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-214БВ-24

Студент: Зайцев Т.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 10.12.25

Москва, 2025

Постановка задачи

Вариант 11.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. 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.

Использованные системные вызовы:

1. Управление процессами:
   fork() - создание нового процесса
   execv() - замена образа процесса (используется через execl())
   waitpid() - ожидание завершения дочернего процесса
   getpid() - получение PID текущего процесса
2. Работа с shared memory:
   shm_open() - создание/открытие именованного shared memory объекта
   ftruncate() - установка размера shared memory
   mmap() - отображение shared memory в адресное пространство процесса
   munmap() - удаление отображения shared memory
   shm_unlink() - удаление именованного shared memory объекта
3. Работа с семафорами (POSIX):
   sem_open() - создание/открытие именованного семафора
   sem_wait() - захват семафора (уменьшение значения)
   sem_post() - освобождение семафора (увеличение значения)
   sem_close() - закрытие семафора
   sem_unlink() - удаление именованного семафора
4. Файловые операции и ввод/вывод:
   open() - открытие файла (используется в shm_open внутри)
   close() - закрытие файлового дескриптора
   read() - чтение из файлового дескриптора
   write() - запись в файловый дескриптор
   ftruncate() - изменение размера файла (для shared memory)

Общий метод и алгоритм решения

Программа создает три именованных shared memory объекта для взаимодействия между процессами:

shm1 - для передачи данных от родительского процесса к Child1
shm2 - для передачи данных от Child1 к Child2
shm3 - для возврата результата от Child2 к родительскому процессу

Для синхронизации доступа к shared memory создаются три семафора:

sem1 - защищает доступ к shm1
sem2 - защищает доступ к shm2
sem3 - защищает доступ к shm3

Дополнительно создаются два семафора синхронизации запуска:

ready1 - сигнализирует о готовности Child1
ready2 - сигнализирует о готовности Child2

Алгоритм работы:

1. Получить PID текущего процесса
2. Сгенерировать уникальные имена для shared memory и семафоров на основе PID
3. Создать три shared memory объекта (shm_open + ftruncate)
4. Отобразить shared memory в адресное пространство (mmap)
5. Создать пять семафоров:
   - sem1, sem2, sem3 (начальное значение = 1) - как мьютексы
   - ready1, ready2 (начальное значение = 0) - для синхронизации запуска
6. Инициализировать структуры shared memory (length = 0)
7. Создать Child1 через fork() + execv():
   - Передать PID родителя как аргумент
   - Child1 откроет существующие shared memory объекты
   - Child1 просигнализирует о готовности (sem_post(ready1))
8. Создать Child2 через fork() + execv():
   - Передать PID родителя как аргумент
   - Child2 откроет существующие shared memory объекты
   - Child2 просигнализирует о готовности (sem_post(ready2))
9. Ожидать готовности обоих детей (sem_wait(ready1), sem_wait(ready2))
10. РОДИТЕЛЬ: Читает строку от пользователя
11. РОДИТЕЛЬ: Блокирует sem1, записывает данные в shm1, разблокирует sem1
12. CHILD1: Обнаруживает данные в shm1 (периодическая проверка)
13. CHILD1: Блокирует sem1, читает данные из shm1, разблокирует sem1
14. CHILD1: Преобразует текст в верхний регистр (toupper)
15. CHILD1: Блокирует sem2, записывает результат в shm2, разблокирует sem2
16. CHILD2: Обнаруживает данные в shm2
17. CHILD2: Блокирует sem2, читает данные из shm2, разблокирует sem2
18. CHILD2: Заменяет пробельные символы на подчеркивания
19. CHILD2: Блокирует sem3, записывает результат в shm3, разблокирует sem3
20. РОДИТЕЛЬ: Обнаруживает данные в shm3 (периодическая проверка)
21. РОДИТЕЛЬ: Блокирует sem3, читает результат из shm3, разблокирует sem3
22. РОДИТЕЛЬ: Выводит результат пользователю
23. РОДИТЕЛЬ: При пустой строке или ошибке записывает UINT32_MAX в shm1

24. CHILD1: Обнаруживает UINT32_MAX, передает его в shm2, завершается
25. CHILD2: Обнаруживает UINT32_MAX в shm2, завершается
26. РОДИТЕЛЬ: Ожидает завершения детей (waitpid)
27. Все процессы закрывают и удаляют shared memory и семафоры

Код программы

Parent.c

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

#define BUFFER_SIZE 1024
#define SHM_SIZE (BUFFER_SIZE + sizeof(uint32_t))

typedef struct {
   uint32_t length;
   char data[BUFFER_SIZE];
} shm_data;

void generate_names(pid_t pid, char* shm1, char* shm2, char* shm3,
                    char* sem1, char* sem2, char* sem3,
                    char* ready1, char* ready2) {
   snprintf(shm1, 64, "/shm1-%d", pid);
   snprintf(shm2, 64, "/shm2-%d", pid);
   snprintf(shm3, 64, "/shm3-%d", pid);
   snprintf(sem1, 64, "/sem1-%d", pid);
   snprintf(sem2, 64, "/sem2-%d", pid);
   snprintf(sem3, 64, "/sem3-%d", pid);
   snprintf(ready1, 64, "/ready1-%d", pid);
   snprintf(ready2, 64, "/ready2-%d", pid);
}

int main() {
   pid_t parent_pid = getpid();
```

```
char shm1_name[64], shm2_name[64], shm3_name[64];
char sem1_name[64], sem2_name[64], sem3_name[64];
char ready1_name[64], ready2_name[64];

generate_names(parent_pid, shm1_name, shm2_name, shm3_name,
               sem1_name, sem2_name, sem3_name, ready1_name, ready2_name);

int shm1_fd = shm_open(shm1_name, O_RDWR | O_CREAT | O_EXCL, 0600);
if (shm1_fd == -1) {
    const char msg[] = "error: failed to create SHM1\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

int shm2_fd = shm_open(shm2_name, O_RDWR | O_CREAT | O_EXCL, 0600);
if (shm2_fd == -1) {
    const char msg[] = "error: failed to create SHM2\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

int shm3_fd = shm_open(shm3_name, O_RDWR | O_CREAT | O_EXCL, 0600);
if (shm3_fd == -1) {
    const char msg[] = "error: failed to create SHM3\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

if (ftruncate(shm1_fd, SHM_SIZE) == -1) {
    const char msg[] = "error: failed to resize SHM1\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

if (ftruncate(shm2_fd, SHM_SIZE) == -1) {
    const char msg[] = "error: failed to resize SHM2\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

if (ftruncate(shm3_fd, SHM_SIZE) == -1) {
```

```
        const char msg[] = "error: failed to resize SHM3\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    shm_data* shm1 = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm1_fd,
0);
    if (shm1 == MAP_FAILED) {
        const char msg[] = "error: failed to map SHM1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    shm_data* shm2 = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm2_fd,
0);
    if (shm2 == MAP_FAILED) {
        const char msg[] = "error: failed to map SHM2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    shm_data* shm3 = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm3_fd,
0);
    if (shm3 == MAP_FAILED) {
        const char msg[] = "error: failed to map SHM3\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    sem_t *sem1 = sem_open(sem1_name, O_CREAT | O_EXCL, 0600, 1);
    if (sem1 == SEM_FAILED) {
        const char msg[] = "error: failed to create semaphore1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    sem_t *sem2 = sem_open(sem2_name, O_CREAT | O_EXCL, 0600, 1);
    if (sem2 == SEM_FAILED) {
        const char msg[] = "error: failed to create semaphore2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }
```

```
sem_t *sem3 = sem_open(sem3_name, O_CREAT | O_EXCL, 0600, 1);
if (sem3 == SEM_FAILED) {
    const char msg[] = "error: failed to create semaphore3\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

sem_t *ready_sem1 = sem_open(ready1_name, O_CREAT | O_EXCL, 0600, 0);
if (ready_sem1 == SEM_FAILED) {
    const char msg[] = "error: failed to create ready semaphore1\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

sem_t *ready_sem2 = sem_open(ready2_name, O_CREAT | O_EXCL, 0600, 0);
if (ready_sem2 == SEM_FAILED) {
    const char msg[] = "error: failed to create ready semaphore2\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

shm1->length = 0;
shm2->length = 0;
shm3->length = 0;

const char parent_msg[] = "Parent started. Enter strings (empty line to exit):\n";
if (write(STDOUT_FILENO, parent_msg, sizeof(parent_msg) - 1) == -1) {
    const char msg[] = "error: failed to write to stdout\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

pid_t child1 = fork();
if (child1 == -1) {
    const char msg[] = "error: failed to fork child1\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

if (child1 == 0) {
    char pid_str[20];
```

```
        snprintf(pid_str, sizeof(pid_str), "%d", parent_pid);
        execl("./child1", "child1", pid_str, NULL);
        const char msg[] = "error: failed to exec child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    pid_t child2 = fork();
    if (child2 == -1) {
        const char msg[] = "error: failed to fork child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    if (child2 == 0) {
        char pid_str[20];
        snprintf(pid_str, sizeof(pid_str), "%d", parent_pid);
        execl("./child2", "child2", pid_str, NULL);
        const char msg[] = "error: failed to exec child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    const char waiting_msg[] = "Waiting for children.\n";
    write(STDOUT_FILENO, waiting_msg, sizeof(waiting_msg) - 1);

    sem_wait(ready_sem1);
    sem_wait(ready_sem2);

    const char ready_msg[] = "Children ready!\n";
    write(STDOUT_FILENO, ready_msg, sizeof(ready_msg) - 1);

    sem_close(ready_sem1);
    sem_close(ready_sem2);
    sem_unlink(ready1_name);
    sem_unlink(ready2_name);

    char buffer[BUFFER_SIZE];

    while (1) {
        const char prompt[] = "Ввод: ";
        if (write(STDOUT_FILENO, prompt, sizeof(prompt) - 1) == -1) {
```

```
        const char msg[] = "error: failed to write prompt\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        break;
    }

    ssize_t bytes = read(STDIN_FILENO, buffer, sizeof(buffer));
    if (bytes == -1) {
        const char msg[] = "error: failed to read from stdin\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        break;
    }

    if (bytes <= 0) break;

    if (bytes > 0 && buffer[bytes-1] == '\n') {
        buffer[--bytes] = '\0';
    }

    if (bytes == 0) break;

    if (sem_wait(sem1) == -1) {
        const char msg[] = "error: sem_wait failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        break;
    }

    shm1->length = bytes;
    memcpy(shm1->data, buffer, bytes + 1);

    if (sem_post(sem1) == -1) {
        const char msg[] = "error: sem_post failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        break;
    }

    int received = 0;
    int attempts = 0;
    while (!received && attempts < 1000) {
        if (sem_wait(sem3) == -1) {
            const char msg[] = "error: sem_wait failed\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            break;
```

```
        }

        if (shm3->length > 0) {
            const char result_msg[] = "Result: ";
            if (write(STDOUT_FILENO, result_msg, sizeof(result_msg) - 1) == -1) {
                const char msg[] = "error: failed to write result\n";
                write(STDERR_FILENO, msg, sizeof(msg) - 1);
            }

            if (write(STDOUT_FILENO, shm3->data, shm3->length) == -1) {
                const char msg[] = "error: failed to write data\n";
                write(STDERR_FILENO, msg, sizeof(msg) - 1);
            }

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

            shm3->length = 0;
            received = 1;
        }

        if (sem_post(sem3) == -1) {
            const char msg[] = "error: sem_post failed\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            break;
        }

        if (!received) {
            usleep(10000);
            attempts++;
        }
    }

    if (!received) {
        const char msg[] = "error: timeout waiting for child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        break;
    }
}
```

```
    if (sem_wait(sem1) == -1) {
        const char msg[] = "error: sem_wait failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    } else {
        shm1->length = UINT32_MAX;
        if (sem_post(sem1) == -1) {
            const char msg[] = "error: sem_post failed\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
        }
    }

    if (waitpid(child1, NULL, 0) == -1) {
        const char msg[] = "error: waitpid for child1 failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (waitpid(child2, NULL, 0) == -1) {
        const char msg[] = "error: waitpid for child2 failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (sem_close(sem1) == -1) {
        const char msg[] = "error: sem_close failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (sem_close(sem2) == -1) {
        const char msg[] = "error: sem_close failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (sem_close(sem3) == -1) {
        const char msg[] = "error: sem_close failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (sem_unlink(sem1_name) == -1) {
        const char msg[] = "error: sem_unlink failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (sem_unlink(sem2_name) == -1) {
```

```
        const char msg[] = "error: sem_unlink failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (sem_unlink(sem3_name) == -1) {
        const char msg[] = "error: sem_unlink failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    sem_unlink(ready1_name);
    sem_unlink(ready2_name);

    if (munmap(shm1, SHM_SIZE) == -1) {
        const char msg[] = "error: munmap failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (munmap(shm2, SHM_SIZE) == -1) {
        const char msg[] = "error: munmap failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (munmap(shm3, SHM_SIZE) == -1) {
        const char msg[] = "error: munmap failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (shm_unlink(shm1_name) == -1) {
        const char msg[] = "error: shm_unlink failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (shm_unlink(shm2_name) == -1) {
        const char msg[] = "error: shm_unlink failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (shm_unlink(shm3_name) == -1) {
        const char msg[] = "error: shm_unlink failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }
```

```
    if (close(shm1_fd) == -1) {
        const char msg[] = "error: close failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (close(shm2_fd) == -1) {
        const char msg[] = "error: close failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (close(shm3_fd) == -1) {
        const char msg[] = "error: close failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    const char end_msg[] = "Parent finished.\n";
    write(STDOUT_FILENO, end_msg, sizeof(end_msg) - 1);

    return 0;
}
```

Child1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <unistd.h>
#include <errno.h>

#define BUFFER_SIZE 1024
#define SHM_SIZE (BUFFER_SIZE + sizeof(uint32_t))

typedef struct {
    uint32_t length;
    char data[BUFFER_SIZE];
} shm_data;

int main(int argc, char* argv[]) {
    if (argc != 2) {
        const char msg[] = "Usage: child1 <parent_pid>\n";
```

```
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    pid_t parent_pid = atoi(argv[1]);

    char shm1_name[64], shm2_name[64], sem1_name[64], sem2_name[64];
    char ready1_name[64];

    snprintf(shm1_name, sizeof(shm1_name), "/shm1-%d", parent_pid);
    snprintf(shm2_name, sizeof(shm2_name), "/shm2-%d", parent_pid);
    snprintf(sem1_name, sizeof(sem1_name), "/sem1-%d", parent_pid);
    snprintf(sem2_name, sizeof(sem2_name), "/sem2-%d", parent_pid);
    snprintf(ready1_name, sizeof(ready1_name), "/ready1-%d", parent_pid);

    int shm1_fd = shm_open(shm1_name, O_RDWR, 0);
    if (shm1_fd == -1) {
        const char msg[] = "error: failed to open SHM1 in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    int shm2_fd = shm_open(shm2_name, O_RDWR, 0);
    if (shm2_fd == -1) {
        const char msg[] = "error: failed to open SHM2 in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    shm_data* shm1 = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm1_fd,
0);
    if (shm1 == MAP_FAILED) {
        const char msg[] = "error: failed to map SHM1 in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    shm_data* shm2 = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm2_fd,
0);
    if (shm2 == MAP_FAILED) {
        const char msg[] = "error: failed to map SHM2 in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    sem_t *sem1 = sem_open(sem1_name, 0);
```

```
if (sem1 == SEM_FAILED) {
    const char msg[] = "error: failed to open semaphore1 in child1\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

sem_t *sem2 = sem_open(sem2_name, 0);
if (sem2 == SEM_FAILED) {
    const char msg[] = "error: failed to open semaphore2 in child1\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

sem_t *ready_sem1 = sem_open(ready1_name, 0);
if (ready_sem1 == SEM_FAILED) {
    const char msg[] = "error: failed to open ready semaphore1 in child1\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

const char start_msg[] = "Child1 gotov!\n";
if (write(STDOUT_FILENO, start_msg, sizeof(start_msg) - 1) == -1) {
    const char msg[] = "error: failed to write start message\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
}

sem_post(ready_sem1);
sem_close(ready_sem1);

int running = 1;

while (running) {
    if (sem_wait(sem1) == -1) {
        const char msg[] = "error: sem_wait failed in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        break;
    }

    if (shm1->length > 0) {
        if (shm1->length == UINT32_MAX) {
            running = 0;
        } else {
            for (int i = 0; i < shm1->length && shm1->data[i]; i++) {
                shm1->data[i] = toupper(shm1->data[i]);
            }
```

```
            if (sem_wait(sem2) == -1) {
                const char msg[] = "error: sem_wait failed in child1\n";
                write(STDERR_FILENO, msg, sizeof(msg) - 1);
                sem_post(sem1);
                break;
            }

            shm2->length = shm1->length;
            memcpy(shm2->data, shm1->data, shm1->length + 1);

            if (sem_post(sem2) == -1) {
                const char msg[] = "error: sem_post failed in child1\n";
                write(STDERR_FILENO, msg, sizeof(msg) - 1);
                sem_post(sem1);
                break;
            }

            shm1->length = 0;
        }
    }

    if (sem_post(sem1) == -1) {
        const char msg[] = "error: sem_post failed in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        break;
    }
}

if (sem_wait(sem2) == -1) {
    const char msg[] = "error: sem_wait failed in child1\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
} else {
    shm2->length = UINT32_MAX;
    if (sem_post(sem2) == -1) {
        const char msg[] = "error: sem_post failed in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }
}

if (sem_close(sem1) == -1) {
    const char msg[] = "error: sem_close failed in child1\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
}

if (sem_close(sem2) == -1) {
    const char msg[] = "error: sem_close failed in child1\n";
```

```
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (munmap(shm1, SHM_SIZE) == -1) {
        const char msg[] = "error: munmap failed in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (munmap(shm2, SHM_SIZE) == -1) {
        const char msg[] = "error: munmap failed in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (close(shm1_fd) == -1) {
        const char msg[] = "error: close failed in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (close(shm2_fd) == -1) {
        const char msg[] = "error: close failed in child1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    const char end_msg[] = "Child1 finished.\n";
    if (write(STDOUT_FILENO, end_msg, sizeof(end_msg) - 1) == -1) {
        const char msg[] = "error: failed to write finish message\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    return 0;
}
```

Child2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <unistd.h>
#include <errno.h>

#define BUFFER_SIZE 1024
#define SHM_SIZE (BUFFER_SIZE + sizeof(uint32_t))
```

```
typedef struct {
   uint32_t length;
   char data[BUFFER_SIZE];
} shm_data;

int main(int argc, char* argv[]) {
   if (argc != 2) {
       const char msg[] = "Usage: child2 <parent_pid>\n";
       write(STDERR_FILENO, msg, sizeof(msg) - 1);
       exit(EXIT_FAILURE);
   }

   pid_t parent_pid = atoi(argv[1]);

   char shm2_name[64], shm3_name[64], sem2_name[64], sem3_name[64];
   char ready2_name[64];

   snprintf(shm2_name, sizeof(shm2_name), "/shm2-%d", parent_pid);
   snprintf(shm3_name, sizeof(shm3_name), "/shm3-%d", parent_pid);
   snprintf(sem2_name, sizeof(sem2_name), "/sem2-%d", parent_pid);
   snprintf(sem3_name, sizeof(sem3_name), "/sem3-%d", parent_pid);
   snprintf(ready2_name, sizeof(ready2_name), "/ready2-%d", parent_pid);

   int shm2_fd = shm_open(shm2_name, O_RDWR, 0);
   if (shm2_fd == -1) {
       const char msg[] = "error: failed to open SHM2 in child2\n";
       write(STDERR_FILENO, msg, sizeof(msg) - 1);
       exit(EXIT_FAILURE);
   }

   int shm3_fd = shm_open(shm3_name, O_RDWR, 0);
   if (shm3_fd == -1) {
       const char msg[] = "error: failed to open SHM3 in child2\n";
       write(STDERR_FILENO, msg, sizeof(msg) - 1);
       exit(EXIT_FAILURE);
   }

   shm_data* shm2 = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm2_fd,
0);
   if (shm2 == MAP_FAILED) {
       const char msg[] = "error: failed to map SHM2 in child2\n";
       write(STDERR_FILENO, msg, sizeof(msg) - 1);
       exit(EXIT_FAILURE);
   }
```

```
    shm_data* shm3 = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm3_fd,
0);
    if (shm3 == MAP_FAILED) {
        const char msg[] = "error: failed to map SHM3 in child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    sem_t *sem2 = sem_open(sem2_name, 0);
    if (sem2 == SEM_FAILED) {
        const char msg[] = "error: failed to open semaphore2 in child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    sem_t *sem3 = sem_open(sem3_name, 0);
    if (sem3 == SEM_FAILED) {
        const char msg[] = "error: failed to open semaphore3 in child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    sem_t *ready_sem2 = sem_open(ready2_name, 0);
    if (ready_sem2 == SEM_FAILED) {
        const char msg[] = "error: failed to open ready semaphore2 in child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    const char start_msg[] = "Child2 gotov!\n";
    if (write(STDOUT_FILENO, start_msg, sizeof(start_msg) - 1) == -1) {
        const char msg[] = "error: failed to write start message\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    sem_post(ready_sem2);
    sem_close(ready_sem2);

    int running = 1;

    while (running) {
        if (sem_wait(sem2) == -1) {
            const char msg[] = "error: sem_wait failed in child2\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            break;
        }
```

```
    if (shm2->length > 0) {
        if (shm2->length == UINT32_MAX) {
            running = 0;
        } else {
            for (int i = 0; i < shm2->length && shm2->data[i]; i++) {
                if (isspace(shm2->data[i])) {
                    shm2->data[i] = '_';
                }
            }

            if (sem_wait(sem3) == -1) {
                const char msg[] = "error: sem_wait failed in child2\n";
                write(STDERR_FILENO, msg, sizeof(msg) - 1);
                sem_post(sem2);
                break;
            }

            shm3->length = shm2->length;
            memcpy(shm3->data, shm2->data, shm2->length + 1);

            if (sem_post(sem3) == -1) {
                const char msg[] = "error: sem_post failed in child2\n";
                write(STDERR_FILENO, msg, sizeof(msg) - 1);
                sem_post(sem2);
                break;
            }

            shm2->length = 0;
        }
    }

    if (sem_post(sem2) == -1) {
        const char msg[] = "error: sem_post failed in child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        break;
    }
}

if (sem_close(sem2) == -1) {
    const char msg[] = "error: sem_close failed in child2\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
}

if (sem_close(sem3) == -1) {
    const char msg[] = "error: sem_close failed in child2\n";
```

```
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (munmap(shm2, SHM_SIZE) == -1) {
        const char msg[] = "error: munmap failed in child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (munmap(shm3, SHM_SIZE) == -1) {
        const char msg[] = "error: munmap failed in child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (close(shm2_fd) == -1) {
        const char msg[] = "error: close failed in child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    if (close(shm3_fd) == -1) {
        const char msg[] = "error: close failed in child2\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    const char end_msg[] = "Child2 finished.\n";
    if (write(STDOUT_FILENO, end_msg, sizeof(end_msg) - 1) == -1) {
        const char msg[] = "error: failed to write finish message\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
    }

    return 0;
}
```

Strace

```
tim@tim-potato-pc:~/OS_labs/lab_03/src$ strace ./parent_exe
execve("./parent_exe", ["./parent_exe"], 0x7ffd9fe4c660 /* 84 vars */) = 0
brk(NULL)                               = 0x5f796737e000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7e6110851000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=69395, ...}) = 0
mmap(NULL, 69395, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7e6110840000
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) = 0x7e6110600000
mmap(0x7e6110628000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7e6110628000
mmap(0x7e61107b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7e61107b0000
mmap(0x7e61107ff000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7e61107ff000
mmap(0x7e6110805000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7e6110805000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7e611083d000
arch_prctl(ARCH_SET_FS, 0x7e611083d740) = 0
set_tid_address(0x7e611083da10)         = 7555
set_robust_list(0x7e611083da20, 24)     = 0
rseq(0x7e611083e060, 0x20, 0, 0x53053053) = 0
mprotect(0x7e61107ff000, 16384, PROT_READ) = 0
mprotect(0x5f7941510000, 4096, PROT_READ) = 0
mprotect(0x7e611088f000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7e6110840000, 69395)           = 0
getpid()                                = 7555
openat(AT_FDCWD, "/dev/shm/shm1-7555", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0600) = 3
openat(AT_FDCWD, "/dev/shm/shm2-7555", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0600) = 4
openat(AT_FDCWD, "/dev/shm/shm3-7555", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0600) = 5
ftruncate(3, 1028)                      = 0
ftruncate(4, 1028)                      = 0
ftruncate(5, 1028)                      = 0
mmap(NULL, 1028, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7e6110850000
mmap(NULL, 1028, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7e611084f000
mmap(NULL, 1028, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7e611084e000
getrandom("\x00\xad\x70\x68\xe7\x96\x0e\xa7", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.sjqiim", 0x7fffc13823d0, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/dev/shm/sem.sjqiim", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0600) = 6
write(6, "\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, 6, 0) = 0x7e611084d000
```

```
link("/dev/shm/sem.sjqiim", "/dev/shm/sem.sem1-7555") = 0
fstat(6, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0
getrandom("\x4f\xa6\xbc\x83\x11\x75\x75\x3d", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5f796737e000
brk(0x5f796739f000)                     = 0x5f796739f000
unlink("/dev/shm/sem.sjqiim")           = 0
close(6)                                = 0
getrandom("\xa1\xa4\x52\x11\xcc\x88\xb6\x70", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.ptaZ96", 0x7fffc13823d0, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/dev/shm/sem.ptaZ96", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0600) = 6
write(6, "\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, 6, 0) = 0x7e611084c000
link("/dev/shm/sem.ptaZ96", "/dev/shm/sem.sem2-7555") = 0
fstat(6, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0
unlink("/dev/shm/sem.ptaZ96")           = 0
close(6)                                = 0
getrandom("\xef\x4f\xe5\x75\xf8\xc7\x64\xb5", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.frrfTd", 0x7fffc13823d0, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/dev/shm/sem.frrfTd", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0600) = 6
write(6, "\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, 6, 0) = 0x7e611084b000
link("/dev/shm/sem.frrfTd", "/dev/shm/sem.sem3-7555") = 0
fstat(6, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0
unlink("/dev/shm/sem.frrfTd")           = 0
close(6)                                = 0
getrandom("\xb6\xbb\x0d\x66\xfe\xa3\x23\x25", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.cAIeo3", 0x7fffc13823d0, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/dev/shm/sem.cAIeo3", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0600) = 6
write(6, "\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, 6, 0) = 0x7e611084a000
link("/dev/shm/sem.cAIeo3", "/dev/shm/sem.ready1-7555") = 0
fstat(6, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0
unlink("/dev/shm/sem.cAIeo3")           = 0
close(6)                                = 0
getrandom("\x98\x0d\xb3\x74\x71\x14\xd1\xfd", 8, GRND_NONBLOCK) = 8
getrandom("\x00\x46\x4e\x76\xb7\x1b\xdc\x2a", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.EjXQOU", 0x7fffc13823d0, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (Нет такого файла или каталога)
```

```
openat(AT_FDCWD, "/dev/shm/sem.EjXQOU", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0600) = 6
write(6, "\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, 6, 0) = 0x7e6110849000
link("/dev/shm/sem.EjXQOU", "/dev/shm/sem.ready2-7555") = 0
fstat(6, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0
unlink("/dev/shm/sem.EjXQOU")           = 0
close(6)                                = 0
write(1, "Parent started. Enter strings (e"..., 52Parent started. Enter strings
(empty line to exit):
) = 52
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7e611083da10) = 7556
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7e611083da10) = 7557
write(1, "Waiting for children.\n", 22Waiting for children.
) = 22
futex(0x7e611084a000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANYChild1 gotov!
) = 0
futex(0x7e6110849000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANYChild2 gotov!
) = 0
write(1, "Children ready!\n", 16Children ready!
)       = 16
munmap(0x7e611084a000, 32)              = 0
munmap(0x7e6110849000, 32)              = 0
unlink("/dev/shm/sem.ready1-7555")      = 0
unlink("/dev/shm/sem.ready2-7555")      = 0
write(1, "\320\222\320\262\320\276\320\264: ", 10Ввод: ) = 10
read(0, Hello, World!           2 Tabs!
"Hello, World!\t\t2 Tabs!\n", 1024) = 23
clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) = 0
write(1, "Result: ", 8Result: )                 = 8
write(1, "HELLO,_WORLD!__2_TABS!", 22HELLO,_WORLD!__2_TABS!)  = 22
write(1, "\n", 1
)                       = 1
write(1, "\320\222\320\262\320\276\320\264: ", 10Ввод: ) = 10
read(0,
"\n", 1024)                     = 1
futex(0x7e611084d000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY) = -1 EAGAIN (Ресурс временно недоступен)
futex(0x7e611084d000, FUTEX_WAKE, 1)    = 0
wait4(7556, Child1 finished.
Child2 finished.
NULL, 0, NULL)              = 7556
```

```
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=7556, si_uid=1000,
si_status=0, si_utime=1869 /* 18.69 s */, si_stime=0} ---
wait4(7557, NULL, 0, NULL)              = 7557
munmap(0x7e611084d000, 32)              = 0
munmap(0x7e611084c000, 32)              = 0
munmap(0x7e611084b000, 32)              = 0
unlink("/dev/shm/sem.sem1-7555")        = 0
unlink("/dev/shm/sem.sem2-7555")        = 0
unlink("/dev/shm/sem.sem3-7555")        = 0
unlink("/dev/shm/sem.ready1-7555")      = -1 ENOENT (Нет такого файла или каталога)
unlink("/dev/shm/sem.ready2-7555")      = -1 ENOENT (Нет такого файла или каталога)
munmap(0x7e6110850000, 1028)            = 0
munmap(0x7e611084f000, 1028)            = 0
munmap(0x7e611084e000, 1028)            = 0
unlink("/dev/shm/shm1-7555")            = 0
unlink("/dev/shm/shm2-7555")            = 0
unlink("/dev/shm/shm3-7555")            = 0
close(3)                                = 0
close(4)                                = 0
close(5)                                = 0
write(1, "Parent finished.\n", 17Parent finished.
)       = 17
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы были успешно изучены и применены на практике механизмы межпроцессного взаимодействия