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

Москва, 2026

Условие

Цель работы

Приобретение практических навыков в:

1. Управление процессами в ОС
2. 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.

Вариант 4

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, 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

Общее описание алгоритма

Программа состоит из двух отдельных исполняемых файлов: `parent` (родительский процесс) и `child` (дочерний процесс).

Родительский процесс выполняет следующие действия:

1. Запрашивает у пользователя имя выходного файла.
2. Создаёт два канала (`pipe`) для двустороннего обмена данными:
   - `to_child`: родитель → дочерний (передача строк с числами);
   - `from_child`: дочерний → родитель (передача однобайтового статуса).
3. Создаёт дочерний процесс через `fork()`.
4. В дочернем процессе перенаправляет `stdin` и `stdout` на каналы с помощью `dup2()` и запускает программу `child` через `execv()`.

5. В родительском процессе в цикле читает строки с терминала (функция `read_line()` поверх `read()`), отправляет их в дочерний процесс через `write()` и после каждой отправки ожидает статус от дочернего процесса (чтение 1 байта из `from_child`).

6. Завершает работу при достижении EOF по вводу, при закрытии канала со стороны дочернего процесса или при получении статуса `'Z'` (деление на ноль).

7. Закрывает каналы и ожидает завершения дочернего процесса через `waitpid()`.

Дочерний процесс выполняет следующие действия:

1. Получает имя выходного файла из аргументов командной строки.

2. Открывает файл системным вызовом `open()` (режим `O_WRONLY|O_CREAT|O_TRUNC`).

3. В цикле читает строки из `stdin` (перенаправленного на pipe) и парсит числа типа `float` функцией `strtof()`.

4. Если в строке некорректный формат — пишет в файл сообщение `bad line` и отправляет родителю статус `'O'`.

5. Если в строке меньше двух чисел — пишет `need at least 2 numbers` и отправляет статус `'O'`.

6. Выполняет деление первого числа на последующие. При обнаружении деления на ноль пишет `division by zero`, отправляет статус `'Z'` и завершает работу.

7. При успешном вычислении формирует строку результата вида `res=<value>` (округление до 3 знаков после запятой) и записывает её в файл, затем отправляет родителю статус `'O'`.

Архитектура программы

- Родительский процесс (parent.c)
  - Создаёт дочерний процесс через `fork()`.
  - Настраивает каналы и перенаправляет потоки `stdin/stdout` дочернего процесса через `dup2()`.
  - Запускает `child` через `execv()`.
  - После каждой отправленной строки ожидает 1-байтовый ответ от дочернего процесса.
- Дочерний процесс (child.c)
  - Читает строки из `stdin` (канал `to_child`).
  - Пишет результаты/сообщения об ошибках в выходной файл через `open()/write()`.
  - Отправляет однобайтовый статус в `stdout` (канал `from_child`):
    * `'O'` — обработка строки завершена (успех или нефатальная ошибка ввода);

    * 'Z' — обнаружено деление на ноль (фатальная ошибка, дочерний процесс завершает работу).

- Каналы:
  - to_child: родитель → дочерний (stdin дочернего процесса)
  - from_child: дочерний → родитель (stdout дочернего процесса)

Описание программы

Программа состоит из двух файлов: parent.c (родительский процесс) и child.c (дочерний процесс).

В отличие от предыдущей версии, ввод/вывод в обоих процессах реализован через низкоуровневые системные вызовы read() и write() (без использования stdio).

Родительский процесс:

- запрашивает имя выходного файла;
- создаёт два канала для обмена с дочерним процессом;
- создаёт дочерний процесс (fork()), перенаправляет стандартные потоки дочернего процесса на каналы (dup2()) и запускает child через execv();
- отправляет дочернему процессу строки с числами и после каждой строки получает однобайтовый статус: 'O' — продолжать работу, 'Z' — обнаружено деление на ноль (нужно завершиться).

Дочерний процесс:

- открывает выходной файл системным вызовом open();
- читает строки из stdin, парсит числа функцией strtof() и выполняет последовательное деление первого числа на последующие;
- пишет результат в файл в формате res=<value> (3 знака после запятой) и сообщает родителю статусом 'O';
- при делении на ноль пишет сообщение division by zero, отправляет статус 'Z' и завершает работу.

Используемые системные вызовы

pipe(int pipefd[2]) Создает канал и возвращает два файловых дескриптора: pipefd[0] для чтения и pipefd[1] для записи.

fork() Создает копию текущего процесса. Возвращает PID дочернего процесса в родительском процессе и 0 в дочернем процессе.

execv(const char *path, char *const argv[]) Заменяет образ текущего процесса новой программой. Принимает путь к программе и массив аргументов командной строки.

`dup2(int oldfd, int newfd)` Дублирует файловый дескриптор, позволяя перенаправить стандартные потоки (`stdin`, `stdout`) на каналы.

`write(int fd, const void *buf, size_t count)` Записывает данные в файловый дескриптор (в данном случае – в канал).

`read(int fd, void *buf, size_t count)` Читает данные из файлового дескриптора (в данном случае – из канала).

`open(const char *pathname, int flags, mode_t mode)` Открывает (или создаёт) файл и возвращает файловый дескриптор. Используется дочерним процессом для создания/очистки выходного файла.

`close(int fd)` Закрывает файловый дескриптор (концы каналов, а также выходной файл в дочернем процессе).

`waitpid(pid_t pid, int *status, int options)` Ожидает завершения дочернего процесса с указанным PID.

`_exit(int status)` Немедленно завершает процесс (используется в функции `die()` при фатальных ошибках).

Обработка ошибок

Программа обрабатывает следующие системные ошибки:

- Ошибки создания каналов (`pipe()`)
- Ошибки создания процесса (`fork()`)
- Ошибки перенаправления потоков (`dup2()`)
- Ошибки запуска программы (`execv()`)
- Ошибки записи в канал (`write()`)
- Ошибки чтения из файловых дескрипторов (`read()`)
- Ошибки открытия/создания выходного файла (`open()`)
- Деление на ноль (проверяется в дочернем процессе)
- Некорректная строка ввода и отсутствие делителей (обрабатывается в дочернем процессе как нефатальная ошибка)

Все ошибки обрабатываются через проверку возвращаемых значений системных вызовов и вывод сообщений об ошибках через `perror()`.

Результаты

Компиляция:

- `gcc parent.c -o parent`
- `gcc child.c -o child`

Запуск: `./parent`. После запроса имени файла вводятся строки с числами (по одной строке). Завершение ввода — EOF (Ctrl+D) или аварийное завершение при делении на ноль.

Проверка корректности

Тест 1 (корректные данные).

- Ввод: имя файла `out.txt`, далее строка `"10 2 5"`
- Ожидаемо: $10/2/5 = 1$
- В выходном файле: `res=1.000`

Тест 2 (деление на ноль).

- Ввод: строка `"7 0 2"`
- Ожидаемо: дочерний процесс записывает `division by zero`, отправляет статус `'Z'` и завершает работу; родительский процесс, получив `'Z'`, также завершает цикл.
- В выходном файле: строка `division by zero`

Тест 3 (ошибка формата и недостаточно чисел).

- Ввод: строка `"abc"` — в файл записывается `bad line`, работа продолжается.
- Ввод: строка `"5"` — в файл записывается `need at least 2 numbers`, работа продолжается.

Выводы

В ходе выполнения лабораторной работы были изучены и реализованы механизмы межпроцессного взаимодействия в операционной системе Linux:

1. **Создание процессов**: Освоен системный вызов `fork()` для создания дочерних процессов и семейство `exec*()` (в работе — `execv()`) для запуска отдельной программы в дочернем процессе.
2. **Каналы (pipes)**: Реализована двусторонняя передача данных между процессами через два канала с использованием `pipe()` и перенаправления стандартных потоков через `dup2()`.
3. **Низкоуровневый ввод-вывод**: Для обмена данными использованы системные вызовы `read()` и `write()` вместо функций `stdio`.

4. **Обработка ошибок и протокол статуса**: Реализован простой протокол синхронизации: дочерний процесс после обработки каждой строки отправляет родителю один байт статуса (`'O'` — продолжать работу, `'Z'` — деление на ноль). При `'Z'` оба процесса корректно завершают работу.

5. **Работа с файлами**: Дочерний процесс создаёт/очищает выходной файл системным вызовом `open()` и записывает результаты через `write()`.

Программа успешно выполняет поставленную задачу, демонстрируя создание процессов, настройку межпроцессного взаимодействия через каналы и работу с файловыми дескрипторами на уровне системных вызовов.

Исходная программа

Родительский процесс (parent.c)

```c
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

static void die(const char *msg) {
    size_t n = 0;
    while (msg[n]) n++;
    write(2, msg, n);
    write(2, "\n", 1);
    _exit(1);
}

static int read_line(int fd, char *buf, int max) {
    int i = 0;
    char c;
    while (i < max - 1) {
        ssize_t r = read(fd, &c, 1);
        if (r == 0) break;
        if (r < 0) return -1;
        buf[i++] = c;
        if (c == '\n') break;
    }
    buf[i] = 0;
    return i;
}

int main(void) {
    int to_child[2];
    int from_child[2];
    pid_t pid;
    char filename[256];
    char line[1024];

    write(1, "Output file name: ", 18);
    if (read_line(0, filename, (int)sizeof(filename)) <= 0) die("No file name");

    for (int i = 0; filename[i]; i++) {
        if (filename[i] == '\n') { filename[i] = 0; break; }
    }

```

```
    if (pipe(to_child) < 0) die("pipe failed");
    if (pipe(from_child) < 0) die("pipe failed");

    pid = fork();
    if (pid < 0) die("fork failed");

    if (pid == 0) {
        close(to_child[1]);
        close(from_child[0]);

        if (dup2(to_child[0], 0) < 0) die("dup2 failed");
        if (dup2(from_child[1], 1) < 0) die("dup2 failed");

        close(to_child[0]);
        close(from_child[1]);

        char *argv[] = { "./child", filename, 0 };
        execv("./child", argv);
        die("execv failed");
    }

    close(to_child[0]);
    close(from_child[1]);

    while (1) {
        int n = read_line(0, line, (int)sizeof(line));
        if (n == 0) break;
        if (n < 0) die("read error");

        if (n == 1 && line[0] == '\n') continue;

        if (write(to_child[1], line, (size_t)n) != n) die("write to child failed");

        char st;
        ssize_t r = read(from_child[0], &st, 1);
        if (r <= 0) break;
        if (st == 'Z') break;
    }

    close(to_child[1]);
    close(from_child[0]);

    waitpid(pid, 0, 0);
    return 0;
}
```

Дочерний процесс (child.c)

```
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

static void die(const char *msg) {
    size_t n = 0;
    while (msg[n]) n++;
    write(2, msg, n);
```

```
    write(2, "\n", 1);
    _exit(1);
}

static int read_line(int fd, char *buf, int max) {
    int i = 0;
    char c;
    while (i < max - 1) {
        ssize_t r = read(fd, &c, 1);
        if (r == 0) break;
        if (r < 0) return -1;
        buf[i++] = c;
        if (c == '\n') break;
    }
    buf[i] = 0;
    return i;
}

static int utoa(unsigned int x, char *out) {
    char tmp[20];
    int n = 0;
    if (x == 0) { out[0] = '0'; return 1; }
    while (x > 0 && n < (int)sizeof(tmp)) {
        tmp[n++] = (char)('0' + (x % 10));
        x /= 10;
    }
    for (int i = 0; i < n; i++) out[i] = tmp[n - 1 - i];
    return n;
}

static int ftoa3(float v, char *out) {
    int pos = 0;
    if (v < 0) { out[pos++] = '-'; v = -v; }

    unsigned int ip = (unsigned int)v;
    float frac = v - (float)ip;
    unsigned int fp = (unsigned int)(frac * 1000.0f + 0.5f);
    if (fp >= 1000) { fp -= 1000; ip += 1; }

    pos += utoa(ip, out + pos);
    out[pos++] = '.';
    out[pos++] = (char)('0' + (fp / 100) % 10);
    out[pos++] = (char)('0' + (fp / 10) % 10);
    out[pos++] = (char)('0' + (fp % 10));
    return pos;
}

int main(int argc, char **argv) {
    if (argc < 2) die("no output file");
    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd < 0) die("open failed");

    char line[1024];

    while (1) {
        int n = read_line(0, line, (int)sizeof(line));
        if (n == 0) break;
        if (n < 0) die("read error");
```

```
67
68          char *p = line;
69          char *end;
70
71          float a = strtof(p, &end);
72          if (end == p) {
73              char msg[] = "bad line\n";
74              write(fd, msg, sizeof(msg)-1);
75              write(1, "0", 1);
76              continue;
77          }
78          p = end;
79
80          float res = a;
81          int have_div = 0;
82
83          while (1) {
84              float b = strtof(p, &end);
85              if (end == p) break;
86              have_div = 1;
87
88              if (b == 0.0f) {
89                  char msg[] = "division by zero\n";
90                  write(fd, msg, sizeof(msg)-1);
91                  write(1, "Z", 1);
92                  close(fd);
93                  return 0;
94              }
95              res = res / b;
96              p = end;
97          }
98
99          if (!have_div) {
100             char msg[] = "need at least 2 numbers\n";
101             write(fd, msg, sizeof(msg)-1);
102             write(1, "0", 1);
103             continue;
104         }
105
106         char out[64];
107         int k = 0;
108         out[k++] = 'r'; out[k++] = 'e'; out[k++] = 's'; out[k++] = '=';
109         k += ftoa3(res, out + k);
110         out[k++] = '\n';
111         write(fd, out, (size_t)k);
112
113         write(1, "0", 1);
114     }
115
116     close(fd);
117     return 0;
118 }
```

Strace

Ниже приведён вывод `strace` для запуска программы `parent` (которая создаёт дочерний процесс и выполняет `execve` программы `child`).

```
4938  12:27:40.311084 execve("./parent", ["./parent"], 0x7ffce344ee98 /* 27 vars */) =
4938  12:27:40.311753 brk(NULL)               = 0x581d6cd40000 <0.000063>
4938  12:27:40.311944 arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd03b285e0) = -1 EINVAL (
4938  12:27:40.312143 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS
4938  12:27:40.312295 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or
4938  12:27:40.312451 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3 <0
4938  12:27:40.312616 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=17920, ...}, AT
4938  12:27:40.312759 mmap(NULL, 17920, PROT_READ, MAP_PRIVATE, 3, 0) = 0x70563553200
4938  12:27:40.312898 close(3)                = 0 <0.000051>
4938  12:27:40.313027 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_
4938  12:27:40.313198 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\
4938  12:27:40.313363 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
4938  12:27:40.313505 pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\
4938  12:27:40.313681 pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O{\f\225\\=\201\327\3
4938  12:27:40.313862 newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...},
4938  12:27:40.314022 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
4938  12:27:40.314171 mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0)
4938  12:27:40.314311 mprotect(0x705635228000, 2023424, PROT_NONE) = 0 <0.000079>
4938  12:27:40.314472 mmap(0x705635228000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|
4938  12:27:40.314618 mmap(0x7056353bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|M
4938  12:27:40.314800 mmap(0x705635416000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|M
4938  12:27:40.314948 mmap(0x70563541c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|M
4938  12:27:40.315095 close(3)                = 0 <0.000049>
4938  12:27:40.315237 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOU
4938  12:27:40.315393 arch_prctl(ARCH_SET_FS, 0x70563552f740) = 0 <0.000053>
4938  12:27:40.315532 set_tid_address(0x70563552fa10) = 4938 <0.000069>
4938  12:27:40.315774 set_robust_list(0x70563552fa20, 24) = 0 <0.000077>
4938  12:27:40.315975 rseq(0x7056355300e0, 0x20, 0, 0x53053053) = 0 <0.000092>
4938  12:27:40.316264 mprotect(0x705635416000, 16384, PROT_READ) = 0 <0.000077>
4938  12:27:40.316452 mprotect(0x581d5344e000, 4096, PROT_READ) = 0 <0.000085>
4938  12:27:40.316640 mprotect(0x705635571000, 8192, PROT_READ) = 0 <0.000076>
4938  12:27:40.316839 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=
4938  12:27:40.317013 munmap(0x705635532000, 17920) = 0 <0.000087>
4938  12:27:40.317225 write(1, "Output file name: ", 18) = 18 <0.000088>
4938  12:27:40.317417 read(0, "t", 1)   = 1 <5.253602>
4938  12:27:45.571203 read(0, "e", 1)   = 1 <0.000079>
4938  12:27:45.571410 read(0, "s", 1)   = 1 <0.000072>
4938  12:27:45.571564 read(0, "t", 1)   = 1 <0.000071>
4938  12:27:45.571726 read(0, ".", 1)   = 1 <0.000096>
4938  12:27:45.571938 read(0, "t", 1)   = 1 <0.000071>
4938  12:27:45.572125 read(0, "x", 1)   = 1 <0.000136>
4938  12:27:45.572383 read(0, "t", 1)   = 1 <0.000103>
4938  12:27:45.572608 read(0, "\n", 1)  = 1 <0.000101>
4938  12:27:45.572874 pipe2([3, 4], 0)  = 0 <0.000092>
4938  12:27:45.573089 pipe2([5, 6], 0)  = 0 <0.000068>
4938  12:27:45.573270 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_
4938  12:27:45.573744 close(3 <unfinished ...>
4959  12:27:45.573793 set_robust_list(0x70563552fa20, 24 <unfinished ...>
```

```
4938  12:27:45.573835 <... close resumed>) = 0 <0.000063>
4959  12:27:45.573864 <... set_robust_list resumed>) = 0 <0.000055>
4938  12:27:45.573892 close(6)          = 0 <0.000062>
4959  12:27:45.573987 close(4 <unfinished ...>
4938  12:27:45.574017 read(0,  <unfinished ...>
4959  12:27:45.574044 <... close resumed>) = 0 <0.000043>
4959  12:27:45.574120 close(5)          = 0 <0.000064>
4959  12:27:45.574267 dup2(3, 0)        = 0 <0.000055>
4959  12:27:45.574404 dup2(6, 1)        = 1 <0.000056>
4959  12:27:45.574535 close(3)          = 0 <0.000055>
4959  12:27:45.574699 close(6)          = 0 <0.000065>
4959  12:27:45.574855 execve("./child", ["./child", "test.txt"], 0x7ffd03b287b8 /* 27
4959  12:27:45.575377 brk(NULL)         = 0x5b7a9c8ee000 <0.000066>
4959  12:27:45.575571 arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc6731fe30) = -1 EINVAL (
4959  12:27:45.575743 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOU
4959  12:27:45.575936 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or
4959  12:27:45.576095 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3 <0
4959  12:27:45.576239 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=17920, ...}, A
4959  12:27:45.576415 mmap(NULL, 17920, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a5fe667300
4959  12:27:45.576577 close(3)          = 0 <0.000080>
4959  12:27:45.576765 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_
4959  12:27:45.576947 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\
4959  12:27:45.577104 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
4959  12:27:45.577276 pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\
4959  12:27:45.577428 pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O{\f\225\\=\201\327\3
4959  12:27:45.577578 newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...},
4959  12:27:45.577753 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
4959  12:27:45.577907 mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0)
4959  12:27:45.578090 mprotect(0x7a5fe6428000, 2023424, PROT_NONE) = 0 <0.000072>
4959  12:27:45.578241 mmap(0x7a5fe6428000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|
4959  12:27:45.578407 mmap(0x7a5fe65bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|M
4959  12:27:45.578568 mmap(0x7a5fe6616000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|M
4959  12:27:45.578787 mmap(0x7a5fe661c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|M
4959  12:27:45.579001 close(3)          = 0 <0.000067>
4959  12:27:45.579170 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMO
4959  12:27:45.579337 arch_prctl(ARCH_SET_FS, 0x7a5fe6670740) = 0 <0.000061>
4959  12:27:45.579476 set_tid_address(0x7a5fe6670a10) = 4959 <0.000065>
4959  12:27:45.579617 set_robust_list(0x7a5fe6670a20, 24) = 0 <0.000051>
4959  12:27:45.579748 rseq(0x7a5fe66710e0, 0x20, 0, 0x53053053) = 0 <0.000072>
4959  12:27:45.579994 mprotect(0x7a5fe6616000, 16384, PROT_READ) = 0 <0.000068>
4959  12:27:45.580169 mprotect(0x5b7a967ae000, 4096, PROT_READ) = 0 <0.000078>
4959  12:27:45.580344 mprotect(0x7a5fe66b2000, 8192, PROT_READ) = 0 <0.000068>
4959  12:27:45.580529 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=
4959  12:27:45.580698 munmap(0x7a5fe6673000, 17920) = 0 <0.000099>
4959  12:27:45.580914 openat(AT_FDCWD, "test.txt", O_WRONLY|O_CREAT|O_TRUNC, 0644) =
4959  12:27:45.581236 read(0,  <unfinished ...>
4938  12:27:48.269277 <... read resumed>"1", 1) = 1 <2.695241>
4938  12:27:48.269431 read(0, "0", 1)   = 1 <0.000074>
```

```
4938  12:27:48.269657 read(0, " ", 1)   = 1 <0.000082>
4938  12:27:48.269847 read(0, "2", 1)   = 1 <0.000066>
4938  12:27:48.270003 read(0, " ", 1)   = 1 <0.000055>
4938  12:27:48.270147 read(0, "5", 1)   = 1 <0.000097>
4938  12:27:48.270353 read(0, "\n", 1)  = 1 <0.000076>
4938  12:27:48.270534 write(4, "10 2 5\n", 7) = 7 <0.000098>
4959  12:27:48.270688 <... read resumed>"1", 1) = 1 <2.689428>
4938  12:27:48.270746 read(5,  <unfinished ...>
4959  12:27:48.270793 read(0, "0", 1)   = 1 <0.000081>
4959  12:27:48.270974 read(0, " ", 1)   = 1 <0.000060>
4959  12:27:48.271132 read(0, "2", 1)   = 1 <0.000053>
4959  12:27:48.271271 read(0, " ", 1)   = 1 <0.000057>
4959  12:27:48.271407 read(0, "5", 1)   = 1 <0.000052>
4959  12:27:48.271543 read(0, "\n", 1)  = 1 <0.000060>
4959  12:27:48.271727 write(3, "res=1.000\n", 10) = 10 <0.000081>
4959  12:27:48.271922 write(1, "0", 1 <unfinished ...>
4938  12:27:48.272031 <... read resumed>"0", 1) = 1 <0.001265>
4959  12:27:48.272076 <... write resumed>) = 1 <0.000129>
4938  12:27:48.272106 read(0,  <unfinished ...>
4959  12:27:48.272134 read(0,  <unfinished ...>
4938  12:27:50.713159 <... read resumed>"7", 1) = 1 <2.441012>
4938  12:27:50.713354 read(0, " ", 1)   = 1 <0.000080>
4938  12:27:50.713549 read(0, "0", 1)   = 1 <0.000128>
4938  12:27:50.720585 read(0, " ", 1)   = 1 <0.000074>
4938  12:27:50.720790 read(0, "1", 1)   = 1 <0.000071>
4938  12:27:50.720969 read(0, "\n", 1)  = 1 <0.000069>
4938  12:27:50.721284 write(4, "7 0 1\n", 6) = 6 <0.000102>
4959  12:27:50.721437 <... read resumed>"7", 1) = 1 <2.449292>
4938  12:27:50.721481 read(5,  <unfinished ...>
4959  12:27:50.721529 read(0, " ", 1)   = 1 <0.000104>
4959  12:27:50.721766 read(0, "0", 1)   = 1 <0.000080>
4959  12:27:50.721967 read(0, " ", 1)   = 1 <0.000071>
4959  12:27:50.722148 read(0, "1", 1)   = 1 <0.000100>
4959  12:27:50.722363 read(0, "\n", 1)  = 1 <0.000065>
4959  12:27:50.722535 write(3, "division by zero\n", 17) = 17 <0.000074>
4959  12:27:50.722696 write(1, "Z", 1 <unfinished ...>
4938  12:27:50.722797 <... read resumed>"Z", 1) = 1 <0.001298>
4959  12:27:50.722837 <... write resumed>) = 1 <0.000120>
4938  12:27:50.722867 close(4 <unfinished ...>
4959  12:27:50.722894 close(3 <unfinished ...>
4938  12:27:50.722921 <... close resumed>) = 0 <0.000043>
4938  12:27:50.722993 close(5)          = 0 <0.000055>
4959  12:27:50.723086 <... close resumed>) = 0 <0.000181>
4938  12:27:50.723116 wait4(4959,  <unfinished ...>
4959  12:27:50.723182 exit_group(0)     = ?
4959  12:27:50.723508 +++ exited with 0 +++
4938  12:27:50.723576 <... wait4 resumed>NULL, 0, NULL) = 4959 <0.000446>
4938  12:27:50.723652 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4959,
```

```
4938  12:27:50.723771 exit_group(0)     = ?
4938  12:27:50.724031 +++ exited with 0 +++
```