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: Жданович Е.Т.

Преподаватель: Бахарев В.Д.

Оценка:

Дата: 26.12.24

Москва, 2024

Постановка задачи

Вариант 11.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем, с использованием shared memory и memory mapping. Для синхронизации чтения и записи из shared memory будем использовать семафор. Родительский процесс создает два дочерних процесса. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода. 11 вариант) Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «_».

Общий метод и алгоритм решения

Использованные системные вызовы:

- **`sem_open`**:
  Открывают или создают именованные семафоры. В данном случае, семафоры `SEM_EMPTY` и `SEM_FULL` используются для синхронизации обмена сообщениями между процессами. `SEM_EMPTY` инициализируется значением 1, а `SEM_FULL` — значением 0. Эти значения позволяют процессам обмениваться данными по принципу "производитель-потребитель".
- **`shm_open`**:
  Создает или открывает объект общей памяти с именем `SHM_NAME`. Этот вызов предоставляет дескриптор для дальнейшего обращения к общей памяти.

- **`ftruncate`**:
  Обрезает или изменяет размер объекта общей памяти. В данном случае, размер общей памяти устанавливается на размер структуры `SharedData`, которая включает в себя массив символов для сообщения и флаг для синхронизации.
- **`mmap`**:
  Открывает отображение в память объекта общей памяти, полученного через `shm_open`. Это позволяет процессам читать и записывать в общую память.

- **`read`**:

Считывает данные из стандартного ввода (консоли) в общую память, в массив `shmaddr->message`. В коде предполагается, что пользователь введет сообщение.

- **`fork`**:
  Создают два дочерних процесса. После каждого `fork` программа продолжает выполнение как в родительском процессе, так и в дочернем. Эти дочерние процессы затем выполняют программы `child1` и `child2`, используя `execl`.

- **`execl`**:
  Выполняет программы `child1` и `child2` в дочерних процессах. Эти процессы будут работать с общими семафорами для синхронизации доступа к общей памяти.

- **`sem_post`**:
  Увеличивает значение семафора `sem_empty`. Это уведомляет другие процессы, что можно работать с общей памятью. Этот вызов может быть использован для синхронизации между производителем и потребителем данных.

- **`wait`**:
  Ожидает завершения дочернего процесса. Используется дважды для ожидания завершения `child1` и `child2`.

- **`write`**:
  Пишет финальное сообщение из общей памяти на стандартный вывод (консоль).

- **`sem_close`**:
  Закрывает дескрипторы семафоров после завершения работы с ними.

- **`sem_unlink:`**
  Удаляет именованные семафоры после их использования.

- **`munmap`**:
  Отключает отображение общей памяти, освобождая ресурсы, связанные с ней.

- **`shm_unlink`**:
  Удаляет объект общей памяти, освобождая ресурсы.

Child1.c

1. **Проверка аргументов командной строки:**

Программа ожидает два аргумента: имена семафоров sem_empty и sem_full. Если аргументы не переданы, выводится сообщение об ошибке, и программа завершает выполнение.

2. Открытие семафоров:

С помощью sem_open открываются два семафора:

sem_empty: отвечает за блокировку чтения до тех пор, пока данные не будут готовы для обработки.

sem_full: отвечает за сигнализацию о завершении обработки данных.

Если открытие семафоров завершается неудачей, программа выводит сообщение об ошибке и завершает выполнение.

3. Подключение к разделяемой памяти:

С помощью shm_open открывается объект разделяемой памяти, имя которого задано в константе SHM_NAME.

Если операция завершается неудачей, выводится сообщение об ошибке, и программа завершает выполнение.

С помощью mmap разделяемая память отображается в адресное пространство программы, предоставляя доступ к структуре SharedData.

4. Ожидание готовности данных:

Программа вызывает sem_wait, ожидая, пока данные станут доступны для обработки.

5. Обработка данных:

В цикле строка из разделяемой памяти преобразуется: каждый символ переводится в верхний регистр с использованием функции toupper.

6. Обновление флага:

После обработки строка помечается как обработанная, устанавливая флаг в значение 1.

7. Сигнал о завершении обработки.

8. Освобождение ресурсов.

9. Завершение программы:

Программа успешно завершает выполнение с кодом 0.

Child2.c

1. Проверка аргументов командной строки:

Программа ожидает два аргумента: имена семафоров sem_empty и sem_full. Если аргументы не переданы, выводится сообщение об ошибке, и программа завершает выполнение.

2. Открытие семафоров:

С помощью sem_open открываются два семафора:

sem_empty: для управления доступом к разделяемой памяти.

sem_full: для ожидания завершения обработки данных другим процессом.

Если открытие семафоров завершается неудачей, программа выводит сообщение об ошибке и завершает выполнение.

3. Подключение к разделяемой памяти:

С помощью shm_open открывается объект разделяемой памяти, имя

которого задано в константе SHM_NAME.

Если операция завершается неудачей, выводится сообщение об ошибке, и программа завершает выполнение.

С помощью mmap разделяемая память отображается в адресное пространство программы, предоставляя доступ к структуре SharedData.

4. Ожидание обработки данных:

Программа вызывает sem_wait(sem_full), ожидая сигнал от другого процесса о завершении обработки данных (например, преобразования строки в верхний регистр).

5. Обработка данных:

В цикле символы строки из разделяемой памяти проверяются: Если символ пробел ' ', он заменяется на символ подчеркивания '_'.

6. Вывод обработанного сообщения:

Сообщение из разделяемой памяти выводится на стандартный вывод. Предварительно печатается строка "Child 2 processed message: ".

Затем — само сообщение.

Вывод завершается символом новой строки.

7. Освобождение ресурсов:

Освобождаются ресурсы:

Операция munmap удаляет отображение разделяемой памяти.

Операция close закрывает дескриптор разделяемой памяти.

8. Завершение программы:

Программа успешно завершает выполнение с кодом 0.

Parent.c

1. Ввод сообщения:

Программа запрашивает у пользователя сообщение для обработки.

Пользователь вводит строку, которая записывается в разделяемую память.

2. Создание общей памяти и семафоров:

Создается общая память с помощью shm_open() и выделяется необходимое пространство с помощью ftruncate().

Память отображается в адресное пространство программы через mmap.

Создаются два семафора: sem_empty для синхронизации записи в память; sem_full для синхронизации обработки данных.

3. Создание дочерних процессов:

С помощью fork() создаются два дочерних процесса:

Первый процесс запускает программу child1 через execl().

Второй процесс запускает программу child2 через execl().

4. Передача данных и ожидание обработки:

Родительский процесс использует sem_post(sem_empty) для сигнализации о готовности данных.

Родитель ожидает завершения дочерних процессов с помощью wait().

5. Вывод результата:

После завершения обработки сообщение из разделяемой памяти выводится на экран.

6. Очистка ресурсов:

Семафоры закрываются и удаляются с помощью sem_close() и sem_unlink().

Разделяемая память удаляется с помощью munmap() и shm_unlink().

Код программы

Parent.c

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <sys/mman.h>
    #include <sys/stat.h>
    #include <fcntl.h>
    #include <semaphore.h>
    #include <unistd.h>
    #include <sys/wait.h>
    #define SHM_NAME "/shared_memory"
    #define SEM_EMPTY "/sem_empty"
    #define SEM_FULL "/sem_full"
    #define BUFFER_SIZE 256
    struct SharedData {
     char message[BUFFER_SIZE];
     int flag;
    };
    int main() {
     sem_t *sem_empty = sem_open(SEM_EMPTY, O_CREAT, 0666, 1);
     sem_t *sem_full = sem_open(SEM_FULL, O_CREAT, 0666, 0);
 if (sem_empty == SEM_FAILED || sem_full == SEM_FAILED) {

perror("sem_open failed");
    exit(EXIT_FAILURE);
```

```
}

int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
if (shm_fd == -1) {
    perror("shm_open failed");
    exit(EXIT_FAILURE);
}

if (ftruncate(shm_fd, sizeof(struct SharedData)) == -1) {
    perror("ftruncate failed");
    exit(EXIT_FAILURE);
}

struct SharedData *shmaddr = mmap(NULL, sizeof(struct SharedData),
                                  PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
if (shmaddr == MAP_FAILED) {
    perror("mmap failed");
    exit(EXIT_FAILURE);
}

shmaddr->flag = 0;

write(1, "Enter a message: ", 17);
ssize_t bytes_read = read(0, shmaddr->message, BUFFER_SIZE);
if (bytes_read == -1) {
    perror("read failed");
    exit(EXIT_FAILURE);
}


if (shmaddr->message[bytes_read - 1] == '\n') {
    shmaddr->message[bytes_read - 1] = '\0';
}

pid_t child1 = fork();
if (child1 == -1) {
    perror("fork failed");
    exit(EXIT_FAILURE);

}
```

```
if (child1 == 0) {
    execl("./child1", "child1", SEM_EMPTY, SEM_FULL, NULL);
    perror("execl failed");
    exit(EXIT_FAILURE);
}

pid_t child2 = fork();
if (child2 == -1) {
    perror("fork failed");
    exit(EXIT_FAILURE);
}

if (child2 == 0) {
    execl("./child2", "child2", SEM_EMPTY, SEM_FULL, NULL);
    perror("execl failed");
    exit(EXIT_FAILURE);
}

sem_post(sem_empty);

wait(NULL);
wait(NULL);

write(1, "Final message: ", 15);
write(1, shmaddr->message, strlen(shmaddr->message));
write(1, "\n", 1);

sem_close(sem_empty);
sem_close(sem_full);
sem_unlink(SEM_EMPTY);
sem_unlink(SEM_FULL);

munmap(shmaddr, sizeof(struct SharedData));
shm_unlink(SHM_NAME);

return 0;


    }
```

Child1.c

```
    #include <stdio.h>

    #include <stdlib.h>

    #include <sys/mman.h>

    #include <fcntl.h>

    #include <semaphore.h>

    #include <unistd.h>

    #include <ctype.h>

    #include <string.h>

    #define SHM_NAME "/shared_memory"

    #define BUFFER_SIZE 256

    struct SharedData {

     char message[BUFFER_SIZE];

     int flag;

     };

    int main(int argc, char *argv[]) {

     if (argc != 3) {

     write(2, "Usage: child1 \n", 37); exit(EXIT_FAILURE);

     }
sem_t*sem_empty=sem_open(argv[1],0);
sem_t*sem_full=sem_open(argv[2],0);

if (sem_empty == SEM_FAILED || sem_full == SEM_FAILED) {


    perror("sem_open                                                failed");
    exit(EXIT_FAILURE);
}

int       shm_fd       =       shm_open(SHM_NAME,       O_RDWR,       0666);
if               (shm_fd               ==               -1)               {

    perror("shm_open                                                failed");
    exit(EXIT_FAILURE);
```

```
}

struct   SharedData   *shmaddr   =   mmap(NULL,   sizeof(struct   SharedData),
                                PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
if             (shmaddr             ==             MAP_FAILED)             {
    perror("mmap                                                  failed");
    exit(EXIT_FAILURE);
}

sem_wait(sem_empty);

for    (int    i    =    0;    shmaddr->message[i]    !=    '\0';    i++)    {
    shmaddr->message[i]   =    toupper((unsigned   char)shmaddr->message[i]);
}

shmaddr->flag                              =                              1;

sem_post(sem_full);

munmap(shmaddr,                  sizeof(struct                  SharedData));
close(shm_fd);
return                                                                    0;


    }
```

Child2.c

```
    #include <stdio.h>

    #include <stdlib.h>

    #include <sys/mman.h>

    #include <fcntl.h>

    #include <semaphore.h>

    #include <unistd.h>

    #include <string.h>

    #define SHM_NAME "/shared_memory"

    #define BUFFER_SIZE 256

    struct SharedData {
```

```
     char message[BUFFER_SIZE];
     int flag;
     };
    int main(int argc, char *argv[]) {
     if (argc != 3) {
     write(2, "Usage: child2 \n", 37);
     exit(EXIT_FAILURE);
    }
    sem_t*sem_empty=sem_open(argv[1],0);
sem_t*sem_full=sem_open(argv[2],0);

if(sem_empty == SEM_FAILED||sem_full == SEM_FAILED) {


    perror("sem_open                                                    failed");
    exit(EXIT_FAILURE);
}

int       shm_fd       =       shm_open(SHM_NAME,       O_RDWR,       0666);
if               (shm_fd               ==               -1)               {
    perror("shm_open                                                    failed");
    exit(EXIT_FAILURE);
}

struct   SharedData   *shmaddr   =   mmap(NULL,   sizeof(struct   SharedData),
                                  PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
if             (shmaddr             ==             MAP_FAILED)             {
    perror("mmap                                                        failed");
    exit(EXIT_FAILURE);
}

sem_wait(sem_full);

for    (int    i    =    0;    shmaddr->message[i]    !=    '\0';    i++)    {
    if         (shmaddr->message[i]         ==         '         ')         {
        shmaddr->message[i]                         =                       '_';
    }
}

write(1,      "Child       2      processed      message:      ",       26);
write(1,            shmaddr->message,             strlen(shmaddr->message));
```

```
write(1,                                "\n",                                1);

munmap(shmaddr,                 sizeof(struct                 SharedData));
close(shm_fd);
return                                                                       0;

    }
```

Протокол работы программы

Тестирование:

```
Enter your string or (Enter / CTRL + D) for stop: ijon kj lk
Processed result: IJON_KJ_LK


Enter your string or (Enter / CTRL + D) for stop: knm kl kjl
Processed result: KNM_KL_KJL


Enter your string or (Enter / CTRL + D) for stop: Hello you
Processed result: HELLO_YOU
```

Strace:

lizka@LizaAlisa:~/ЛАБЫ_ОС/Лаба3$ strace -f ./parent

execve("./parent", ["./parent"], 0x7fff6743b7d8 /* 27 vars */) = 0

brk(NULL) = 0x55c5697d0000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f08ec146000

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=19163, ...}) = 0

mmap(NULL, 19163, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f08ec141000

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\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) = 0x7f08ebf2f000

mmap(0x7f08ebf57000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f08ebf57000

mmap(0x7f08ec0df000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f08ec0df000

mmap(0x7f08ec12e000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f08ec12e000

mmap(0x7f08ec134000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f08ec134000 close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f08ebf2c000

arch_prctl(ARCH_SET_FS, 0x7f08ebf2c740) = 0

set_tid_address(0x7f08ebf2ca10) = 450

set_robust_list(0x7f08ebf2ca20, 24) = 0 rseq(0x7f08ebf2d060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f08ec12e000, 16384, PROT_READ) = 0

mprotect(0x55c545d2e000, 4096, PROT_READ) = 0

mprotect(0x7f08ec17e000, 8192, PROT_READ) = 0

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

munmap(0x7f08ec141000, 19163) = 0

openat(AT_FDCWD, "/dev/shm/sem.sem_empty", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1

ENOENT (No such file or directory)

getrandom("\x5b\xc4\x5d\x48\x02\x8d\x09\x75", 8, GRND_NONBLOCK) = 8

newfstatat(AT_FDCWD, "/dev/shm/sem.lA7jSB", 0x7ffdbe8b3550, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/dev/shm/sem.lA7jSB", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 3

write(3, "\1\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\0\0", 32) = 32

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f08ec145000

link("/dev/shm/sem.lA7jSB", "/dev/shm/sem.sem_empty") = 0

fstat(3, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

getrandom("\x7a\xa9\x64\xa2\xb9\x6a\x90\x73", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x55c5697d0000 brk(0x55c5697f1000) = 0x55c5697f1000

unlink("/dev/shm/sem.lA7jSB") = 0

close(3) = 0

openat(AT_FDCWD, "/dev/shm/sem.sem_full", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)

getrandom("\x64\xb3\x41\x69\xf0\x78\xcd\xbc", 8, GRND_NONBLOCK) = 8

newfstatat(AT_FDCWD, "/dev/shm/sem.0iaE9V", 0x7ffdbe8b3550, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/dev/shm/sem.0iaE9V", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 3

write(3, "\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\0\0", 32) = 32

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f08ec144000

link("/dev/shm/sem.0iaE9V", "/dev/shm/sem.sem_full") = 0

fstat(3, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

unlink("/dev/shm/sem.0iaE9V") = 0

close(3) = 0

openat(AT_FDCWD, "/dev/shm/shared_memory", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3

ftruncate(3, 260) = 0

mmap(NULL, 260, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f08ec143000

write(1, "Enter a message: ", 17Enter a message:) = 17

read(0, "\n", 256) = 1

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 451 attached , child_tidptr=0x7f08ebf2ca10) = 451

[pid 451] set_robust_list(0x7f08ebf2ca20, 24 <unfinished ...>

[pid 450] clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD <unfinished ...>

[pid 451] <... set_robust_list resumed>) = 0

[pid 451] execve("./child1", ["child1", "/sem_empty", "/sem_full"], 0x7ffdbe8b3a38 /* 27 vars */*strace: Process 452 attached <unfinished ...>*

[pid 450] <... clone resumed>, child_tidptr=0x7f08ebf2ca10) = 452

[pid 452] set_robust_list(0x7f08ebf2ca20, 24 <unfinished ...>

[pid 450] wait4(-1, <unfinished ...>

[pid 452] <... set_robust_list resumed>) = 0

[pid 452] execve("./child2", ["child2", "/sem_empty", "/sem_full"], 0x7ffdbe8b3a38 / 27 vars */) = 0*

[pid 452] brk(NULL <unfinished ...>

[pid 451] <... execve resumed>) = 0

[pid 452] <... brk resumed>) = 0x55bf94a50000

[pid 451] brk(NULL <unfinished ...>

[pid 452] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 451] <... brk resumed>) = 0x561388fcd000

[pid 452] <... mmap resumed>) = 0x7f1c9e26c000

[pid 452] access("/etc/ld.so.preload", R_OK <unfinished ...>

[pid 451] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 452] <... access resumed>) = -1 ENOENT (No such file or directory)

[pid 451] <... mmap resumed>) = 0x7f0150483000

[pid 452] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC <unfinished ...>

[pid 451] access("/etc/ld.so.preload", R_OK <unfinished ...>

[pid 452] <... openat resumed>) = 3

[pid 451] <... access resumed>) = -1 ENOENT (No such file or directory)

[pid 452] fstat(3, <unfinished ...>

[pid 451] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC <unfinished ...>

[pid 452] <... fstat resumed>{st_mode=S_IFREG|0644, st_size=19163, ...}) = 0

[pid 451] <... openat resumed>) = 3

[pid 452] mmap(NULL, 19163, PROT_READ, MAP_PRIVATE, 3, 0 <unfinished ...>

[pid 451] fstat(3, <unfinished ...>

[pid 452] <... mmap resumed>) = 0x7f1c9e267000

[pid 451] <... fstat resumed>{st_mode=S_IFREG|0644, st_size=19163, ...}) = 0

[pid 452] close(3 <unfinished ...>

[pid 451] mmap(NULL, 19163, PROT_READ, MAP_PRIVATE, 3, 0 <unfinished ...>

[pid 452] <... close resumed>) = 0

[pid 451] <... mmap resumed>) = 0x7f015047e000

[pid 452] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC <unfinished ...>

[pid 451] close(3 <unfinished ...> [pid 452] <... openat resumed>) = 3

[pid 451] <... close resumed>) = 0

*[**pid 452] read(3, <unfinished ...>***

[pid 451] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC <unfinished ...>

[pid 452] <... read resumed>"\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

[pid 451] <... openat resumed>) = 3

[pid 452] pread64(3, <unfinished ...>

[pid 451] read(3, <unfinished ...>

[pid 452] <... pread64 resumed>"\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 451] <... read resumed>"\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

[pid 452] fstat(3, <unfinished ...>

[pid 451] pread64(3, <unfinished ...>

[pid 452] <... fstat resumed>{st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 451] <... pread64 resumed>"\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 452] 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 451] fstat(3, <unfinished ...>

[pid 452] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0 <unfinished ...>

[pid 451] <... fstat resumed>{st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 452] <... mmap resumed>) = 0x7f1c9e055000

[pid 451] pread64(3, <unfinished ...>

[pid 452] mmap(0x7f1c9e07d000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000 <unfinished ...>

[pid 451] <... pread64 resumed>"\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 452] <... mmap resumed>) = 0x7f1c9e07d000

[pid 451] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0 <unfinished ...>

[pid 452] mmap(0x7f1c9e205000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000 <unfinished ...>

[pid 451] <... mmap resumed>) = 0x7f015026c000

[pid 452] <... mmap resumed>) = 0x7f1c9e205000

[pid 451] mmap(0x7f0150294000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000 <unfinished ...>

[pid 452] mmap(0x7f1c9e254000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000 <unfinished ...>

[pid 451] <... mmap resumed>) = 0x7f0150294000

[pid 452] <... mmap resumed>) = 0x7f1c9e254000

*[**pid 451] mmap(0x7f015041c000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,***
0x1b0000 <unfinished ...>

[pid 452] mmap(0x7f1c9e25a000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 451] <... mmap resumed>) = 0x7f015041c000

[pid 452] <... mmap resumed>) = 0x7f1c9e25a000

[pid 451] mmap(0x7f015046b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000 <unfinished ...>

[pid 452] close(3 <unfinished ...>

[pid 451] <... mmap resumed>) = 0x7f015046b000

[pid 452] <... close resumed>) = 0

[pid 451] mmap(0x7f0150471000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 452] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 451] <... mmap resumed>) = 0x7f0150471000

[pid 452] <... mmap resumed>) = 0x7f1c9e052000

[pid 452] arch_prctl(ARCH_SET_FS, 0x7f1c9e052740 <unfinished ...>

[pid 451] close(3 <unfinished ...>

[pid 452] <... arch_prctl resumed>) = 0

[pid 451] <... close resumed>) = 0

[pid 452] set_tid_address(0x7f1c9e052a10 <unfinished ...>

[pid 451] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 452] <... set_tid_address resumed>) = 452

[pid 451] <... mmap resumed>) = 0x7f0150269000

[pid 452] set_robust_list(0x7f1c9e052a20, 24) = 0

[pid 451] arch_prctl(ARCH_SET_FS, 0x7f0150269740 <unfinished ...>

[pid 452] rseq(0x7f1c9e053060, 0x20, 0, 0x53053053 <unfinished ...>

[pid 451] <... arch_prctl resumed>) = 0 [pid 452] <... rseq resumed>) = 0

[pid 451] set_tid_address(0x7f0150269a10) = 451

[pid 452] mprotect(0x7f1c9e254000, 16384, PROT_READ <unfinished ...>

[pid 451] set_robust_list(0x7f0150269a20, 24 <unfinished ...>

[pid 452] <... mprotect resumed>) = 0

[pid 451] <... set_robust_list resumed>) = 0

[pid 452] mprotect(0x55bf90af4000, 4096, PROT_READ <unfinished ...>

[pid 451] rseq(0x7f015026a060, 0x20, 0, 0x53053053 <unfinished ...>

[pid 452] <... mprotect resumed>) = 0

[pid 451] <... rseq resumed>) = 0

[pid 452] mprotect(0x7f1c9e2a4000, 8192, PROT_READ <unfinished ...>

[pid 451] mprotect(0x7f015046b000, 16384, PROT_READ <unfinished ...>

[pid 452] <... mprotect resumed>) = 0

[pid 451] <... mprotect resumed>) = 0

[pid 452] prlimit64(0, RLIMIT_STACK, NULL, <unfinished ...>

[pid 451] mprotect(0x56136e570000, 4096, PROT_READ <unfinished ...>

*[pid 452] <... prlimit64 resumed>{rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 451] <... mprotect resumed>) = 0

[pid 452] munmap(0x7f1c9e267000, 19163) = 0

[pid 451] mprotect(0x7f01504bb000, 8192, PROT_READ <unfinished ...>

[pid 452] openat(AT_FDCWD, "/dev/shm/sem.sem_empty", O_RDWR|O_NOFOLLOW|O_CLOEXEC <unfinished ...>

[pid 451] <... mprotect resumed>) = 0

[pid 452] <... openat resumed>) = 3

[pid 452] fstat(3, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

[pid 451] prlimit64(0, RLIMIT_STACK, NULL, <unfinished ...>

[pid 452] getrandom(<unfinished ...>

[pid 451] <... prlimit64 resumed>{rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 452] <... getrandom resumed>"\x52\xec\xf4\x31\x77\x3f\xf9\x57", 8, GRND_NONBLOCK) = 8

[pid 451] munmap(0x7f015047e000, 19163 <unfinished ...>

[pid 452] brk(NULL <unfinished ...>

[pid 451] <... munmap resumed>) = 0

[pid 452] <... brk resumed>) = 0x55bf94a50000

[pid 452] brk(0x55bf94a71000 <unfinished ...>

[pid 451] openat(AT_FDCWD, "/dev/shm/sem.sem_empty", O_RDWR|O_NOFOLLOW|O_CLOEXEC <unfinished ...>

[pid 452] <... brk resumed>) = 0x55bf94a71000

[pid 451] <... openat resumed>) = 3

[pid 452] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0 <unfinished ...>

[pid 451] fstat(3, <unfinished ...>

[pid 452] <... mmap resumed>) = 0x7f1c9e26b000

[pid 451] <... fstat resumed>{st_mode=S_IFREG|0644, st_size=32, ...}) = 0

[pid 452] close(3 <unfinished ...>

[pid 451] getrandom(<unfinished ...>

[pid 452] <... close resumed>) = 0

[pid 451] <... getrandom resumed>"\x3e\xf7\xb9\x09\xd9\x5e\x7c\x8b", 8, GRND_NONBLOCK) = 8

[pid 452] openat(AT_FDCWD, "/dev/shm/sem.sem_full", O_RDWR|O_NOFOLLOW|O_CLOEXEC <unfinished ...>

[pid 451] brk(NULL <unfinished ...>

[pid 452] <... openat resumed>) = 3

[pid 451] <... brk resumed>) = 0x561388fcd000

[pid 452] fstat(3, <unfinished ...>

[pid 451] brk(0x561388fee000 <unfinished ...>

[pid 452] <... fstat resumed>{st_mode=S_IFREG|0644, st_size=32, ...}) = 0

[pid 451] <... brk resumed>) = 0x561388fee000

[pid 452] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f1c9e26a000

[pid 452] close(3) = 0

[pid 452] openat(AT_FDCWD, "/dev/shm/shared_memory", O_RDWR|O_NOFOLLOW|O_CLOEXEC <unfinished ...>

[pid 451] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0 <unfinished ...>

[pid 452] <... openat resumed>) = 3

[pid 451] <... mmap resumed>) = 0x7f0150482000

[pid 452] mmap(NULL, 260, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0 <unfinished ...>

[pid 451] close(3 <unfinished ...>

[pid 452] <... mmap resumed>) = 0x7f1c9e269000

[pid 451] <... close resumed>) = 0

[pid 452] futex(0x7f1c9e26a000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 451] openat(AT_FDCWD, "/dev/shm/sem.sem_full", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3

[pid 451] fstat(3, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

[pid 451] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f0150481000

[pid 451] close(3) = 0

[pid 451] openat(AT_FDCWD, "/dev/shm/shared_memory", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3

[pid 451] mmap(NULL, 260, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f0150480000

[pid 451] futex(0x7f0150481000, FUTEX_WAKE, 1 <unfinished ...>

[pid 452] <... futex resumed>) = 0

[pid 451] <... futex resumed>) = 1

[pid 452] write(1, "Child 2 processed message:", 26 <unfinished ...> Child 2 processed message:

[pid 451] munmap(0x7f0150480000, 260 <unfinished ...>

[pid 452] <... write resumed>) = 26

[pid 452] write(1, "", 0 <unfinished ...>

[pid 451] <... munmap resumed>) = 0

[pid 452] <... write resumed>) = 0

[pid 451] close(3 <unfinished ...>

[pid 452] write(1, "\n", 1 <unfinished ...>

[pid 451] <... close resumed>) = 0

[pid 452] <... write resumed>) = 1

[pid 451] exit_group(0 <unfinished ...>

[pid 452] munmap(0x7f1c9e269000, 260 <unfinished ...>

[pid 451] <... exit_group resumed>) = ?

[pid 452] <... munmap resumed>) = 0

[pid 452] close(3) = 0 [pid 451]

+++ exited with 0 +++

[pid 452] exit_group(0 <unfinished ...>

[pid 450] <... wait4 resumed>NULL, 0, NULL) = 451

[pid 452] <... exit_group resumed>) = ?

[pid 450] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=451, si_uid=1000, si_status=0, si_utime=1 /* 0.01 s */, si_stime=1 /* 0.01 s */} ---

[pid 450] wait4(-1, <unfinished ...>

[pid 452] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL) = 452

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=452, si_uid=1000, si_status=0, si_utime=0, si_stime=1 / 0.01 s */} ---*

write(1, "Final message: ", 15Final message:) = 15

write(1, "", 0) = 0

write(1, "\n", 1) = 1

munmap(0x7f08ec145000, 32) = 0

munmap(0x7f08ec144000, 32) = 0

unlink("/dev/shm/sem.sem_empty") = 0

unlink("/dev/shm/sem.sem_full") = 0

munmap(0x7f08ec143000, 260) = 0

```
unlink("/dev/shm/shared_memory") = 0
exit_group(0) = ?
+++ exited with 0 +++
```

Вывод

Во время выполнения лабораторной работы я разработала программу, которая использует механизмы межпроцессного взаимодействия, такие как общая память и семафоры, для обработки строк, вводимых пользователем, с участием нескольких процессов. Основная сложность возникла при управлении ресурсами, такими как семафоры и общая память, а также в синхронизации процессов. Проблемы возникали из-за некорректной работы с семафорами и общей памятью в дочерних процессах, что приводило к возможным гонкам состояний и неправильному завершению процессов. Я решила эту проблему, добавив дополнительные проверки ошибок и убедившись, что дочерние процессы корректно завершаются.

В будущем хотелось бы больше времени уделить отладке и тестированию многозадачности и синхронизации между процессами, чтобы избежать подобных проблем с ресурсами. Также важно будет улучшить код дочерних процессов, чтобы они корректно работали с общей памятью и семафорами. В целом, эта работа была полезной и помогла мне лучше понять принципы работы с межпроцессным взаимодействием в операционной системе.