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: Д. А. Кузнецов
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 C++, 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.

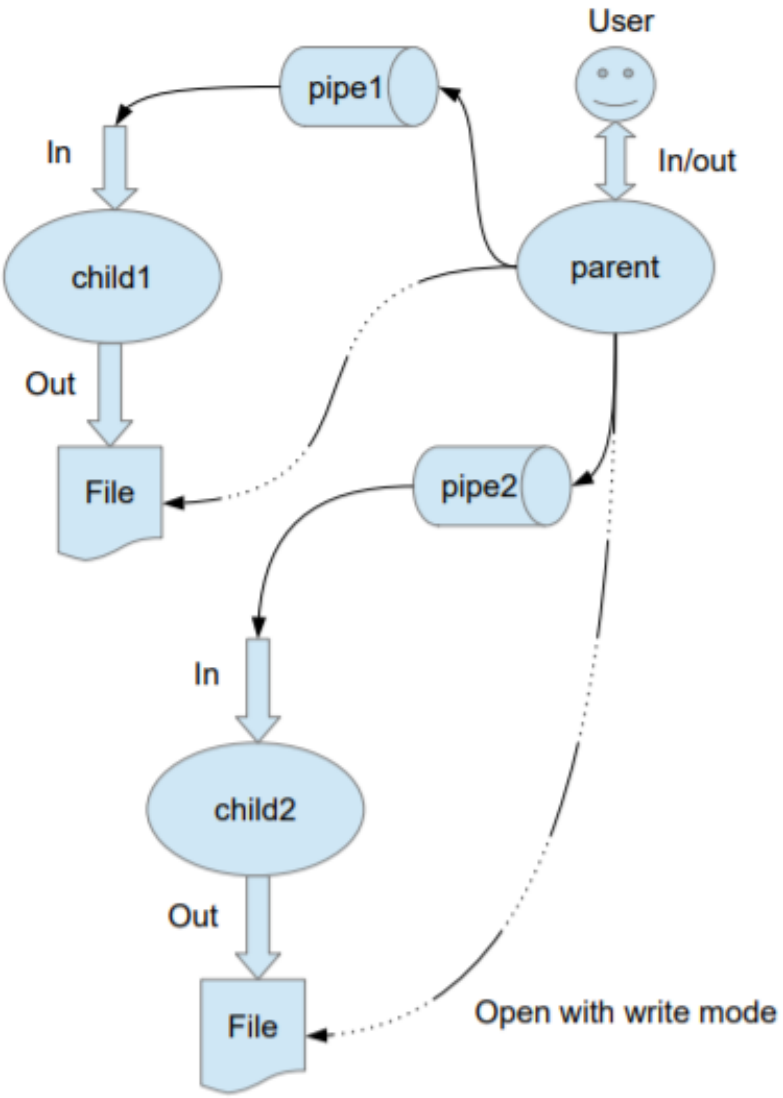

Рис. 1: Схема работы процессов.

Вариант: 18
Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их дочерним процесса. Процессы child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод. Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы удаляют все гласные из строк.

Метод решения
Алгоритм решения задачи:

1. Пользователь указывает имена файлов для двух дочерних процессов через консоль родительского процесса.
2. Создается управляющий объект Parent, который координирует работу всей системы.
3. Создаются две независимые области разделяемой памяти для обмена данными между родительским и каждым дочерним процессом.
4. Родительский процесс создает первого дочернего процесса через механизм fork(), запускает исполняемый файл дочернего процесса, передавая параметры: имя области памяти и целевой файл.
5. Каждый дочерний процесс подключается к своей области разделяемой памяти, открывает указанный файл в рабочей директории для записи результатов.
6. Родительский процесс постоянно проверяет статус дочерних процессов.
7. Родительский процесс читает пользовательский ввод из консоли и отправляет строки данных в соответствующие области разделяемой памяти, затем отправляет сигнал одному из дочерних процессов, уведомляя о наличии данных.
8. Дочерний процесс ожидает сигнал, читает строку, удаляет все гласные и записывает в соответствующий файл.
9. По команде “exit” или “quit” родитель останавливает дочерние процессы и программа завершается.

Ссылки:

- https://pubs.opengroup.org/onlinepubs/009696799/functions/write.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/execl.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/getppid.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/getpid.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/waitpid.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/sleep.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/exit.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/kill.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/shm_open.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/mmap.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/fork.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/signal.html

Описание программы

`main.cpp` — точка входа в программу, создается объект класса Parent, обрабатываются исключения.

`child-main.cpp` — точка входа в программу для дочернего процесса, создается объект класса Child.

`os.h` — объявление функций управления процессами и разделяемой памятью ОС.
`src/os.cpp` — реализация.

Поля класса:

- `SIGNAL-DATA-READY;` — сигнал о готовности данных (10).

Основные функции:

- `CreateProcess(const ProcessParams& params);` — создает новый процесс с заданными параметрами.
- `void TerminateProcess(ProcessHandle handle);` — принудительно завершает процесс.
- `bool IsAliveProcess(ProcessHandle handle);` — проверяет, выполняется ли процесс еще.

- `int GetChildPid(ProcessHandle handle);` — возвращает PID дочернего процесса.
- `int GetChildPid(ProcessHandle handle);` — возвращает PID дочернего процесса.
- `SharedMemHandle CreateSharedMemory(const char* name, size_t size);` — создает блок разделяемой памяти.
- `char* MapSharedMemory(SharedMemHandle handle, size_t size);` — подключает разделяемую память к адресному пространству.
- `void UnmapSharedMemory(char* ptr, size_t size);` — отключает разделяемую память.
- `void CloseSharedMemory(SharedMemHandle handle);` — закрывает handle разделяемой памяти.
- `void SendSignal(ProcessHandle handle, int signal);` — отправляет сигнал процессу.
- `int GetPid();` — получение pid текущего процесса. Используется системный вызов `getpid()`.
- `int Pause();` — приостанавливает выполнение процесса.
- `void Sleep(int seconds);` — приостанавливает выполнение на указанное время.
- `void Exit(int status);` — завершает процесс с заданным статусом.

`child.h` — объявление класса Child.
`src/child.cpp` — реализация.
Поля класса:

- `int pid;` — pid процесса.
- `std::string filename;` — название файла для открытия (создания).
- `std::ofstream file;` — поток для записи в файлы.
- `char* shm_ptr;` — указатель на отображённую разделяемую память.
- `size_t shm_size;` — размер разделяемой памяти.
- `std::string shm_name;` — имя разделяемой памяти.

Основные функции (`методы`):

- `void Work();` — ожидает сигнал, читает строку из разделяемой памяти, удаляет гласные и записывает в файл.

parent.h — объявление класса Parent.
src/parent.cpp — реализация.
Поля класса:

- `ProcessHandle child1;` — дескриптор 1 дочернего процесса.
- `ProcessHandle child2;` — дескриптор 2 дочернего процесса.
- `SharedMemHandle shm_handle1;` — дескриптор разделяемой памяти для 1 дочернего процесса.
- `SharedMemHandle shm_handle2;` — дескриптор разделяемой памяти для 2 дочернего процесса.
- `char* shm_ptr1;` — указатель на разделяемую память 1.
- `char* shm_ptr2;` — указатель на разделяемую память 2.
- `std::string shm_name1;` — имя разделяемой памяти 1.
- `std::string shm_name2;` — имя разделяемой памяти 2.
- `const size_t shm_size;` — размер разделяемой памяти.

Основные функции (методы):

- `void CreateChildProcesses(std::string filename1, std::string filename2);` — создает 2 дочерних процесса и запускает бинарный файл для дочернего процесса, при ошибке дочерний процесс завершается.
- `void Work();` — получает пользовательский ввод и записывает в одну из областей разделяемой памяти для дочерних процессов по правилу фильтрации, затем отправляет сигнал.
- `void EndChildren();` — завершает дочерние процессы.

stringprocessor.h — объявление класса Utils.
src/stringprocessor.cpp — реализация.
Поля класса:

- `static const std::string VOWELS;` — гласные буквы

Основные функции (методы):

- `static std::string RemoveVowels(const std::string str);` — удаляет гласную букву.
- `static bool IsVowel(char c);` — проверяет, является ли буква гласной.

Результаты

Программа принимает в качестве входных данных имена двух файлов. Затем она запускает два дочерних процесса, которые в текущей рабочей директории открывают указанные файлы.

После этого программа обрабатывает вводимые пользователем строки — за исключением команд «exit» и «quit». Из каждой строки удаляются все гласные буквы, а полученный результат записывается в один из двух файлов согласно заранее определённому правилу фильтрации.

Для организации взаимодействия между родительским процессом и двумя дочерними используются два механизма: разделяемая память (для передачи данных) и сигналы (для синхронизации и управления процессами).

По завершении работы программа формирует два файла в текущей директории.

В случае возникновения ошибок — например, если не удаётся создать или отобразить разделяемую память, либо если дочерние процессы аварийно завершаются — программа выполняет корректное завершение работы, обеспечивая безопасность и предотвращая утечки ресурсов.

Выводы

В ходе выполнения лабораторной работы были приобретены практические навыки в области межпроцессного взаимодействия. Была разработана и отлажена программа на языке C++, реализующая взаимодействие между родительским и дочерними процессами через разделяемую память и сигналы.

Программа демонстрирует:

- Создание дочерних процессов.
- Запуск новых исполняемых файлов в дочерних процессах.
- Управление разделяемой памятью.
- Синхронизацию процессов с помощью сигналов.
- Обработку системных ошибок и корректное завершение процессов.
- Абстракцию системных вызовов для обеспечения кроссплатформенности.

Исходная программа

```
#pragma once

#include <algorithm>
#include <fstream>
#include <iostream>
#include <string>
#include <unistd.h>
#include <signal.h>

#include "os.h"
#include "stringprocessor.h"

inline std::string BASEDIRECTORYFORFILES = "";

namespace child {
class Child {
private:
    int pid;
    std::string filename;
    std::ofstream file;
    char* shm_ptr;
    size_t shm_size = SHMSIZE;
    std::string shm_name;

    static volatile sig_atomic_t data_ready;
    static void SignalHandler(int sig);

public:
    Child(const std::string& shm_name, const std::string& filename);
    void Work();
    ~Child();
};
}
```

Листинг 1: child.h

```
#pragma once

#include <string>

namespace utils {
class StringProcessor {
private:
    static const std::string VOWELS;
public:
    static std::string RemoveVowels(const std::string& str);
    static bool IsVowel(char c);
};
}
```

Листинг 2: stringprocessor.h

```
#pragma once

#include <cstddef>
```

```cpp
#include <initializer_list>

using ProcessHandle = void*;
using SharedMemHandle = void*;

using SignalHandle = void(*)(int);

constexpr size_t SHMSIZE = 2048;

struct ProcessParams {
    const char* app;
    const char* arg1 = nullptr;
    const char* arg2 = nullptr;
    const char* arg3 = nullptr;
};

namespace os {

ProcessHandle CreateProcess(const ProcessParams& params);
void TerminateProcess(ProcessHandle handle);
bool IsAliveProcess(ProcessHandle handle);

int GetChildPid(ProcessHandle handle);

SharedMemHandle CreateSharedMemory(const char* name, size_t size);
char* MapSharedMemory(SharedMemHandle handle, size_t size);
void UnmapSharedMemory(char* ptr, size_t size);
void CloseSharedMemory(SharedMemHandle handle);

const int SIGNAL_DATA_READY = 10;
void Signal(int signal, SignalHandle handle);
void SendSignal(ProcessHandle handle, int signal);

int GetPid();
int Pause();
void Sleep(int seconds);
void Exit(int status);
}
```

Листинг 3: os.h

```cpp
#pragma once

#include <cerrno>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <random>
#include <string>

#include "os.h"

namespace parent {
class Parent {
private:
    ProcessHandle child1 = nullptr;
    ProcessHandle child2 = nullptr;
    SharedMemHandle shm_handle1 = nullptr;
```

```
    SharedMemHandle shm_handle2 = nullptr;
    char* shm_ptr1 = nullptr;
    char* shm_ptr2 = nullptr;
    std::string shm_name1 = "/shm_child1";
    std::string shm_name2 = "/shm_child2";
    const size_t shm_size = SHMSIZE;

    int lineCount = 0;

    void CreateSharedMemory();
    void SendToChild(const std::string& data, int childNum);
    void CheckChildrenAlive();

public:
    Parent();
    void CreateChildProcesses(std::string filename1, std::string filename2);
    void Work();
    void EndChildren();
    ~Parent();
};
}
```

Листинг 4: parent.h

```
#include <stringprocessor.h>

namespace utils {
const std::string StringProcessor::VOWELS = "aeiouAEIOU";

bool StringProcessor::IsVowel(char c) {
    return VOWELS.find(c) != std::string::npos;
}

std::string StringProcessor::RemoveVowels(const std::string& str) {
    std::string res;

    for (char c : str) {
        if (!IsVowel(c)) {
            res += c;
        }
    }

    return res;
}
}
```

Листинг 5: stringprocessor.cpp

```
#include <stdexcept>
#include <csignal>
#include <cstring>
#include <algorithm>

#include "child.h"
#include "os.h"
#include "stringprocessor.h"

```

```cpp
namespace child {

volatile sig_atomic_t Child::data_ready = 0;

void Child::SignalHandler(int sig) {
    if (sig == os::SIGNAL_DATA_READY) {
        data_ready = 1;
    }
}

Child::Child(const std::string& shm_name, const std::string& filename)
    : shm_name(shm_name), filename(BASEDIRECTORYFORFILES + filename), shm_ptr(nullptr)
        {

    pid = os::GetPid();


    SharedMemHandle shm_handle = os::CreateSharedMemory(shm_name.c_str(), shm_size);
    if (!shm_handle) {
        throw std::runtime_error("Failed to open shared memory: " + shm_name);
    }

    shm_ptr = os::MapSharedMemory(shm_handle, shm_size);
    if (!shm_ptr) {
        throw std::runtime_error("Failed to map shared memory: " + shm_name);
    }


    file.open(this->filename);
    if (!file.is_open()) {
        throw std::runtime_error("Failed to open file: " + this->filename);
    }


    os::Signal(os::SIGNAL_DATA_READY, SignalHandler);

    std::cout << "Child " << pid << " initialized. SHM: " << shm_name
              << ", File: " << this->filename << std::endl;
}

void Child::Work() {
    std::cout << "Child [" << pid << "] started working..." << std::endl;

    while (true) {

        while (!data_ready) {
            os::Pause();
        }

        data_ready = 0;

        std::string input_str(shm_ptr);

        if (input_str == "TERMINATE") {
            std::cout << "Child [" << pid << "] received termination signal" << std::
                endl;
            break;
        }
```

```
66
67          std::string result = utils::StringProcessor::RemoveVowels(input_str);
68
69          file << result << std::endl;
70          file.flush();
71
72          std::cout << "Child " << pid << " processed: \"" << input_str
73                    << "\" -> \"" << result << "\"" << std::endl;
74
75          memset(shm_ptr, 0, shm_size);
76      }
77
78      std::cout << "Child " << pid << " finished work" << std::endl;
79  }
80
81  Child::~Child() {
82      if (file.is_open()) {
83          file.close();
84      }
85
86      if (shm_ptr) {
87          os::UnmapSharedMemory(shm_ptr, shm_size);
88      }
89  }
90  }
```

Листинг 6: child.cpp

```
1   #include <sys/mman.h>
2   #include <sys/wait.h>
3   #include <fcntl.h>
4   #include <unistd.h>
5   #include <signal.h>
6   #include <cstring>
7   #include <iostream>
8   #include <stdexcept>
9   #include <cstdint>
10
11  #include "os.h"
12
13  namespace os {
14
15  ProcessHandle CreateProcess(const ProcessParams& params) {
16      pid_t pid = fork();
17
18      if (pid == -1) {
19          return nullptr;
20      }
21
22      if (pid == 0) {
23
24          if (params.arg3) {
25              execl(params.app, params.app, params.arg1, params.arg2, params.arg3,
                      nullptr);
26          } else if (params.arg2) {
27              execl(params.app, params.app, params.arg1, params.arg2, nullptr);
28          } else if (params.arg1) {
29              execl(params.app, params.app, params.arg1, nullptr);
```

```
        } else {
            execl(params.app, params.app, nullptr);
        }

        Exit(1);
    }

    return reinterpret_cast<ProcessHandle>(static_cast<uintptr_t>(pid));
}

void TerminateProcess(ProcessHandle handle) {
    if (handle == nullptr) return;
    pid_t pid = static_cast<pid_t>(reinterpret_cast<uintptr_t>(handle));
    kill(pid, SIGTERM);
    waitpid(pid, nullptr, 0);
}

bool IsAliveProcess(ProcessHandle handle) {
    if (handle == nullptr) return false;
    pid_t pid = static_cast<pid_t>(reinterpret_cast<uintptr_t>(handle));
    return kill(pid, 0) == 0;
}

int GetChildPid(ProcessHandle handle) {
    if (handle == nullptr) return -1;
    return static_cast<int>(reinterpret_cast<uintptr_t>(handle));
}

SharedMemHandle CreateSharedMemory(const char* name, size_t size) {
    int shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        return nullptr;
    }

    if (ftruncate(shm_fd, size) == -1) {
        close(shm_fd);
        return nullptr;
    }

    return reinterpret_cast<SharedMemHandle>(static_cast<uintptr_t>(shm_fd));
}

char* MapSharedMemory(SharedMemHandle handle, size_t size) {
    if (handle == nullptr) return nullptr;
    int shm_fd = static_cast<int>(reinterpret_cast<uintptr_t>(handle));
    void* ptr = mmap(nullptr, size, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);

    if (ptr == MAP_FAILED) {
        return nullptr;
    }

    return static_cast<char*>(ptr);
}

void UnmapSharedMemory(char* ptr, size_t size) {
    if (ptr != nullptr) {
        munmap(ptr, size);
    }
```

```
}

void CloseSharedMemory(SharedMemHandle handle) {
    if (handle == nullptr) return;
    int shm_fd = static_cast<int>(reinterpret_cast<uintptr_t>(handle));
    close(shm_fd);
}

void Signal(int signal, SignalHandle handler) {
    struct sigaction sa;
    sa.sa_handler = handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(signal, &sa, nullptr);
}

void SendSignal(ProcessHandle handle, int signal) {
    if (handle == nullptr) return;
    pid_t pid = static_cast<pid_t>(reinterpret_cast<uintptr_t>(handle));
    kill(pid, signal);
}

int GetPid() {
    return getpid();
}

int Pause() {
    return pause();
}

void Sleep(int seconds) {
    sleep(seconds);
}

void Exit(int status) {
    _exit(status);
}
}
```

Листинг 7: os.cpp

```
#include <stdexcept>
#include <csignal>

#include "parent.h"
#include "os.h"

namespace parent {

Parent::Parent() {
    CreateSharedMemory();
}

void Parent::CreateSharedMemory() {
    shm_handle1 = os::CreateSharedMemory(shm_name1.c_str(), shm_size);
    if (shm_handle1 == nullptr) {
        throw std::runtime_error("Failed to create shared memory for child1");
    }
```

```
18
19      shm_ptr1 = os::MapSharedMemory(shm_handle1, shm_size);
20      if (!shm_ptr1) {
21          throw std::runtime_error("Failed to map shared memory for child1");
22      }
23      memset(shm_ptr1, 0, shm_size);
24
25      shm_handle2 = os::CreateSharedMemory(shm_name2.c_str(), shm_size);
26      if (shm_handle2 == nullptr) {
27          throw std::runtime_error("Failed to create shared memory for child2");
28      }
29
30      shm_ptr2 = os::MapSharedMemory(shm_handle2, shm_size);
31      if (!shm_ptr2) {
32          throw std::runtime_error("Failed to map shared memory for child2");
33      }
34      memset(shm_ptr2, 0, shm_size);
35
36      std::cout << "Shared memory created successfully" << std::endl;
37  }
38
39  void Parent::CreateChildProcesses(std::string filename1, std::string filename2) {
40      ProcessParams params1;
41      params1.app = "./child";
42      params1.arg1 = shm_name1.c_str();
43      params1.arg2 = filename1.c_str();
44
45      child1 = os::CreateProcess(params1);
46      if (child1 == nullptr) {
47          throw std::runtime_error("Failed to create child process 1");
48      }
49
50      ProcessParams params2;
51      params2.app = "./child";
52      params2.arg1 = shm_name2.c_str();
53      params2.arg2 = filename2.c_str();
54
55      child2 = os::CreateProcess(params2);
56      if (child2 == nullptr) {
57          os::TerminateProcess(child1);
58          throw std::runtime_error("Failed to create child process 2");
59      }
60
61      std::cout << "Child processes created successfully: PID1=" << child1
62                << ", PID2=" << child2 << std::endl;
63      std::cout << "Waiting for children to initialize..." << std::endl;
64
65      os::Sleep(1);
66  }
67
68  void Parent::Work() {
69      std::cout << "Enter strings (type 'exit' or 'quit' to stop):" << std::endl;
70
71      std::string line;
72      while (std::getline(std::cin, line)) {
73          if (line == "exit" || line == "quit") {
74              break;
75          }
```

```
76
77          if (line.empty()) {
78              continue;
79          }
80
81          CheckChildrenAlive();
82
83          lineCount++;
84
85          if (lineCount % 2 == 1) {
86              SendToChild(line, 1);
87              std::cout << "Sent to child1 (odd): " << line << std::endl;
88          } else {
89              SendToChild(line, 2);
90              std::cout << "Sent to child2 (even): " << line << std::endl;
91          }
92      }
93
94      std::cout << "Finished processing input" << std::endl;
95  }
96
97  void Parent::SendToChild(const std::string& data, int childNum) {
98      char* shm_ptr = (childNum == 1) ? shm_ptr1 : shm_ptr2;
99      ProcessHandle child = (childNum == 1) ? child1 : child2;
100
101     strncpy(shm_ptr, data.c_str(), shm_size - 1);
102     shm_ptr[shm_size - 1] = '\0';
103
104     os::SendSignal(child, os::SIGNAL_DATA_READY);
105 }
106
107 void Parent::CheckChildrenAlive() {
108     if (!os::IsAliveProcess(child1)) {
109         throw std::runtime_error("Child process 1 is not alive");
110     }
111     if (!os::IsAliveProcess(child2)) {
112         throw std::runtime_error("Child process 2 is not alive");
113     }
114 }
115
116 void Parent::EndChildren() {
117
118     if (shm_ptr1) {
119         strncpy(shm_ptr1, "TERMINATE", shm_size - 1);
120         os::SendSignal(child1, os::SIGNAL_DATA_READY);
121     }
122
123     if (shm_ptr2) {
124         strncpy(shm_ptr2, "TERMINATE", shm_size - 1);
125         os::SendSignal(child2, os::SIGNAL_DATA_READY);
126     }
127
128     os::Sleep(1);
129
130     if (os::IsAliveProcess(child1)) {
131         os::TerminateProcess(child1);
132     }
133     if (os::IsAliveProcess(child2)) {
```

```
        os::TerminateProcess(child2);
    }
}

Parent::~Parent() {
    EndChildren();

    if (shm_ptr1) {
        os::UnmapSharedMemory(shm_ptr1, shm_size);
    }
    if (shm_ptr2) {
        os::UnmapSharedMemory(shm_ptr2, shm_size);
    }

    if (shm_handle1 != nullptr) {
        os::CloseSharedMemory(shm_handle1);
    }
    if (shm_handle2 != nullptr) {
        os::CloseSharedMemory(shm_handle2);
    }
}
}
```

Листинг 8: parent.cpp

```
#include <iostream>
#include <string>

#include "parent.h"

int main() {
    try {
        std::string filename1, filename2;

        std::cout << "Enter filename for child1: ";
        std::getline(std::cin, filename1);

        std::cout << "Enter filename for child2: ";
        std::getline(std::cin, filename2);

        parent::Parent parent;
        parent.CreateChildProcesses(filename1, filename2);
        parent.Work();

        std::cout << "Program finished successfully" << std::endl;

    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

Листинг 9: main.cpp

```cpp
#include "child.h"
#include <iostream>
#include <string>
#include <stdexcept>

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

    std::string shm_name = argv[1];
    std::string filename = argv[2];

    try {
        child::Child child(shm_name, filename);
        child.Work();
    } catch (const std::exception& e) {
        std::cerr << "Child error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

Листинг 10: child-main.cpp

Strace

```
execve("./main",["./main"],0x7ffe3b957960 /* 75 vars */) = 0
brk(NULL)                                       = 0x60aa3b74c000
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7e2ecd275000
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=71087,...}) = 0
mmap(NULL,71087,PROT_READ,MAP_PRIVATE,3,0) = 0x7e2ecd263000
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
fstat(3,{st_mode=S_IFREG|0644,st_size=2592224,...}) = 0
mmap(NULL,2609472,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7e2ecce00000
mmap(0x7e2ecce9d000,1343488,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3
= 0x7e2ecce9d000
mmap(0x7e2eccfe5000,552960,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x1e5000)
= 0x7e2eccfe5000
mmap(0x7e2ecd06c000,57344,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x7e2ecd06c000
mmap(0x7e2ecd07a000,12608,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1
= 0x7e2ecd07a000
```

```
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
fstat(3,{st_mode=S_IFREG|0644,st_size=183024,...}) = 0
mmap(NULL,185256,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7e2ecd235000
mmap(0x7e2ecd239000,147456,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x7e2ecd239000
mmap(0x7e2ecd25d000,16384,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x28000)
= 0x7e2ecd25d000
mmap(0x7e2ecd261000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x7e2ecd261000
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) = 0x7e2ecca00000
mmap(0x7e2ecca28000,1605632,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3
= 0x7e2ecca28000
mmap(0x7e2eccbb0000,323584,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x1b0000)
= 0x7e2eccbb0000
mmap(0x7e2eccbff000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x7e2eccbff000
mmap(0x7e2eccc05000,52624,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1
= 0x7e2eccc05000
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
fstat(3,{st_mode=S_IFREG|0644,st_size=952616,...}) = 0
mmap(NULL,950296,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7e2ecd14c000
mmap(0x7e2ecd15c000,520192,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x7e2ecd15c000
mmap(0x7e2ecd1db000,360448,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x8f000)
= 0x7e2ecd1db000
mmap(0x7e2ecd233000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x7e2ecd233000
close(3)                                = 0
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7e2ecd14a000
mmap(NULL,12288,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7e2ecd147000
arch_prctl(ARCH_SET_FS,0x7e2ecd147740) = 0
set_tid_address(0x7e2ecd147a10)         = 90617
```

```
set_robust_list(0x7e2ecd147a20,24)     = 0
rseq(0x7e2ecd148060,0x20,0,0x53053053) = 0
mprotect(0x7e2eccbff000,16384,PROT_READ) = 0
mprotect(0x7e2ecd233000,4096,PROT_READ) = 0
mprotect(0x7e2ecd261000,4096,PROT_READ) = 0
mprotect(0x7e2ecd06c000,45056,PROT_READ) = 0
mprotect(0x60aa0c7e2000,4096,PROT_READ) = 0
mprotect(0x7e2ecd2b3000,8192,PROT_READ) = 0
prlimit64(0,RLIMIT_STACK,NULL,{rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY})
= 0
munmap(0x7e2ecd263000,71087)           = 0
futex(0x7e2ecd07a7bc,FUTEX_WAKE_PRIVATE,2147483647) = 0
getrandom("\x63\x06\xff\x05\xf4\xa7\x2e\x1d",8,GRND_NONBLOCK) = 8
brk(NULL)                               = 0x60aa3b74c000
brk(0x60aa3b76d000)                     = 0x60aa3b76d000
fstat(1,{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0),...}) = 0
getpid()                                = 90617
write(1,"Parent[90617]: Enter filename fo"...,41) = 41
fstat(0,{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0),...}) = 0
read(0,"1.txt\n",1024)                 = 6
getpid()                                = 90617
write(1,"Parent[90617]: Enter filename fo"...,41) = 41
read(0,"2.txt\n",1024)                 = 6
openat(AT_FDCWD,"/dev/shm/shm_child1",O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC,0666)
= 3
ftruncate(3,2048)                       = 0
mmap(NULL,2048,PROT_READ|PROT_WRITE,MAP_SHARED,3,0) = 0x7e2ecd274000
openat(AT_FDCWD,"/dev/shm/shm_child2",O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC,0666)
= 4
ftruncate(4,2048)                       = 0
mmap(NULL,2048,PROT_READ|PROT_WRITE,MAP_SHARED,4,0) = 0x7e2ecd273000
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_ti
= 90672
write(1,"Child1[90672]: process was creat"...,36) = 36
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_ti
= 90673
write(1,"Child2[90673]: process was creat"...,36) = 36
clock_nanosleep(CLOCK_REALTIME,0,{tv_sec=1,tv_nsec=0},0x7ffd604a08e0) = 0
wait4(90672,0x7ffd604a0914,WNOHANG,NULL) = 0
wait4(90673,0x7ffd604a0914,WNOHANG,NULL) = 0
read(0,"aaaaaa\n",1024)                = 7
wait4(90672,0x7ffd604a0924,WNOHANG,NULL) = 0
wait4(90673,0x7ffd604a0924,WNOHANG,NULL) = 0
kill(90672,SIGUSR1)                     = 0
read(0,"sssssssssss\n",1024)           = 12
wait4(90672,0x7ffd604a0924,WNOHANG,NULL) = 0
wait4(90673,0x7ffd604a0924,WNOHANG,NULL) = 0
kill(90673,SIGUSR1)                     = 0
```

```
read(0,"fffffffffff\n",1024)         = 13
wait4(90672,0x7ffd604a0924,WNOHANG,NULL) = 0
wait4(90673,0x7ffd604a0924,WNOHANG,NULL) = 0
kill(90672,SIGUSR1)                   = 0
read(0,"exit\n",1024)                = 5
write(1,"Parent: termination command was "...,42) = 42
kill(90672,SIGTERM)                   = 0
wait4(90672,[{WIFSIGNALED(s) && WTERMSIG(s) == SIGTERM}],0,NULL) = 90672
---SIGCHLD {si_signo=SIGCHLD,si_code=CLD_KILLED,si_pid=90672,si_uid=1000,si_status=SI
---
write(1,"Child1: process was terminated.\n",32) = 32
kill(90673,SIGTERM)                   = 0
wait4(90673,[{WIFSIGNALED(s) && WTERMSIG(s) == SIGTERM}],0,NULL) = 90673
---SIGCHLD {si_signo=SIGCHLD,si_code=CLD_KILLED,si_pid=90673,si_uid=1000,si_status=SI
---
write(1,"Child2: process was terminated.\n",32) = 32
munmap(0x7e2ecd274000,2048)           = 0
close(3)                               = 0
munmap(0x7e2ecd273000,2048)           = 0
close(4)                               = 0
exit_group(0)                          = ?
+++ exited with 0 +++
```