Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-213Б-23

Студент: Османова В.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 23.11.24

Москва, 2024

Постановка задачи

Вариант 6.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода.

Дочерний процесс передаёт данные родительскому с помощью shared memory. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork() - создание дочернего процесса
- int execve(const char *filename, char *const argv[], char *const envp[]) (и другие вариации exec) - замена образа памяти процесса
- pid_t waitpid(pid_t pid, int *status, int options) - Ожидание завершения дочернего процесса
- void exit(int status) - завершения выполнения процесса и возвращение статуса
- int shm_open(const char *name, int oflag, mode_t mode) – создание/открытие именованной области разделенной памяти
- int shm_unlink(const char *name) – уничтожение именованной области памяти
- int ftruncate(int fd, off_t length) – устанавливает размер файла
- mmap
- sem_t *sem_open(const char *, int, ...) – создание/открытие именованного семефора
- int sem_wait(sem_t *) – уменьшить семафор (ожидание если 0)
- int sem_post(sem_t *) – увеличить семафор
- int sem_unlink(const char *) – уничтожить именованный семафор
- int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
- int open(const char *pathname, int flags, mode_t mode) – открытие/создание файла
- int close(int fd) - закрыть файл

Программа считывает из стандартного ввода имя файла, открывает его функцией open, создает shared memory и 2 семафора: full и empty. Shared memory устанавливается размер достаточный для хранения 2 значений типа int: одно передаваемое число, второе флаг конца файла. Далее порождается дочерний процесс, у которого с помощью функции dup2 стандартный ввод перенаправлен на открытый ранее файл. Образ дочернего процесса заменяется на child.out с помощью функции execv.

Родительский процесс считывает значения типа int по индексу 0 из shared memory, пока дочерний процесс не обнулит значение по индексу 1, каждое значение выводится в стандартный вывод с помощью функции print_int, в конце родительский процесс ждёт завершения дочернего и завершается.

Дочерний процесс считывает данные из стандартного ввода (перенаправленного на файл) с помощью функции read_line_of_int, которая считывает целые числа, разделённые пробелом, пока не достигнет конца строки, после чего записывает сумму чисел в переменную адрес, которой передаётся в качестве аргумента, и возвращает 1. Если строка была пустой возвращается 0.

Результат работы read_line_of_int записывается по индексу 0 в shared memory, пока функция не вернёт 0. После этого по индексу 1 записывается 0 и процесс завершается.

Код программы

main.c

```
#define _XOPEN_SOURCE 900
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <wait.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <semaphore.h>

int print_int(const int num) {
    char buf[16];
    char res[32];
    int n = 0;
    int sign = (num < 0);
    int x = abs(num);
    if (x == 0) {
        write(STDOUT_FILENO, "0\n", 2);
        return 0;
    }
    while (x) {
        buf[n] = (x % 10) + '0';
        x = x / 10;
        n++;
    }
    if (sign) {
        res[0] = '-';
    }
    for (int i = 0; i < n; i++) {
        res[i + sign] = buf[n - i - 1];
    }
    res[n + sign] = '\n';
    write(STDOUT_FILENO, res, (n + sign + 1));
    return 0;
}

int read_line(char** buf, int* n){
    char c;
    int i = 0;
    while(1) {
        if (read(STDIN_FILENO, &c, sizeof(char)) == -1) {
            return -1;
        }
        (*buf)[i] = c;
        i++;
        if (i >= *n) {
            *buf = realloc(*buf, (*n) * 2 * sizeof(char));
            *n = *n * 2;
        }
        if (c == '\n') {
            (*buf)[i - 1] = '\0';
            break;
        }
    }
```

```
    return 0;
}

int main() {
    char* buf = malloc(128 * sizeof(char));
    int n = 128;
    if (read_line(&buf, &n) == -1) {
        char* msg = "fail to read file name\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }

    int file_fd = open(buf, O_RDONLY);
    if (file_fd == -1) {
        char* msg = "fail to open file\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }
    free(buf);

    int shm = shm_open("/memory", O_RDWR | O_CREAT, 0666);
    if (shm == -1) {
        char* msg = "fail to create shared memory\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }

    if (ftruncate(shm, sizeof(int) * 2) == -1) {
        char* msg = "fail to set size of shared memory\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    };

    sem_t* sem_empty = sem_open("/semaphore_empty", O_CREAT, 0666, 1);
    if (sem_empty == SEM_FAILED) {
        char* msg = "fail to create semaphore\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }

    sem_t* sem_full = sem_open("/semaphore_full", O_CREAT, 0666, 0);
    if (sem_full == SEM_FAILED) {
        char* msg = "fail to create semaphore\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }

    __pid_t pid = fork();
    if (pid == -1) {
        char* msg = "fail to fork\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }

    if (pid == 0) {
        if (dup2(file_fd, STDIN_FILENO) == -1) {
            char* msg = "fail to reassign file descriptor\n";
            write(STDOUT_FILENO, msg, strlen(msg));
            exit(-1);
        }
        close(file_fd);
        char *arg[] = {"./child.out", NULL};
        if (execv("./child.out", arg) == -1) {
            char* msg = "fail to replace process image\n";
```

```
            write(STDOUT_FILENO, msg, strlen(msg));
            exit(-1);
        }
    } else {
        char c;
        char* ptr = mmap(0, sizeof(int) * 2, PROT_READ|PROT_WRITE, MAP_SHARED, shm, 0);
        if (ptr == (char*)-1) {
            char* msg = "fail to memory map\n";
            write(STDOUT_FILENO, msg, strlen(msg));
            exit(-1);
        }
        ptr[1] = 1;
        while (ptr[1]) {
            sem_wait(sem_full);
            print_int(ptr[0]);
            sem_post(sem_empty);
        }
        waitpid(pid, 0 , 0);
        shm_unlink("/memory");
    }
    return 0;
}
```

child.c

```
#define _XOPEN_SOURCE 900
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <wait.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <semaphore.h>

int is_num(char c) {
    return (c >= '0') && (c <= '9');
}

int read_line_of_int(int* res) {
    char c = 0;
    int i = 0, k = 0, sign = 1, r;
    int flag = 1;
    int buf = 0;
    int sum = 0;
    while (c != '\n') {
        while(1) {
            r = read(STDIN_FILENO, &c, sizeof(char));
            if (r < 1) {
                return r;
            }
            if ((c == '\n') || (c == ' ')) {
                break;
            }
            if (flag && (c == '-')) {
                sign = -1;
            } else if (!is_num(c)) {
                return -1;
            } else {
                buf = buf * 10 + c - '0';
                i++;
```

```
                if (flag) {
                    k++;
                    flag = 0;
                }
            }
        }
        sum = sum + buf * sign;
        buf = 0;
        flag = 1;
        sign = 1;
    }
    if (k == 0) {
        return 0;
    }
    *res = sum;
    return 1;
}

int main() {
    int shm = shm_open("/memory", O_RDWR, 0666);
    if (shm == -1) {
        char* msg = "fail to open shared memory\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }

    sem_t* sem_empty = sem_open("/semaphore_empty", O_EXCL);
    if (sem_empty == SEM_FAILED) {
        char* msg = "fail to open semaphore\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }

    sem_t* sem_full = sem_open("/semaphore_full", O_EXCL);
    if (sem_full == SEM_FAILED) {
        char* msg = "fail to open semaphore\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }
    char* ptr = mmap(0, sizeof(int) * 2, PROT_READ|PROT_WRITE, MAP_SHARED, shm,
0);
    int res = 0;
    while (read_line_of_int(&res) > 0) {
        sem_wait(sem_empty);
        ptr[0] = res;
        sem_post(sem_full);
    }
    ptr[1] = 0;
    close(STDOUT_FILENO);
    return 0;
}
```

Протокол работы программы

Тестирование:

```
$ cat > a.txt
1 1 1
0 0
```

```
1 -2 -3
20 -14
1 2 3 4 5
$ cat > run.txt
a.txt
$ ./main.out < run.txt
3
0
-4
6
15
```

Strace:

```
$ strace -f ./main.out < run.txt
execve("./main.out", ["./main.out"], 0x7ffcab9cfca8 /* 57 vars */) = 0
brk(NULL)                               = 0x563e8de74000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe28e77ef0) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3a802d6000
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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=61763, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 61763, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3a802c6000
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\0P<\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\0\0\0"..., 784, 64) = 784
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2072888, ...}, AT_EMPTY_PATH) = 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, 2117488, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f3a80000000
mmap(0x7f3a80022000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000)
= 0x7f3a80022000
mmap(0x7f3a8019a000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) =
0x7f3a8019a000
mmap(0x7f3a801f2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1f1000)
= 0x7f3a801f2000
mmap(0x7f3a801f8000, 53104, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
0x7f3a801f8000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3a802c3000
arch_prctl(ARCH_SET_FS, 0x7f3a802c3740) = 0
set_tid_address(0x7f3a802c3a10)         = 5824
set_robust_list(0x7f3a802c3a20, 24)     = 0
rseq(0x7f3a802c4060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f3a801f2000, 16384, PROT_READ) = 0
mprotect(0x563e8bee9000, 4096, PROT_READ) = 0
mprotect(0x7f3a8030b000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f3a802c6000, 61763)           = 0
getrandom("\xdf\x6a\x4d\x02\x9b\xd4\xb3\x79", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x563e8de74000
brk(0x563e8de95000)                     = 0x563e8de95000
read(0, "a", 1)                         = 1
read(0, ".", 1)                         = 1
read(0, "t", 1)                         = 1
read(0, "x", 1)                         = 1
read(0, "t", 1)                         = 1
read(0, "\n", 1)                        = 1
openat(AT_FDCWD, "a.txt", O_RDONLY)     = 3
openat(AT_FDCWD, "/dev/shm/memory", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 4
ftruncate(4, 8)                         = 0
openat(AT_FDCWD, "/dev/shm/sem.semaphore_empty", O_RDWR|O_NOFOLLOW) = 5
newfstatat(5, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7f3a802d5000
```

```
close(5)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.semaphore_full", O_RDWR|O_NOFOLLOW) = 5
newfstatat(5, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7f3a802d4000
close(5)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 5825 attached
, child_tidptr=0x7f3a802c3a10) = 5825
[pid  5825] set_robust_list(0x7f3a802c3a20, 24 <unfinished ...>
[pid  5824] mmap(NULL, 8, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0 <unfinished ...>
[pid  5825] <... set_robust_list resumed>) = 0
[pid  5825] dup2(3, 0)                  = 0
[pid  5825] close(3)                    = 0
[pid  5824] <... mmap resumed>)         = 0x7f3a802d3000
[pid  5825] execve("./child.out", ["./child.out"], 0x7ffe28e78058 /* 57 vars */ <unfinished ...>
[pid  5824] futex(0x7f3a802d4000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY
<unfinished ...>
[pid  5825] <... execve resumed>)       = 0
[pid  5825] brk(NULL)                   = 0x562ab810c000
[pid  5825] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdddef1220) = -1 EINVAL (Invalid argument)
[pid  5825] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f05ed85f000
[pid  5825] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
[pid  5825] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
[pid  5825] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=61763, ...}, AT_EMPTY_PATH) = 0
[pid  5825] mmap(NULL, 61763, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05ed84f000
[pid  5825] close(3)                    = 0
[pid  5825] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
[pid  5825] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P<\2\0\0\0\0\0"..., 832) = 832
[pid  5825] 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  5825] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2072888, ...}, AT_EMPTY_PATH) = 0
[pid  5825] 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  5825] mmap(NULL, 2117488, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f05ed600000
[pid  5825] mmap(0x7f05ed622000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x22000) = 0x7f05ed622000
[pid  5825] mmap(0x7f05ed79a000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) =
0x7f05ed79a000
[pid  5825] mmap(0x7f05ed7f2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1f1000) = 0x7f05ed7f2000
[pid  5825] mmap(0x7f05ed7f8000, 53104, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -
1, 0) = 0x7f05ed7f8000
[pid  5825] close(3)                    = 0
[pid  5825] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f05ed84c000
[pid  5825] arch_prctl(ARCH_SET_FS, 0x7f05ed84c740) = 0
[pid  5825] set_tid_address(0x7f05ed84ca10) = 5825
[pid  5825] set_robust_list(0x7f05ed84ca20, 24) = 0
[pid  5825] rseq(0x7f05ed84d060, 0x20, 0, 0x53053053) = 0
[pid  5825] mprotect(0x7f05ed7f2000, 16384, PROT_READ) = 0
[pid  5825] mprotect(0x562ab6e1d000, 4096, PROT_READ) = 0
[pid  5825] mprotect(0x7f05ed894000, 8192, PROT_READ) = 0
[pid  5825] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
[pid  5825] munmap(0x7f05ed84f000, 61763) = 0
[pid  5825] openat(AT_FDCWD, "/dev/shm/memory", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3
[pid  5825] openat(AT_FDCWD, "/dev/shm/sem.semaphore_empty", O_RDWR|O_EXCL|O_NOFOLLOW) = 4
[pid  5825] newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0
[pid  5825] getrandom("\xc4\x7a\x23\x0d\x3d\x9f\x73\x80", 8, GRND_NONBLOCK) = 8
[pid  5825] brk(NULL)                   = 0x562ab810c000
[pid  5825] brk(0x562ab812d000)         = 0x562ab812d000
[pid  5825] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f05ed85e000
[pid  5825] close(4)                    = 0
[pid  5825] openat(AT_FDCWD, "/dev/shm/sem.semaphore_full", O_RDWR|O_EXCL|O_NOFOLLOW) = 4
[pid  5825] newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0
[pid  5825] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f05ed85d000
[pid  5825] close(4)                    = 0
[pid  5825] mmap(NULL, 8, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f05ed85c000
```

```
[pid  5825] read(0, "1", 1)           = 1
[pid  5825] read(0, " ", 1)           = 1
[pid  5825] read(0, "1", 1)           = 1
[pid  5825] read(0, " ", 1)           = 1
[pid  5825] read(0, "1", 1)           = 1
[pid  5825] read(0, "\n", 1)          = 1
[pid  5825] futex(0x7f05ed85d000, FUTEX_WAKE, 1) = 1
[pid  5824] <... futex resumed>)      = 0
[pid  5825] read(0, "0", 1)           = 1
[pid  5824] write(1, "3\n", 2 <unfinished ...>
[pid  5825] read(0, " ", 1)           = 1
3
[pid  5824] <... write resumed>)      = 2
[pid  5825] read(0, "0", 1)           = 1
[pid  5824] futex(0x7f3a802d4000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY
<unfinished ...>
[pid  5825] read(0, "\n", 1)          = 1
[pid  5825] futex(0x7f05ed85d000, FUTEX_WAKE, 1) = 1
[pid  5824] <... futex resumed>)      = 0
[pid  5825] read(0,  <unfinished ...>
[pid  5824] write(1, "0\n", 2 <unfinished ...>
0
[pid  5825] <... read resumed>"1", 1)  = 1
[pid  5824] <... write resumed>)      = 2
[pid  5824] futex(0x7f3a802d4000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY
<unfinished ...>
[pid  5825] read(0, " ", 1)           = 1
[pid  5825] read(0, "-", 1)           = 1
[pid  5825] read(0, "2", 1)           = 1
[pid  5825] read(0, " ", 1)           = 1
[pid  5825] read(0, "-", 1)           = 1
[pid  5825] read(0, "3", 1)           = 1
[pid  5825] read(0, "\n", 1)          = 1
[pid  5825] futex(0x7f05ed85d000, FUTEX_WAKE, 1) = 1
[pid  5824] <... futex resumed>)      = 0
[pid  5825] read(0,  <unfinished ...>
[pid  5824] write(1, "-4\n", 3 <unfinished ...>
-4
[pid  5825] <... read resumed>"2", 1)  = 1
[pid  5824] <... write resumed>)      = 3
[pid  5825] read(0,  <unfinished ...>
[pid  5824] futex(0x7f3a802d4000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY
<unfinished ...>
[pid  5825] <... read resumed>"0", 1)  = 1
[pid  5825] read(0, " ", 1)           = 1
[pid  5825] read(0, "-", 1)           = 1
[pid  5825] read(0, "1", 1)           = 1
[pid  5825] read(0, "4", 1)           = 1
[pid  5825] read(0, "\n", 1)          = 1
[pid  5825] futex(0x7f05ed85d000, FUTEX_WAKE, 1) = 1
[pid  5824] <... futex resumed>)      = 0
[pid  5825] read(0,  <unfinished ...>
[pid  5824] write(1, "6\n", 2 <unfinished ...>
[pid  5825] <... read resumed>"1", 1)  = 1
6
[pid  5824] <... write resumed>)      = 2
[pid  5825] read(0,  <unfinished ...>
[pid  5824] futex(0x7f3a802d4000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY
<unfinished ...>
[pid  5825] <... read resumed>" ", 1)  = 1
[pid  5825] read(0, "2", 1)           = 1
[pid  5825] read(0, " ", 1)           = 1
[pid  5825] read(0, "3", 1)           = 1
```

```
[pid  5825] read(0, " ", 1)           = 1
[pid  5825] read(0, "4", 1)           = 1
[pid  5825] read(0, " ", 1)           = 1
[pid  5825] read(0, "5", 1)           = 1
[pid  5825] read(0, "\n", 1)          = 1
[pid  5825] futex(0x7f05ed85d000, FUTEX_WAKE, 1) = 1
[pid  5824] <... futex resumed>)      = 0
[pid  5825] read(0,  <unfinished ...>
[pid  5824] write(1, "15\n", 3 <unfinished ...>
[pid  5825] <... read resumed>"", 1)    = 0
15
[pid  5824] <... write resumed>)      = 3
[pid  5825] close(1 <unfinished ...>
[pid  5824] wait4(5825,  <unfinished ...>
[pid  5825] <... close resumed>)      = 0
[pid  5825] exit_group(0)             = ?
[pid  5825] +++ exited with 0 +++
<... wait4 resumed>NULL, 0, NULL)     = 5825
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=5825, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
unlink("/dev/shm/memory")             = 0
exit_group(0)                         = ?
+++ exited with 0 ++++++ exited with 0 +++
```

Вывод

Создание этой программы потребовало глубокого понимания взаимодействия между родительским и дочерним процессами на языке C. Необходимо было тщательно разработать механизм обмена данными с использованием общей памяти и семафоров, а также реализовать эффективные функции для работы с целыми числами. Важно было уделить особое внимание обработке возможных ошибок на каждом этапе выполнения программы, чтобы обеспечить надежность и устойчивость. В конечном итоге была создана программа, демонстрирующая высокое понимание ключевых концепций межпроцессного взаимодействия и обработки данных в системном программировании на C.