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-209Б-24

Студент: Филь Н.А.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 16.12.25

Москва, 2025

Постановка задачи

Вариант 1.

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

В программе используются системные вызовы POSIX:
open(), ftruncate(), mmap(), fork(), wait(), munmap(), unlink().

Алгоритм:

1. Создание и отображение файла в память.
2. Создание дочернего процесса.
3. Ввод данных родительским процессом.
4. Вычисление сумм в дочернем процессе.
5. Запись результатов в файл и вывод на экран.
6. Освобождение ресурсов.

Код программы

lab3.c

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

#define FILENAME "data.bin"
#define MAX_LINES 100
#define MAX_NUMBERS_PER_LINE 100

typedef struct {
    int numbers[MAX_LINES][MAX_NUMBERS_PER_LINE];
    int counts[MAX_LINES];
    int sums[MAX_LINES];
    int line_count;
} shared_data;

int main() {
    size_t file_size = sizeof(shared_data);

    int fd = open(FILENAME, O_CREAT | O_RDWR | O_TRUNC, 0666);
    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    if (ftruncate(fd, file_size) == -1) {
        perror("ftruncate");
        close(fd);
        exit(EXIT_FAILURE);
    }
```

```
    shared_data *data = mmap(NULL, file_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
0);
    if (data == MAP_FAILED) {
        perror("mmap");
        close(fd);
        exit(EXIT_FAILURE);
    }

    data->line_count = 0;
    for (int i = 0; i < MAX_LINES; i++) {
        data->counts[i] = 0;
        data->sums[i] = 0;
    }

    pid_t pid = fork();
    if (pid == -1) {
        perror("fork");
        munmap(data, file_size);
        close(fd);
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        while (data->line_count == 0) {
            usleep(100);
        }

        for (int line = 0; line < data->line_count; line++) {
            data->sums[line] = 0;
            for (int i = 0; i < data->counts[line]; i++) {
                data->sums[line] += data->numbers[line][i];
            }
        }

        FILE *result_file = fopen("result.txt", "w");
        if (result_file == NULL) {
            perror("fopen");
            munmap(data, file_size);
            close(fd);
            exit(EXIT_FAILURE);
        }

        for (int line = 0; line < data->line_count; line++) {
            fprintf(result_file, "%d\n", data->sums[line]);
        }
        fclose(result_file);

        munmap(data, file_size);
        close(fd);
        exit(EXIT_SUCCESS);

    } else {
        printf("Введите строки с числами:\n");
        printf("Для завершения введите 'end' или нажмите Ctrl+D:\n");

        char input[256];
        int line = 0;

        while (fgets(input, sizeof(input), stdin) != NULL && line < MAX_LINES) {
            input[strcspn(input, "\n")] = 0;

            if (strlen(input) == 0) {
                continue;
            }

            if (strcmp(input, "end") == 0 || strcmp(input, "END") == 0) {
                printf("Завершение ввода по команде 'end'\n");
```

```
                break;
            }

            char *token = strtok(input, " \t");
            int count = 0;

            while (token != NULL && count < MAX_NUMBERS_PER_LINE) {
                if (sscanf(token, "%d", &data->numbers[line][count]) == 1) {
                    count++;
                }
                token = strtok(NULL, " \t");
            }

            if (count > 0) {
                data->counts[line] = count;
                line++;
            } else {
                printf("Пропущена строка без чисел\n");
            }
        }

        data->line_count = line;

        if (data->line_count == 0) {
            printf("Ошибка: не введено ни одной строки.\n");
            munmap(data, file_size);
            close(fd);
            unlink(FILENAME);
            return 1;
        }

        wait(NULL);

        printf("\nРезультаты:\n");
        for (int line = 0; line < data->line_count; line++) {
            printf("Строка %d сумма: %d\n", line + 1, data->sums[line]);
        }

        printf("\nСуммы записаны в result.txt\n");

        munmap(data, file_size);
        close(fd);
        unlink(FILENAME);
    }

    return 0;
}
```

Протокол работы программы

Тестирование:

gcc -o lab3 lab3.c

./lab3
Пример ввода:
1 1 1
1 2 3

0
1 2 3
end

Пример вывода:
Строка 1 сумма: 3
Строка 2 сумма: 6
Строка 3 сумма: 0
Строка 4 сумма: 6

Результаты также сохраняются в файл result.txt.

Strace:

Фрагмент вывода strace (ключевые места):

Запуск программы
execve("./lab3", ["./lab3"], 0xfffffc0a9e50 /* 75 vars */) = 0
Системный вызов execve() используется ядром Linux для запуска исполняемого файла программы lab3. В данный момент происходит загрузка бинарного файла в адресное пространство процесса и передача управления функции main().

Инициализация адресного пространства процесса
brk(NULL) = 0xaaaafe824000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
Вызовы brk() и mmap() используются для выделения динамической памяти под нужды программы и стандартной библиотеки C. На данном этапе происходит настройка кучи и вспомогательных структур процесса.

Загрузка стандартной библиотеки C
openat(AT_FDCWD, "/lib/aarch64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
mmap(NULL, 1519552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0)
Происходит загрузка библиотеки libc.so.6, которая содержит реализацию стандартных функций и POSIX-вызовов. Это стандартный этап запуска любой программы в Linux.

Создание и отображение файла (file mapping)
openat(AT_FDCWD, "data.bin", O_RDWR|O_CREAT|O_TRUNC, 0666) = 3
ftruncate(3, 40804) = 0
mmap(NULL, 40804, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0)
Данные системные вызовы соответствуют созданию файла data.bin и его отображению в память:

openat() — создаёт файл для хранения разделяемых данных;
ftruncate() — устанавливает размер файла, равный размеру структуры данных;
mmap() — отображает файл в виртуальное адресное пространство процесса.
Использование флага MAP_SHARED обеспечивает совместный доступ к данным между родительским и дочерним процессами.

Создание дочернего процесса

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, ...) = 4452
Системный вызов clone() является низкоуровневой реализацией функции fork(). Он создаёт дочерний процесс, который получает копию адресного пространства родителя, включая отображаемый файл.
Флаг SIGCHLD указывает, что при завершении дочернего процесса родитель получит сигнал.

Ввод пользовательских данных

read(0, "1 1 1\n", 1024) = 6
read(0, "1 2 3\n", 1024) = 6
read(0, "0\n", 1024) = 2
read(0, "1 2 3\n", 1024) = 6
read(0, "end\n", 1024) = 4
Системный вызов read() используется для чтения данных со стандартного потока ввода (клавиатуры). Введённые пользователем строки чисел записываются родительским процессом в разделяемую область памяти.

Ожидание завершения дочернего процесса

wait4(-1, NULL, 0, NULL) = 4452
--- SIGCHLD ---
Родительский процесс вызывает wait4() для ожидания завершения дочернего процесса.
Получение сигнала SIGCHLD подтверждает, что дочерний процесс успешно завершил работу.

Вывод результатов

write(1, "Строка 1 сумма: 3\n", 29) = 29
write(1, "Строка 2 сумма: 6\n", 29) = 29
Системный вызов write() используется для вывода вычисленных сумм на стандартный поток вывода (экран).

Освобождение ресурсов и завершение программы

munmap(0xffffb59b8000, 40804) = 0
close(3) = 0
unlinkat(AT_FDCWD, "data.bin", 0) = 0
exit_group(0)
munmap() — освобождает отображаемую область памяти;
close() — закрывает файловый дескриптор;
unlinkat() — удаляет временный файл data.bin;
exit_group() — завершает выполнение программы.
Корректное завершение всех системных вызовов свидетельствует об отсутствии ошибок в работе программы.

Вывод

В ходе выполнения лабораторной работы была реализована программа межпроцессного взаимодействия с использованием отображаемых файлов. Подробный анализ вывода утилиты strace подтвердил корректное использование системных вызовов POSIX и правильную организацию обмена данными между процессами.