

**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. М. Мишин
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.

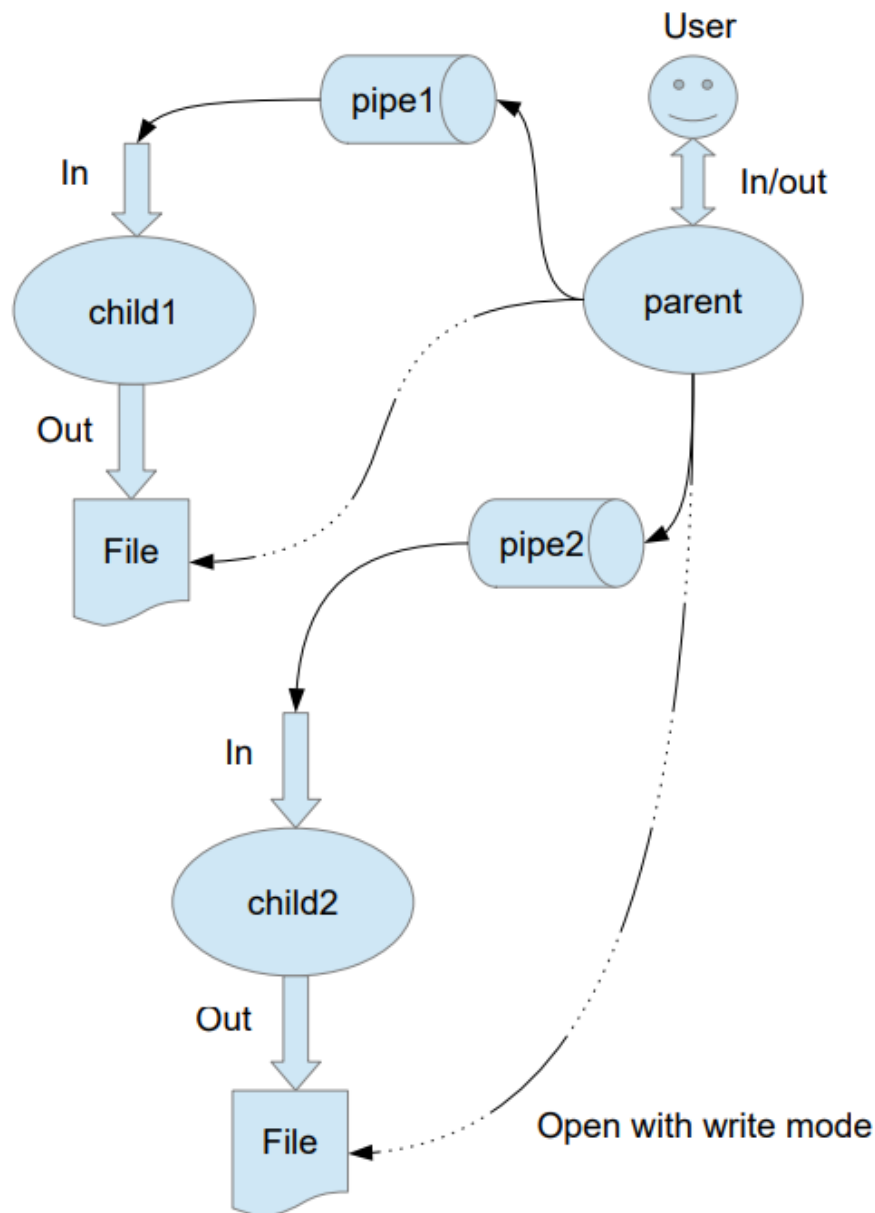


Рис. 1: Схема работы процессов.

Вариант: 22

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 процессам. Процессы child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод. Правило фильтрации: с вероятностью 80% над строками работает child1, иначе в child2. Дочерние процессы инвертируют строки.

Метод решения

Алгоритм решения задачи:

1. Пользователь в консоль родительского процесса вводит имена файлов, которые будут использованы для открытия (создания) файлов с таким именем для child1 и child2 соответственно.
2. Создается объект класса Parent, так называемый “родитель”.
3. Родительский процесс создает два объекта разделяемой памяти для передачи данных между родительским и дочерними процессами.
4. Затем делает fork для создания первого дочернего процесса. Пытается запустить бинарный файл для дочернего процесса и передать аргументы - имя разделяемой памяти и имя файла для записи. Если не получилось — процесс завершается. Аналогично для второго дочернего процесса.
5. Дочерний процесс открывает свою область разделяемой памяти. И пытается открыть (создать) указанный файл в специальной директории. Если не получилось — процесс завершается.
6. Родительский процесс проверяет “жизнеспособность” процессов, если любой из них уже завершен, то программа прекращает работу со статусом ошибки.
7. Родительский процесс считывает пользовательский ввод 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 из разделяемой памяти, инвертирует её и записывает в соответствующий файл.
9. По команде “exit” или “quit” родитель останавливает дочерние процессы и программа завершается.

Архитектура программы:

```
lab1/
├── bin/
│   └── child.cpp
├── build/
├── include/
│   ├── child.h
│   ├── exceptions.h
│   ├── os.h
│   └── parent.h
├── src/
│   ├── child.cpp
│   ├── os.cpp
│   └── parent.cpp
├── testfiles/
└── main.cpp
```

Описание программы

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

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

`exceptions.h` — объявление необходимых исключений.

- `ChildProcessEndException` — ошибка внезапного завершения любого из дочерних процессов, программа завершается.

`os.h` — объявление функций управления процессами и разделяемой памятью ОС.

`src/os.cpp` — реализация.

Основные функции:

- `ProcessHandle CreateProcess(const ProcessParams& params);` — создание дочернего процесса. Используется системный вызов `fork()` и `execl()`.
- `void TerminateProcess(ProcessHandle handle);` — завершение процесса по `handle` и очистка ресурсов. Используется системный вызов `kill()` и `waitpid()`.
- `bool IsAliveProcess(ProcessHandle handle);` — проверка “жизни/деятельности” процесса. Используется системный вызов `waitpid()` с макросом `WNOHANG`.
- `int GetChildPid(ProcessHandle handle);` — получение `pid` дочернего процесса.
- `SharedMemHandle CreateSharedMemory(const char* name, size_t size);` — создание POSIX shared memory объекта. Используется системный вызов `shm_open()` и `ftruncate()`.
- `char* MapSharedMemory(SharedMemHandle handle, size_t size);` — отображение shared memory в адресное пространство. Используется системный вызов `mmap()`.
- `void UnmapSharedMemory(char* ptr, size_t size);` — отмена отображения памяти. Используется системный вызов `munmap()`.

- `void CloseSharedMemory(SharedMemHandle handle);` — закрытие дескриптора shared memory. Используется системный вызов `close()`.
- `void SendSignal(ProcessHandle handle, int signal);` — отправка сигнала дочернему процессу. Используется системный вызов `kill()`.
- `void Signal(int sig, SignalHandle handle);` — установка обработчика сигнала. Используется системный вызов `signal()`.
- `int GetPid();` — получение pid текущего процесса. Используется системный вызов `getpid()`.
- `int Pause();` — ожидание сигнала. Используется системный вызов `pause()`.
- `void Sleep(int seconds);` — ожидание. Используется системный вызов `sleep()`.
- `void Exit(int status);` — завершение текущего процесса. Используется системный вызов `_exit()`.

`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, инвертирует её и записывает в файл.

Основные функции (методы):

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

`parent.h` — объявление класса `Parent`.

`src/parent.cpp` — реализация.

Поля класса:

- `std::random_device rd;` — генератор случайных чисел.
- `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.

Результаты

Программа получает на вход названия двух файлов, создает два дочерних процесса, эти процессы в специальной директории (testfiles) открывают (создают) эти файлы. Далее все введенные пользователем строки, исключая “exit” и “quit”, инвертируются и записываются в один из файлов в зависимости от правила фильтрации. Взаимодействие между родительским и дочерними процессами осуществляется через разделяемую память и сигналы. Результатом являются два файла в директории “testfiles”. Если не удалось создать разделяемую память, отобразить её, или дочерние процессы завершились, программа безопасно прекращает свою работу.

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

```
1 | #pragma once
2 |
3 | #include <exception>
4 | #include <string>
5 |
6 | namespace exceptions {
7 |     class ChildProcessEndException: std::exception {
8 |     public:
9 |         explicit ChildProcessEndException(const std::string& text): error_message_(text)
10 |             {}
11 |         const char* what() const noexcept override {
12 |             return error_message_.data();
13 |         }
14 |     private:
15 |         std::string error_message_;
16 |     };
17 | }
```

Листинг 1: exceptions.h

```
1 | #pragma once
2 |
3 | #include <cstddef>
4 | #include <initializer_list>
5 |
6 | using ProcessHandle = void*;
7 | using SharedMemHandle = void*;
8 |
9 | using SignalHandle = void(*)(int);
10 |
11 | namespace {
12 |     constexpr size_t SHMSIZE = 2048;
13 | }
14 |
15 | struct ProcessParams {
16 |     const char* app;
17 |     const char* arg1 = nullptr;
18 |     const char* arg2 = nullptr;
19 |     const char* arg3 = nullptr;
20 | };
21 |
22 | namespace os {
23 |     ProcessHandle CreateProcess(const ProcessParams& params);
24 |     void TerminateProcess(ProcessHandle handle);
25 |     bool IsAliveProcess(ProcessHandle handle);
26 |
27 |     int GetChildPid(ProcessHandle handle);
28 |
29 |     SharedMemHandle CreateSharedMemory(const char* name, size_t size);
30 |     char* MapSharedMemory(SharedMemHandle handle, size_t size);
31 |     void UnmapSharedMemory(char* ptr, size_t size);
32 |     void CloseSharedMemory(SharedMemHandle handle);
33 |
34 |     const int SIGNAL_DATA_READY = 10;
35 |     const int SIGNAL_TERMINATE = 15;
36 |     void Signal(int signal, SignalHandle handle);
37 |     void SendSignal(ProcessHandle handle, int signal);
38 | }
```

```

38
39     int GetPid();
40     int Pause();
41     void Sleep(int seconds);
42     void Exit(int status);
43 }

```

Листинг 2: os.h

```

1  #include <sys/mman.h>
2  #include <sys/stat.h>
3  #include <fcntl.h>
4  #include <unistd.h>
5  #include <signal.h>
6  #include <sys/wait.h>
7  #include <cstring>
8  #include <cerrno>
9  #include <cstdlib>
10 #include <iostream>
11 #include <stdexcept>
12
13 #include "os.h"
14
15 struct ProcessHandleData {
16     pid_t pid;
17 };
18
19 struct SharedMemHandleData {
20     int fd;
21     size_t size;
22 };
23
24 namespace os {
25     ProcessHandle CreateProcess(const ProcessParams& params) {
26         pid_t pid = fork();
27         if (pid == -1) {
28             perror("Error: failed to fork");
29             throw std::runtime_error("Fork failed");
30         }
31         if (pid == 0) {
32             execl(params.app, params.arg1, params.arg2, params.arg3, NULL);
33             perror("Error: failed to execl");
34             _exit(1);
35         }
36         auto* data = new ProcessHandleData{pid};
37         return reinterpret_cast<ProcessHandle>(data);
38     }
39
40     int GetChildPid(ProcessHandle handle) {
41         ProcessHandleData* data = reinterpret_cast<ProcessHandleData*>(handle);
42         return data->pid;
43     }
44
45     void TerminateProcess(ProcessHandle handle) {
46         ProcessHandleData* data = reinterpret_cast<ProcessHandleData*>(handle);
47         pid_t pid = data->pid;
48         if (kill(pid, SIGTERM) == -1) {
49             perror("Warning: failed to terminate process");
50         }

```



```

51     int status;
52     waitpid(pid, &status, 0);
53     delete data;
54 }
55
56 void SendSignal(ProcessHandle handle, int signal) {
57     ProcessHandleData* data = reinterpret_cast<ProcessHandleData*>(handle);
58     pid_t pid = data->pid;
59     if (kill(pid, signal) == -1) {
60         perror("Warning: failed to send signal");
61     }
62 }
63
64 bool IsAliveProcess(ProcessHandle handle) {
65     ProcessHandleData* data = reinterpret_cast<ProcessHandleData*>(handle);
66     pid_t pid = data->pid;
67     int status;
68     pid_t result = waitpid(pid, &status, WNOHANG);
69     if (result == -1) {
70         perror("Error: failed to waitpid");
71         return false;
72     }
73     return (result == 0);
74 }
75
76 void Sleep(int seconds) {
77     sleep(seconds);
78 }
79
80 SharedMemHandle CreateSharedMemory(const char* name, size_t size) {
81     int fd = shm_open(name, O_CREAT | O_RDWR, 0666);
82     if (fd == -1) {
83         perror("Error: failed to shm_open");
84         throw std::runtime_error("Shm_open failed");
85     }
86     if (ftruncate(fd, size) == -1) {
87         perror("Error: failed to ftruncate");
88         close(fd);
89         shm_unlink(name);
90         throw std::runtime_error("Ftruncate failed");
91     }
92     auto* data = new SharedMemHandleData{fd, size};
93     return reinterpret_cast<SharedMemHandle>(data);
94 }
95
96 char* MapSharedMemory(SharedMemHandle handle, size_t size) {
97     SharedMemHandleData* data = reinterpret_cast<SharedMemHandleData*>(handle);
98     void* ptr = mmap(nullptr, size, PROT_READ | PROT_WRITE, MAP_SHARED, data->fd, 0);
99     if (ptr == MAP_FAILED) {
100         throw std::runtime_error("Error: failed to mmap");
101     }
102     return reinterpret_cast<char*>(ptr);
103 }
104
105 void UnmapSharedMemory(char* ptr, size_t size) {
106     if (munmap(ptr, size) == -1) {
107         perror("Warning: failed to munmap");
108     }
109 }
110

```

```

111     void CloseSharedMemory(SharedMemHandle handle) {
112         SharedMemHandleData* data = reinterpret_cast<SharedMemHandleData*>(handle);
113         if (close(data->fd) == -1) {
114             perror("Warning: failed to close fd");
115         }
116         delete data;
117     }
118
119     void Signal(int sig, SignalHandle handle) {
120         if (signal(sig, handle) == SIG_ERR) {
121             perror("Warning: signal failed");
122         }
123     }
124
125     int Pause() {
126         return pause();
127     }
128
129     int GetPid() {
130         return getpid();
131     }
132
133     void Exit(int status) {
134         _exit(status);
135     }
136 }

```

Листинг 3: os.cpp

```

1  #pragma once
2
3  #include <algorithm>
4  #include <fstream>
5  #include <iostream>
6  #include <string>
7  #include <unistd.h>
8
9  #include "os.h"
10
11 namespace {
12     inline std::string BASEDIRECTORYFORFILES = "../testfiles/";
13 }
14
15 static void signal_handler(int sig) {}
16
17 namespace child {
18     class Child {
19     private:
20         int pid;
21
22         std::string filename;
23         std::ofstream file;
24
25         char* shm_ptr;
26         size_t shm_size = SHMSIZE;
27         std::string shm_name;
28     public:
29         Child(const std::string& shm_name, const std::string& filename);
30         void Work();

```

```

31         ~Child();
32     };
33 }

```

Листинг 4: child.h

```

1  #include <signal.h>
2  #include <cstring>
3
4  #include "child.h"
5  #include "os.h"
6
7
8
9  namespace child {
10     Child::Child(const std::string& shm_name, const std::string& filename): pid(os::
        GetPid()), filename(BASEDIRECTORYFORFILES + filename), shm_name(shm_name) {
11         auto shm_handle = os::CreateSharedMemory(shm_name.c_str(), shm_size);
12         if (!shm_handle) {
13             std::cout << "Child[" << pid << "]: failed to open shared memory. Process is
                over.\n";
14             os::Exit(1);
15         }
16
17         shm_ptr = os::MapSharedMemory(shm_handle, shm_size);
18         if (!shm_ptr) {
19             std::cout << "Child[" << pid << "]: failed to map shared memory. Process is
                over.\n";
20             os::Exit(1);
21         }
22
23         file.open(Child::filename, std::ios::app);
24         if (!file.is_open()) {
25             std::cout << "Child[" << pid << "]: failed to open " << Child::filename << ".
                Process is over.\n";
26             os::Exit(1);
27         }
28         std::cout << "Child[" << pid << "]: file " << Child::filename << " opened." << std
            ::endl;
29         os::Signal(os::SIGNAL_DATA_READY, signal_handler);
30     }
31
32     void Child::Work() {
33         if (!file.is_open()) {
34             std::cout << "Child[" << pid << "]: the file is not open! Work is impossible."
                << std::endl;
35             os::Exit(1);
36         }
37
38         std::string line;
39         std::string inv;
40
41         while (true) {
42
43             os::Pause();
44
45             line = shm_ptr;
46
47             if (line.empty()) continue;

```

```

48
49         std::cout << "Child[" << pid << "]: the string " << line << " was got." << std
           ::endl;
50
51         inv = line;
52         std::reverse(inv.begin(), inv.end());
53         std::cout << "Child[" << pid << "]: the string " << line << " was reversed."
           << std::endl;
54
55         file << inv << std::endl;
56         std::cout << "Child[" << pid << "]: the string " << inv << " was written in
           file " << filename << "." << std::endl;
57     }
58 }
59
60 Child::~Child() {
61     if (file.is_open()) {
62         file.close();
63     }
64 }
65 }

```

Листинг 5: child.cpp

```

1  #include "child.h"
2
3  int main(int argc, char* argv[]) {
4      if (argc < 3) {
5          std::cerr << "Error: invalid agrs" << std::endl;
6          return 1;
7      }
8
9      std::string shm_name = argv[1];
10     std::string filename = argv[2];
11     child::Child child(shm_name, filename);
12     child.Work();
13     return 0;
14 }

```

Листинг 6: bin/child.cpp

```

1  #pragma once
2
3  #include <cerrno>
4  #include <cstdlib>
5  #include <cstring>
6  #include <iostream>
7  #include <random>
8
9
10 #include "exceptions.h"
11 #include "os.h"
12
13 namespace parent {
14     class Parent {
15     private:
16         std::random_device rd;

```

```

17     ProcessHandle child1;
18     ProcessHandle child2;
19     SharedMemHandle shm_handle1;
20     SharedMemHandle shm_handle2;
21     char* shm_ptr1;
22     char* shm_ptr2;
23     std::string shm_name1 = "/shm_child1";
24     std::string shm_name2 = "/shm_child2";
25     const size_t shm_size = SHMSIZE;
26
27     public:
28         Parent();
29         void CreateChildProcesses(std::string filename1, std::string filename2);
30         void Work();
31         void EndChildren();
32         ~Parent();
33     };
34 }

```

Листинг 7: parent.h

```

1  #include "exceptions.h"
2  #include "parent.h"
3  #include "os.h"
4
5  namespace parent {
6      Parent::Parent() {
7          shm_handle1 = os::CreateSharedMemory(shm_name1.c_str(), shm_size);
8          shm_ptr1 = os::MapSharedMemory(shm_handle1, shm_size);
9
10         shm_handle2 = os::CreateSharedMemory(shm_name2.c_str(), shm_size);
11         shm_ptr2 = os::MapSharedMemory(shm_handle2, shm_size);
12     }
13
14     void Parent::CreateChildProcesses(std::string filename1, std::string filename2) {
15         child1 = os::CreateProcess(ProcessParams{"/child", "child1", shm_name1.c_str(),
16             filename1.c_str()});
17         std::cout << "Child1[" << os::GetChildPid(child1) << "]: process was created." <<
18             std::endl;
19
20         child2 = os::CreateProcess(ProcessParams{"/child", "child2", shm_name2.c_str(),
21             filename2.c_str()});
22         std::cout << "Child2[" << os::GetChildPid(child2) << "]: process was created." <<
23             std::endl;
24
25         os::Sleep(1);
26
27         if (!os::IsAliveProcess(child1) || !os::IsAliveProcess(child2)) {
28             throw exceptions::ChildProcessEndException("Child process already terminated!");
29         }
30     }
31
32     void Parent::Work() {
33         std::string input;
34         while (std::getline(std::cin, input)) {
35             if (input == "exit" || input == "quit") {
36                 std::cout << "Parent: termination command was received." << std::endl;
37                 break;
38             }
39         }
40     }
41 }

```

```

34     }
35     if (!os::IsAliveProcess(child1) || !os::IsAliveProcess(child2)) {
36         throw exceptions::ChildProcessEndException("Child process already
37             terminated!");
38     }
39     if (input.empty()) continue;
40     if (rd() % 100 < 80) {
41         strncpy(shm_ptr1, input.c_str(), shm_size - 1);
42         os::SendSignal(child1, os::SIGNAL_DATA_READY);
43     } else {
44         strncpy(shm_ptr2, input.c_str(), shm_size - 1);
45         os::SendSignal(child2, os::SIGNAL_DATA_READY);
46     }
47 }
48
49 void Parent::EndChildren() {
50     os::TerminateProcess(child1);
51     std::cout << "Child1: process was terminated." << std::endl;
52     os::TerminateProcess(child2);
53     std::cout << "Child2: process was terminated." << std::endl;
54 }
55
56 Parent::~~Parent() {
57     os::UnmapSharedMemory(shm_ptr1, shm_size);
58     os::CloseSharedMemory(shm_handle1);
59
60     os::UnmapSharedMemory(shm_ptr2, shm_size);
61     os::CloseSharedMemory(shm_handle2);
62 }
63 }

```

Листинг 8: parent.cpp

```

1  #include "parent.h"
2
3  int main() {
4      std::string filename1;
5      std::string filename2;
6      std::cout << "Parent[" << os::GetPid() << "]: Enter filename for child1" << std::endl
7          ;
8      std::getline(std::cin, filename1);
9      std::cout << "Parent[" << os::GetPid() << "]: Enter filename for child2" << std::endl
10         ;
11     std::getline(std::cin, filename2);
12     parent::Parent p;
13     try {
14         p.CreateChildProcesses(filename1, filename2);
15         p.Work();
16         p.EndChildren();
17     } catch (const exceptions::ChildProcessEndException&) {
18         std::cerr << "Parent[" << os::GetPid() << "]: Child process is ended with error.
19             Ending programm." << std::endl;
20         return 1;
21     } catch (const std::runtime_error&) {
22         std::cerr << "Parent[" << os::GetPid() << "]: System calls error. Ending programm.
23             " << std::endl;
24         return 1;
25     }
26 }

```

```

22 |     return 0;
23 | }

```

Листинг 9: 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\0\3\0>\0\1\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, 0x9d000) = 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, 0x26b000) = 0x7e2ecd06c000
mmap(0x7e2ecd07a000, 12608, PROT_READ|PROT_WRITE,
    ↪ MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 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\0\3\0>\0\1\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, 0x4000) = 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, 0x2b000) = 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\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, 0x28000) = 0x7e2ecca28000

```

```

mmap(0x7e2eccbb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↳ 0x1b0000) = 0x7e2eccbb0000
mmap(0x7e2eccbfff000, 24576, PROT_READ|PROT_WRITE,
↳ MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7e2eccbfff000
mmap(0x7e2ecc05000, 52624, PROT_READ|PROT_WRITE,
↳ MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7e2ecc05000
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\0\0\3\0>\0\1\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, 0x10000) = 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, 0xe7000) = 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(0x7e2eccbfff000, 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
futexp(0x7e2ecd07a7bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
getrandom("\x63\x06\xff\x05\xf4\xa7\xe\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_tidptr=0x7e2ecd147a10) = 90672
write(1, "Child1[90672]: process was creat"... , 36) = 36
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
↳ child_tidptr=0x7e2ecd147a10) = 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, "ssssssssss\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=SIGTERM, si_utime=0, si_stime=0} ---
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=SIGTERM, si_utime=0, si_stime=0} ---
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 +++
```