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-213Б-23

Студент: Аксельрод А.М.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 26.12.24

Москва, 2024

Постановка задачи

Вариант 10.

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. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число ». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- int open(const char *pathname, int flags); Открывает или создаёт файл по пути pathname с флагами flags.
- int close(int fd); Закрывает файловый дескриптор fd.
- pid_t fork(void); Создаёт дочерний процесс.
- int dup2(int oldfd, int newfd); Создаёт дубликат файлового дескриптора oldfd newfd.
- int execv(const char *path, char *const argv[]); Заменяет текущий образ процесса новым, загружая исполняемый файл по пути path с аргументами argv.
- pid_t waitpid(pid_t pid, int *status, int options); Приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс pid не совершит выполнение, и возвращает его статус в status.
- void exit(int status); Завершает процесс с кодом завершения status
- int shm_open(const char *name, int oflag, mode_t mode); Создаёт и открывает объект разделённой памяти.
- int shm_unlink(const char *name); Удаляет имя объекта разделяемой памяти и очищает память.
- int ftruncate(int fd, off_t length); Устанавливает длину файла
- mmap(void* start, size_t length, int prot, int flags, int fd, off_t offset); Отражает length байтов, начиная со смещения offset файла в память, начиная с адреса start.

- sem_t *sem_open(const char *name, int, ...); Создаёт и открывает семефор.
- int sem_wait(sem_t *sem); Уменьшает семафор.
- int sem_post(sem_t * sem); Увеличивает семафор.
- int sem_unlink(const char * sem); Удаляет семафор.

Программа считывает имя файла из стандартного ввода, открывает его функцией open. Создаёт разделённую память (shared memory) и два семафора. Разделённая память вмещает два значения int: число и флаг. Потом создаётся дочерний процесс с помощью fork. Дочерний процесс перенаправляет стандартный ввод на открытый файл с помощью dup2 и перенаправляет стандартный вывод на конец записи канала, чтобы отправлять данные обратно родительскому процессу. Выполняет исполняемый файл child.out с помощью execv.

Родительский процесс считывает из ячейки 0 разделённой памяти числа, отправленные дочерним процессом, и передаёт их функции print_int для вывода на экран. Если флаг станет 0, он завершится. Он ожидает завершения дочернего процесса с помощью waitpid.

Дочерний процесс считывает числа из перенаправленного стандартного ввода (файла, открытого родительским процессом) с помощью функции process_line.

- Если число составное, записывает его в стандартный вывод (который перенаправлен в канал).
- Если число отрицательное, простое или некорректное, завершает работу с кодом ошибки (сюда же относится 0).

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* 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);
        }

        memset(ptr, 0, sizeof(int) * 2);
        ptr[1] = 1;
        while (ptr[1] != 0) {
            sem_wait(sem_full);
            print_int(ptr[0]);
            sem_post(sem_empty);
        }

        waitpid(pid, 0, 0);
        shm_unlink("/memory");
    }
    return 0;
}
```

child.c

```
#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>

#define BUFFER_SIZE 512
#define _XOPEN_SOURCE 900
```

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

int is_space(char c) {
    return (c == ' ') || (c == '\t') || (c == '\n');
}

int is_prime(int num) {
    if (num < 2) {
        return 0;
    }

    for (int i = 2; i * i <= num; i++){
        if (num % i == 0) {
            return 0;
        }
    }
    return 1;
}

int process_line(int* res) {
    char buffer[BUFFER_SIZE];
    int index = 0;
    char c;
    int sign = 1;
    *res = 0;
```

```
    while(read(STDIN_FILENO, &c, sizeof(char)) > 0) {
        if (c == '-') {
            if (index != 0) {
                return -1;
            }
            sign = -1;
        } else if(is_num(c)) {
            *res = *res * 10 + (c - '0');
        } else if (is_space(c)) {
            break;
        } else {
            return -1;
        }
        index++;
    }
    *res *= sign;
    return (index > 0) ? 0 : -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 num = 0;

    while (process_line(&num) != -1) {
        if (num <= 0) {
            sem_wait(sem_empty);
            ptr[0] = 0;
            sem_post(sem_full);
            break;
        }

        if (!is_prime(num)) {
            sem_wait(sem_empty);
            ptr[0] = num;
            sem_post(sem_full);
        } else {
            sem_wait(sem_empty);
```

```
            ptr[0] = 0;
            sem_post(sem_full);
            break;
        }
    }
    ptr[1] = 0;
    close(STDOUT_FILENO);
    return 0;
}
```

Протокол работы программы

Тестирование:

1. Все числа составные

```
$cat > a.txt
8 12
16
4
18 22 10
$ cat > run.txt
a.txt
$ ./main.out < run.txt
8
12
16
4
18
22
10
```

2. Встречается простое число

```
$ cat > a.txt
```

8 12

16 11

4

18 22 10

$ ./main.out < run.txt

8

12

16

3. Встречается отрицательное число

$ cat > a.txt

8 12

16 -4

4

18 22 10

$ ./main.out < run.txt

8

12

16

Strace:

$ strace -f ./main.out < run.txt

execve("./main.out", ["./main.out"], 0x7fff1bb73c08 /* 68 vars */) = 0

brk(NULL) = 0x5d597b1e4000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe07870590) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x70f153704000

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=58791, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 58791, PROT_READ, MAP_PRIVATE, 3, 0) = 0x70f1536f5000

```
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\237\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\0"..., 784,
64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\0300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48,
848) = 48
pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68,
896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...},
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\0"..., 784,
64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3,
0) = 0x70f153400000
mprotect(0x70f153428000, 2023424, PROT_NONE) = 0
mmap(0x70f153428000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x70f153428000
mmap(0x70f1535bd000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x70f1535bd000
mmap(0x70f153616000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x70f153616000
mmap(0x70f15361c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x70f15361c000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x70f1536f2000
arch_prctl(ARCH_SET_FS, 0x70f1536f2740) = 0
set_tid_address(0x70f1536f2a10)         = 25141
set_robust_list(0x70f1536f2a20, 24)     = 0
rseq(0x70f1536f30e0, 0x20, 0, 0x53053053) = 0
mprotect(0x70f153616000, 16384, PROT_READ) = 0
```

mprotect(0x5d597a24c000, 4096, PROT_READ) = 0

mprotect(0x70f15373e000, 8192, PROT_READ) = 0

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

munmap(0x70f1536f5000, 58791) = 0

getrandom("\x7e\xb9\x4e\x95\x3e\x74\x90\xbf", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x5d597b1e4000

brk(0x5d597b205000) = 0x5d597b205000

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) = 0x70f15373d000

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) = 0x70f153703000

close(5) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 25142 attached, child_tidptr=0x70f1536f2a10) = 25142

[pid 25141] mmap(NULL, 8, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x70f153702000

[pid 25141] write(1, "0\n", 20) = 2

[pid 25142] set_robust_list(0x70f1536f2a20, 24 <unfinished ...>

[pid 25141] futex(0x70f153703000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 25142] <...set_robust_list resumed>) = 0

[pid 25142] dup2(3, 0) = 0

[pid 25142] close(3) = 0

[pid 25142] execve("./child.out", ["./child.out"], 0x7ffe07870768 /* 68 vars */) = 0

[pid 25142] brk(NULL) = 0x60339dc06000

[pid 25142] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffeceee3f80) = -1 EINVAL (Invalid argument)

[pid 25142] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7185fb4a7000

[pid 25142] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 25142] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 25142] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58791, ...}, AT_EMPTY_PATH) = 0

[pid 25142] mmap(NULL, 58791, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7185fb498000

[pid 25142] close(3) = 0

[pid 25142] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 25142] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

[pid 25142] 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 25142] pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

[pid 25142] pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

[pid 25142] newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

[pid 25142] 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 25142] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7185fb200000

[pid 25142] mprotect(0x7185fb228000, 2023424, PROT_NONE) = 0

[pid 25142] mmap(0x7185fb228000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7185fb228000

[pid 25142] mmap(0x7185fb3bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7185fb3bd000

[pid 25142] mmap(0x7185fb416000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7185fb416000

[pid 25142] mmap(0x7185fb41c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7185fb41c000

[pid 25142] close(3) = 0

[pid 25142] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7185fb495000

[pid 25142] arch_prctl(ARCH_SET_FS, 0x7185fb495740) = 0

[pid 25142] set_tid_address(0x7185fb495a10) = 25142

[pid 25142] set_robust_list(0x7185fb495a20, 24) = 0

[pid 25142] rseq(0x7185fb4960e0, 0x20, 0, 0x53053053) = 0

[pid 25142] mprotect(0x7185fb416000, 16384, PROT_READ) = 0

[pid 25142] mprotect(0x60339d76e000, 4096, PROT_READ) = 0

[pid 25142] mprotect(0x7185fb4e1000, 8192, PROT_READ) = 0

[pid 25142] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 25142] munmap(0x7185fb498000, 58791) = 0

[pid 25142] openat(AT_FDCWD, "/dev/shm/memory", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3

[pid 25142] openat(AT_FDCWD, "/dev/shm/sem.semaphore_empty", O_RDWR|O_EXCL|O_NOFOLLOW) = 4

[pid 25142] newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0

[pid 25142] getrandom("\xc6\x02\x1d\x26\xc5\x01\xd1\x8e", 8, GRND_NONBLOCK) = 8

[pid 25142] brk(NULL) = 0x60339dc06000

[pid 25142] brk(0x60339dc27000) = 0x60339dc27000

[pid 25142] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7185fb4e0000

[pid 25142] close(4) = 0

[pid 25142] openat(AT_FDCWD, "/dev/shm/sem.semaphore_full", O_RDWR|O_EXCL|O_NOFOLLOW) = 4

[pid 25142] newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0

[pid 25142] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7185fb4a6000

[pid 25142] close(4) = 0

[pid 25142] mmap(NULL, 8, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7185fb4a5000

[pid 25142] read(0, "8", 1) = 1

[pid 25142] read(0, " ", 1) = 1

[pid 25142] futex(0x7185fb4a6000, FUTEX_WAKE, 1 <unfinished ...>

[pid 25141] <... futex resumed>) = 0

[pid 25142] <... futex resumed>) = 1

[pid 25141] write(1, "8\n", 28) = 2

[pid 25142] read(0, <unfinished ...>

[pid 25141] futex(0x70f153703000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 25142] <... read resumed>"1", 1) = 1

[pid 25142] read(0, "2", 1) = 1

[pid 25142] read(0, "\n", 1) = 1

[pid 25142] futex(0x7185fb4a6000, FUTEX_WAKE, 1 <unfinished ...>

[pid 25141] <... futex resumed>) = 0

[pid 25142] <... futex resumed>) = 1

[pid 25141] write(1, "12\n", 312

<unfinished ...>

[pid 25142] read(0, <unfinished ...>

[pid 25141] <... write resumed>) = 3

[pid 25141] futex(0x70f153703000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 25142] <... read resumed>"1", 1) = 1

[pid 25142] read(0, "6", 1) = 1

[pid 25142] read(0, "\n", 1) = 1

[pid 25142] futex(0x7185fb4a6000, FUTEX_WAKE, 1 <unfinished ...>

[pid 25141] <... futex resumed>) = 0

[pid 25142] <... futex resumed>) = 1

[pid 25141] write(1, "16\n", 316) = 3

[pid 25141] futex(0x70f153703000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 25142] read(0, "4", 1) = 1

[pid 25142] read(0, "\n", 1) = 1

[pid 25142] futex(0x7185fb4a6000, FUTEX_WAKE, 1 <unfinished ...>

[pid 25141] <... futex resumed>) = 0

[pid 25142] <... futex resumed>) = 1

[pid 25141] write(1, "4\n", 24) = 2

[pid 25141] futex(0x70f153703000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 25142] read(0, "1", 1) = 1

[pid 25142] read(0, "8", 1) = 1

[pid 25142] read(0, " ", 1) = 1

[pid 25142] futex(0x7185fb4a6000, FUTEX_WAKE, 1 <unfinished ...>

[pid 25141] <... futex resumed>) = 0

[pid 25142] <... futex resumed>) = 1

[pid 25141] write(1, "18\n", 318) = 3

[pid 25141] futex(0x70f153703000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 25142] read(0, "2", 1) = 1

[pid 25142] read(0, "2", 1) = 1

[pid 25142] read(0, " ", 1) = 1

[pid 25142] futex(0x7185fb4a6000, FUTEX_WAKE, 1) = 1

[pid 25141] <... futex resumed>) = 0

[pid 25141] write(1, "22\n", 322) = 3

[pid 25141] futex(0x70f153703000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 25142] read(0, "1", 1) = 1

[pid 25142] read(0, "0", 1) = 1

[pid 25142] read(0, "\n", 1) = 1

[pid 25142] futex(0x7185fb4a6000, FUTEX_WAKE, 1) = 1

[pid 25141] <... futex resumed>) = 0

[pid 25142] read(0, <unfinished ...>

[pid 25141] write(1, "10\n", 3 <unfinished ...>

[pid 25142] <... read resumed>"", 1) = 0

10

[pid 25141] <... write resumed>) = 3

[pid 25142] close(1 <unfinished ...>

[pid 25141] wait4(25142, <unfinished ...>

[pid 25142] <... close resumed>) = 0

[pid 25142] exit_group(0) = ?

[pid 25142] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL)   = 25142

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=25142, si_uid=1000, si_status=0, si_utime=0, si_stime=1} ---

unlink("/dev/shm/memory")   = 0

exit_group(0)   = ?

+++ exited with 0 +++

Вывод

Для выполнения этой лабораторной работы мне потребовалось разобраться в написании программы с родительскими и дочерними процессами на Си. Нужно было хорошо продумать механизм работы программы с перенаправлением ввода и вывода, а также разделённой памятью(shared memory). Я писала подобную программу впервые и пришлось долго изучать информацию и разбираться с тем, как это работает. Одной из сложностей, с которыми я столкнулась была синхронизация родительского и дочернего процессов.