

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-214Б-24

Студент: Шведова Е. В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 26.11.25

Москва, 2025

Постановка задачи

Вариант 3.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через `ripe1`, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через `ripe2`. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа на последующие, а результат выводит в файл. 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 `int`. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

В рамках лабораторной работы создана программа, использующая разделяемую память и семафоры для межпроцессного взаимодействия. Серверный процесс (`parent.c`) и клиентский процесс (`child.c`) работают как независимые программы.

Серверный процесс создает область разделяемой памяти (`shm_open`) и семафор (`sem_open`) для синхронизации. После запроса имени файла у пользователя, процесс ожидает ввода чисел для обработки.

Клиентский процесс открывает созданные объекты, читает имя файла и открывает его для записи. Получая числа через разделяемую память, процесс выполняет деление первого числа на последующие и записывает результаты в файл. При обнаружении деления на ноль оба процесса завершаются.

Синхронизация доступа к разделяемой памяти обеспечивается операциями `sem_wait` и `sem_post`. По завершении работы ресурсы освобождаются с помощью `munmap`, `shm_unlink`, `sem_close` и `sem_unlink`.

Использованные системные вызовы:

- `int shm_open(const char *name, int oflag, mode_t mode);` – создает объект разделяемой памяти
- `int ftruncate(int fd, off_t length);` – устанавливает размер разделяемой памяти
- `void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);` – отображает разделяемую память в адресное пространство
- `sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);` – создает именованный семафор
- `int sem_wait(sem_t *sem);` – уменьшает значение семафора перед записью в shared memory
- `int sem_post(sem_t *sem);` – увеличивает значение семафора после записи в shared memory

- int sem_wait(sem_t *sem); – ожидает освобождения семафора для чтения ответа
- int sem_post(sem_t *sem); – освобождает семафор после чтения ответа
- int munmap(void *addr, size_t length); – удаляет отображение разделяемой памяти
- int shm_unlink(const char *name); – удаляет объект разделяемой памяти
- int sem_close(sem_t *sem); – закрывает семафор
- int sem_unlink(const char *name); – удаляет именованный семафор
- int shm_open(const char *name, int oflag, mode_t mode); – открывает существующий объект разделяемой памяти
- void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset); – отображает разделяемую память в адресное пространство
- sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value); – открывает существующий именованный семафор
- int sem_wait(sem_t *sem); – уменьшает значение семафора перед чтением из shared memory
- int sem_post(sem_t *sem); – увеличивает значение семафора после чтения из shared memory
- int open(const char *pathname, int flags, mode_t mode); – открывает файл для записи результатов
- int close(int fd); – закрывает файл
- int munmap(void *addr, size_t length); – удаляет отображение разделяемой памяти
- int sem_close(sem_t *sem); – закрывает семафор

Код программы

parent.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <semaphore.h>
#include <time.h>

#define SHM_SIZE 4096

typedef struct {
    char filename[256];
    char numbers[1024];
    char result[256];
    int has_data;
    int division_by_zero;
    int shutdown;
} shared_data_t;

int main(int argc, char **argv) {
    if (argc != 2) {
        printf("Usage: %s <unique_id>\n", argv[0]);
```

```

    return 1;
}

char shm_name[256], sem_name[256];
snprintf(shm_name, sizeof(shm_name), "/lab_shm_%s", argv[1]);
snprintf(sem_name, sizeof(sem_name), "/lab_sem_%s", argv[1]);

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

if (ftruncate(shm_fd, SHM_SIZE) == -1) {
    perror("ftruncate failed");
    return 1;
}

shared_data_t *shared_data = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
if (shared_data == MAP_FAILED) {
    perror("mmap failed");
    return 1;
}

memset(shared_data, 0, sizeof(shared_data_t));

sem_t *semaphore = sem_open(sem_name, O_CREAT, 0666, 1);
if (semaphore == SEM_FAILED) {
    perror("sem_open failed");
    return 1;
}

printf("Server started with ID: %s\n", argv[1]);
printf("Shared memory: %s\n", shm_name);
printf("Semaphore: %s\n", sem_name);
printf("Waiting for client...\n");

printf("Enter filename: ");
char filename[256];
if (fgets(filename, sizeof(filename), stdin) == NULL) {
    printf("Error reading filename\n");
    return 1;
}
filename[strcspn(filename, "\n")] = '\0';

sem_wait(semaphore);
strcpy(shared_data->filename, filename);
sem_post(semaphore);

```

```

printf("Enter numbers separated by spaces (e.g., '12 3 4'):\n");
printf("Press Enter on empty line to exit\n");

char input[1024];
while (fgets(input, sizeof(input), stdin)) {
    if (input[0] == '\n') {
        break;
    }

    input[strcspn(input, "\n")] = '\0';

    sem_wait(semaphore);
    strcpy(shared_data->numbers, input);
    shared_data->has_data = 1;
    shared_data->division_by_zero = 0;
    memset(shared_data->result, 0, sizeof(shared_data->result));
    sem_post(semaphore);

    int processed = 0;
    while (!processed) {
        sem_wait(semaphore);
        if (!shared_data->has_data) {
            processed = 1;
            if (shared_data->division_by_zero) {
                printf("Error: Division by zero! Exiting...\n");
                sem_post(semaphore);

                munmap(shared_data, SHM_SIZE);
                close(shm_fd);
                shm_unlink(shm_name);
                sem_close(semaphore);
                sem_unlink(sem_name);
                return 1;
            }
        }
        sem_post(semaphore);
        usleep(100000);
    }

    sem_wait(semaphore);
    if (strlen(shared_data->result) > 0) {
        printf("Result: %s", shared_data->result);
    }
    sem_post(semaphore);
}

sem_wait(semaphore);
shared_data->shutdown = 1;

```

```

    sem_post(semaphore);

    printf("Program finished successfully.\n");

    munmap(shared_data, SHM_SIZE);
    close(shm_fd);
    shm_unlink(shm_name);
    sem_close(semaphore);
    sem_unlink(sem_name);

    return 0;
}

```

child.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <semaphore.h>

```

```
#define SHM_SIZE 4096
```

```

typedef struct {
    char filename[256];
    char numbers[1024];
    char result[256];
    int has_data;
    int division_by_zero;
    int shutdown;
} shared_data_t;

```

```

int main(int argc, char **argv) {
    if (argc != 2) {
        printf("Usage: %s <unique_id>\n", argv[0]);
        return 1;
    }
}

```

```

char shm_name[256], sem_name[256];
snprintf(shm_name, sizeof(shm_name), "/lab_shm_%s", argv[1]);
snprintf(sem_name, sizeof(sem_name), "/lab_sem_%s", argv[1]);

```

```

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

```

```

    return 1;
}

shared_data_t *shared_data = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
if (shared_data == MAP_FAILED) {
    perror("mmap failed");
    return 1;
}

sem_t *semaphore = sem_open(sem_name, 0);
if (semaphore == SEM_FAILED) {
    perror("sem_open failed");
    return 1;
}

printf("Client started with ID: %s\n", argv[1]);

char filename[256];
sem_wait(semaphore);
strcpy(filename, shared_data->filename);
sem_post(semaphore);

FILE *file = fopen(filename, "w");
if (file == NULL) {
    printf("Error: failed to open file %s\n", filename);
    return 1;
}

printf("Output file: %s\n", filename);

while (1) {
    sem_wait(semaphore);

    if (shared_data->shutdown) {
        sem_post(semaphore);
        break;
    }

    if (shared_data->has_data) {
        char numbers[1024];
        strcpy(numbers, shared_data->numbers);
        sem_post(semaphore);

        int nums[100];
        int count = 0;
    }
}

```

```

char *token = strtok(numbers, " \t");

while (token != NULL && count < 100) {
    nums[count++] = atoi(token);
    token = strtok(NULL, " \t");
}

if (count < 2) {
    const char *error_msg = "Error: need at least 2 numbers\n";
    fprintf(file, "%s", error_msg);
    fflush(file);

    sem_wait(semaphore);
    strcpy(shared_data->result, error_msg);
    shared_data->has_data = 0;
    sem_post(semaphore);
    continue;
}

int division_by_zero = 0;
for (int i = 1; i < count; i++) {
    if (nums[i] == 0) {
        division_by_zero = 1;
        break;
    }
}

if (division_by_zero) {
    const char *error = "Error: division by zero\n";
    fprintf(file, "%s", error);
    fflush(file);

    sem_wait(semaphore);
    shared_data->division_by_zero = 1;
    shared_data->has_data = 0;
    sem_post(semaphore);

    fclose(file);
    return 1;
}

float result = (float)nums[0];
for (int i = 1; i < count; i++) {
    result /= nums[i];
}

fprintf(file, "%d", nums[0]);

```

```

        for (int i = 1; i < count; i++) {
            fprintf(file, " / %d", nums[i]);
        }
        fprintf(file, " = %.2f\n", result);
        fflush(file);

        char response[50];
        sprintf(response, sizeof(response), "% .2f\n", result);

        sem_wait(semaphore);
        strcpy(shared_data->result, response);
        shared_data->has_data = 0;
        sem_post(semaphore);
    } else {
        sem_post(semaphore);
        usleep(100000);
    }
}

fclose(file);
printf("Client finished successfully.\n");

munmap(shared_data, SHM_SIZE);
close(shm_fd);
sem_close(semaphore);

return 0;
}

```

Протокол работы программы

```
rane@Assus:~/op/Lab3/src$ ./parent test1
Server started with ID: test1
Shared memory: /lab_shm_test1
Semaphore: /lab_sem_test1
Waiting for client...
Enter filename: result.txt
Enter numbers separated by spaces (e.g., '12 3 4'):
Press Enter on empty line to exit
12 3 4
Result: 1.00
10 4 2
Result: 1.25
```

10 0 5

Error: Division by zero! Exiting...

crane@Assus:~/op/Lab3/src\$./child test1

Client started with ID: test1

Output file: result.txt

crane@Assus:~/op/Lab3/src\$./parent test1

Server started with ID: test1

Shared memory: /lab_shm_test1

Semaphore: /lab_sem_test1

Waiting for client...

Enter filename: result.txt

Enter numbers separated by spaces (e.g., '12 3 4'):

Press Enter on empty line to exit

1000000 1000 10

Result: 100.00

100 -5 2

Result: -10.00

5

Result: Error: need at least 2 numbers

Program finished successfully.

crane@Assus:~/op/Lab3/src\$./child test1

Client started with ID: test1

Output file: result.txt

Client finished successfully.

crane@Assus:~/op/Lab3/src\$ strace -f ./parent test1

execve("./parent", ["../parent", "test1"], 0x7fff8b687170 /* 78 vars */) = 0

brk(NULL) = 0x5d9dae544000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x74fb283bc000


```

unlink("/dev/shm/sem.gok0H0")      = 0
close(4)                      = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "Server started with ID: test1\n", 30Server started with ID: test1
) = 30
write(1, "Shared memory: /lab_shm_test1\n", 30Shared memory: /lab_shm_test1
) = 30
write(1, "Semaphore: /lab_sem_test1\n", 26Semaphore: /lab_sem_test1
) = 26
write(1, "Waiting for client...\n", 22Waiting for client...
) = 22
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "Enter filename: ", 16Enter filename: )      = 16
read(0, result.txt
"result.txt\n", 1024)      = 11
write(1, "Enter numbers separated by space"..., 52Enter numbers separated by spaces (e.g., '12 3
4'):
) = 52
write(1, "Press Enter on empty line to exit"..., 34Press Enter on empty line to exit
) = 34
read(0, 12 3 4
"12 3 4\n", 1024)      = 7
clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=100000000}, NULL) = 0
clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=100000000}, NULL) = 0
write(1, "Result: 1.00\n", 13Result: 1.00
)      = 13
read(0,
"\n", 1024)      = 1
write(1, "Program finished successfully.\n", 31Program finished successfully.
) = 31
munmap(0x74fb283bb000, 4096)      = 0
close(3)                      = 0
unlink("/dev/shm/lab_shm_test1")      = 0
munmap(0x74fb283ba000, 32)      = 0
unlink("/dev/shm/sem.lab_sem_test1")      = 0
exit_group(0)      = ?
+++ exited with 0 +++
crane@Assus:~/op/Lab3/src$
```

Вывод

В ходе выполнения лабораторной работы была успешно реализована программа межпроцессного взаимодействия с использованием механизмов разделяемой памяти и семафоров.