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»

Выполнил: В. А. Гузова
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы:
Приобретение практических навыков в:

- Освоение принципов работы с файловыми системами
- Обеспечение обмена данных между процессами посредством технологии «File mapping»

Задание:
Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант: 7
Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами. В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

Метод решения

Parent.cpp запрашивает у пользователя имя файла, создаёт разделяемую память и child. Ребенок читает числа из файла, складывает их и записывает результат в виде строки в разделяемую память. Потом он посылает родителю сигнал SIGUSR2, говоря о готовности ответа. Родительский процесс ожидает этот сигнал, после чего извлекает и выводит результат.

Описание программы

Разделение по файлам, описание основных типов данных и функций. Обязательно написать используемые системные вызовы.

`parent.cpp` — точка входа в программу.Запрашивает имя файла, создаёт разделяемую память, запускает дочерний процесс, ожидает сигнала и выводит результат.

`child.cpp` — исполняемый файл дочернего процесса.Получает имя файла и имя разделяемой памяти, читает числа из файла, суммирует их и записывает результат в разделяемую память, посылает сигнал родителю.

`os.h` — объявление функций-обёрток над системными вызовами ОС для управления процессами, сигналами и разделяемой памятью.

`os.cpp` — реализация для Unix-подобных ОС.

`exception.cpp` — объявление пользовательчких исключений.

Основные функции:

- `CreateSharedMemory(name, size)` — создаёт объект разделяемой памяти с заданным именем и размером (вызывает `shm_open`, `ftruncate`, `mmap`).
- `Open(name, size)` — подключается к существующей разделяемой памяти (выполняет `shm_open` + `mmap`).
- `Destroy(shm)` — деинициализирует и удаляет разделяемую память (`munmap`, `close`, `shm_unlink`).
- `Unmap(shm)` — отсоединяет текущий процесс от разделяемой памяти (`munmap` + `close`).
- `CreateChildProcess(exe, arg1, arg2)` — создаёт дочерний процесс через `fork` + `execl`.
- `SetSignalHandler(signum, handler)` — регистрирует обработчик сигнала (`signal`).
- `SendSignal(pid, signum)` — посылает сигнал указанному процессу (`kill`).
- `WaitSignal()` — ожидает поступления сигнала через цикл с `pause()` и флагом `signal_flag`, устанавливаемым в обработчике.
- `WaitChild(pid)` — ждёт завершения дочернего процесса (`waitpid`).

Архитектура взаимодействия:

Родитель создаёт разделяемую память /lab размером 1024 байта.Устанавливает обработчик для сигнала SIGUSR2 (символ CONFIRM).Запускает дочерний процесс, передавая ему имя файла и имя разделяемой памяти.Блокируется в WaitSignal(), ожидая сигнала от ребёнка.Дочерний процесс подключается к той же разделяемой памяти. Вычисляет сумму чисел из файла.Преобразует результат в строку и копирует в разделяемую память.Посылает SIGUSR2 родителю через SendSignal(GetParentPID(), CONFIRM).Отключается от разделяемой памяти и завершается.Родитель читает строку из shm.ptr и выводит её.После завершения ребёнка (WaitChild) и чтения результата — освобождает разделяемую память.

Результаты

Программа получает имя файла, создаёт дочерний процесс, который считывает из файла числа и вычисляет их сумму. Результат передаётся родителю через разделяемую память и выводится в стандартный поток. При ошибках программа корректно освобождает ресурсы и завершается.

Выводы

В ходе выполнения лабораторной работы были приобретены навыки в управлении процессами в ОС,обмен данными через разделяемую память и синхронизация с помощью сигналов.

Была составлена и отлажена программа на языке C++. Обрабатаны системные ошибки, которые могут возникнуть в результате работы.

Есть возможность поддержки кроссплатформенности за счет изменений системных вызовов в файле

Исходная программа

```
#pragma once

#include <csignal>
#include <string>
#include <sys/types.h>

namespace os {

using ProcessHandle = pid_t;
using FileHandle = int;
using SignalHandler = void (*)(int);

constexpr int CONFIRM = SIGUSR2;

constexpr size_t SHARED_MEMORY_SIZE = 1024;

struct SharedMemory {
    char* ptr;
    size_t size;
    FileHandle fd;
    std::string name;
};

ProcessHandle CreateChildProcess(const std::string& exe_name, const std::string&
    filename, const std::string& shm_name);

void WaitChild(ProcessHandle process);

int GetCurrentPID();

ProcessHandle GetParentPID();

SharedMemory CreateSharedMemory(const std::string& name, size_t size);

SharedMemory Open(const std::string& name, size_t size);

void Destroy(SharedMemory& shm);

void Unmap(SharedMemory& shm);

void SetSignalHandler(int signum, SignalHandler handler);

void DefaultSignalHandler(int signum);

void SendSignal(ProcessHandle pid, int signum);

void WaitSignal();

}
```

Листинг 1: include/os.h

```
#include <csignal>
#include <cstring>
#include <iostream>
```

```cpp
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <unistd.h>

#include "os.h"
#include "exception.h"

namespace os {

volatile sig_atomic_t signal_flag = 0;

void DefaultSignalHandler(int) {
    signal_flag = 1;
}

ProcessHandle CreateChildProcess(const std::string& exe_name, const std::string&
    filename, const std::string& shm_name) {
    pid_t pid = fork();
    if (pid == 0) {
        execl(exe_name.c_str(), exe_name.c_str(), filename.c_str(), shm_name.c_str(),
            nullptr);
        perror("Execl failed");
        _exit(1);
    }
    return pid;
}

void WaitChild(ProcessHandle process) {
    if (process > 0) {
        int status;
        if (waitpid(process, &status, 0) == -1) {
            throw except::ProcessExcept("Waitpid failed.");
        }
    }
}

int GetCurrentPID() {
    return getpid();
}

ProcessHandle GetParentPID() {
    return getppid();
}

SharedMemory CreateSharedMemory(const std::string& name, size_t size) {
    SharedMemory shm;
    shm.name = name;
    shm.size = size;
    shm.fd = shm_open(name.c_str(), O_CREAT | O_RDWR, 0666);
    if (shm.fd == -1) {
        throw except::ProcessExcept("Shm_open failed.");
    }
    if (ftruncate(shm.fd, size) == -1) {
        close(shm.fd);
        throw except::ProcessExcept("Ftruncate failed.");
    }

```

```
60      shm.ptr = (char*)mmap(0, size, PROT_WRITE | PROT_READ, MAP_SHARED, shm.fd, 0);
61      if (shm.ptr == MAP_FAILED) {
62          close(shm.fd);
63          throw except::ProcessExcept("Mmap failed.");
64      }
65      return shm;
66  }
67
68  SharedMemory Open(const std::string& name, size_t size) {
69      SharedMemory shm;
70      shm.name = name;
71      shm.size = size;
72      shm.fd = shm_open(name.c_str(), O_RDWR, 0666);
73      if (shm.fd == -1) {
74          throw except::ProcessExcept("Shm_open failed.");
75      }
76      shm.ptr = (char*)mmap(0, size, PROT_WRITE | PROT_READ, MAP_SHARED, shm.fd, 0);
77      if (shm.ptr == MAP_FAILED) {
78          close(shm.fd);
79          throw except::ProcessExcept("Mmap failed in child.");
80      }
81      return shm;
82  }
83
84  void Unmap(SharedMemory& shm) {
85      if (shm.ptr && shm.ptr != MAP_FAILED) {
86          munmap(shm.ptr, shm.size);
87          shm.ptr = nullptr;
88      }
89      if (shm.fd != -1) {
90          close(shm.fd);
91          shm.fd = -1;
92      }
93  }
94
95  void Destroy(SharedMemory& shm) {
96      Unmap(shm);
97      shm_unlink(shm.name.c_str());
98  }
99
100 void SetSignalHandler(int signum, SignalHandler handler) {
101     if (signal(signum, handler) == SIG_ERR) {
102         throw except::ProcessExcept("Signal failed.");
103     }
104 }
105
106 void SendSignal(ProcessHandle pid, int signum) {
107     if (kill(pid, signum) == -1) {
108         throw except::ProcessExcept("Send signal failed.");
109     }
110 }
111
112 void WaitSignal() {
113     while (!signal_flag) {
114         pause();
115     }
116     signal_flag = 0;
117 }
```

```
118 
119 }
```

Листинг 2: src/os.cpp

```
1  #include <iostream>
2  #include <string>
3  #include <cstdlib>
4  #include "os.h"
5  #include "exception.h"
6  
7  int main() {
8      std::string filename;
9      std::cout << "Enter filename: ";
10     std::cin >> filename;
11 
12     const std::string shm_name = "/lab";
13     os::SharedMemory shm = {nullptr, 0, -1, ""};
14     os::ProcessHandle child_pid = -1;
15 
16     try {
17         shm = os::CreateSharedMemory(shm_name, os::SHARED_MEMORY_SIZE);
18         os::SetSignalHandler(os::CONFIRM, os::DefaultSignalHandler);
19         child_pid = os::CreateChildProcess("./child", filename, shm_name);
20         os::WaitSignal();
21         std::string answer_str(shm.ptr);
22         std::cout << "Answer: " << answer_str << std::endl;
23     } catch (const except::ProcessExcept& e) {
24         std::cerr << "Parent Error: " << e.what() << std::endl;
25     }
26 
27     if (child_pid > 0) {
28         try {
29             os::WaitChild(child_pid);
30         } catch (const except::ProcessExcept& e) {
31             std::cerr << "Parent Error : " << e.what() << std::endl;
32         }
33     }
34     if (shm.ptr) {
35         os::Destroy(shm);
36     }
37     return 0;
38 }
```

Листинг 3: src/parent.cpp

```
1  #include <iostream>
2  #include <fstream>
3  #include <sstream>
4  #include <string>
5  #include <cstring>
6  #include <cstdio>
7  #include "os.h"
8  #include "exception.h"
9  
```

```cpp

float SumFile(const std::string& filename) {
    std::ifstream file(filename);
    if (!file) {
        std::cerr << "Child: Error opening file " << filename << std::endl;
        return 0.0f;
    }
    float sum = 0.0f;
    std::string line;
    while (std::getline(file, line)) {
        std::istringstream iss(line);
        float num;
        while (iss >> num) {
            sum += num;
        }
    }
    return sum;
}

int main(int argc, char* argv[]) {
    if (argc < 3) {
        std::cerr << "Usage: " << argv[0] << " <filename> <shm_name>" << std::endl;
        _exit(1);
    }

    const std::string filename = argv[1];
    const std::string shm_name = argv[2];
    float answer = SumFile(filename);

    os::SharedMemory shm;
    try {
        shm = os::Open(shm_name, os::SHARED_MEMORY_SIZE);
        std::string answer_str = std::to_string(answer);
        strncpy(shm.ptr, answer_str.c_str(), shm.size - 1);
        shm.ptr[shm.size - 1] = '\0';
        os::SendSignal(os::GetParentPID(), os::CONFIRM);
        os::Unmap(shm);
    } catch (const except::ProcessExcept& e) {
        std::cerr << "Child Error: " << e.what() << std::endl;
        os::Unmap(shm);
        _exit(1);
    }
    return 0;
}
```

Листинг 4: src/child.cpp

```cpp
#pragma once

#include <stdexcept>
#include <string>

namespace except {
class ProcessExcept : public std::runtime_error {
public:
    explicit ProcessExcept(const std::string& message)
        : std::runtime_error(message) {}
```

```
11 };
12
13 }
```

Листинг 5: src/exception.h

Strace

```
execve("./parent", ["./parent"], 0x7ffef5265970 /* 27 vars */) = 0
brk(NULL)                               = 0x61261ca8e000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdc2325030) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↪  0x75e138eef000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=29000, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 29000, PROT_READ, MAP_PRIVATE, 3, 0) = 0x75e138ee7000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.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\0\0\0\0\0\0\0\0"...,
↪  832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2260296, ...}, AT_EMPTY_PATH) =
↪  0
mmap(NULL, 2275520, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x75e138c00000
mprotect(0x75e138c9a000, 1576960, PROT_NONE) = 0
mmap(0x75e138c9a000, 1118208, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x9a000) = 0x75e138c9a000
mmap(0x75e138dab000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x1ab000) = 0x75e138dab000
mmap(0x75e138e1b000, 57344, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x21a000) = 0x75e138e1b000
mmap(0x75e138e29000, 10432, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x75e138e29000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"...,
↪  832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=125488, ...}, AT_EMPTY_PATH) =
↪  0
mmap(NULL, 127720, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x75e138ec7000
mmap(0x75e138eca000, 94208, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x75e138eca000
mmap(0x75e138ee1000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x1a000) = 0x75e138ee1000
mmap(0x75e138ee5000, 8192, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d000) = 0x75e138ee5000
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\0P\237\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
```

```
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"...,
↪  48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O{\f\225\\=\201\327\312\301P\32$\230
↪  \266\235"..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) =
↪  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, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x75e138800000
mprotect(0x75e138828000, 2023424, PROT_NONE) = 0
mmap(0x75e138828000, 1658880, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x75e138828000
mmap(0x75e1389bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x1bd000) = 0x75e1389bd000
mmap(0x75e138a16000, 24576, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x75e138a16000
mmap(0x75e138a1c000, 52816, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x75e138a1c000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.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\0\0\0\0\0\0\0\0"...,
↪  832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=940560, ...}, AT_EMPTY_PATH) =
↪  0
mmap(NULL, 942344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x75e138b19000
mmap(0x75e138b27000, 507904, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe000) = 0x75e138b27000
mmap(0x75e138ba3000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x8a000) = 0x75e138ba3000
mmap(0x75e138bfe000, 8192, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe4000) = 0x75e138bfe000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↪  0x75e138ec5000
arch_prctl(ARCH_SET_FS, 0x75e138ec63c0) = 0
set_tid_address(0x75e138ec6690)         = 939
set_robust_list(0x75e138ec66a0, 24)     = 0
rseq(0x75e138ec6d60, 0x20, 0, 0x53053053) = 0
mprotect(0x75e138a16000, 16384, PROT_READ) = 0
mprotect(0x75e138bfe000, 4096, PROT_READ) = 0
mprotect(0x75e138ee5000, 4096, PROT_READ) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↪  0x75e138ec3000
mprotect(0x75e138e1b000, 45056, PROT_READ) = 0
mprotect(0x6125e174d000, 4096, PROT_READ) = 0
mprotect(0x75e138f29000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
↪  rlim_max=RLIM64_INFINITY}) = 0
munmap(0x75e138ee7000, 29000)           = 0
getrandom("\xa5\x0f\x58\xe6\x85\x95\x21\x0d", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x61261ca8e000
brk(0x61261caaf000)                     = 0x61261caaf000
```

```
futex(0x75e138e2977c, FUTEX_WAKE_PRIVATE, 2147483647) = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...},
↪  AT_EMPTY_PATH) = 0
write(1, "Enter filename: ", 16)        = 16
newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...},
↪  AT_EMPTY_PATH) = 0
read(0, "\n", 1024)                     = 1
read(0, "test.txt\n", 1024)             = 9
openat(AT_FDCWD, "/dev/shm/lab", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 1024)                      = 0
mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x75e138f28000
rt_sigaction(SIGUSR2, {sa_handler=0x6125e174a117, sa_mask=[USR2],
↪  sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x75e138842520},
↪  {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
↪  child_tidptr=0x75e138ec6690) = 940
pause()                                 = ? ERESTARTNOHAND (To be restarted if
↪  no handler)
--- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=940, si_uid=1000} ---
rt_sigreturn({mask=[]})                 = -1 EINTR (Interrupted system call)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=940, si_uid=1000,
↪  si_status=0, si_utime=0, si_stime=0} ---
write(1, "Answer: 30.000000\n", 18)     = 18
wait4(940, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 940
munmap(0x75e138f28000, 1024)            = 0
close(3)                                = 0
unlink("/dev/shm/lab")                  = 0
lseek(0, -1, SEEK_CUR)                  = -1 ESPIPE (Illegal seek)
exit_group(0)                           = ?
+++ exited with 0 +++
```