Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-214Б-24

Студент: Дылдин С.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 27.11.25

Москва, 2025

Постановка задачи

Вариант 9.

9 вариант) В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

Условие такое: нужно взять свою первую лабу и переделать её с использованием shared memory и memory mapping. Варианты остаются те же, что и у первой лабораторно

Общий метод и алгоритм решения

Использованные системные вызовы:

- int shm_open(const char *name, …); - создает или открывает объект разделяемой памяти
- int ftruncate(int fd, off_t length); - задает размер объекта разделяемой памяти
- sem_t *sem_open(const cahr *name, …); - создает или открывает именованный семафор
- int sem_wait(sem_t *sem); - захватывает семафор(лекрементирует счётчик), блокирует процесс до появления данных
- int sem_post(sem_t *sem); освобождает семафор(инкрементирует счётчик), сигнализирует о готовности данных
- int munmap(void *addr, size_t length); - отключает отображение памяти
- int shm_unlink(const char *name); удаляет имя объекта разделяемой памяти из системы

Описание:

Взаимодействие двух процессов.

- Родитель читает команды из файла и передает ребенку.
- Ребенок просыпается по семафору, парсит числа из памяти и выполняет деление, данные запихивает в буфер shared memory и сигнализирует родителю.
- При ошибке деления на 0 - процессы завершаются, ресурсы очищаются

Код программы

parent.c

```c
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
```

```c
#include <sys/stat.h>
#include <semaphore.h>
#include <time.h>

static char SERVER_PROGRAM_NAME[] = "div-server";

#define SHM_SIZE 8192

typedef struct {
   char data[SHM_SIZE];
   size_t size;
   bool finished;
} SharedData;

int main(int argc, char **argv){
   if (argc == 1){
      char msg[1024];
      uint32_t len = snprintf(msg, sizeof(msg), "usage: %s filename\n", argv[0]);
      write(STDERR_FILENO, msg, len);
      exit(EXIT_SUCCESS);
   }

   char progpath[1024];
   {
      ssize_t len = readlink("/proc/self/exe", progpath, sizeof(progpath) - 1);
      if (len == -1){
         const char msg[] = "error: failed to read full path of programm\n";
         write(STDERR_FILENO, msg, sizeof(msg));
         exit(EXIT_FAILURE);
      }
      progpath[len] = '\0';

      ssize_t i = len - 1;
      while (i >= 0 && progpath[i] != '/'){
         --i;
      }
      if (i >= 0){
         progpath[i] = '\0';
      } else{
         progpath[0] = '\0';
      }
   }

   char shm_in_name[256], shm_out_name[256];
   char sem_in_name[256], sem_out_name[256];
   pid_t pid = getpid();
   snprintf(shm_in_name, sizeof(shm_in_name), "/shm_in_%d", pid);
   snprintf(shm_out_name, sizeof(shm_out_name), "/shm_out_%d", pid);
   snprintf(sem_in_name, sizeof(sem_in_name), "/sem_in_%d", pid);
```

```
    snprintf(sem_out_name, sizeof(sem_out_name), "/sem_out_%d", pid);

    int shm_in = shm_open(shm_in_name, O_CREAT | O_RDWR, 0666);
    if (shm_in == -1){
        const char msg[] = "error: failed to create input shared memory\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    int shm_out = shm_open(shm_out_name, O_CREAT | O_RDWR, 0666);
    if (shm_out == -1){
        const char msg[] = "error: failed to create output shared memory\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        shm_unlink(shm_in_name);
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shm_in, sizeof(SharedData)) == -1 || ftruncate(shm_out,
sizeof(SharedData)) == -1){
        const char msg[] = "error: failed to set shared memory size\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        shm_unlink(shm_in_name);
        shm_unlink(shm_out_name);
        exit(EXIT_FAILURE);
    }

    SharedData *data_in = mmap(NULL, sizeof(SharedData), PROT_READ | PROT_WRITE,
MAP_SHARED, shm_in, 0);
    SharedData *data_out = mmap(NULL, sizeof(SharedData), PROT_READ | PROT_WRITE,
MAP_SHARED, shm_out, 0);

    if (data_in == MAP_FAILED || data_out == MAP_FAILED){
        const char msg[] = "error: failed to map shared memory\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        shm_unlink(shm_in_name);
        shm_unlink(shm_out_name);
        exit(EXIT_FAILURE);
    }

    data_in->size = 0;
    data_in->finished = false;
    data_out->size = 0;
    data_out->finished = false;

    sem_t *sem_in = sem_open(sem_in_name, O_CREAT, 0666, 0);
    sem_t *sem_out = sem_open(sem_out_name, O_CREAT, 0666, 0);

    if (sem_in == SEM_FAILED || sem_out == SEM_FAILED){
        const char msg[] = "error: failed to create semaphores\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        munmap(data_in, sizeof(SharedData));
```

```
        munmap(data_out, sizeof(SharedData));
        shm_unlink(shm_in_name);
        shm_unlink(shm_out_name);
        exit(EXIT_FAILURE);
    }

    const pid_t child = fork();
    switch (child){
        case (-1): {
            const char msg[] = "error failed to create new process\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        case(0):{
            {
                pid_t child_pid = getpid();
                char msg[64];
                const int32_t len = snprintf(msg, sizeof(msg), "%d: child process
created\n", child_pid);
                write(STDOUT_FILENO, msg, len);
            }

            char path[2048];
            snprintf(path, sizeof(path), "%s/%s", progpath, SERVER_PROGRAM_NAME);
            char *const args[] = {SERVER_PROGRAM_NAME, shm_in_name, shm_out_name,
sem_in_name, sem_out_name, NULL};
            int32_t status = execv(path, args);
            if (status == -1){
                const char msg[] = "error: failed to exec into new image\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
        } break;
        default: {
            {
                char msg[64];
                const int32_t len = snprintf(msg, sizeof(msg), "%d: parent process,
child PID %d\n", pid, child);
                write(STDOUT_FILENO, msg, len);
            }

            int32_t file = open(argv[1], O_RDONLY);
            if (file == -1){
                const char msg[] = "error: failed to open input file\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }

            char buf[4096];
            ssize_t bytes;
            bool child_alive = true;
```

```
        while (child_alive && (bytes = read(file, buf, sizeof(buf))) > 0){
            if ((size_t)bytes < SHM_SIZE){
                memcpy(data_in->data, buf, bytes);
                data_in->size = bytes;
                data_in->finished = false;

                sem_post(sem_in);
                sem_wait(sem_out);

                if (data_out->size > 0){
                    write(STDOUT_FILENO, data_out->data, data_out->size);
                }

                if (data_out->finished){
                    const char msg[] = "parent: child process terminated (div by
0)\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    child_alive = false;
                    break;
                }
            }
        }

        if (bytes == -1) {
            const char msg[] = "error: failed to read input file\n";
            write(STDERR_FILENO, msg, sizeof(msg));
        }

        data_in->finished = true;
        data_in->size = 0;
        sem_post(sem_in);

        close(file);

        int status;
        wait(&status);
        if (WIFEXITED(status)){
            int exitCode = WEXITSTATUS(status);
            if (exitCode != 0){
                const char msg[] = "parent: terminating due to child error\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
        }
        const char msg[] = "parent: programm completed successfully\n";
        write(STDOUT_FILENO, msg, sizeof(msg));

        munmap(data_in, sizeof(SharedData));
        munmap(data_out, sizeof(SharedData));
```

```
            shm_unlink(shm_in_name);
            shm_unlink(shm_out_name);
            sem_close(sem_in);
            sem_close(sem_out);
            sem_unlink(sem_in_name);
            sem_unlink(sem_out_name);
        } break;
    }
    return 0;
}
```

child.c

```c
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>

#define MAX_NUMS 100
#define SHM_SIZE 8192

typedef struct {
   char data[SHM_SIZE];
   size_t size;
   bool finished;
} SharedData;

int main(int argc, char **argv){
   if (argc != 5){
      const char msg[] = "error: incorrect arguments\n";
      write(STDERR_FILENO, msg, sizeof(msg));
      exit(EXIT_FAILURE);
   }

   char *shm_in_name = argv[1];
   char *shm_out_name = argv[2];
   char *sem_in_name = argv[3];
   char *sem_out_name = argv[4];

   // shared memory
   int shm_in = shm_open(shm_in_name, O_RDWR, 0666);
   int shm_out = shm_open(shm_out_name, O_RDWR, 0666);

   if (shm_in == -1 || shm_out == -1){
      const char msg[] = "error: failed to open shared memory\n";
      write(STDERR_FILENO, msg, sizeof(msg));
      exit(EXIT_FAILURE);
   }

   SharedData *data_in = mmap(NULL, sizeof(SharedData), PROT_READ |
PROT_WRITE, MAP_SHARED, shm_in, 0);
   SharedData *data_out = mmap(NULL, sizeof(SharedData), PROT_READ |
PROT_WRITE, MAP_SHARED, shm_out, 0);

   if (data_in == MAP_FAILED || data_out == MAP_FAILED){
```

```
        const char msg[] = "error: failed to map shared memory\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    // семафоры
    sem_t *sem_in = sem_open(sem_in_name, 0);
    sem_t *sem_out = sem_open(sem_out_name, 0);

    if (sem_in == SEM_FAILED || sem_out == SEM_FAILED){
        const char msg[] = "error: failed to open semaphores\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    pid_t pid = getpid();
    char str[4096];
    uint32_t strInd = 0;

    while(true){
        sem_wait(sem_in);
        data_out->size = 0;

        if (data_in->finished){
            char msg[128];
            int len = snprintf(msg, sizeof(msg), "PID %d successfully
terminated\n", pid);
            write(STDERR_FILENO, msg, len);
            break;
        }

        ssize_t bytes = data_in->size;
        char *buf = data_in->data;

        for(ssize_t i = 0; i < bytes; i++){
            if (buf[i] == '\n') {
                if (strInd == 0){
                    continue;
                }
                str[strInd] = '\0';

                float nums[MAX_NUMS];
                uint32_t cnt = 0;

                char *ptr = str;
                char *endptr = NULL;

                while(*ptr != '\0' && cnt < MAX_NUMS) {
                    while(*ptr == ' ' || *ptr == '\t'){
```

```
                ptr++;
            }
            if (*ptr == '\0'){
                break;
            }
            nums[cnt] = strtof(ptr, &endptr);
            if (endptr == ptr){
                break;
            }
            cnt++;
            ptr = endptr;
        }

        if (cnt < 2){
            const char msg[] = "error occured, need at least 2 nums\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            strInd = 0;
            continue;
        }

        float res = nums[0];
        bool divByZero = false;
        char output[4096];
        int outLen = snprintf(output, sizeof(output), "processing %.2f",
nums[0]);

        for (uint32_t j = 1; j < cnt; j++){
            if(nums[j] == 0.0f){
                char msg[256];
                int msgLen = snprintf(msg, sizeof(msg), "div by zero
occeured, child PID: %d is terminatied\n", pid);
                write(STDERR_FILENO, msg, msgLen);
                divByZero = true;
                break;
            }
            outLen += snprintf(output + outLen, sizeof(output) - outLen,
" / %.2f", nums[j]);
            res /= nums[j];
        }

        if (divByZero) {
            data_out->finished = true;
            data_out->size = 0;
            sem_post(sem_out);

            munmap(data_in, sizeof(SharedData));
            munmap(data_out, sizeof(SharedData));
            sem_close(sem_in);
            sem_close(sem_out);
            exit(EXIT_FAILURE);
```

```
            }

            outLen += snprintf(output + outLen, sizeof(output) - outLen, " =
%.6f\n", res);

            if (data_out->size + outLen <= SHM_SIZE) {
               memcpy(data_out->data + data_out->size, output, outLen);
               data_out->size += outLen;
            }

            data_out->finished = false;

            char log[256];
            int logLen = snprintf(log, sizeof(log), "PID: %d with result :
%.6f\n", pid, res);
            write(STDERR_FILENO, log, logLen);
            strInd = 0;

         } else {
            if(strInd < sizeof(str) - 1){
               str[strInd++] = buf[i];
            }
         }
      }
      sem_post(sem_out);
   }

   munmap(data_in, sizeof(SharedData));
   munmap(data_out, sizeof(SharedData));
   sem_close(sem_in);
   sem_close(sem_out);

   return 0;
}
```

Протокол работы программы

Тестирование:

```
/mnt/d/MAI/3semestr/OSI/laba3 → ./div-client inp.txt
40176: parent process, child PID 40177
40177: child process created
PID: 40177 with result : 1.000000
PID: 40177 with result : 3.000000
PID: 40177 with result : 4.000000
processing 10.00 / 2.00 / 5.00 = 1.000000
processing 9.00 / 3.00 / 1.00 = 3.000000
```

```
processing 8.00 / 2.00 = 4.000000
PID 40177 successfully terminated
parent: programm completed successfully
/mnt/d/MAI/3semestr/OSI/laba3 →
```

Strace:

```
/mnt/d/MAI/3semestr/OSI/laba3 → strace ./div-client inp.txt
execve("./div-client", ["./div-client", "inp.txt"], 0x7ffe0aa06108 /* 30 vars */) = 0
brk(NULL)                               = 0x56bf9f5b9000
mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7bc2a397f000
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=34311, ...}) = 0
mmap(NULL, 34311, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7bc2a3976000
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) =
0x7bc2a3600000
mmap(0x7bc2a3628000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7bc2a3628000
mmap(0x7bc2a37b0000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7bc2a37b0000

mmap(0x7bc2a37ff000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7bc2a37ff000
mmap(0x7bc2a3805000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7bc2a3805000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7bc2a3973000
arch_prctl(ARCH_SET_FS, 0x7bc2a3973740) = 0
set_tid_address(0x7bc2a3973a10)         = 40263
set_robust_list(0x7bc2a3973a20, 24)     = 0
rseq(0x7bc2a3974060, 0x20, 0, 0x53053053) = 0
mprotect(0x7bc2a37ff000, 16384, PROT_READ) = 0
mprotect(0x56bf70c03000, 4096, PROT_READ) = 0
mprotect(0x7bc2a39b7000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
```

rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7bc2a3976000, 34311)   = 0
readlink("/proc/self/exe", "/mnt/d/MAI/3semestr/OSI/laba3/di"..., 1023) = 40
getpid()   = 40263
openat(AT_FDCWD, "/dev/shm/shm_in_40263",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
openat(AT_FDCWD, "/dev/shm/shm_out_40263",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 4
ftruncate(3, 8208)   = 0
ftruncate(4, 8208)   = 0
mmap(NULL, 8208, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) =
0x7bc2a397c000
mmap(NULL, 8208, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) =
0x7bc2a3979000
openat(AT_FDCWD, "/dev/shm/sem.sem_in_40263",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)

getrandom("\xac\x7d\x3e\x6e\x19\x77\xc0\x6b", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.IxcEe1", 0x7ffcf11fb320,
AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.IxcEe1",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5
write(5, "\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, 5, 0) =
0x7bc2a3978000
link("/dev/shm/sem.IxcEe1", "/dev/shm/sem.sem_in_40263") = 0
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
getrandom("\xd0\x95\x97\xa2\x59\xe8\xfc\x10", 8, GRND_NONBLOCK) = 8
brk(NULL)   = 0x56bf9f5b9000
brk(0x56bf9f5da000)   = 0x56bf9f5da000
unlink("/dev/shm/sem.IxcEe1")   = 0
close(5)   = 0
openat(AT_FDCWD, "/dev/shm/sem.sem_out_40263",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)
getrandom("\xdb\x84\x18\xe9\xfe\x54\xb4\x03", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.HHbttZ", 0x7ffcf11fb320,
AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/dev/shm/sem.HHbttZ",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5

write(5, "\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, 5, 0) =
0x7bc2a3977000

link("/dev/shm/sem.HHbttZ", "/dev/shm/sem.sem_out_40263") = 0
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
unlink("/dev/shm/sem.HHbttZ")   = 0
close(5)   = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7bc2a3973a10) = 40264
40264: child process created
write(1, "40263: parent process, child PID"..., 3940263: parent process, child PID 40264
) = 39
openat(AT_FDCWD, "inp.txt", O_RDONLY)   = 5
read(5, "10 2 5\n9 3 1\n8 2\n", 4096)   = 17
futex(0x7bc2a3977000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0,
NULL, FUTEX_BITSET_MATCH_ANYPID: 40264 with result : 1.000000
PID: 40264 with result : 3.000000
PID: 40264 with result : 4.000000
) = 0
write(1, "processing 10.00 / 2.00 / 5.00 ="..., 117processing 10.00 / 2.00 / 5.00 = 1.000000
processing 9.00 / 3.00 / 1.00 = 3.000000
processing 8.00 / 2.00 = 4.000000
) = 117
read(5, "", 4096)   = 0
futex(0x7bc2a3978000, FUTEX_WAKE, 1PID 40264 successfully terminated
)   = 1
close(5)   = 0
wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 40264
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=40264, si_uid=1000,
 si_status=0, si_utime=0, si_stime=0} ---
write(1, "parent: programm completed succe"..., 41parent: programm completed
 successfully
) = 41
munmap(0x7bc2a397c000, 8208)   = 0
munmap(0x7bc2a3979000, 8208)   = 0
unlink("/dev/shm/shm_in_40263")   = 0
unlink("/dev/shm/shm_out_40263")   = 0
munmap(0x7bc2a3978000, 32)   = 0
munmap(0x7bc2a3977000, 32)   = 0
unlink("/dev/shm/sem.sem_in_40263")   = 0
unlink("/dev/shm/sem.sem_out_40263")   = 0
exit_group(0)   = ?
+++ exited with 0 +++
/mnt/d/MAI/3semestr/OSI/laba3 →

Вывод

Научился организовывать взаимодействие между процессами через разделяемую память и синхронизировать их работу с помощью именованных семафоров.