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: М. А. Понизяйкин
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы:
Приобретение практических навыков в:

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

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

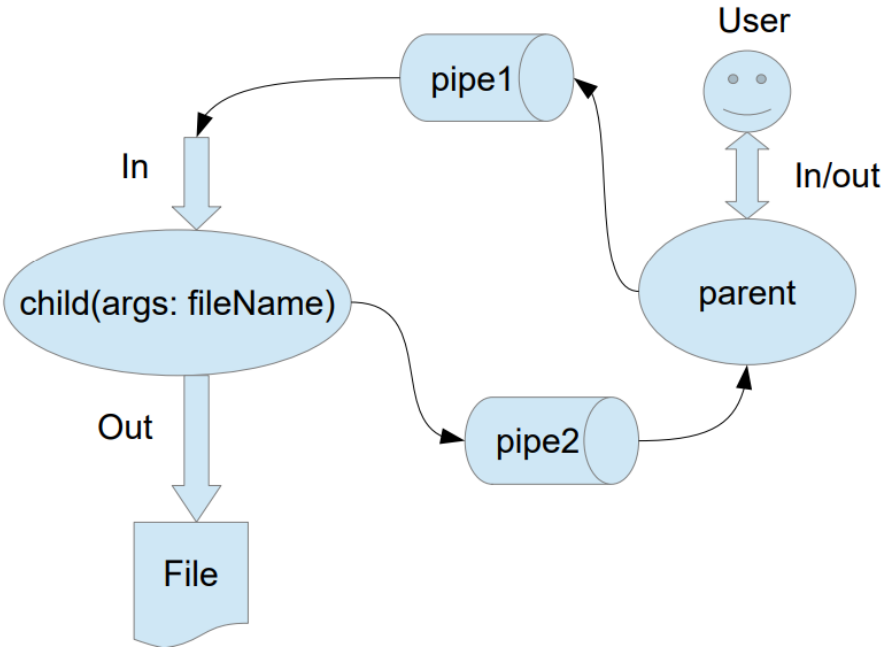

Рис. 1: Схема работы процессов.

Вариант: 4
Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода. Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна

осуществляться на стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

Метод решения

Алгоритм решения задачи:

1. Пользователь в консоль родительского процесса вводит имя файла, в который дочерний процесс запишет результат вычислений, а затем — последовательность чисел типа `float`, разделённых пробелами.
2. Создаётся объект логики родительского процесса (реализован в `parent.cpp`), инициализирующий работу основной программы.
3. Родительский процесс создаёт объект разделяемой памяти с именем `/shm` и устанавливает обработчики сигналов `SIGUSR1` (для получения уведомлений от дочернего процесса) и `SIGCHLD` (для отслеживания завершения дочернего процесса).
4. Вызывается `fork` для создания дочернего процесса.
5. В контексте дочернего процесса:
   - Производится подключение к разделяемой памяти по имени `/shm`.;
   - Устанавливается обработчик сигнала `SIGUSR2` для получения команды от родителя.;
   - Отправляется сигнал `SIGUSR1` родителю, чтобы сообщить, что дочерний процесс готов к работе.;
   - Вызывается pause() для ожидания сигнала `SIGUSR2`.
6. Родительский процесс ожидает получения сигнала `SIGUSR1` от дочернего процесса, чтобы убедиться, что обработчик установлен..
7. После получения сигнала `SIGUSR1`, родитель записывает в разделяемую память введённые пользователем числа.
8. Затем родитель отправляет сигнал `SIGUSR2` дочернему процессу, чтобы начать вычисления.
9. Дочерний процесс, получив сигнал `SIGUSR2`, читает числа из разделяемой памяти, выполняет последовательное деление первого числа на все последующие.
10. Перед каждым делением проверяется, не равен ли делитель нулю. В случае деления на ноль процесс немедленно завершается с выводом сообщения об ошибке в `stderr`.
11. Если деление выполнено успешно, дочерний процесс открывает (создаёт) указанный файл и записывает в него результат вычислений с точностью до шести знаков после запятой.
12. После завершения вычислений дочерний процесс отправляет сигнал `SIGUSR1` родителю, чтобы сообщить об успешном завершении, и завершает свою работу.

13. Родительский процесс, получив сигнал SIGUSR1, выводит сообщение о том, что результат записан в файл, и завершает работу.

Архитектура программы:

```
lab3-var4/
├── bin/
├── child.cpp
├── build/
├── include/
│   ├── child.h
│   ├── exceptions.h
│   ├── os.h
│   └── parent.h
├── src/
│   ├── child.cpp
│   ├── os.cpp
│   └── parent.cpp
└── 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/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

vavv

Описание программы

main.cpp — точка входа в родительский процесс. Вызывает функцию RunParentProcess(), реализующую основную логику взаимодействия с пользователем и дочерним процессом.

bin/child.cpp — Точка входа в дочерний процесс. Принимает имя выходного файла и имя разделяемой памяти как аргументы командной строки и вызывает функцию RunChildProcess() для выполнения вычислений.

exceptions.h — объявление пользовательских исключений.

os.h — объявление функций-обёрток над системными вызовами операционной системы (для поддержки кроссплатформенности).
src/os.cpp — реализация функций.

Основные функции:

- void* CreateSharedMemory(const char* name, size_t size); — создаёт разделяемую память. Использует shm_open и mmap.
- int CloseSharedMemory(void* addr, size_t size); — отключает разделяемую память. Использует munmap.
- int UnlinkSharedMemory(const char* name); — удаляет разделяемый объект. Использует shm_unlink.
- void SendSignal(pid_t pid, signal_t sig); — отправляет сигнал процессу. Использует kill.
- pid_t CloneProcess(); — создаёт копию текущего процесса. Использует fork.
- int Exec(const char* path); — заменяет текущий процесс на новый исполняемый файл. Используется системный вызов execl().
- void AddSignalHandler(signal_t sig, SignalHandler_t handler); — регистрирует обработчик сигнала. Используется системный вызов signal().
- void PrintLastError(); — выводит описание последней ошибки ОС. Используется функция perror().

child.h — объявление функции логики дочернего процесса.
src/child.cpp — реализация вычислительной логики.
Основные функции:

- void RunChildProcess(const char* output_file, const char* shared_mem_name); — подключается к разделяемой памяти, устанавливает обработчик сигнала, отправляет сигнал готовности родителю, ждёт команду, читает данные, производит вычисления, записывает результат в файл.
- static void OnParentSignal(int sig); — обрабатывает SIGUSR2, читает числа из разделяемой памяти, выполняет деление, записывает результат в файл и отправляет сигнал SIGUSR1 родителю.

parent.h — объявление функции логики родительского процесса.
src/parent.cpp — реализация управления процессами и взаимодействия с пользователем.
Основные функции:

- void RunParentProcess(); — запрашивает у пользователя имя выходного файла и последовательность чисел, создаёт разделяемую память, устанавливает обработчики сигналов, порождает дочерний процесс через fork(), запускает исполняемый файл child, записывает числа в разделяемую память, отправляет сигнал SIGUSR2, ожидает сигнал SIGUSR1 и обрабатывает возможные ошибки.
- void OnChildDone(signal_t signum); — обработчик сигнала SIGUSR1. Выводит сообщение о завершении работы дочернего процесса.
- void OnChildExit(signal_t signum); — обработчик сигнала SIGCHLD. Проверяет статус завершения дочернего процесса и при необходимости завершает родительский процесс.

Результаты

Программа получает на вход название файла, создает дочерний процесс, этот процесс в директории открывает (создает) этот файл. После этого дочерний процесс обрабатывает все введеные пользователем числа, последовательно делит первое число на остальные, при этом избегая деления на 0, и результат записывает в указанный файл.
Программа корректно обрабатывает ошибки, такие как деление на ноль, ошибки доступа к файлу, а также завершение дочернего процесса с ошибкой.
Результатом является файл в директории. Если не удалось создать канал 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 сигналов и разделяемой памяти.
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) создаёт один дочерний процесс. Взаимодействие между процессами осуществляется через сигналы и отображаемые файлы (mmap). Обработка ошибок в дочернем процессе происходит через сигналы и завершение с ненулевым кодом, что отслеживается родительским процессом с помощью waitpid.
Системные ошибки, возникающие в результате работы, обрабатываются с помощью perror и errno. Программа может быть адаптирована для поддержки кроссплатформенности за счёт изоляции системных вызовов в отдельном модуле os.

Исходная программа

```cpp
#include "child.h"
#include <cstdio>

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::fprintf(stderr, "Usage: %s <output_file>\n", argv[0]);
        return 1;
    }

    const char* shared_mem_name = "/shm";
    RunChildProcess(argv[1], shared_mem_name);
    return 0;
}
```

Листинг 1: bin/child.cpp

```cpp
#pragma once

void RunChildProcess(const char* output_file, const char* shared_mem_name);
```

Листинг 2: child.h

```cpp
#pragma once

#include <stdexcept>
#include <string>

class OsException : public std::runtime_error {
public:
    explicit OsException(const std::string& message) : std::runtime_error(message) {}
};
```

Листинг 3: exceptions.h

```cpp
#pragma once

#include <string>
#include <cstddef>
#include <signal.h>

using pipe_t = int;
using pid_t = int;
using signal_t = int;
using SignalHandler_t = void(*)(int);

extern const int ChildDeathSig;
extern const int BrokenPipeSig;
extern const int ChildDoneSig;

int CreatePipe(pipe_t fd[2]);
pid_t CloneProcess();
int Exec(const char* path);
void PrintLastError();
int WaitForChild();
int ClosePipe(pipe_t pipe);
```

```
int WritePipe(pipe_t pipe, void* buffer, std::size_t bytes);
int ReadPipe(pipe_t pipe, void* buffer, std::size_t bytes);
int LinkStdinWithPipe(pipe_t pipe);
int LinkStderrWithPipe(pipe_t pipe);
void AddSignalHandler(signal_t sig, SignalHandler_t handler);
int ReadFromStdin(char* buffer, std::size_t size);
void* CreateSharedMemory(const char* name, size_t size);
int CloseSharedMemory(void* addr, size_t size);
int UnlinkSharedMemory(const char* name);
void SendSignal(pid_t pid, signal_t sig);
```

Листинг 4: os.h

```
#pragma once
#include "os.h"

void RunParentProcess();
void OnChildKilled(signal_t signum);
void OnChildDone(signal_t signum);
```

Листинг 5: parent.h

```
#include "child.h"
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <signal.h>

static void* shared_mem = nullptr;
static pid_t parent_pid = -1;
const size_t SHARED_MEM_BUFFER = 1024;
static const char* g_output_file = nullptr;

static void OnParentSignal(int sig) {
    if (sig == SIGUSR2) {
        char* input = static_cast<char*>(shared_mem);
        float num = 0.0;
        int scanned = std::sscanf(input, "%f", &num);
        if (scanned != 1) {
            std::fprintf(stderr, "Error: failed to read first number\n");
            std::exit(1);
        }

        const char* p = input;
        while (*p && *p != ' ') p++;
        if (*p) p++;
        float divider = 0.0;
        while (std::sscanf(p, "%f", &divider) == 1) {
            if (divider == 0.0) {
                std::fprintf(stderr, "Error: division by zero occurred\n");
                std::exit(1);
            }
            num /= divider;
```

```
            while (*p && *p != ' ') p++;
            if (*p) p++;
        }

        FILE* out = std::fopen(g_output_file, "w");
        if (!out) {
            std::fprintf(stderr, "Error: failed to open output file '%s'\n",
                g_output_file);
            std::exit(1);
        }

        std::fprintf(out, "%.6f\n", num);
        std::fclose(out);

        kill(parent_pid, SIGUSR1);

        std::exit(0);
    }
}

void RunChildProcess(const char* output_file, const char* shared_mem_name) {
    g_output_file = output_file;
    parent_pid = getppid();

    int fd = shm_open(shared_mem_name, O_RDWR, 0666);
    if (fd == -1) {
        std::perror("child can't open shared memory");
        std::exit(-1);
    }

    shared_mem = mmap(nullptr, SHARED_MEM_BUFFER, PROT_READ | PROT_WRITE, MAP_SHARED,
        fd, 0);
    close(fd);

    if (shared_mem == MAP_FAILED) {
        std::perror("child: mmap");
        std::exit(1);
    }

    signal(SIGUSR2, OnParentSignal);

    kill(parent_pid, SIGUSR1);

    pause();
}
```

Листинг 6: child.cpp

```
#include "os.h"
#include "exceptions.h"
#include <unistd.h>
#include <sys/wait.h>
#include <cstring>
#include <csignal>
#include <cstdio>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/stat.h>
```

```cpp
#include <signal.h>

const int ChildDeathSig = SIGCHLD;
const int ChildDoneSig = SIGUSR1;
const int BrokenPipeSig = SIGPIPE;

int CreatePipe(pipe_t fd[2]) {
    return pipe(fd);
}

pid_t CloneProcess() {
    return static_cast<pid_t>(fork());
}

int Exec(const char* path) {
    return execl(path, path, nullptr);
}

void PrintLastError() {
    // std::fprintf(stderr, "DEBUG: errno = %d\n", errno);
    perror("OS Error");
}

int WaitForChild() {
    return wait(nullptr);
}

int ClosePipe(pipe_t pipe) {
    return close(pipe);
}

int WritePipe(pipe_t pipe, void* buffer, std::size_t bytes) {
    return static_cast<int>(write(pipe, buffer, bytes));
}

int ReadPipe(pipe_t pipe, void* buffer, std::size_t bytes) {
    return static_cast<int>(read(pipe, buffer, bytes));
}

int LinkStdinWithPipe(pipe_t pipe) {
    return dup2(pipe, STDIN_FILENO);
}

int LinkStderrWithPipe(pipe_t pipe) {
    return dup2(pipe, STDERR_FILENO);
}

void AddSignalHandler(signal_t sig, SignalHandler_t handler) {
    signal(sig, handler);
}

int ReadFromStdin(char* buffer, std::size_t size) {
    return static_cast<int>(read(STDIN_FILENO, buffer, size));
}

void* CreateSharedMemory(const char* name, size_t size) {
    int fd = shm_open(name, O_CREAT | O_RDWR, 0666);
    if (fd == -1) {
```

```
69          PrintLastError();
70          return nullptr;
71      }
72
73      if (ftruncate(fd, size) == -1) {
74          PrintLastError();
75          close(fd);
76          return nullptr;
77      }
78
79      void* addr = mmap(nullptr, size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
80      close(fd);
81
82      if (addr == MAP_FAILED) {
83          PrintLastError();
84          return nullptr;
85      }
86
87      return addr;
88  }
89
90  int CloseSharedMemory(void* addr, size_t size) {
91      return munmap(addr, size);
92  }
93
94  int UnlinkSharedMemory(const char* name) {
95      return shm_unlink(name);
96  }
97
98  void SendSignal(pid_t pid, signal_t sig) {
99      kill(pid, sig);
100 }
```

Листинг 7: os.cpp

```
1   #include "parent.h"
2   #include "os.h"
3   #include <cstdio>
4   #include <cstdlib>
5   #include <cstring>
6   #include <string>
7   #include <unistd.h>
8   #include <signal.h>
9   #include <sys/wait.h>
10
11  const std::size_t KBUFFER = 100;
12  const std::size_t KSHARED_MEM_SIZE = 1024;
13  static pid_t child_pid = -1;
14  static void* shared_mem = nullptr;
15  static const char* KSHARED_MEM_NAME = "/shm";
16  static int child_finished = 0;
17  static int child_exited_with_error = 0;
18  static int child_ready = 0;
19
20  void OnChildExit(signal_t signum) {
21      if (signum == SIGCHLD) {
22          int status;
23          pid_t pid = waitpid(child_pid, &status, WNOHANG);
```

```
24          if (pid == child_pid) {
25              if (WIFEXITED(status)) {
26                  int exit_code = WEXITSTATUS(status);
27                  if (exit_code != 0) {
28                      child_exited_with_error = 1;
29                      std::fprintf(stderr, "Child process exited with error code: %d\n",
                            exit_code);
30                      std::exit(-1);
31                  }
32              } else if (WIFSIGNALED(status)) {
33                  std::fprintf(stderr, "Child process was terminated by signal: %d\n",
                        WTERMSIG(status));
34                  std::exit(-1);
35              }
36          }
37      }
38  }
39
40  void OnChildKilled(signal_t signum) {
41      if (shared_mem) {
42          CloseSharedMemory(shared_mem, KSHARED_MEM_SIZE);
43      }
44      UnlinkSharedMemory(KSHARED_MEM_NAME);
45      std::fprintf(stderr, "Child process was killed by signal %d\n", signum);
46      std::exit(-1);
47  }
48
49  void OnChildDone(signal_t signum) {
50      if (signum == SIGUSR1 && !child_ready) {
51          child_ready = 1;
52      } else if (signum == SIGUSR1 && child_ready) {
53          child_finished = 1;
54          std::printf("Child process completed succesfully.\n");
55          std::printf("Result written to file.\n");
56      }
57  }
58
59  void RunParentProcess() {
60      AddSignalHandler(SIGCHLD, OnChildExit);
61      AddSignalHandler(SIGUSR1, OnChildDone);
62
63      std::printf("Enter result file name: ");
64      char output_file[KBUFFER];
65      if (!std::fgets(output_file, KBUFFER, stdin)) {
66          std::fprintf(stderr, "Error reading file name from stdin\n");
67          std::exit(1);
68      }
69      output_file[strcspn(output_file, "\n")] = '\0';
70
71      std::printf("Enter numbers (separator - space): ");
72      char numbers_line[KBUFFER];
73      if (!std::fgets(numbers_line, KBUFFER, stdin)) {
74          std::fprintf(stderr, "Error reading numbers from stdin\n");
75          std::exit(1);
76      }
77
78      shared_mem = CreateSharedMemory(KSHARED_MEM_NAME, KSHARED_MEM_SIZE);
79      if (!shared_mem) {
```

```
        std::fprintf(stderr, "Failed to create shared memory\n");
        std::exit(-1);
    }

    pid_t pid = CloneProcess();

    if (pid == 0) { // child
        execl("./child", "./child", output_file, nullptr);

        PrintLastError();
        std::exit(1);
    } else if (pid == -1) {
        PrintLastError();
        std::exit(1);
    } else { // parent
        child_pid = pid;

        while (!child_ready) {
            pause();
        }

        std::strncpy(static_cast<char*>(shared_mem), numbers_line, KSHARED_MEM_SIZE -
            1);
        static_cast<char*>(shared_mem)[KSHARED_MEM_SIZE - 1] = '\0';

        SendSignal(pid, SIGUSR2);

        while (!child_finished && !child_exited_with_error) {
            pause();
        }

        CloseSharedMemory(shared_mem, KSHARED_MEM_SIZE);
        UnlinkSharedMemory(KSHARED_MEM_NAME);
    }
}
```

Листинг 8: parent.cpp

```
#include "parent.h"

int main() {
    RunParentProcess();
    return 0;
}
```

Листинг 9: main.cpp

Strace

```
execve("./main", ["./main"], 0x7ffc5410c180 /* 27 vars */) = 0
brk(NULL)                               = 0x63b60b7ac000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7757d246
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
fstat(3, {st_mode=S_IFREG|0644, st_size=20919, ...}) = 0
```

```
mmap(NULL, 20919, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7757d245f000
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)
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, 6
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, 6
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7757d2200000
mmap(0x7757d2228000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRIT
mmap(0x7757d23b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0
mmap(0x7757d23ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7757d2405000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7757d24
arch_prctl(ARCH_SET_FS, 0x7757d245c740) = 0
set_tid_address(0x7757d245ca10)         = 44592
set_robust_list(0x7757d245ca20, 24)     = 0
rseq(0x7757d245d060, 0x20, 0, 0x53053053) = 0
mprotect(0x7757d23ff000, 16384, PROT_READ) = 0
mprotect(0x63b5d7a98000, 4096, PROT_READ) = 0
mprotect(0x7757d249d000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7757d245f000, 20919)           = 0
rt_sigaction(SIGCHLD, {sa_handler=0x63b5d7a9649d, sa_mask=[CHLD], sa_flags=SA_RESTORE
rt_sigaction(SIGUSR1, {sa_handler=0x63b5d7a96602, sa_mask=[USR1], sa_flags=SA_RESTORE
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
getrandom("\x4a\xd4\x7e\xa6\x4e\x7c\xe5\x59", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x63b60b7ac000
brk(0x63b60b7cd000)                     = 0x63b60b7cd000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "Enter result file name: ", 24) = 24
read(0, "out.txt\n", 1024)              = 8
write(1, "Enter numbers (separator - space"..., 35) = 35
read(0, "1 2 3 4 5\n", 1024)            = 10
openat(AT_FDCWD, "/dev/shm/shm", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 1024)                      = 0
mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7757d2464000
close(3)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_
pause()                                 = ? ERESTARTNOHAND (To be restarted if no han
--- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=44611, si_uid=1000} ---
rt_sigreturn({mask=[]})                 = -1 EINTR (Interrupted system call)
kill(44611, SIGUSR2)                    = 0
pause()                                 = ? ERESTARTNOHAND (To be restarted if no han
--- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=44611, si_uid=1000} ---
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=44611, si_uid=1000, si_stat
wait4(44611, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], WNOHANG, NULL) = 44611
rt_sigreturn({mask=[USR1]})             = 0
```

```
write(1, "Child process completed succesfu"..., 37) = 37
write(1, "Result written to file.\n", 24) = 24
rt_sigreturn({mask=[]})                 = -1 EINTR (Interrupted system call)
munmap(0x7757d2464000, 1024)            = 0
unlink("/dev/shm/shm")                  = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```