n", sum);
            fclose(file);
        } else {
            perror("Ошибка при открытии файла");
        }
    }

    exit(EXIT_SUCCESS);
}
```

```
}
```

Протокол работы программы

Тестирование:

```
kotlasboy@kotlasboy-Modern-15-B12M:~/Programming/Projects/OS/lab_1$ ./a.out output.txt
Введите числа (или 'exit' для выхода): 10.2 Введите
числа (или 'exit' для выхода): 10.2 10.2  Введите
числа (или 'exit' для выхода):
Введите числа (или 'exit' для выхода): exit
```

Strace:

```
kotlasboy@kotlasboy-Modern-15-B12M:~/Programming/Projects/OS/lab_1$ strace -f ./a.out output.txt

execve("./a.out", ["./a.out", "output.txt"], 0x7ffe85f5ffa0 /* 59 vars */) = 0

brk(NULL)                               = 0x647dac0a8000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a480bf60000

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=68847, ...}) = 0

mmap(NULL, 68847, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a480bf4f000

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\0\3\0>\0\1\0\0\0\220\243\2\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\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\0\0\0"..., 784, 64) = 784

mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a480bc00000

mmap(0x7a480bc28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7a480bc28000

mmap(0x7a480bdb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7a480bdb0000

mmap(0x7a480bdff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7a480bdff000

mmap(0x7a480be05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7a480be05000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a480bf4c000

arch_prctl(ARCH_SET_FS, 0x7a480bf4c740) = 0

set_tid_address(0x7a480bf4ca10)         = 29679

set_robust_list(0x7a480bf4ca20, 24)     = 0

rseq(0x7a480bf4d060, 0x20, 0, 0x53053053) = 0
```

mprotect(0x7a480bdff000, 16384, PROT_READ) = 0

mprotect(0x647d9f520000, 4096, PROT_READ) = 0

mprotect(0x7a480bf98000, 8192, PROT_READ) = 0

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

munmap(0x7a480bf4f000, 68847)   = 0

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

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 29680
attached

, child_tidptr=0x7a480bf4ca10) = 29680

[pid 29679] close(3)   = 0

[pid 29679] write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\207\320\270\321\201\320\273\320\260 (\320\270\320\273\320"..., 63Введите числа (или 'exit'
для выхода): <unfinished ...>

[pid 29680] set_robust_list(0x7a480bf4ca20, 24 <unfinished ...>

[pid 29679] <... write resumed>)   = 63

[pid 29679] read(0, <unfinished ...>

[pid 29680] <... set_robust_list resumed>) = 0

[pid 29680] close(4)   = 0

[pid 29680] dup2(3, 0)   = 0

[pid 29680] close(3)   = 0

[pid 29680] execve("./child", ["./child", "output.txt"], 0x7ffd859667f0 /* 59 vars */) = 0

[pid 29680] brk(NULL)   = 0x60691d7fd000

[pid 29680] mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7279da298000

[pid 29680] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 29680] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 29680] fstat(3, {st_mode=S_IFREG|0644, st_size=68847, ...}) = 0

[pid 29680] mmap(NULL, 68847, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7279da287000

[pid 29680] close(3)   = 0

[pid 29680] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) =
3

[pid 29680] 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

[pid 29680] 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

[pid 29680] fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 29680] 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

[pid 29680] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7279da000000

[pid 29680] mmap(0x7279da028000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7279da028000

[pid 29680] mmap(0x7279da1b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7279da1b0000

[pid 29680] mmap(0x7279da1ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7279da1ff000

[pid 29680] mmap(0x7279da205000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7279da205000

[pid 29680] close(3)   = 0

[pid 29680] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7279da284000

[pid 29680] arch_prctl(ARCH_SET_FS, 0x7279da284740) = 0

[pid 29680] set_tid_address(0x7279da284a10) = 29680

[pid 29680] set_robust_list(0x7279da284a20, 24) = 0

[pid 29680] rseq(0x7279da285060, 0x20, 0, 0x53053053) = 0

[pid 29680] mprotect(0x7279da1ff000, 16384, PROT_READ) = 0

[pid 29680] mprotect(0x606912a20000, 4096, PROT_READ) = 0

[pid 29680] mprotect(0x7279da2d0000, 8192, PROT_READ) = 0

[pid 29680] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 29680] munmap(0x7279da287000, 68847) = 0

[pid 29680] read(0, 10.2

 <unfinished ...>

[pid 29679] <... read resumed>"10.2\n", 1024) = 5

[pid 29679] write(4, "10.2\0", 5)   = 5

[pid 29680] <... read resumed>"10.2\0", 1024) = 5

[pid 29679] write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260 (\320\270\320\273\320"..., 63Введите числа (или 'exit' для выхода):) = 63

[pid 29680] getrandom(<unfinished ...>

[pid 29679] read(0, <unfinished ...>

[pid 29680] <... getrandom resumed>"\xf3\xdc\x86\xf1\x9f\xe2\x81\x16", 8, GRND_NONBLOCK) = 8

[pid 29680] brk(NULL)   = 0x60691d7fd000

[pid 29680] brk(0x60691d81e000)   = 0x60691d81e000

[pid 29680] openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_APPEND, 0666) = 3

[pid 29680] lseek(3, 0, SEEK_END)   = 89

[pid 29680] fstat(3, {st_mode=S_IFREG|0664, st_size=89, ...}) = 0

[pid 29680] write(3, "\320\241\321\203\320\274\320\274\320\260: 10.20\n", 18) = 18

[pid 29680] close(3)   = 0

[pid 29680] read(0, 10.2 10.2

 <unfinished ...>

[pid 29679] <... read resumed>"10.2 10.2\n", 1024) = 10

[pid 29679] write(4, "10.2 10.2\0", 10) = 10

[pid 29680] <... read resumed>"10.2 10.2\0", 1024) = 10

[pid 29679] write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260 (\320\270\320\273\320"..., 63 <unfinished ...>

[pid 29680] openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_APPEND, 0666Введите числа (или 'exit' для выхода): <unfinished ...>

[pid 29679] <... write resumed>)   = 63

[pid 29680] <... openat resumed>)   = 3

[pid 29679] read(0, <unfinished ...>

[pid 29680] lseek(3, 0, SEEK_END)   = 107

[pid 29680] fstat(3, {st_mode=S_IFREG|0664, st_size=107, ...}) = 0

[pid 29680] write(3, "\320\241\321\203\320\274\320\274\320\260: 20.40\n", 18) = 18

[pid 29680] close(3)   = 0

[pid 29680] read(0,

<unfinished ...>

[pid 29679] <... read resumed>"\n", 1024) = 1

[pid 29679] write(4, "\0", 1)   = 1

[pid 29680] <... read resumed>"\0", 1024) = 1

[pid 29679] write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260 (\320\270\320\273\320"..., 63Введите числа (или 'exit' для выхода):) = 63

[pid 29680] openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_APPEND, 0666 <unfinished ...>

[pid 29679] read(0, <unfinished ...>

[pid 29680] <... openat resumed>)   = 3

[pid 29680] lseek(3, 0, SEEK_END)   = 125

[pid 29680] fstat(3, {st_mode=S_IFREG|0664, st_size=125, ...}) = 0

[pid 29680] write(3, "\320\241\321\203\320\274\320\274\320\260: 0.00\n", 17) = 17

[pid 29680] close(3)   = 0

[pid 29680] read(0, exit

<unfinished ...>

[pid 29679] <... read resumed>"exit\n", 1024) = 5

[pid 29679] write(4, "exit\0", 5)   = 5

[pid 29679] close(4 <unfinished ...>

[pid 29680] <... read resumed>"exit\0", 1024) = 5

[pid 29679] <... close resumed>)   = 0

[pid 29679] wait4(-1, <unfinished ...>

[pid 29680] exit_group(0)   = ?

[pid 29680] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL)   = 29680

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=29680, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

exit_group(0)   = ?

+++ exited with 0 +++

Вывод

Программа, использующая общую память для взаимодействия между родительским и дочерним процессами, демонстрирует эффективный способ передачи данных в многопоточных или многопроцессорных системах. Родительский процесс считывает ввод пользователя и записывает его в сегмент общей памяти, в то время как дочерний процесс обрабатывает эти данные и записывает результаты в файл. Использование общей памяти позволяет избежать накладных расходов на межпроцессное взаимодействие, обеспечивая более быструю и эффективную передачу информации. В конечном итоге, программа иллюстрирует важность управления памятью и синхронизации процессов для достижения оптимальной производительности.