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-212БВ-24

Студент: Мышакин М.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.11.25

Москва, 2025

Постановка задачи

Вариант 7.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным (модифицированная: нужно переделать свою первую лабораторную работу с использованием shared memory и memory mapping)

Общий метод и алгоритм решения

В данной лабораторной работе реализовано взаимодействие между двумя процессами — родительским и дочерним — с использованием именованной общей памяти (shared memory) и семафоров для синхронизации доступа к разделяемым данным.

Общий метод основан на использовании системных вызовов:

- shm_open — для создания и открытия именованного объекта общей памяти в ядре ОС;
- ftruncate — для задания требуемого размера памяти;
- mmap — для отображения (mapping) общей памяти в виртуальное адресное пространство процесса;
- sem_open, sem_wait, sem_post — для создания и управления семафорами синхронизации;
- fork и execl — для порождения дочернего процесса и запуска отдельной программы;
- munmap, shm_unlink, sem_close, sem_unlink — для корректного освобождения ресурсов.

Алгоритм работы системы:

1. Родительский процесс запрашивает у пользователя имя входного файла.
2. Создаётся именованный объект общей памяти и два семафора (sem_write, sem_read) для координации работы между процессами.
3. Родитель вызывает fork(). В дочернем процессе выполняется execl() для запуска программы child.
4. Программа child считывает данные из указанного файла, вычисляет сумму чисел в каждой строке и записывает результат в общую память.
   После записи она подаёт сигнал родителю через sem_post(sem_read) и ожидает разрешения на следующую итерацию (sem_wait(sem_write)).

5. Родитель, в свою очередь, ждёт появления данных (sem_wait(sem_read)), выводит их на экран, и затем разрешает запись следующего результата (sem_post(sem_write)).

6. После завершения обработки файла ребёнок записывает в память сообщение "END", родитель получает этот сигнал и завершает работу, освобождая ресурсы.

Описание работы программы

Программа состоит из двух частей:

- **parent.c** — родительский процесс, который управляет созданием общей памяти, синхронизацией и выводом результатов;

- **child.c** — дочерний процесс, выполняющий обработку входного файла.

Работа родительского процесса (parent.c):

1. Запрашивает у пользователя имя файла для обработки.

2. Создаёт именованный объект общей памяти (shm_open) и задаёт ему размер (ftruncate).

3. Отображает память в адресное пространство (mmap) и создаёт два семафора для синхронизации.

4. Вызывает fork() и в дочернем процессе запускает программу child через execl(), передавая ей имена объектов (shared memory и семафоров).

5. После этого родительский процесс циклически:

    - ожидает сигнал от ребёнка (sem_wait(sem_read)),

    - читает результат из общей памяти и выводит его на экран,

    - разрешает следующую запись (sem_post(sem_write)).

6. После получения сигнала "END" родитель завершает цикл и освобождает все системные ресурсы.

Работа дочернего процесса (child.c):

1. Подключается к существующим объектам общей памяти и семафорам по переданным именам.

2. Считывает из файла строки, содержащие произвольное количество чисел с плавающей точкой.

3. Для каждой строки вычисляет сумму чисел, записывает результат в shared memory и подаёт сигнал родителю (sem_post(sem_read)).

4. Ждёт, пока родитель разрешит запись следующего результата (sem_wait(sem_write)).

5. После окончания файла записывает строку "END", сигнализируя о завершении вычислений.

Код программы

common.h

```
#pragma once

#include <semaphore.h>


#define SHM_SIZE 1024


// префиксы имён для shared memory и семафоров

#define SHM_NAME_PREFIX "/my_shm_"

#define SEM_WRITE_PREFIX "/sem_write_"

#define SEM_READ_PREFIX "/sem_read_"
```

parent.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/wait.h>

#include <semaphore.h>

#include "common.h"


int main()
```

```
{
	char shm_name[64], sem_write_name[64], sem_read_name[64];
	pid_t pid = getpid();

	// создаёт уникальное имя для общей памяти
	snprintf(shm_name, sizeof(shm_name), "%s%d", SHM_NAME_PREFIX, pid);
	snprintf(sem_write_name, sizeof(sem_write_name), "%s%d",
SEM_WRITE_PREFIX, pid);
	snprintf(sem_read_name, sizeof(sem_read_name), "%s%d", SEM_READ_PREFIX,
pid);

	// создаёт именованный объект общей памяти в ядре; 0666 - остальные
процессы имеют права на чтение и запись
	int shm_fd = shm_open(shm_name, O_CREAT | O_RDWR, 0666);
	if (shm_fd == -1)
	{
		perror("shm_open");
		exit(1);
	}

	// задает размер области общей памяти в ядре
	ftruncate(shm_fd, SHM_SIZE);

	// создает виртуальную память с которой взаимодействует программа, а также
таблицу MMU для соответствия учатсков памяти.
	char *shm_ptr = mmap(0, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0); // memorymap
	{
		perror("mmap");
		exit(1);
	}

	// создаёт или открывает именованный семафор с разрешением на запись
```

```
sem_t *sem_write = sem_open(sem_write_name, O_CREAT, 0666, 1);

// создаёт или открывает именованный семафор без разрешения на чтение
sem_t *sem_read = sem_open(sem_read_name, O_CREAT, 0666, 0);
if (sem_write == SEM_FAILED || sem_read == SEM_FAILED)
{
	perror("sem_open");
	exit(1);
}

printf("Введите имя файла с числами: ");
char filename[256];
scanf("%255s", filename);

// открываем файл для чтения
int file_fd = open(filename, O_RDONLY);
if (file_fd == -1)
{
	perror("open");
	exit(1);
}

// создаём дочерний процесс
pid_t child_pid = fork();
if (child_pid == -1)
{
	perror("fork");
	exit(1);
}

if (child_pid == 0)
```

```
	{
		// перенаправляем ввод дочернего процесса из файла
		dup2(file_fd, STDIN_FILENO);
		close(file_fd);

		// заменяет выполнение текущего процесса на дочерний и передает ему соответствующие аргументы
		execl("./child", "child", shm_name, sem_write_name, sem_read_name, NULL);
		perror("execl");
		exit(1);
	}

	close(file_fd);

	// родитель читает данные из shared memory
	printf("\nРезультаты вычислений:\n");

	while (1)
	{
		// ждём пока ребёнок что-то запишет
		sem_wait(sem_read); // переводим его в 0, то есть родитель теперь может читать, но внешние процессы не могут этого делать

		if (strcmp(shm_ptr, "END") == 0)
			break;

		printf("%s\n", shm_ptr);

		// разрешаем следующую запись
		sem_post(sem_write);
	}
```

```
	// очищаем ресурсы
	wait(NULL);
	munmap(shm_ptr, SHM_SIZE);
	close(shm_fd);

	shm_unlink(shm_name);
	sem_unlink(sem_write_name);
	sem_unlink(sem_read_name);

	printf("\nРабота завершена.\n");
	return 0;
}
```

child.c

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

int main(int argc, char *argv[])
{
	if (argc != 4)
	{
		fprintf(stderr, "Usage: %s <shm_name> <sem_write> <sem_read>\n",
argv[0]);
```

```
        return 1;
    }

    const char *shm_name = argv[1];
    const char *sem_write_name = argv[2];
    const char *sem_read_name = argv[3];

    // подключаемся к существующей shared memory
    int shm_fd = shm_open(shm_name, O_RDWR, 0666);
    if (shm_fd == -1)
    {
        perror("shm_open");
        return 1;
    }

    char *shm_ptr = mmap(0, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shm_ptr == MAP_FAILED)
    {
        perror("mmap");
        return 1;
    }

    // подключаемся к существующим семафорам
    sem_t *sem_write = sem_open(sem_write_name, 0);
    sem_t *sem_read = sem_open(sem_read_name, 0);
    if (sem_write == SEM_FAILED || sem_read == SEM_FAILED)
    {
        perror("sem_open");
        return 1;
    }
```

```
char line[1024];

// читаем строки из stdin ( у нас это файл отправленный родителем )
while (1)
{
	if (!fgets(line, sizeof(line), stdin))
	{
		if (feof(stdin))
			break; // достигнут конец файла
		continue;  // ошибка чтения – пропускаем
	}

	// убираем возможный '\n' в конце строки
	size_t len = strlen(line);
	if (len > 0 && line[len - 1] == '\n')
		line[len - 1] = '\0';

	// разбираем строку на произвольное количество чисел
	char *ptr = line;
	float sum = 0.0f;
	int hasNumbers = 0;

	while (*ptr)
	{
		char *end;
		float value = strtof(ptr, &end);
		if (end == ptr)
		{
			// если не смогли считать число – двигаемся дальше
			ptr++;
```

```
                        continue;
                }
                sum += value;
                hasNumbers = 1;
                ptr = end;
        }

        // если в строке были числа — передаём результат в shared memory
        if (hasNumbers)
        {
                char buffer[64];
                snprintf(buffer, sizeof(buffer), "Сумма: %.2f", sum);

                sem_wait(sem_write); // помечаем, что ребенок начал запись
shared memory, поэтому блокируем запись от других процессов
                strncpy(shm_ptr, buffer, SHM_SIZE - 1);
                shm_ptr[SHM_SIZE - 1] = '\0';
                sem_post(sem_read); // сигнализируем, что данные готовы,
меняем значение семафора на 1, то есть даем родителю возможность читать данные
        }
    }

    // после завершения отправляем сигнал конца
    sem_wait(sem_write);
    strcpy(shm_ptr, "END");
    sem_post(sem_read);

    // освобождаем ресурсы
    munmap(shm_ptr, SHM_SIZE);
    close(shm_fd);
    return 0;
}
```

Протокол работы программы

```
data.txt
1.2 3.4 5.6 7.8 9.0
0.5 -1.5
10.0 20.0 30.0 40.0 50.0


misha@misha-VirtualBoxUbuntu:~/Documents/cApps/OS-LABS$ gcc -o child child.c
misha@misha-VirtualBoxUbuntu:~/Documents/cApps/OS-LABS$ gcc -o parent parent.c
misha@misha-VirtualBoxUbuntu:~/Documents/cApps/OS-LABS$ ./parent
Введите имя файла: da
Ошибка открытия файла: da
misha@misha-VirtualBoxUbuntu:~/Documents/cApps/OS-LABS$ ./parent
Введите имя файла: data.txt
Сумма: 27.00
Сумма: -1.00
Сумма: 150.00
misha@misha-VirtualBoxUbuntu:~/Documents/cApps/OS-LABS$
```

Вывод

В ходе выполнения лабораторной работы были успешно реализованы механизмы межпроцессного взаимодействия с использованием именованной общей памяти и семафоров. Программа корректно создаёт дочерний процесс, отображает общую память в адресное пространство обоих процессов, обеспечивает синхронизацию доступа к разделяемым данным и выводит результаты вычислений в терминал.