Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-209БВ-24

Студент: Фимина А.О.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 16.11.25

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 ее в файл. Числа имеют.

Общий метод и алгоритм решения

Использованные системные вызовы

- *int open(const char path, int flags, mode_t mode) — создаёт или открывает файл
- int ftruncate(int fd, off_t length) — изменяет размер файла (выделяет нужный объём под shared memory)
- *void* mmap(void addr, size_t length, int prot, int flags, int fd, off_t offset) — отображает файл в память и предоставляет общий участок памяти для процессов
- pid_t fork(void) — создаёт дочерний процесс
- *pid_t waitpid(pid_t pid, int status, int options) — ожидает завершения дочернего процесса
- int kill(pid_t pid, int sig) — отправляет сигнал дочернему процессу
- **int sigaction(int signum, const struct sigaction act, struct sigaction oldact) — устанавливает обработчик сигнала в дочернем процессе
- unsigned int sleep(unsigned int seconds) / int usleep(useconds_t usec) — временная задержка
- int close(int fd) — закрывает файловый дескриптор
- *int munmap(void addr, size_t length) — освобождает участок памяти, созданный через mmap
- ssize_t write(int fd, const void buf, size_t count) — запись данных в файл
- *ssize_t read(int fd, void buf, size_t count) — чтение данных из файла
- exit(int status) — завершает процесс

Алгоритм работы:

1. Родительский процесс запрашивает имя выходного файла.
   Пользователь вводит имя файла, в который будут записаны результаты, например output.txt.
2. Создание файла и выделение разделяемой памяти
   Родитель:
   1. Открывает файл shared.dat
   2. Увеличивает его до нужного размера через ftruncate
   3. Отображает его в память (mmap)

4. Получает указатель на общий буфер, куда будет писать числа

Этот участок памяти будет доступен **и родителю, и дочернему процессу**.

3. Создание дочернего процесса

   Родитель вызывает fork().

   - Родитель → продолжает логическую работу
   - Дочерний процесс → устанавливает обработчик сигнала и ждёт уведомления

4. Дочерний процесс

   После fork() дочерний процесс:

   1. Настраивает обработчик сигнала SIGUSR1 с помощью sigaction
   2. Переходит в режим ожидания (обычно pause())
   3. При получении сигнала SIGUSR1:
      - Читает разделяемую память (строка чисел)
      - Разбивает строку на токены
      - Вычисляет сумму/среднее (то, что делает твоя программа)
      - Записывает результат в выходной файл
   4. Ждёт следующего сигнала или завершения

5. Родительский процесс

   Родитель:

   1. Ждёт ввода строк чисел от пользователя
   2. Каждую введённую строку:
      - Записывает в разделяемую память
      - Посылает дочернему процессу SIGUSR1 через kill(pid, SIGUSR1)
      - Делает задержку (например 100ms), чтобы дать дочернему процессу обработать данные
   3. Когда пользователь вводит пустую строку:
      - Родитель прекращает отправку сигналов
      - Посылает дочернему процессу сигнал завершения или просто закрывает разделяемую память

6. Завершение работы
   1. Родитель вызывает waitpid() — ожидает завершения дочернего процесса
   2. Освобождает shared memory (munmap)
   3. Закрывает файлы и корректно завершает выполнение

Код программы

parent.c

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

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>   // ftruncate, usleep
#include <sys/wait.h>
#include <signal.h>   // kill
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>

static const size_t SHARED_SIZE = 4096;

int main(void) {
    char outname[256];
    printf("Enter output filename: ");
    if (fgets(outname, sizeof(outname), stdin) == NULL) {
        fprintf(stderr, "Error reading filename\n");
        return 1;
    }

    size_t len = strlen(outname);
    if (len > 0 && outname[len - 1] == '\n') outname[len - 1] = '\0';
    if (strlen(outname) == 0) {
        fprintf(stderr, "Filename cannot be empty\n");
        return 1;
    }

    const char* shared_file = "shared.dat";

    int fd = open(shared_file, O_RDWR | O_CREAT | O_TRUNC, 0666);
    if (fd < 0) { perror("open"); return 1; }

    if (ftruncate(fd, SHARED_SIZE) < 0) { perror("ftruncate"); close(fd); return 1; }
```

```
    void* shared = mmap(NULL, SHARED_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (shared == MAP_FAILED) { perror("mmap"); close(fd); return 1; }
    close(fd);

    pid_t pid = fork();
    if (pid == -1) { perror("fork"); munmap(shared, SHARED_SIZE); return 1; }

    if (pid == 0) {
        execl("./child", "child", outname, shared_file, NULL);
        perror("exec"); munmap(shared, SHARED_SIZE); return 1;
    }

    usleep(100000); // 100ms delay

    printf("Enter float numbers (empty line to exit):\n");

    char line[SHARED_SIZE];
    while (1) {
        if (!fgets(line, sizeof(line), stdin)) break;
        len = strlen(line);
        if (len > 0 && line[len - 1] == '\n') line[len - 1] = '\0';
        if (strlen(line) == 0) break;

        memset(shared, 0, SHARED_SIZE);
        memcpy(shared, line, strlen(line));

        if (kill(pid, SIGUSR1) < 0) perror("kill");
    }

    if (kill(pid, SIGTERM) < 0) perror("kill");
    int status;
    waitpid(pid, &status, 0);
    munmap(shared, SHARED_SIZE);
```

```
    return 0;
}
```

child.c

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

static char* shared_data = NULL;
static const size_t SHARED_SIZE = 4096;
static int should_exit = 0;
static FILE* outfile = NULL;

void handle_signal(int sig) {
    if (sig == SIGTERM) { should_exit = 1; return; }
    if (!shared_data) return;

    char buffer[SHARED_SIZE];
    memcpy(buffer, shared_data, SHARED_SIZE);

    float num, sum = 0;
    char* token;
    char* rest = buffer;

    while ((token = strtok_r(rest, " \t\n", &rest)) != NULL) {
        if (sscanf(token, "%f", &num) == 1) sum += num;
    }

    fprintf(outfile, "Sum: %f\n", sum);
    fflush(outfile);
```

```
}

int main(int argc, char* argv[]) {
    if (argc != 3) { fprintf(stderr, "Usage: child <output_file> <shared_file>\n"); return 1; }

    const char* outname = argv[1];
    const char* shared_name = argv[2];

    outfile = fopen(outname, "w");
    if (!outfile) { perror("fopen"); return 1; }

    int fd = open(shared_name, O_RDONLY);
    if (fd < 0) { perror("open shared file"); fclose(outfile); return 1; }

    shared_data = mmap(NULL, SHARED_SIZE, PROT_READ, MAP_SHARED, fd, 0);
    if (shared_data == MAP_FAILED) { perror("mmap"); close(fd); fclose(outfile); return 1; }

    close(fd);

    signal(SIGUSR1, handle_signal);
    signal(SIGTERM, handle_signal);

    while (!should_exit) pause();

    if (shared_data != MAP_FAILED) munmap(shared_data, SHARED_SIZE);
    if (outfile) fclose(outfile);

    return 0;
}
```

Протокол работы программы

Тестирование:

```
$ ./parent
Enter output filename: output.txt
```

Enter float numbers (empty line to exit):

1 2 3 4

-0.5

<empty line>

$ cat output.txt

Sum: 10

Sum: -0.5

Strace(через docker):

$ strace -f ./parent

29 syscall_0xf00004d3c6e0(0xf00004d3d2f0, 0, 0x1, 0, 0x1, 0xac) = 0x800000090000

29 syscall_0xf00004d3c6e0(0xf00004d3d2f0, 0, 0x1, 0, 0x1c, 0x87) = 0xeffff7df5000

29 syscall_0x7fffffffe040(0x555555557d30, 0x555555555389, 0x1, 0, 0, 0x63) = 0x7fffff5963d0

29 syscall_0xeffff7df4d01(0xeffff7df4d00, 0xeffff7df4c18, 0x2, 0, 0x8000001905c8, 0xdd) = 0

29 read(0, NULL, 0) = 8388608

29 read(0, NULL, 0) = 8388608

29 read(0, NULL, 0) = 8388608

29 read(0, NULL, 0) = 0

29 read(0, NULL, 0) = 0

29 read(0, NULL, 0) = 0

29 read(0, NULL, 0) = 0

29 read(0, NULL, 0) = 0

29 read(0, NULL, 0) = 0

29 read(0, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 56) = 140737489999304

29 read(0, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 56) = 140737489999304

29 read(0, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 56) = 140737489999304

29 read(0, 0xfffffffffffff000, 28) = 140737489995112

29 read(0, 0xfffffffffffff000, 28) = 8190

29 read(0, 0xfffffffffffff000, 28) = 8190

29 read(0, 0xfffffffffffff000, 28) = 8190

29 write(5, 0x1999999999999999, 10) = 10

29 write(5, "\1", 1) = 18446744073709547520

29 write(5, "\1", 1) = 18446744073709547520

29 syscall_0xf00004d3d530(0xf00004d3d650, 0xf00004d3d650, 0xf00004d3d6e0, 0, 0, 0x43) = 0x800000009db3

29 syscall_0xf00004d3d530(0xf00004d3d650, 0xf00004d3d650, 0xf00004d3d6e0, 0, 0, 0x39) = 0x800000009db3

29 syscall_0xffffd87703a9(0xffffd87703a8, 0xffffd876f228, 0x98, 0, 0x800000009f15, 0x30) = 0

29 syscall_0xffffd87703a9(0xffffd87703a8, 0xffffd876f228, 0x98, 0, 0x80000000722a, 0x30) = 0

29 syscall_0xffffd876f2e1(0xffffd876f2e0, 0xffffd876f1f8, 0x1, 0, 0x80000000c93e, 0x4e) = 0

29 syscall_0xffffd876f2e1(0xffffd876f2e0, 0xffffd876f1f8, 0x1, 0, 0x80000000c93e, 0x3f) = 0

29 syscall_0xffffd876f2e1(0xffffd876f2e0, 0xffffd876f1f8, 0x1, 0, 0x80000000c93e, 0x50) = 0

29 syscall_0xffffd876f2e1(0xffffd876f2e0, 0xffffd876f1f8, 0x1, 0, 0xf00000000000, 0xde) = 0xf00004d3d260

29 write(0, 0x1, 15680) = 281474313422324

29 syscall_0xf00004d3d020(0xf00004d3d800, 0xf00004d3d800, 0xf00004d3d890, 0, 0xffffd87703a0, 0xde) = 0xffffd87705f4

29 write(80992400, "\1\0\0\0\0\0\0@\321\323\4\0\360\0\0 \331\323\4\0\360\0\0\0\330\323\4\0\360\0\0"..., 263882871656768) = 281474313422324

29 syscall_0x555555556000(0x555555555000, 0xf00004d3d9b0, 0xf00004d3d890, 0, 0xffffd87703a0, 0xde) = 0xffffd87705f4

29 syscall_0x555555557000(0x555555556000, 0xf00004d3da40, 0xf00004d3d9b0, 0, 0xffffd87703a0, 0xde) = 0xffffd87705f4

29 syscall_0xf00004d3da40(0xf00004d3d020, 0xf00004d3d020, 0xf00004d3da40, 0, 0xffffd87703a0, 0xd7) = 0xffffd87705f4

29 read(0, NULL, 0) = 0

29 read(0, NULL, 0) = 0

29 read(0, NULL, 0) = 0

29 read(0, NULL, 0) = 263882871658352

29 write(0, NULL, 230944) = 281474313422324

29 syscall_0xf00004d3d800(0xf00004d3d890, 0xf00004d3d890, 0xf00004d3dad0, 0, 0xffffd87703a0, 0xde) = 0xffffd87705f4

29 write(80993120, "\1\0\0\0\0\0\0@\321\323\4\0\360\0\0\0\330\323\4\0\360\0\0\220\330\323\4\0\360\0\0"..., 263882871656768) = 281474313422324

29 syscall_0xf00004d3dbf0(0xf00004d3dad0, 0xf00004d3dad0, 0xf00004d3dbf0, 0, 0xffffd87703a0, 0xde) = 0xffffd87705f4

29 syscall_0x7fffffffb000(0x7fffffff0000, 0xf00004d3dc80, 0xf00004d3d890, 0, 0xffffd87703a0, 0xde) = 0xffffd87705f4

29 syscall_0x800000000000(0x7fffffffc000, 0xf00004d3dd10, 0xf00004d3d890, 0, 0xffffd87703a0, 0xd7) = 0xffffd87705f4

29 syscall_0x555555554000(0, 0, 0xffffd876f358, 0, 0x2af0, 0x39) = 0x1

29 syscall_0x555555554000(0, 0, 0xffffd876f358, 0, 0x7, 0xa3) = 0x1

29 syscall_0x555555554000(0, 0, 0xffffd876f358, 0, 0x800000, 0x38) = 0x1

29 syscall_0x555555554000(0, 0, 0xffffd876f358, 0, 0x800000, 0x3f) = 0x1

29 syscall_0x555555554000(0, 0, 0xffffd876f358, 0, 0x800000, 0x39) = 0x1

29 syscall_0x555555554000(0, 0, 0xffffd876f358, 0, 0x800000, 0x116) = 0x1

29 syscall_0x555555554000(0, 0, 0xffffd876f358, 0, 0xffffd87703a0, 0xde) = 0xffffd87705f4

29 write(80992112, "\1\0\0\0\0\0\0\0\360\333\323\4\0\360\0\0\0\330\323\4\0\360\0\0\320\332\323\4\0\360\0\0"..., 263882871659504) = 281474313422324

29 syscall_0xf00004d3d800(0xf00004d3d770, 0xf00004d3d770, 0xf00004d3dda0, 0, 0x800000195000, 0xa0) = 0x800000195000

29 poll(0xcccccccccccccccc, 10, 10) = 5 ([0xcccccccccccccccc])

29 syscall_0x80000007d370(0x1, 0xffffffffffffffff, 0xffffd8771970, 0, 0xf00000000000, 0xde) = 0xf00004d3d260

29 write(0, "\1\0\0\0\0\0\0`\322\323\4\0\360\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 263882871657056) = 263882871657056

29 syscall_0xffffffffffffffff(0xd503201f000001b1, 0xfffffe00000001b0, 0xfffffe00000001af, 0, 0xeffff7ff8000, 0xde) = 0xf00004d3d260

29 syscall_0xf00004d3d2f0(0xf00004d3dec0, 0xf00004d3dec0, 0xeffff7ff7ec0, 0, 0xeffff7ff7000, 0xde) = 0xf00004d3d260

29 syscall_0x7ffffffe4290(0, 0xf00004d3df50, 0xffffffff, 0, 0xeffff7ff6000, 0xde) = 0xf00004d3d260

29 read(0, 0x1, 0) = 263882871657056

29 write(0, "\1\0\0\0\0\0\0`\322\323\4\0\360\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 263882871657056) = 263882871657056

29 write(0, "\1\0\0\0\0\0\0`\322\323\4\0\360\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 263882871657056) = 281474313422324

29 write(80994128, "\1\0\0\0\0\0\0\20\324\323\4\0\360\0\0\360\322\323\4\0\360\0\0\300\336\323\4\0\360\0\0"..., 263882871657488) = 263882871660368

29 syscall_0xf00004d3d2f0(0xf00004d3c020, 0xf00004d3c020, 0xf00004d3df50, 0, 0xffffd87703a0, 0xe2) = 0xffffd87705f4

29 write(80986288, "\1\0\0\0\0\0\0`\322\323\4\0\360\0\0\360\322\323\4\0\360\0\0\300\323\4\0\360\0\0"..., 263882871657056) = 32458

29 write(80986288, "\1\0\0\0\0\0\0`\322\323\4\0\360\0\0\360\322\323\4\0\360\0\0\300\323\4\0\360\0\0"..., 263882871657056) = 32458

29 write(80986288, "\1\0\0\0\0\0\0`\322\323\4\0\360\0\0\360\322\323\4\0\360\0\0\300\323\4\0\360\0\0"..., 263882871657056) = 32458

29 write(80986288, "\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\0\0\0\0\0\0\0\0\0"..., 140737489997824) = 140737489997824

29 write(80986288, NULL, 0) = 0

29 write(80986288, NULL, 0) = 0

29 write(80986288, NULL, 0) = 0

29 syscall_0x24a7f2879353fd7(0xa1b4d1e6625697e9, 0xeb4b4a1e4e5adaa7, 0xf2e9a9fcb1f4d4fd, 0, 0xeffff7de7000, 0xde) = 0xf00004d3c0b0

29 syscall_0x1696a52e52d4da9(0x6b5a5b534a5a96b1, 0xd6b536a52e52d4a5, 0x25a67fb76adb54ff, 0, 0xeffff7de6000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7dfca40(0xeffff7dfca42, 0x10204, 0, 0, 0xeffff7de5000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7dfb140(0xeffff7dfb142, 0x10204, 0, 0, 0xeffff7de4000, 0xde) = 0xf00004d3c0b0

29 syscall_0x7fffffff2e0(0, 0, 0x6fffff41, 0, 0xefffef5, 0xa0) = 0x32

29 syscall_0xeffff7dfca00(0xeffff7dfca20, 0x10204, 0, 0, 0xeffff7de3000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7df4d01(0xeffff7df4d00, 0xeffff7df4c18, 0x2, 0, 0x8000001905c8, 0x30) = 0

29 syscall_0xeffff7dfa920(0xeffff7dfa922, 0x10204, 0, 0, 0xeffff7de2000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7dfb710(0xeffff7dfb71e, 0x10204, 0, 0, 0xeffff7de1000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7df48e1(0xeffff7df48e0, 0xeffff7df47f8, 0x2, 0, 0x8000effff7df4f48, 0x38) = 0

29 syscall_0xeffff7df4c81(0xeffff7df4c80, 0xeffff7df4b98, 0x2, 0, 0x8000001905c8, 0x4f) = 0

29 syscall_0x5555555546a4(0xa, 0x1, 0xffffffffffffffff, 0, 0x7fffff7c3000, 0xde) = 0xf00004d3dda0

29 syscall_0x5555555546a4(0xa, 0x1, 0x7fffff7c0000, 0, 0, 0x39) = 0x2

29 syscall_0xeffff7df48e1(0xeffff7df48e0, 0xeffff7df47f8, 0x2, 0, 0x8000effff7df4f48, 0x38) = 0

29 munmap(0x7fffffffaf0, 140737488103472) = 0

29 munmap(0x7fffffffaf0, 140737488103472) = 64

29 munmap(0x7fffffffaf0, 140737488103472) = 848

29 munmap(0x7fffffffaf0, 140737488103472) = 896

29 syscall_0xeffff7df4c81(0xeffff7df4c80, 0xeffff7df4b98, 0x2, 0, 0x8000001905c8, 0x4f) = 0

29 syscall_0xeffff7dfb830(0xeffff7dfb832, 0x10204, 0, 0, 0xeffff7de0000, 0xde) = 0xf00004d3c0b0

29 syscall_0x7fffffffb10(0xa, 0x5555555546a4, 0x7fffffff2e0, 0, 0x7fffffffff98, 0x43) = 0x40

29 syscall_0x7fffffffb10(0x1000, 0, 0x4, 0, 0x7fffff7c0000, 0xde) = 0xf00004d3dda0

29 syscall_0x7fffffffb10(0x7fffff597000, 0, 0x4, 0, 0, 0xe2) = 0

29 syscall_0x7fffffffb10(0x7fffff597000, 0x7ffffffc2038, 0x4, 0, 0, 0xde) = 0

29 syscall_0x7fffffffb10(0x7fffff597000, 0x7ffffffc2070, 0x4, 0, 0, 0xde) = 0

29 syscall_0x7fffffffb10(0x7fffff597000, 0x7ffffffc20a8, 0x4, 0, 0, 0xde) = 0

29 syscall_0x7fffffffb10(0x7fffff7b3000, 0x7ffffffc20a8, 0x4, 0, 0, 0xde) = 0

29 syscall_0x7fffffffb10(0xc0000002, 0x7fffff597380, 0x7fffff597040, 0, 0x30, 0x39) = 0xffff800000a68cbc

29 syscall_0xeffff7dfa7a0(0xeffff7dfa7a2, 0x10204, 0, 0, 0xeffff7ddf000, 0xde) = 0xf00004d3c0b0

29 syscall_0x7fffffffb10(0x1, 0x7fffff5b53b9, 0x7fffff7b0d40, 0, 0x7fffff597000, 0xde) = 0xf00004d3dda0

29 syscall_0x92ca7fadab0d8fb(0x75a72d7197fc36b6, 0xeb4d4ff52c94ff52, 0xedd2dbfffffb15ff, 0, 0xeffff7dde000, 0xde) = 0xf00004d3c0b0

29 syscall_0x7fffffff2e0(0, 0, 0x7fffff596140, 0, 0x90, 0x60) = 0x1

29 syscall_0x7fffffff2e0(0, 0, 0x7fffff596140, 0, 0x90, 0x63) = 0x1

29 syscall_0xeffff7dfa7d0(0xeffff7dfa7d2, 0x10204, 0, 0, 0xeffff7ddd000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7dfb5b0(0xeffff7dfb5c0, 0x10204, 0, 0, 0xeffff7ddc000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7dfbfe0(0xeffff7dfbff8, 0x10204, 0, 0, 0xeffff7ddb000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7dfbe70(0xeffff7dfbe8a, 0x10204, 0, 0, 0xeffff7dda000, 0xde) = 0xf00004d3c0b0

29 syscall_0x7fffff5befc8(0x7fffffffb10, 0x7fffffffb10, 0x7fffff597000, 0, 0, 0xe2) = 0

29 syscall_0xeffff7dfca30(0xeffff7dfca36, 0x10204, 0, 0, 0xeffff7dd9000, 0xde) = 0xf00004d3c0b0

29 syscall_0x7fffffff2e0(0x7fffffff2e0, 0x5555555545a0, 0, 0, 0, 0xe2) = 0

29 syscall_0xeffff7dfada0(0xeffff7dfada8, 0x10204, 0, 0, 0xeffff7dd8000, 0xde) = 0xf00004d3c0b0

29 syscall_0x7fffffffeaf0(0x7fffffffeaf0, 0, 0, 0, 0, 0xe2) = 0

29 syscall_0x7fffffff2e0(0x7fffff596140, 0xffffffffffffffff8, 0x7fffffff2e0, 0, 0xf, 0x105) = 0x7ffffffc3210

29 syscall_0xeffff7dfaf10(0xeffff7dfaf18, 0x10204, 0, 0, 0xeffff7dd7000, 0xde) = 0xf00004d3c0b0

29 syscall_0x7fffffff2e0(0x7fffff596140, 0xffffffffffffffff8, 0x7fffffff2e0, 0, 0, 0xd7) = 0

29 syscall_0xeffff7dfa800(0xeffff7dfa802, 0x500010204, 0, 0, 0xeffff7dd6000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7dfbf00(0xeffff7dfbf16, 0x10204, 0, 0, 0xeffff7dd5000, 0xde) = 0xf00004d3c0b0

29 syscall_0x7fffffffe040(0x555555557d48, 0x55555555604a, 0x7ffffffc3678, 0, 0x7ffffffca040, 0x116) = 0x7ffffffc4908

29 syscall_0x2b0(0x1000, 0x7fffff7b1ce0, 0x555555559000, 0, 0x555555559000, 0xa3) = 0x7fffff7b1ce0

29 syscall_0x2b0(0x1000, 0x7fffff7b1ce0, 0x555555559000, 0, 0, 0xde) = 0x7fffff7b1ce0

29 syscall_0xeffff7dfabd0(0xeffff7dfabd2, 0x10204, 0, 0, 0xeffff7dd4000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7df48e1(0xeffff7df48e0, 0xeffff7df47f8, 0x2, 0, 0x8000effff7df4f48, 0x38) = 0

29 syscall_0xeffff7df48e1(0xeffff7df48e0, 0xeffff7df47f8, 0x2, 0, 0x8000effff7df4f48, 0x38) = 0

29 syscall_0x7fffffffe040(0x555555557d48, 0x55555555550e, 0x7ffffffc3678, 0, 0x7fffff7c3000, 0xde) = 0xf00004d3da40

29 syscall_0x7fffffffe040(0x555555557d48, 0x55555555550e, 0x7ffffffc3678, 0, 0, 0x39) = 0x1

29 syscall_0xeffff7dfa820(0xeffff7dfa822, 0x700010204, 0, 0, 0xeffff7dd3000, 0xde) = 0xf00004d3c0b0

29 syscall_0x7fffffffe040(0x555555557d48, 0x55555555550e, 0x7ffffffc3678, 0, 0x1, 0x86) = 0x8

29 read(0, NULL, 0) = 140737488875908

29 syscall_0x7fffffffe040(0x555555557d48, 0x55555555550e, 0x7ffffffc3678, 0, 0x1, 0x86) = 0x8

29 read(0, NULL, 0) = 140737488875908

29 syscall_0x7fffffffe040(0x555555557d48, 0x55555555550e, 0x7ffffffc3678, 0, 0, 0x49) = 0x8

29 --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=28, si_uid=0} ---

29 syscall_0x7fffffffe040(0x555555557d48, 0x55555555550e, 0x6, 0, 0xeffff7dd2000, 0xde) = 0xf00004d3c0b0

29 syscall_0xeffff7dd26e0(0xeffff7dd2920, 0xeffff7dd2958, 0, 0, 0x800000191ef0, 0x87) = 0x800000192110

29 syscall_0xeffff7dfd860(0xeffff7dfd866, 0x10204, 0, 0, 0xeffff7dd0000, 0xde) = 0xf00004d3c0b0

29 --- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0xfffffffffffc1010} ---

29 syscall_0x7fffffffe040(0x7ffffffc0e40, 0xfffffffffffc1010, 0x4, 0, 0x800000192440, 0x86) = 0x90000040

29 syscall_0x7fffffffe040(0x7ffffffc0e40, 0xfffffffffffc1010, 0x4, 0, 0x800000192440, 0xf0) = 0x90000040

29 syscall_0x7fffffffe040(0x7ffffffc0e40, 0xfffffffffffc1010, 0x4, 0, 0x800000192440, 0x8b) = 0x1000

29 --- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0xfffffffffffc1010} ---

29 +++ killed by SIGSEGV +++

28 +++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы была реализована система взаимодействия двух процессов — родительского и дочернего — с использованием механизмов операционной системы Linux: сигналов, разделяемой памяти и системных вызовов работы с файлами. Родительский процесс запрашивает у пользователя строки с числами и передает их дочернему процессу через общий участок памяти, после чего с помощью сигнала уведомляет дочерний процесс о необходимости обработки данных. Дочерний процесс принимает сигнал, считывает строку из разделяемой памяти, вычисляет сумму чисел и дописывает результат в выходной файл.

Для анализа работы программы использовался инструмент **strace**, позволивший подробно проследить системные вызовы обоих процессов. Анализ trace-лога подтвердил корректность реализации механизмов межпроцессного взаимодействия: родитель действительно создаёт файл, выделяет разделяемую память, передаёт данные и будит дочерний процесс с помощью kill(), а дочерний процесс правильно ждёт сигнал, обрабатывает данные и записывает результат. Завершение процессов происходит корректно: родитель ожидает дочерний процесс через waitpid(), дочерний завершается через exit().

В результате работы было изучено практическое применение следующих механизмов ОС Linux:

- создание процессов (fork);
- обработка сигналов (sigaction, kill);

- синхронное ожидание событий (sigsuspend);
- работа с разделяемой памятью на базе mmap-файла (open, ftruncate, mmap);
- работа с файлами (open, write, close);
- наблюдение за системными вызовами через strace.

Выполнение лабораторной работы позволило получить практические навыки разработки приложений, использующих сигналы и разделяемую память для организации обмена данными между процессами, а также закрепить понимание того, как подобные механизмы выглядят внутри ядра операционной системы. Программа работает корректно, а результаты strace подтверждают правильность логики и последовательности системных вызовов.