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. А. Кузнецов
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.

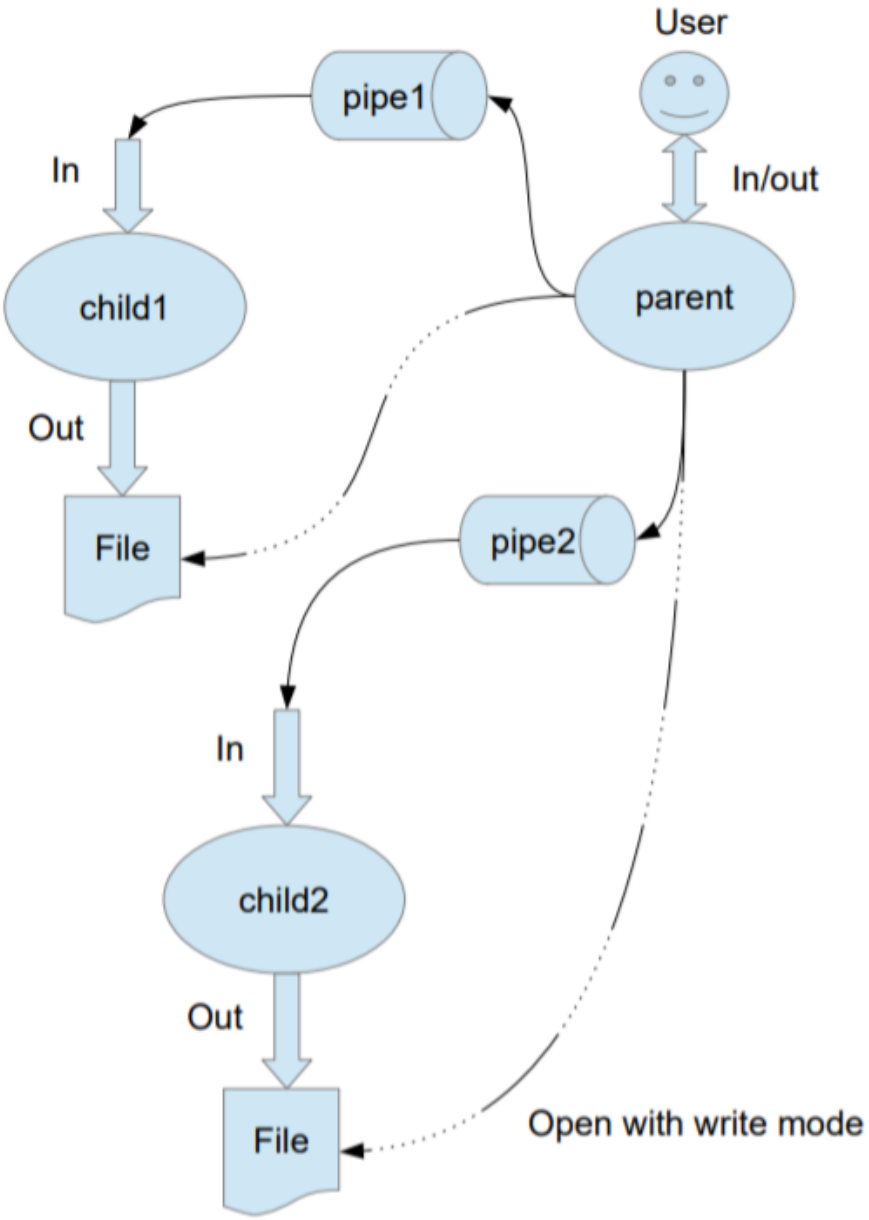

Рисунок 1: Схема работы процессов.

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 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: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы удаляют все гласные из строк.

Метод решения

Алгоритм решения задачи:

1. Пользователь вводит имена файлов для дочерних процессов child1 и child2 через консоль родительского процесса.
2. Создается объект класса Parent, который управляет работой родительского процесса.
3. Родительский процесс создает два канала (pipe1 и pipe2) для межпроцессного взаимодействия, затем выполняет два последовательных вызова fork() для создания дочерних процессов.
4. Каждый дочерний процесс перенаправляет стандартный ввод на соответствующий канал чтения, закрывает неиспользуемые дескрипторы каналов и запускает обработку данных.
5. Родительский процесс закрывает концы каналов для чтения и оставляет открытыми концы для записи.
6. Пользователь вводит произвольные строки, которые родительский процесс распределяет по каналам согласно правилу фильтрации: нечётные строки отправляются в pipe1, чётные — в pipe2.
7. Каждый дочерний процесс читает строки из своего канала, удаляет все гласные буквы и записывает результат как в соответствующий файл, так и в стандартный вывод.
8. При вводе пустой строки родительский процесс завершает работу, закрывает каналы, ожидает завершения дочерних процессов и выводит сообщение о завершении работы.
9. В случае ошибок создания каналов, создания процессов или системных сбоев программа безопасно завершает работу с соответствующими сообщениями об ошибках.

Архитектура программы:

```
lab1/
  bin/
  child.cpp
  build/
  include/
    child.h
    stringprocessor.h
    os.h
    parent.h
  src/
    child.cpp
    os.cpp
    stringprocessor.cpp
    parent.cpp
  CMakeLists.txt
  main.cpp
```

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/waitpid.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/dup2.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/pipe.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/fork.html
- https://pubs.opengroup.org/onlinepubs/009696799/functions/close.html

Описание программы

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

`stringprocessor.h` — объявление необходимых функций дочерних процессов.
`src/stringprocessor.cpp` — реализация.

- `std::string RemoveVowels(const std::string str);` — убирает гласные из строки.
- `bool IsVowel(char c);` — определяет, является ли буква гласной.

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

- `int Pipe(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 Close(int fd);` — закрытие файлового дескриптора. Используется системный вызов `close()`.
- `int Execl(const char* processPath, const char* processName);` — запуск исполняемого файла процессом. Используется системный вызов `execl()`.
- `ssize_t Write(int fd, const char* buf, size_t bytes);` — запись в канал. Используется системный вызов `write()`.
- `void Exit(int status);` — завершение текущего процесса. Используется системный вызов: `_exit()`.
- `void Perror(const char* s);` — обёртка вокруг стандартной функции perror(), которая выводит системные ошибки.
- `pid_t Wait(pid_t pid);` – обёртка вокруг `waitpid()` для ожидания завершения дочернего процесса

`child.h` — объявление класса Child.
`src/child.cpp` — реализация.
Основные функции (`методы`):

- `void Run(int childId, const std::string outputFilename);` — запускает обработку данных.
- `void ProcessInput(const std::string outputFilename);` — получает из канала строки, убирает гласные их и записывает в файл.

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

- `int lineCount;` — счетчик строк, для определения в какой канал передавать данные.

- `int pipe1[2];` — канал для 1 дочернего процесса.
- `int pipe2[2];` — канал для 2 дочернего процесса.
- `pid_t pid1;` — pid 1 дочернего процесса.
- `pid_t pid2;` — pid 2 дочернего процесса.

Основные функции (`методы`):

- `void Run();` — главная функция, координирующая весь workflow родительского процесса.
- `void Clean();` — закрывает каналы, ожидает завершения дочерних процессов и выводит финальное сообщение.
- `bool CreateChildProcess(const std::string filename1, const std::string filename2);` — порождает два дочерних процесса и настраивает перенаправление потоков.
- `bool CreatePipes();` — создаёт два канала pipe1 и pipe2 для межпроцессного взаимодействия.
- `void ProcessUserInput();` — читает строки от пользователя и распределяет их по каналам согласно правилу чётности..
- `void SendToPipe(const std::string line, int pipeNum);` — отправляет конкретную строку в указанный канал pipe1 или pipe2.

Результаты

Программа получает на вход названия двух файлов, создаёт два дочерних процесса, которые открывают (создают) файлы с указанными именами для записи результатов. Все введённые пользователем строки, исключая команды завершения, обрабатываются по правилу чётности: нечётные строки направляются первому процессу, чётные — второму. Каждый дочерний процесс удаляет гласные буквы из полученных строк и записывает результат в соответствующий файл.

Результатом работы программы являются два файла с обработанными данными. В случае ошибок создания каналов передачи данных или аварийного завершения дочерних процессов, программа безопасно прекращает работу с соответствующими сообщениями об ошибках.

Выводы

В ходе выполнения лабораторной работы были успешно приобретены практические навыки в области управления процессами в операционной системе и организации межпроцессного взаимодействия с использованием каналов (pipe).

Была разработана и отлажена программа на языке C++, реализующая механизм управления процессами и их взаимодействия в среде операционной системы Linux.

Основной (родительский) процесс создаёт два дочерних процесса, которые выполняют параллельную обработку данных. Связь и синхронизация между процессами организованы посредством системных сигналов и каналов.

В программе предусмотрена обработка системных ошибок, которые могут возникать на различных этапах работы, что обеспечивает повышенную надежность и отказоустойчивость приложения.

Архитектура программы предусматривает возможность кроссплатформенной поддержки за счет вынесения системно-зависимых вызовов в отдельный модуль `src/os.cpp`, что позволяет адаптировать приложение для других операционных систем с минимальными изменениями кодовой базы.

Исходная программа

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

int main() {
    std::string filename;
    std::getline(std::cin, filename);

    child::Child::Run(1, filename);

    return 0;
}
```

Листинг 1: bin/child.cpp

```cpp
#pragma once

#include <string>
#include <iostream>

namespace child {
    class Child {
        public:
            static void Run(int childId, const std::string& outputFilename);

        private:
            static void ProcessInput(const std::string& outputFilename);

            static constexpr int BUFFER_SIZE = 1024;
    };
}
```

Листинг 2: child.h

```cpp
#pragma once

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

namespace os {
    int Pipe(int pipefd[2]);

    pid_t Fork();

    int Dup2(int fd1, int fd2);

    int Close(int fd);

    int Execl(const char* processPath, const char* processName);

    ssize_t Write(int fd, const char* buf, size_t bytes);

    void Exit(int status);
```

```
22
23      void Perror(const char* s);
24
25      pid_t Wait(pid_t pid);
26  }
```

Листинг 3: os.h

```
1   #pragma once
2
3   #include <cstddef>
4   #include <iostream>
5   #include <unistd.h>
6   #include <sys/wait.h>
7   #include <string>
8   #include <child.h>
9
10  namespace parent {
11      class Parent {
12          private:
13              static constexpr int BUFFER_SIZE = 1024;
14              static constexpr int PIPE_READ = 0;
15              static constexpr int PIPE_WRITE = 1;
16
17              int pipe1[2];
18              int pipe2[2];
19              pid_t pid1;
20              pid_t pid2;
21              int lineCount;
22
23          public:
24              Parent() = default;
25              void Run();
26              void Clean();
27              bool CreateChildProcess(const std::string& filename1, const std::string&
                    filename2);
28              ~Parent() = default;
29
30          private:
31              bool CreatePipes();
32              void ProcessUserInput();
33              void SendToPipe(const std::string& line, int pipeNum);
34      };
35  }
```

Листинг 4: parent.h

```
1   #include "child.h"
2   #include "stringprocessor.h"
3   #include <iostream>
4   #include <fstream>
5   #include <unistd.h>
6
7   namespace child {
8
9       void Child::Run(int childId, const std::string& outputFilename) {
```

```
        std::cout << "Child" << childId << " started, output file: " << outputFilename
            << std::endl;

        ProcessInput(outputFilename);

        std::cout << "Child" << childId << " finished" << std::endl;
    }

    void Child::ProcessInput(const std::string& outputFilename) {
        std::ofstream outfile(outputFilename);
        if (!outfile.is_open()) {
            std::cerr << "Error opening file: " << outputFilename << std::endl;
            return;
        }

        char buffer[BUFFER_SIZE];
        ssize_t bytesRead;

        while ((bytesRead = read(STDIN_FILENO, buffer, sizeof(buffer) - 1)) > 0) {
            buffer[bytesRead] = '\0';
            std::string input_str(buffer);
            std::string result = utils::StringProcessor::RemoveVowels(input_str);

            outfile << result << std::endl;
            std::cout << "Child processed: \"" << input_str << "\" -> \"" << result <<
                "\"" << std::endl;
            std::cout.flush();
        }

        outfile.close();
    }
}
```

Листинг 5: child.cpp

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

namespace os {
    int Pipe(int pipefd[2]) {
        return pipe(pipefd);
    }

    pid_t Fork() {
        return fork();
    }

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

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

    int Execl(const char* processPath, const char* processName) {
        execl(processPath, processName, NULL);
        return -1;
```

```
    }

    ssize_t Write(int fd, const char* buf, size_t bytes) {
        return write(fd, buf, bytes);
    }

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

    void Perror(const char* s) {
        perror(s);
    }

    pid_t Wait(pid_t pid) {
        return waitpid(pid, nullptr, 0);
    }
}
```

Листинг 6: os.cpp

```
#include <parent.h>
#include <os.h>

namespace parent {
    void Parent::Run() {
        std::string filename1;
        std::string filename2;

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

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

        if (!CreatePipes()) {
            return;
        }

        if (!CreateChildProcess(filename1, filename2)) {
            return;
        }

        ProcessUserInput();
        Clean();
    }

    bool Parent::CreatePipes() {
        if (os::Pipe(pipe1) == -1 || os::Pipe(pipe2) == -1) {
            os::Perror("pipe");
            return false;
        }
        return true;
    }

    bool Parent::CreateChildProcess(const std::string& filename1, const std::string&
        filename2) {
        pid1 = os::Fork();
```

```
37          if (pid1 == -1) {
38              os::Perror("fork pid1 failed");
39              return false;
40          }
41
42          if (pid1 == 0) {
43              os::Close(pipe1[PIPE_WRITE]);
44              os::Close(pipe2[PIPE_READ]);
45              os::Close(pipe2[PIPE_WRITE]);
46
47              os::Dup2(pipe1[PIPE_READ], STDIN_FILENO);
48              os::Close(pipe1[PIPE_READ]);
49
50              child::Child::Run(1, filename1);
51              os::Exit(1);
52          }
53
54          pid2 = os::Fork();
55          if (pid2 == -1) {
56              os::Perror("fork pid2 failed");
57              return false;
58          }
59
60          if (pid2 == 0) {
61              os::Close(pipe2[PIPE_WRITE]);
62              os::Close(pipe1[PIPE_READ]);
63              os::Close(pipe1[PIPE_WRITE]);
64
65              os::Dup2(pipe2[PIPE_READ], STDIN_FILENO);
66              os::Close(pipe2[PIPE_READ]);
67
68              child::Child::Run(2, filename2);
69              os::Exit(1);
70          }
71
72          os::Close(pipe1[PIPE_READ]);
73          os::Close(pipe2[PIPE_READ]);
74
75          return true;
76      }
77
78      void Parent::ProcessUserInput() {
79          std::cout << "Enter strings (empty line to stop): " << std::endl;
80
81          std::string line;
82          lineCount = 0;
83
84          while(std::getline(std::cin, line)) {
85              if (line.empty()) {
86                  break;
87              }
88
89              lineCount++;
90
91              if (lineCount % 2 == 1) {
92                  SendToPipe(line, 1);
93                  std::cout << "Sent to pipe1: " << line << std::endl;
94              } else {
```

```
                SendToPipe(line, 2);
                std::cout << "Sent to pipe2: " << line << std::endl;
            }
        }
    }

    void Parent::SendToPipe(const std::string& line, int pipeNum) {
        if (pipeNum == 1) {
            os::Write(pipe1[PIPE_WRITE], line.c_str(), line.size());
        } else {
            os::Write(pipe2[PIPE_WRITE], line.c_str(), line.size());
        }
    }

    void Parent::Clean() {
        os::Close(pipe1[PIPE_WRITE]);
        os::Close(pipe2[PIPE_WRITE]);

        os::Wait(pid1);
        os::Wait(pid2);

        std::cout << "Finished" << std::endl;
    }
}
```

Листинг 7: parent.cpp

```
#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 8: stringprocessor.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)) {
```

```
15              res += c;
16          }
17      }
18
19      return res;
20  }
21 }
```

Листинг 9: stringprocessor.cpp

```
1 #include "parent.h"
2
3 int main() {
4     parent::Parent p;
5     p.Run();
6     return 0;
7 }
```

Листинг 10: main.cpp

Strace

```
execve("./parent",["./parent"],0x7fff1d500d00 /* 28 vars */) = 0
brk(NULL)                                       = 0x6477c59eb000
arch_prctl(0x3001 /* ARCH_??? */,0x7ffc51d28230) = -1 EINVAL (Invalid argument)
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7c197f0fc000
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=21304,...},AT_EMPTY_PATH) = 0
mmap(NULL,21304,PROT_READ,MAP_PRIVATE,3,0) = 0x7c197f0f6000
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) = 0x7c197ee00000
mprotect(0x7c197ee9a000,1576960,PROT_NONE) = 0
mmap(0x7c197ee9a000,1118208,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3
= 0x7c197ee9a000
mmap(0x7c197efab000,454656,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x1ab000)
= 0x7c197efab000
mmap(0x7c197f01b000,57344,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x7c197f01b000
mmap(0x7c197f029000,10432,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1
= 0x7c197f029000
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) = 0x7c197f0d6000
mmap(0x7c197f0d9000,94208,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x7c197f0d9000
mmap(0x7c197f0f0000,16384,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x1a000)
= 0x7c197f0f0000
mmap(0x7c197f0f4000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x7c197f0f4000
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\0\325\31p\226\367\t\200\30)\261\30\257\33|\36
= 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) = 0x7c197ea00000
mprotect(0x7c197ea28000,2023424,PROT_NONE) = 0
mmap(0x7c197ea28000,1658880,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3
= 0x7c197ea28000
mmap(0x7c197ebbd000,360448,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x1bd000)
= 0x7c197ebbd000
mmap(0x7c197ec16000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x7c197ec16000
mmap(0x7c197ec1c000,52816,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1
= 0x7c197ec1c000
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) = 0x7c197ed19000
mmap(0x7c197ed27000,507904,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x7c197ed27000
mmap(0x7c197eda3000,372736,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x8a000)
= 0x7c197eda3000
mmap(0x7c197edfe000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x7c197edfe000
```

```
close(3)                                = 0
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7c197f0d4000
arch_prctl(ARCH_SET_FS,0x7c197f0d53c0) = 0
set_tid_address(0x7c197f0d5690)         = 360086
set_robust_list(0x7c197f0d56a0,24)     = 0
rseq(0x7c197f0d5d60,0x20,0,0x53053053) = 0
mprotect(0x7c197ec16000,16384,PROT_READ) = 0
mprotect(0x7c197edfe000,4096,PROT_READ) = 0
mprotect(0x7c197f0f4000,4096,PROT_READ) = 0
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7c197f0d2000
mprotect(0x7c197f01b000,45056,PROT_READ) = 0
mprotect(0x6477b9eac000,4096,PROT_READ) = 0
mprotect(0x7c197f136000,8192,PROT_READ) = 0
prlimit64(0,RLIMIT_STACK,NULL,{rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY})
= 0
munmap(0x7c197f0f6000,21304)           = 0
getrandom("\x85\x22\xfb\x73\x5e\xa8\x82\x6b",8,GRND_NONBLOCK) = 8
brk(NULL)                               = 0x6477c59eb000
brk(0x6477c5a0c000)                     = 0x6477c5a0c000
futex(0x7c197f02977c,FUTEX_WAKE_PRIVATE,2147483647) = 0
newfstatat(1,"",{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0x5),...},AT_EMPTY_PATH)
= 0
write(1,"Enter the name for child1: ",27Enter the name for child1: ) = 27
newfstatat(0,"",{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0x5),...},AT_EMPTY_PATH)
= 0
read(0,
"\n",1024)                     = 1
write(1,"Enter the name for child2: ",27Enter the name for child2: ) = 27
read(0,
"\n",1024)                     = 1
pipe2([3,4],0)                        = 0
pipe2([5,6],0)                        = 0
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_ti
= 360135
Child1 started,output file:
Error opening file:
Child1 finished
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_ti
= 360136
Child2 started,output file:
Error opening file:
Child2 finished
---SIGCHLD {si_signo=SIGCHLD,si_code=CLD_EXITED,si_pid=360135,si_uid=1000,si_status=1
---
---SIGCHLD {si_signo=SIGCHLD,si_code=CLD_EXITED,si_pid=360136,si_uid=1000,si_status=1
---
close(3)                                = 0
close(5)                                = 0
```

```
write(1,"Enter strings (empty line to sto"...,37Enter strings (empty line to
stop):
) = 37
read(0,
"\n",1024)                      = 1
close(4)                                = 0
close(6)                                = 0
wait4(360135,NULL,0,NULL)              = 360135
wait4(360136,NULL,0,NULL)              = 360136
write(1,"Finished\n",9Finished
)               = 9
exit_group(0)                           = ?
+++ exited with 0 +++
```