МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

# Лабораторная работа №1
# 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:

- Управлении процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

**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 каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

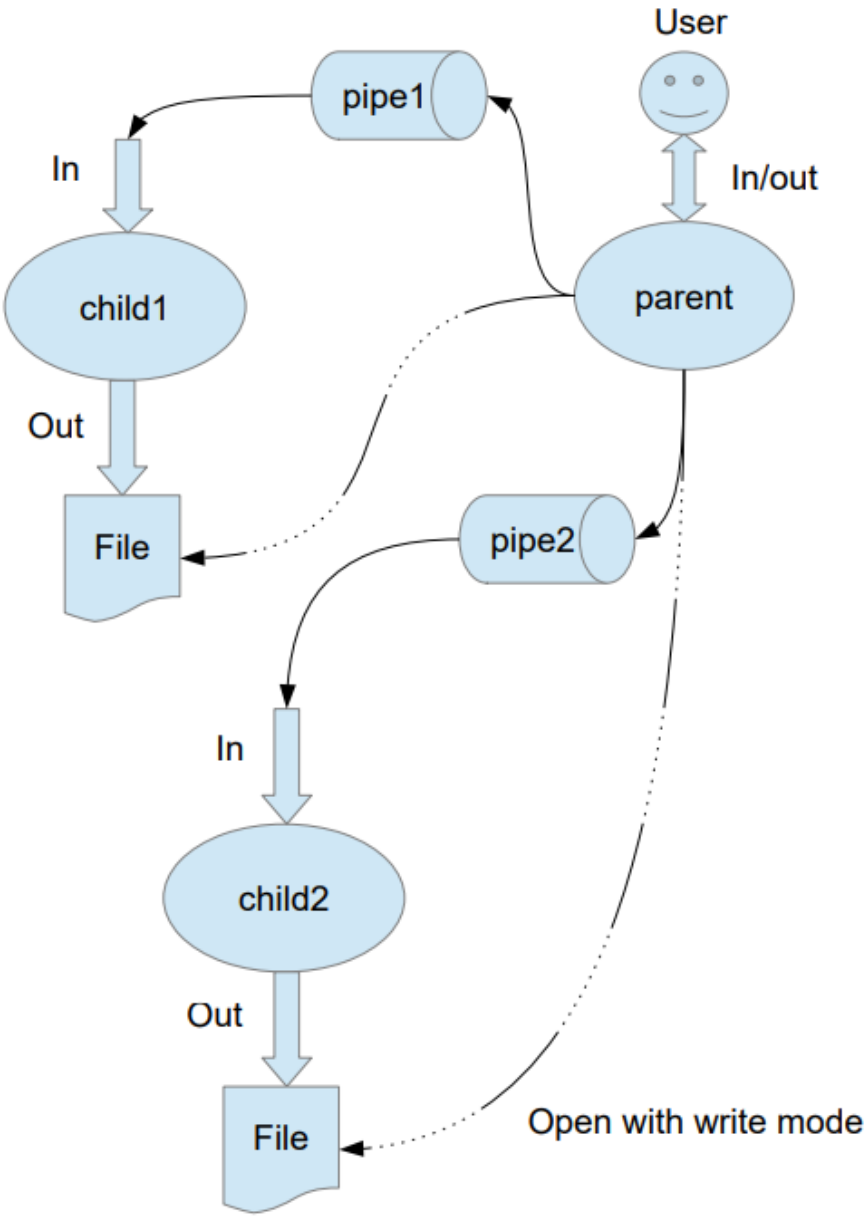


Рис. 1: Схема работы процессов.

**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 в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод. Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки.

## Метод решения
Алгоритм решения задачи:

1. Пользователь в консоль родительского процесса вводит имена файлов, которые будет использовано 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 канала (pipe) для передачи данных между родительским и дочерними процессами. Затем делает первый fork для создания дочернего процеса. Первый дочерний процесс закрывает соответсвующий канал для записи, “перенаправляет” второй “конец” канала для чтения на стандартный ввод, закрывает второй “конец” канала. Дочерний процесс пытается запустить бинарный файл для дочернего процесса, если не получилось — завершается. Аналогично для второго дочернего процесса.

4. Родительский процесс закрывает “концы” каналов для чтения и отправляет имена файлов для их открытия (создания) дочерними процессами.

5. Каждый из дочерних процессов пытается открыть (создать) указанный файл в спец. директории, если не получилось — процесс завершается.

6. Родительский процесс проверяет “жизнидеятельность” процессов, если любой из них уже завершен, то программа прекращает работу со статусом ошибки, затем считывает пользовательский ввод и отправляет дочернему процессу через pipe по правилу фильтрации.

7. Дочерний процесс инвертирует строку и записывает её в соответствующий файл.

8. 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.

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

`os.h` — объявление функций управления процессами ОС.
`src/os.cpp` — реализация.
Основные функции:

- `int CreatePipe(int pipefd[2]);` — создание канала. Используется системный вызов `pipe()`.
- `pid_t Fork();` — создание клона текущего процесса. Используется системный вызов `fork()`.
- `int Dup2(int fd1, int fd2);` — “перенаправление” файлового дескриптора. Используется системный вызов `dup2()`.
- `int CloseFd(int fd);` — закрытие файлового дескриптора. Используется системный вызов `close()`.
- `int Exec(const char* processPath, const char* processName);` — запуск исполняемого файла процессом. Используется системный вызов `execl()`.
- `int KillProcess(pid_t pid);` — завершение процесса по pid. Используется системный вызов `kill()`.
- `ssize_t WriteStr(int fd, const char* buf, size_t bytes);` — запись в канал. Используется системный вызов `write()`.

- `pid_t GetPid();` — получение pid текущего процесса. Используется системный вызов `getpid()`.
- `pid_t GetPPid();` — получение pid родительского процесса. Используется системный вызов `getppid()`.
- `bool IsAliveProcess(pid_t pid);` — проверка "жизнидеятельности" процесса. Используется системный вызов `waitpid()` с макросом `WNOHANG`.
- `void Exit(int status);` — завершение текущего процесса. Используется системный вызов: `_exit()`.
- `unsigned int Sleep(unsigned int);` — ожидание. Используется системный вызов: `sleep()`

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

- `std::string filename;` — название файла для открытия (создания).
- `std::ofstream file;` — поток для записи в файлы.
- `pid_t pid;` — pid процесса.

Основные функции (`методы`):

- `void Work();` — получает из канала строки, инвертирует их и записывает в файл.

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

- `std::random_device rd;` — генератор случайных чисел.
- `int pipe1[2];` — канал для 1 дочернего процесса.
- `int pipe2[2];` — канал для 2 дочернего процесса.
- `pid_t child1;` — pid 1 дочернего процесса.
- `pid_t child2;` — pid 2 дочернего процесса.

Основные функции (`методы`):

- `void CreateChildProcesses(std::string filename1, std::string filename2);` — Создает 2 дочерних процесса и запускает запускает бинарный файл для дочернего процесса, при ошибке дочерний процесс завершается.
- `void Work();` — Получает пользовательский ввод и записывает в один из каналов для дочерних процессов по правилу фильтрации.
- `void EndChildren();` — Завершает дочерние процессы.

## Результаты

Программа получает на вход названия двух файлов, создает два дочерних процесса, эти процессы в спец. указанной директории (testfiles) открывают (создают) эти файлы, далее все введеные пользователем строки, исключая "exit"и "quit инвертируются и записываются в один из файлов.
Результатом являются два файла в директории "testfiles". Если не удалось создать каналы для передачи данных между процессами или дочерние процессы завершились, то программа безопасно прекращает свою работу.

## Выводы

В ходе выполнения лабораторной работы были приобретены практические навыки в управлении процессами в ОС, обеспечение обмена данных между процессами посредством каналов.
Была составлена и отлажена программа на языке C++, осуществляющая работу с процессами и взаимодействие между ними в операционной системе с ядром Linux.
В результате работы программа (основной процесс) создает два дочерних процесса.
Взаимодействие между процессами осуществляется через системные сигналы и каналы.
Обрабатаны системные ошибки, которые могут возникнуть в результате работы.
Есть возможно поддержки кроссплатформенности за счет изменений системных вызовов в файле `inc/src/os.cpp`.

## Исходная программа

```cpp
#pragma once

#include <exception>
#include <string>

namespace exceptions {
    class CreatePipeException: std::exception {
    public:
        explicit CreatePipeException(const std::string& text): error_message_(text) {}
        const char* what() const noexcept override {
            return error_message_.data();
        }
    private:
        std::string error_message_;
    };

    class ChildProcessEndException: std::exception {
    public:
        explicit ChildProcessEndException(const std::string& text): error_message_(text)
            {}
        const char* what() const noexcept override {
            return error_message_.data();
        }
    private:
        std::string error_message_;
    };
}
```

Листинг 1: exceptions.h

```cpp
#pragma once

#include <cstddef>
#include <unistd.h>
#include <sys/wait.h>

namespace os {
    int CreatePipe(int pipefd[2]);
    pid_t Fork();
    int Dup2(int fd1, int fd2);
    int CloseFd(int fd);
    int Exec(const char* processPath, const char* processName);
    int KillProcess(pid_t pid);
    ssize_t WriteStr(int fd, const char* buf, size_t bytes);
    pid_t GetPid();
    pid_t GetPPid();
    bool IsAliveProcess(pid_t pid);
    void Exit(int status);
    unsigned int Sleep(unsigned int);
}
```

Листинг 2: os.h

```cpp
#include "os.h"
#include <unistd.h>

```

```cpp
namespace os {
    int CreatePipe(int pipefd[2]) {return pipe(pipefd);}

    pid_t Fork() {return fork();};

    int Dup2(int fd1, int fd2) {return dup2(fd1, fd2);}

    int CloseFd(int fd) {return close(fd);}

    int Exec(const char* processPath, const char* processName) {
        execl(processPath, processName, NULL);
        return -1;
    }
    int KillProcess(pid_t pid) {
        return kill(pid, SIGTERM);
    }
    ssize_t WriteStr(int fd, const char* buf, size_t bytes) {
        return write(fd, buf, bytes);
    }
    pid_t GetPid() {
        return getpid();
    }

    pid_t GetPPid() {
        return getppid();
    }

    bool IsAliveProcess(pid_t pid) {
        return waitpid(pid, nullptr, WNOHANG) == 0;
    }

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

    unsigned int Sleep(unsigned int t) {
        return sleep(t);
    }
}
```

Листинг 3: os.cpp

```cpp
#pragma once

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

#include "os.h"

namespace child {
    inline std::string BASEDIRECTORYFORFILES = "../testfiles/";

    class Child {
      private:
        std::string filename;
        std::ofstream file;
```

```
18          pid_t pid;
19        public:
20          Child();
21          Child(const std::string& filename);
22          void Work();
23          ~Child();
24      };
25  }
```

Листинг 4: child.h

```
1   #include "child.h"
2   #include "os.h"
3
4   namespace child {
5       Child::Child(): Child::Child("") {
6       }
7
8       Child::Child(const std::string& filename): pid(os::GetPid()), filename(
            BASEDIRECTORYFORFILES + filename) {
9           file.open(Child::filename, std::ios::app);
10          if (!file.is_open()) {
11              std::cout << "Child[" << pid << "]: failed to open " << Child::filename << ".
                    Process is over.\n";
12              os::Exit(1);
13          }
14          std::cout << "Child[" << pid << "]: file " << Child::filename << " opened." << std
                ::endl;
15      }
16
17      void Child::Work() {
18          if (!file.is_open()) {
19              std::cout << "Child[" << pid << "]: the file is not open! Work is impossible."
                     << std::endl;
20              os::Exit(1);
21          }
22          std::string line;
23          std::string inv;
24          while (std::getline(std::cin, line)) {
25              if (line.empty()) {continue;}
26              std::cout << "Child[" << pid << "]: the string \"" << line << "\" was got." <<
                     std::endl;
27              inv = line;
28              std::reverse(inv.begin(), inv.end());
29              std::cout << "Child[" << pid << "]: the string \"" << line << "\" was reversed
                    ." << std::endl;
30              file << inv << std::endl;
31              std::cout << "Child[" << pid << "]: the string \"" << inv << "\" was written
                    in file \"" << filename << "\"." << std::endl;
32          }
33      }
34
35      Child::~Child() {
36          if (file.is_open()) {
37              file.close();
38          }
39      }
```

```
}
```

Листинг 5: child.cpp

```
#include "child.h"

int main() {
    std::string filename;
    std::getline(std::cin, filename);
    child::Child child(filename);
    child.Work();
    return 0;
}
```

Листинг 6: bin/child.cpp

```
#pragma once

#include <iostream>
#include <random>

#include "exceptions.h"
#include "os.h"

namespace parent {
    class Parent {
      private:
        std::random_device rd;
        int pipe1[2];
        int pipe2[2];
        pid_t child1;
        pid_t child2;
      public:
        Parent() = default;
        void CreateChildProcesses(std::string filename1, std::string filename2);
        void Work();
        void EndChildren();
        ~Parent();
    };
}
```

Листинг 7: parent.h

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

namespace parent {
    void Parent::CreateChildProcesses(std::string filename1, std::string filename2) {
        if (os::CreatePipe(pipe1) == -1 || os::CreatePipe(pipe2) == -1) {
            throw exceptions::CreatePipeException("Failed to create pipe!");
        }

        child1 = os::Fork();
        if (child1 == 0) {
            os::CloseFd(pipe1[1]);
            os::Dup2(pipe1[0], STDIN_FILENO);
```

```
14            os::CloseFd(pipe1[0]);
15            os::Exec("./child", "child1");
16            os::Exit(1);
17        }
18        std::cout << "Child1[" << child1 << "]: process was created." << std::endl;
19
20        child2 = os::Fork();
21        if (child2 == 0) {
22            os::CloseFd(pipe2[1]);
23            os::Dup2(pipe2[0], STDIN_FILENO);
24            os::CloseFd(pipe2[0]);
25            os::Exec("./child", "child2");
26            os::Exit(1);
27        }
28        std::cout << "Child2[" << child2 << "]: process was created." << std::endl;
29
30        os::Sleep(1);
31
32        os::CloseFd(pipe1[0]);
33        os::CloseFd(pipe2[0]);
34
35        filename1 += "\n";
36        filename2 += "\n";
37
38        os::WriteStr(pipe1[1], filename1.c_str(), filename1.length());
39        os::WriteStr(pipe2[1], filename2.c_str(), filename2.length());
40
41        if (!os::IsAliveProcess(child1) || !os::IsAliveProcess(child2)) {
42            Parent::~Parent();
43            throw exceptions::ChildProcessEndException("Child process already terminated!"
                  );
44        }
45    }
46
47    void Parent::Work() {
48        std::string input;
49        while (std::getline(std::cin, input)) {
50            if (input == "exit" || input == "quit") {
51                std::cout << "Parent[" << os::GetPid() << "]: termination command was
                      received." << std::endl;
52                break;
53            }
54            if (!os::IsAliveProcess(child1) || !os::IsAliveProcess(child2)) {
55                Parent::~Parent();
56                throw exceptions::ChildProcessEndException("Child process already
                      terminated!");
57            }
58            if (input.empty()) {continue;}
59            input += "\n";
60            if (rd() % 100 < 80) {
61                os::WriteStr(pipe1[1], input.c_str(), input.length());
62            } else {
63                os::WriteStr(pipe2[1], input.c_str(), input.length());
64            }
65        }
66    }
67
68    void Parent::EndChildren() {
69        os::KillProcess(child1);
70        std::cout << "Child[" << child1 << "]: process was terminated." << std::endl;
```

```
71          os::KillProcess(child2);
72          std::cout << "Child[" << child2 << "]: process was terminated." << std::endl;
73      }
74
75      Parent::~Parent() {
76          os::CloseFd(pipe1[1]);
77          os::CloseFd(pipe2[1]);
78      }
79  }
```

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       std::getline(std::cin, filename1);
8       std::cout << "Parent[" << os::GetPid() << "]: Enter filename for child2" << std::endl
            ;
9       std::getline(std::cin, filename2);
10      parent::Parent p;
11      try {
12          p.CreateChildProcesses(filename1, filename2);
13          p.Work();
14          p.EndChildren();
15      } catch (const exceptions::CreatePipeException&) {
16          std::cout << "Parent[" << os::GetPid() << "]: Failed to create pipe. Ending
                programm." << std::endl;
17          return 1;
18      } catch (const exceptions::ChildProcessEndException&) {
19          std::cout << "Parent[" << os::GetPid() << "]: Child process is ended with error.
                Ending programm." << std::endl;
20          return 1;
21      }
22      return 0;
23  }
```

Листинг 9: main.cpp

## Strace

```
execve("./main", ["./main"], 0x7ffd66416120 /* 77 vars */) = 0
brk(NULL)                               = 0x6114081ef000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↪  0x77177838e000
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=74431, ...}) = 0
mmap(NULL, 74431, PROT_READ, MAP_PRIVATE, 3, 0) = 0x77177837b000
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\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) = 0x771778000000
mmap(0x77177809d000, 1343488, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x9d000) = 0x77177809d000
mmap(0x7717781e5000, 552960, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x1e5000) = 0x7717781e5000
mmap(0x77177826c000, 57344, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26b000) = 0x77177826c000
mmap(0x77177827a000, 12608, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x77177827a000
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) = 0x77177834d000
mmap(0x771778351000, 147456, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x4000) = 0x771778351000
mmap(0x771778375000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x28000) = 0x771778375000
mmap(0x771778379000, 8192, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2b000) = 0x771778379000
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) = 0x771777c00000
mmap(0x771777c28000, 1605632, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x771777c28000
mmap(0x771777db0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x1b0000) = 0x771777db0000
mmap(0x771777dff000, 24576, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x771777dff000
mmap(0x771777e05000, 52624, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x771777e05000
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) = 0x771777f17000
mmap(0x771777f27000, 520192, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x10000) = 0x771777f27000
mmap(0x771777fa6000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x8f000) = 0x771777fa6000
mmap(0x771777ffe000, 8192, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe7000) = 0x771777ffe000
close(3)                                = 0
```

```
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↪  0x77177834b000
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↪  0x771778348000
arch_prctl(ARCH_SET_FS, 0x771778348740) = 0
set_tid_address(0x771778348a10)         = 14797
set_robust_list(0x771778348a20, 24)     = 0
rseq(0x771778349060, 0x20, 0, 0x53053053) = 0
mprotect(0x771777dff000, 16384, PROT_READ) = 0
mprotect(0x771777ffe000, 4096, PROT_READ) = 0
mprotect(0x771778379000, 4096, PROT_READ) = 0
mprotect(0x77177826c000, 45056, PROT_READ) = 0
mprotect(0x611401a7e000, 4096, PROT_READ) = 0
mprotect(0x7717783cc000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) =
↪  0
munmap(0x77177837b000, 74431)           = 0
futex(0x77177827a7bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
getrandom("\x04\x1a\xbd\xc6\x6e\x41\xe3\xa5", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x6114081ef000
brk(0x611408210000)                     = 0x611408210000
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
getpid()                                = 14797
write(1, "Parent[14797]: 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()                                = 14797
write(1, "Parent[14797]: Enter filename fo"..., 41) = 41
read(0, "2.txt\n", 1024)                = 6
pipe2([3, 4], 0)                        = 0
pipe2([5, 6], 0)                        = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
↪  child_tidptr=0x771778348a10) = 14827
write(1, "Child1[14827]: process was creat"..., 36) = 36
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
↪  child_tidptr=0x771778348a10) = 14828
write(1, "Child2[14828]: process was creat"..., 36) = 36
clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=1, tv_nsec=0}, 0x7ffda76da7a0) = 0
close(3)                                = 0
close(5)                                = 0
write(4, "1.txt\n", 6)                  = 6
write(6, "2.txt\n", 6)                  = 6
wait4(14827, NULL, WNOHANG, NULL)       = 0
wait4(14828, NULL, WNOHANG, NULL)       = 0
read(0, "fdsfsfsfs\n", 1024)            = 10
wait4(14827, NULL, WNOHANG, NULL)       = 0
wait4(14828, NULL, WNOHANG, NULL)       = 0
write(4, "fdsfsfsfs\n", 10)             = 10
read(0, "okmgdkfgmdkmgkdgd\n", 1024)    = 18
wait4(14827, NULL, WNOHANG, NULL)       = 0
wait4(14828, NULL, WNOHANG, NULL)       = 0
write(4, "okmgdkfgmdkmgkdgd\n", 18)     = 18
read(0, "gmkkdmgmdklgmdlkgmd\n", 1024)  = 20
```

```
wait4(14827, NULL, WNOHANG, NULL)       = 0
wait4(14828, NULL, WNOHANG, NULL)       = 0
write(4, "gmkkdmgmdklgmdlkgmd\n", 20)   = 20
read(0, "exit\n", 1024)                 = 5
getpid()                                = 14797
write(1, "Parent[14797]: termination comma"..., 49) = 49
kill(14827, SIGTERM)                    = 0
write(1, "Child[14827]: process was termin"..., 38) = 38
kill(14828, SIGTERM)                    = 0
write(1, "Child[14828]: process was termin"..., 38) = 38
close(4)                                = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_KILLED, si_pid=14827, si_uid=1000,
↪  si_status=SIGTERM, si_utime=0, si_stime=0} ---
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_KILLED, si_pid=14828, si_uid=1000,
↪  si_status=SIGTERM, si_utime=0, si_stime=0} ---
close(6)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```