

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-210БВ-24

Студент: Тодуа Н.Г.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 02.11.25

Москва, 2025

Постановка задачи

Вариант 21.

Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `int ftruncate()` - установка размера shared memory объекта
- `void mmap()` - отображение shared memory в адресное пространство процесса
- `int munmap()` - удаление отображения shared memory
- `int shm_unlink()` - удаление именованного объекта shared memory

Системные вызовы для работы с именованными семафорами:

- `sem_t sem_open()` - создание/открытие именованного семафора
- `sem_wait()` - уменьшение значения семафора
- `sem_post()` - увеличение значения семафора
- `sem_close()` - закрытие семафора
- `sem_unlink()` - удаление именованного семафора

Системные вызовы для управления процессами:

- `fork()` - создание нового процесса
- `execl()` - Заменяет текущий процесс новым процессом из исполняемого файла.
- `waitpid()` - ожидание завершения дочернего процесса
- `getpid()` - получение PID текущего процесса

Межпроцессное взаимодействие реализовано с использованием именованной разделяемой памяти и именованных семафоров. Родительский процесс создаёт два дочерних: первый обрабатывает нечётные строки и реверсирует символы, второй обрабатывает чётные строки и реверсирует символы. Обмен данными между процессами происходит через общие области памяти, созданные с помощью `shm_open` и отображённые в адресное пространство каждого процесса через `mmap`. Синхронизация доступа к разделяемой памяти обеспечивается с помощью семафоров, создаваемых через `sem_open`. Взаимодействие с пользователем (ввод текста и вывод результата) осуществляется родительским и дочерними процессами.

Код программы

parent.c

```

#include "header.h"
#include <sys/wait.h>

int main() {
    char filename1[256], filename2[256];
    printf(format: "Введите название выходного файла для нечётных строк: \n");
    if (!fgets(s: filename1, n: sizeof(filename1), stream: stdin)) exit(status: EXIT_FAILURE);
    filename1[strcspn(s: filename1, reject: "\n")] = 0;

    printf(format: "Введите название выходного файла для чётных строк: \n");
    if (!fgets(s: filename2, n: sizeof(filename2), stream: stdin)) exit(status: EXIT_FAILURE);
    filename2[strcspn(s: filename2, reject: "\n")] = 0;

    // Генерируем уникальные имена
    char shm_name1[256], shm_name2[256];
    char sem_empty1[256], sem_full1[256];
    char sem_empty2[256], sem_full2[256];

    generate_name(buf: shm_name1, len: sizeof(shm_name1), prefix: "shm1");
    generate_name(buf: shm_name2, len: sizeof(shm_name2), prefix: "shm2");
    generate_name(buf: sem_empty1, len: sizeof(sem_empty1), prefix: "empty1");
    generate_name(buf: sem_full1, len: sizeof(sem_full1), prefix: "full1");
    generate_name(buf: sem_empty2, len: sizeof(sem_empty2), prefix: "empty2");
    generate_name(buf: sem_full2, len: sizeof(sem_full2), prefix: "full2");

    // Создаём shared memory
    int fd1 = shm_open(name: shm_name1, oflag: O_CREAT | O_RDWR, mode: 0666);
    int fd2 = shm_open(name: shm_name2, oflag: O_CREAT | O_RDWR, mode: 0666);
    if (fd1 == -1 || fd2 == -1) { perror(s: "shm_open"); exit(status: EXIT_FAILURE); }

    ftruncate(fd: fd1, length: sizeof(SharedBuffer));
    ftruncate(fd: fd2, length: sizeof(SharedBuffer));

    SharedBuffer *buf1 = mmap(addr: NULL, len: sizeof(SharedBuffer), prot: PROT_READ | PROT_WRITE, flags: MAP_SHARED, fd: fd1, offset: 0);
    SharedBuffer *buf2 = mmap(addr: NULL, len: sizeof(SharedBuffer), prot: PROT_READ | PROT_WRITE, flags: MAP_SHARED, fd: fd2, offset: 0);
    if (buf1 == MAP_FAILED || buf2 == MAP_FAILED) { perror(s: "mmap"); exit(status: EXIT_FAILURE); }

    // Инициализируем буферы
    memset(s: buf1, c: 0, n: sizeof(SharedBuffer));
    memset(s: buf2, c: 0, n: sizeof(SharedBuffer));

```

```

// Создаём семафоры
sem_t *empty1 = sem_open(name: sem_empty1, oflag: O_CREAT | O_EXCL, 0666, 1); // 1 слот
sem_t *full1 = sem_open(name: sem_full1, oflag: O_CREAT | O_EXCL, 0666, 0);
sem_t *empty2 = sem_open(name: sem_empty2, oflag: O_CREAT | O_EXCL, 0666, 1);
sem_t *full2 = sem_open(name: sem_full2, oflag: O_CREAT | O_EXCL, 0666, 0);

if (empty1 == SEM_FAILED || full1 == SEM_FAILED ||
    empty2 == SEM_FAILED || full2 == SEM_FAILED) {
    perror(s: "sem_open");
    exit(status: EXIT_FAILURE);
}

// Запуск потомков
pid_t pid1 = fork();
if (pid1 == 0) {
    // Child 1
    execl(path: "./child", arg: "child", filename1, shm_name1, sem_empty1, sem_full1, (char*)NULL);
    perror(s: "execl child1");
    exit(status: EXIT_FAILURE);
}

pid_t pid2 = fork();
if (pid2 == 0) {
    // Child 2
    execl(path: "./child", arg: "child", filename2, shm_name2, sem_empty2, sem_full2, (char*)NULL);
    perror(s: "execl child2");
    exit(status: EXIT_FAILURE);
}

// Родитель: записывает в буферы
char *line = NULL;
size_t n = 0;
ssize_t read;
int line_number = 0;

while ((read = getline(lineptr: &line, n: &n, stream: stdin)) != -1) {
    line_number++;

```

```

while ((read = getline(lineptr: &line, n: &n, stream: stdin)) != -1) {
    line_number++;

    if (line_number % 2 == 1) {
        // Нечётная - buf1
        sem_wait(sem: empty1);
        if (read <= MAX_LINE) {
            memcpy(dest: buf1->data, src: line, n: read);
            buf1->size = read;
            buf1->ready = 1;
        }
        sem_post(sem: full1);
    } else {
        // Чётная - buf2
        sem_wait(sem: empty2);
        if (read <= MAX_LINE) {
            memcpy(dest: buf2->data, src: line, n: read);
            buf2->size = read;
            buf2->ready = 1;
        }
        sem_post(sem: full2);
    }

    // Пошлaем сигнал завершения: отправляем пустую строку с ready=0
    sem_wait(sem: empty1);
    buf1->ready = 0;
    sem_post(sem: full1);

    sem_wait(sem: empty2);
    buf2->ready = 0;
    sem_post(sem: full2);

    // Ожидаем завершения
    waitpid(pid: pid1, stat_loc: NULL, options: 0);
    waitpid(pid: pid2, stat_loc: NULL, options: 0);

    // Очистка

```

// Очистка

```

munmap(addr: buf1, len: sizeof(SharedBuffer));
munmap(addr: buf2, len: sizeof(SharedBuffer));
close(fd: fd1);
close(fd: fd2);

```

```

shm_unlink(name: shm_name1);
shm_unlink(name: shm_name2);

```

```

sem_close(sem: empty1); sem_close(sem: full1);
sem_close(sem: empty2); sem_close(sem: full2);
sem_unlink(name: sem_empty1); sem_unlink(name: sem_full1);
sem_unlink(name: sem_empty2); sem_unlink(name: sem_full2);

```

```

free(ptr: line);
return 0;

```

child.c

```

#include "header.h"

void reverse_string(char *str, size_t len) {
    if (len == 0) return;
    if (str[len - 1] == '\n') len--;
    for (size_t i = 0; i < len / 2; i++) {
        char tmp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = tmp;
    }
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, stderr, format: "Usage: %s <output_file> <shm_name> <sem_empty> <sem_full>\n", argv[0]);
        exit(status: EXIT_FAILURE);
    }

    const char *filename = argv[1];
    const char *shm_name = argv[2];
    const char *sem_empty_name = argv[3];
    const char *sem_full_name = argv[4];

    FILE *fout = fopen(filename, modes: "a");
    if (!fout) { perror(s: "fopen"); exit(status: EXIT_FAILURE); }

    // Подключаемся к shared memory
    int fd = shm_open(name: shm_name, oflag: O_RDWR, mode: 0666);
    if (fd == -1) { perror(s: "shm_open child"); exit(status: EXIT_FAILURE); }

    SharedBuffer *buf = mmap(addr: NULL, len: sizeof(SharedBuffer), prot: PROT_READ | PROT_WRITE, flags: MAP_SHARED, fd, offset: 0);
    if (buf == MAP_FAILED) { perror(s: "mmap child"); exit(status: EXIT_FAILURE); }

    // Подключаемся к семафорам
    sem_t *empty = sem_open(name: sem_empty_name, oflag: 0);
    sem_t *full = sem_open(name: sem_full_name, oflag: 0);
    if (empty == SEM_FAILED || full == SEM_FAILED) {
        perror(s: "sem_open child");
        exit(status: EXIT_FAILURE);
    }
}

```

```

// Цикл обработки
while (1) {
    sem_wait(sem: full); // Ждём, пока данные появятся

    if (!buf->ready) {
        // Сигнал завершения
        break;
    }

    // Копируем данные (чтобы не менять оригинал в буфере)
    char line_copy[MAX_LINE];
    size_t size = buf->size;
    if (size > MAX_LINE - 1) size = MAX_LINE - 1;
    memcpy(dest: line_copy, src: buf->data, n: size);
    line_copy[size] = '\0';

    // Реверс
    reverse_string(str: line_copy, len: size);

    // Выводим
    fputs(s: line_copy, stream: stdout);
    fflush(stream: stdout);

    // Пишем в файл
    fputs(s: line_copy, stream: fout);
    fflush(stream: fout);

    sem_post(sem: empty); // Освобождаем слот
}

fclose(stream: fout);
munmap(addr: buf, len: sizeof(SharedBuffer));
close(fd);
sem_close(sem: empty);
sem_close(sem: full);

return 0;
}

```

utils.c

```
#include "header.h"

void generate_name(char *buf, size_t len, const char *prefix) {
    snprintf(s: buf, maxlen: len, format: "%s_%ld", prefix, (long)getpid());
}
```

header.h

```
#ifndef HEADER_H
#define HEADER_H

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

#define BUFFER_SIZE 4096
#define MAX_LINE 4096

typedef struct {
    char data[BUFFER_SIZE];
    size_t size;
    int ready;
} SharedBuffer;

void generate_name(char *buf, size_t len, const char *prefix);

#endif
```

Протокол работы программы

Тестирование:

```
[mafonchik@mafonchik-aspirea31544p lab3]$ ./parent
Введите название выходного файла для нечётных строк:
out1.txt
Введите название выходного файла для чётных строк:
out2.txt
string1
1gnirts
string2
2gnirts
lfasdjg
gjdsaf1
ashviusau
uasuivhsa
kssfsdkofg siofjaosijdf
fdjisoajfois gfokdsfssk
ssss ewdsdg
gdsdwe ssss
[mafonchik@mafonchik-aspirea31544p lab3]$ cat out1.txt
1gnirts
gjdsaf1
fdjisoajfois gfokdsfssk
[mafonchik@mafonchik-aspirea31544p lab3]$ cat out2.txt
2gnirts
uasuivhsa
gdsdwe ssss
```

Вывод

В ходе выполнения лабораторной работы было реализовано межпроцессное взаимодействие в Unix-подобной операционной системе с использованием именованной разделяемой памяти и именованных семафоров. Родительский процесс создаёт два дочерних, каждый из которых выполняет отдельную обработку текста. Обмен данными осуществляется через общую область памяти, созданную с помощью `shm_open` и отображённую в адресное пространство процессов через `mmap`, а синхронизация доступа к ней обеспечивается семафорами, созданными с помощью `sem_open`. Такой подход позволяет эффективно координировать работу процессов и безопасно разделять данные без использования каналов или копирования.