

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-210Б-23

Студент: Гула Дмитрий Александрович

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_

Дата: 01.11.24

Москва, 2024

# Постановка задачи

## Вариант 7.

С использованием shared memory и memory mapping. 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 число число<newline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

## Общий метод и алгоритм решения

Использованные системные вызовы:

- **fork()** — создаёт новый процесс, который является копией текущего (родительского) процесса. Возвращает PID дочернего процесса в родительском процессе и 0 в дочернем.
- **read()** — считывает данные из файла или ввода, возвращая количество считанных байт. Используется для чтения данных из файла или стандартного ввода.
- **write()** — записывает данные в файл или вывод. Возвращает количество записанных байт. Используется для вывода данных в консоль или запись в файл.
- **shm\_open()** — создаёт или открывает объект разделяемой памяти, возвращая файловый дескриптор для доступа к нему.
- **ftruncate()** — устанавливает размер объекта разделяемой памяти. Используется для выделения нужного объёма памяти.
- **mmap()** — отображает объект разделяемой памяти в адресное пространство процесса, возвращая указатель на начало отображения.
- **munmap()** — удаляет отображение объекта разделяемой памяти из адресного пространства процесса.
- **shm\_unlink()** — удаляет объект разделяемой памяти, освобождая ресурсы.
- **sem\_open()** — создаёт или открывает именованный семафор, возвращая указатель на него.
- **sem\_post()** — увеличивает значение семафора, сигнализируя, что ресурс доступен.
- **sem\_wait()** — уменьшает значение семафора, блокируя процесс, пока ресурс не станет доступным.
- **sem\_close()** — закрывает семафор, освобождая связанные ресурсы.
- **sem\_unlink()** — удаляет именованный семафор, освобождая системные ресурсы.
- **open()** — открывает файл и возвращает файловый дескриптор. Используется для открытия файла в определённом режиме (например, только для чтения).
- **close()** — закрывает дескриптор файла, освобождая ресурсы, связанные с файлом.

- **exec1()** — заменяет текущий процесс новым, исполняя указанную программу. Завершает выполнение текущего кода.
- **waitpid()** — приостанавливает выполнение родительского процесса до завершения указанного дочернего процесса, позволяя получить его статус.
- **perror()** — выводит сообщение об ошибке на основе значения глобальной переменной **errno**.
- **exit()** — завершает выполнение процесса с указанным кодом возврата.
- 

В рамках лабораторной работы мы разработали две программы: **родительскую** и **дочернюю**, которые взаимодействуют между собой с использованием shared memory и memory mapping. Для синхронизации чтения и записи из shared memory будем использовать семафор.

### Родительский процесс

1. **Запрос имени файла:**  
Родительский процесс запрашивает у пользователя имя файла, в котором содержатся строки с числами для обработки. Используется системный вызов **write()** для вывода запроса и **read()** для считывания ввода пользователя.
2. **Создание разделяемой памяти:**  
Родительский процесс создаёт объект разделяемой памяти с помощью **shm\_open()**. Далее с помощью **ftruncate()** задаётся размер разделяемой памяти.
3. **Отображение памяти:**  
Разделяемая память отображается в адресное пространство родительского процесса с помощью **mmap()**.
4. **Создание семафоров:**  
Родительский процесс создаёт два семафора:
  - **data\_sem**: для уведомления дочернего процесса о том, что данные готовы.
  - **processing\_sem**: для уведомления родительского процесса о завершении обработки данных.
5. **Создание дочернего процесса:**  
Системный вызов **fork()** создаёт новый (дочерний) процесс. Родительский и дочерний процессы начинают выполнение разных задач.
6. **Передача данных в разделяемую память:**  
Родительский процесс открывает файл, указанный пользователем, используя **open()**. Затем считывает данные из файла с помощью **read()** и записывает их в разделяемую память. После каждой записи:
  - Увеличивает значение семафора **data\_sem**, сигнализируя дочернему процессу, что данные готовы.
  - Ожидает сигнал от дочернего процесса через семафор **processing\_sem**, подтверждающий, что данные обработаны.
7. **Завершение передачи данных:**  
Когда все данные из файла переданы, родительский процесс записывает символ конца строки (**\0**) в разделяемую память и сигнализирует дочернему процессу о завершении передачи данных через **data\_sem**.
8. **Очистка ресурсов:**  
После завершения работы дочернего процесса родительский процесс:
  - Закрывает дескрипторы файлов.

- Удаляет отображение разделяемой памяти и семафоров.
- Освобождает ресурсы, используя `shm_unlink()` и `sem_unlink()`.

### Дочерний процесс

#### 1. Подключение к разделяемой памяти:

Дочерний процесс открывает существующий объект разделяемой памяти с помощью `shm_open()` и отображает его в своё адресное пространство через `mmap()`.

#### 2. Подключение к семафорам:

Дочерний процесс открывает существующие семафоры `data_sem` и `processing_sem` с помощью `sem_open()`.

#### 3. Обработка данных:

В цикле:

- Ожидает сигнал от родительского процесса через `sem_wait(data_sem)`, указывающий на готовность данных.
- Проверяет, содержит ли разделяемая память символ конца строки (`\0`). Если да, завершает выполнение.
- Разбивает строки на числа, используя `strtok()`, и подсчитывает их сумму.
- Выводит результат на экран через `write()`.

#### 4. Уведомление о завершении обработки:

После завершения обработки данных в текущей строке дочерний процесс увеличивает значение семафора `processing_sem`, чтобы уведомить родительский процесс.

#### 5. Очистка ресурсов:

Дочерний процесс освобождает свои ресурсы:

- Удаляет отображение разделяемой памяти.
- Закрывает дескрипторы семафоров.

#### 6. Завершение работы:

После обработки всех данных дочерний процесс завершает выполнение.

## Код программы

### parent.cpp

```
##include <iostream>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

#include <unistd.h>

#include <sys/wait.h>
```

```
#include <cstring>

const int SHM_SIZE = 4096;

const char* SHM_NAME = "/my_shm";

const char* DATA_SEM_NAME = "/my_data_sem";

const char* PROCESSING_SEM_NAME = "/my_processing_sem";

int main() {

    char filename[256];

    write(STDOUT_FILENO, "Введите имя файла: ", 35);

    ssize_t bytesRead = read(STDIN_FILENO, filename, sizeof(filename) - 1);

    if (bytesRead == -1) {

        perror("read error");

        exit(EXIT_FAILURE);

    }

    if (bytesRead > 0 && filename[bytesRead - 1] == '\n') {

        filename[bytesRead - 1] = '\0';

    } else {

        filename[bytesRead] = '\0';

    }

    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);

    if (shm_fd == -1) {

        perror("shm_open error");

        exit(EXIT_FAILURE);

    }

}
```

```
if (ftruncate(shm_fd, SHM_SIZE) == -1) {

    perror("ftruncate error");

    exit(EXIT_FAILURE);

}


char* shm_ptr = (char*)mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);

if (shm_ptr == MAP_FAILED) {

    perror("mmap error");

    exit(EXIT_FAILURE);

}


sem_t* data_sem = sem_open(DATA_SEM_NAME, O_CREAT, 0666, 0);

if (data_sem == SEM_FAILED) {

    perror("sem_open data_sem error");

    exit(EXIT_FAILURE);

}


sem_t* processing_sem = sem_open(PROCESSING_SEM_NAME, O_CREAT, 0666, 0);

if (processing_sem == SEM_FAILED) {

    perror("sem_open processing_sem error");

    exit(EXIT_FAILURE);

}


pid_t pid = fork();

if (pid == -1) {

    perror("fork error");

    exit(EXIT_FAILURE);

} else if (pid == 0) {
```

```
    execl("./child", "./child", nullptr);

    perror("execl error");

    exit(EXIT_FAILURE);
} else {

    int file_fd = open(filename, O_RDONLY);

    if (file_fd == -1) {

        perror("open error");

        exit(EXIT_FAILURE);

    }

    ssize_t bytesRead;

    while ((bytesRead = read(file_fd, shm_ptr, SHM_SIZE - 1)) > 0) {

        shm_ptr[bytesRead] = '\0';

        sem_post(data_sem);

        sem_wait(processing_sem);

    }

    shm_ptr[0] = '\0';

    sem_post(data_sem);

    close(file_fd);

    waitpid(pid, nullptr, 0);

    munmap(shm_ptr, SHM_SIZE);

    shm_unlink(SHM_NAME);

    sem_close(data_sem);

    sem_unlink(DATA_SEM_NAME);

    sem_close(processing_sem);
```

```
        sem_unlink(PROCESSING_SEM_NAME);

    }

    return 0;
}
```

### **child.cpp**

```
#include <iostream>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

#include <unistd.h>

#include <sstream>

#include <cstring>

const int SHM_SIZE = 4096;

const char* SHM_NAME = "/my_shm";

const char* DATA_SEM_NAME = "/my_data_sem";

const char* PROCESSING_SEM_NAME = "/my_processing_sem";

int main() {

    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);

    if (shm_fd == -1) {

        perror("shm_open error");

        exit(EXIT_FAILURE);

    }
```



```
char* shm_ptr = (char*)mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);

if (shm_ptr == MAP_FAILED) {

    perror("mmap error");

    exit(EXIT_FAILURE);

}

sem_t* data_sem = sem_open(DATA_SEM_NAME, 0);

if (data_sem == SEM_FAILED) {

    perror("sem_open data_sem error");

    exit(EXIT_FAILURE);

}

sem_t* processing_sem = sem_open(PROCESSING_SEM_NAME, 0);

if (processing_sem == SEM_FAILED) {

    perror("sem_open processing_sem error");

    exit(EXIT_FAILURE);

}

while (true) {

    sem_wait(data_sem);

    if (shm_ptr[0] == '\\0') {

        break;

    }

    char* line = strtok(shm_ptr, "\\n");

    while (line) {

        std::istringstream iss(line);
```

```
float number, sum = 0;

while (iss >> number) {

    sum += number;

}

char output[256];

int output_len = snprintf(output, sizeof(output), "Сумма:
%.2f\n", sum);

write(STDOUT_FILENO, output, output_len);

line = strtok(nullptr, "\n");

}

sem_post(processing_sem);

}

munmap(shm_ptr, SHM_SIZE);

sem_close(data_sem);

sem_close(processing_sem);

return 0;

}
```

# Протокол работы программы

## Тестирование:

```
→ src git:(main) ./parent
```

Введите имя файла: data.txt

Сумма: 60.9

Сумма: 13

Сумма: 11

Сумма: 151

Сумма: 36.5

Сумма: 9.5

```
→ src git:(main) cat < data.txt
```

10.5 20.3 30.1

5.5 3.2 4.3

1.1 2.2 3.3 4.4

100.25 50.75

7.7 8.8 9.9 10.10

7.7 8.8 9.9 10.10 -1.5 -25.5

**Dtruss:**

```
dmitry@dmitry-HP-Laptop-15-da3xxx:~/C/Labs-OS/lab3/src$ strace ./parent
```

```
execve("./parent", ["./parent"], 0x7ffcef193f60 /* 73 vars */) = 0
```

```
brk(NULL) = 0x58f06fb16000
```

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

0x74438f8d6000

```
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=58511, ...}) = 0
```

```
mmap(NULL, 58511, PROT_READ, MAP_PRIVATE, 3, 0) = 0x74438f8c7000
```

```
close(3) = 0
```

```
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.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\\0\\0\\0\\0\\0\\0\\0\\0"..., 832) = 832
```

```
fstat(3, {st mode=S IFREG|0644, st size=2592224, ...}) = 0
```

```
mmap(NULL, 2609472, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x74438f600000
```

```
mmap(0x74438f69d000, 1343488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
```

$$3, 0x9d000) = 0x74438f69d000$$

```
mmap(0x74438f7e5000, 552960, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
```

$$0x1e5000) = 0x74438f7e5000$$

```
mmap(0x74438f86c000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
```

$$3, 0x26b000) = 0x74438f86c000$$

```
mmap(0x74438f87a000, 12608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
```

-1, 0) = 0x74438f87a000

```
close(3) = 0
```

```
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", 0_RDONLY|0_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\\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) = 0x74438f200000
mmap(0x74438f228000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x74438f228000
mmap(0x74438f3b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x74438f3b0000
mmap(0x74438f3ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1fe000) = 0x74438f3ff000
mmap(0x74438f405000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x74438f405000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"... , 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=952616, ...}) = 0
mmap(NULL, 950296, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x74438f517000
mmap(0x74438f527000, 520192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x10000) = 0x74438f527000
mmap(0x74438f5a6000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x8f000) = 0x74438f5a6000
mmap(0x74438f5fe000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0xe7000) = 0x74438f5fe000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"... , 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=183024, ...}) = 0
mmap(NULL, 185256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x74438f899000
mmap(0x74438f89d000, 147456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x4000) = 0x74438f89d000
mmap(0x74438f8c1000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000)
= 0x74438f8c1000
mmap(0x74438f8c5000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x2b000) = 0x74438f8c5000
close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x74438f897000
arch_prctl(ARCH_SET_FS, 0x74438f898500) = 0
set_tid_address(0x74438f8987d0) = 41738
set_robust_list(0x74438f8987e0, 24) = 0
rseq(0x74438f898e20, 0x20, 0, 0x53053053) = 0
mprotect(0x74438f3ff000, 16384, PROT_READ) = 0
mprotect(0x74438f8c5000, 4096, PROT_READ) = 0
mprotect(0x74438f5fe000, 4096, PROT_READ) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x74438f895000
mprotect(0x74438f86c000, 45056, PROT_READ) = 0

```

```

mprotect(0x58f06f092000, 4096, PROT_READ) = 0
mprotect(0x74438f90e000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x74438f8c7000, 58511) = 0
futex(0x74438f87a7bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
getrandom("\xc9\x0b\xb6\xb4\x79\x36\xca\x76", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x58f06fb16000
brk(0x58f06fb37000) = 0x58f06fb37000
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"... , 35Введите имя
файла: ) = 35
read(0, data.txt
"data.txt\n", 255) = 9
openat(AT_FDCWD, "/dev/shm/my_shm", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 4096) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x74438f8d5000
openat(AT_FDCWD, "/dev/shm/sem.my_data_sem", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT
(No such file or directory)
getrandom("\xc2\x50\xf2\x04\x3f\xbe\x2a\xdc", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.68AL2f", 0x7ffcbade260, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.68AL2f", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0666) = 4
write(4, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x74438f8d4000
link("/dev/shm/sem.68AL2f", "/dev/shm/sem.my_data_sem") = 0
fstat(4, {st_mode=S_IFREG|0664, st_size=32, ...}) = 0
unlink("/dev/shm/sem.68AL2f") = 0
close(4) = 0
openat(AT_FDCWD, "/dev/shm/sem.my_processing_sem", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1
ENOENT (No such file or directory)
getrandom("\xeb\x54\xf7\x82\x4a\x76\x31\xd2", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.5ZwiIx", 0x7ffcbade260, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.5ZwiIx", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0666) = 4
write(4, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x74438f8d3000
link("/dev/shm/sem.5ZwiIx", "/dev/shm/sem.my_processing_sem") = 0
fstat(4, {st_mode=S_IFREG|0664, st_size=32, ...}) = 0
unlink("/dev/shm/sem.5ZwiIx") = 0
close(4) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x74438f8987d0) = 41820
openat(AT_FDCWD, "data.txt", O_RDONLY) = 4
read(4, "10.5 20.3 30.1\n5.5 3.2 4.3\n1.1 2"... , 4095) = 102
futex(0x74438f8d3000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANYСымма: 60.90
Сымма: 13.00

```

```

Сумма: 11.00
Сумма: 151.00
Сумма: 36.50
Сумма: 9.50
) = 0
read(4, "", 4095) = 0
futex(0x74438f8d4000, FUTEX_WAKE, 1) = 1
close(4) = 0
wait4(41820, NULL, 0, NULL) = 41820
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=41820, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
munmap(0x74438f8d5000, 4096) = 0
unlink("/dev/shm/my_shm") = 0
munmap(0x74438f8d4000, 32) = 0
unlink("/dev/shm/sem.my_data_sem") = 0
munmap(0x74438f8d3000, 32) = 0
unlink("/dev/shm/sem.my_processing_sem") = 0
exit_group(0) = ?
+++ exited with 0 +++

```

## Вывод

Эта программа реализует взаимодействие между двумя процессами с использованием общей памяти и семафоров. Программа делится на две части: родительский процесс и дочерний процесс. Родительский процесс читает данные из файла, передает их в общую память, а дочерний процесс обрабатывает эти данные, выполняя подсчет суммы чисел в каждой строке и выводя результат.