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»**

Группа: М8О-211Б-23

Студент: Тимофеева И.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 11.12.2024

Москва, 2024

**Постановка задачи**

**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, 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. Необходимо выполнить операции с использованием shared memory и memory mapping.**

**Общий метод и алгоритм решения**

Использованные системные вызовы:

● pid\_t fork(void); – создает дочерний процесс.

● pid\_t getpid(void); – функция возвращает идентификатор (PID) вызвавшего процесса.

● int pipe(int fd[2]) – создание неименованного канала для передачи данных между

процессами.

● int shm\_open(const char \*name, int oflag, mode\_t mode); – функция устанавливает соединение между объектом разделяемой памяти и файловым дескриптором.

● int shm\_unlink(const char \*name); – функция удаляет объект разделяемой области памяти.

● void \*mmap (void \*address, size\_t length, int protect, int flags, int filedes, off\_t offset); - функция для сопоставления адресного пространства процесса с файлами или устройствами.

● int \*munmap (void \*addr, size\_t len); - функция удаляет сопоставление страниц памяти в заданном диапазоне адресов процесса.

● int ftruncate(int fd, off\_t length); – функция задает размер разделяемой памяти.

● ssize\_t read(int fd, void \*buf, size\_t count); – считывает до count байт из файлового

дескриптора fd в буфер buf.

● int32\_t snprintf(char \*msg, size\_t buffer\_size, const char \*restrict format, int x); - функция для преобразования числа x в отформатированную строку.

● ssize\_t write(int fd, const void \*buf, size\_t count); – записывает до count байт из буфера

buf в файловый дескриптор fd.

● ssize\_t readlink(const char \*restrict pathname, char \*restrict buf,size\_t bufsize); – считывает значение символьной ссылки pathname в буфер buf, у которого размер bufsize.

● int dup2(int oldfd, int newfd); – переназначение файлового дескриптора.

● int32\_t execv(char \*fname, const char \*argv, NULL); – заменяет текущий образ процесса новым образом процесса, загружает и выполняет новый дочерний процесс с исполняемым файлом fname и строкой аргументов argv.

● pid\_t wait(int \*stat\_loc); ожидание изменений состояния программы в дочерних процессах и получения соответствующей информации. Если дочерний процесс завершается, wait() возвращает PID завершённого дочернего процесса.

● float strtof(const char\* str, char\*\* endptr); - преобразует строку с представлением числа с плавающей точкой в значение типа float.

● int open(const char \*pathname, int flags, mode\_t mode) – открытие\создание файла.

● int close(int fd); – Закрывает файловый дескриптор fd.

● void perror(const char \*s) – функция выводит сообщение об ошибке в поток stderr.

● void exit(int status) – завершения выполнения процесса и возвращение статуса.

Создание каналов: Программа создает два канала (массив pipe и каналы pipe[1] и pipe[2]), чтобы организовать передачу данных между родительским процессом и одним дочерним процесом.

Далее с помощью функции **shm\_open** программа выделяет область разделяемой памяти, которая может использоваться другими процессами, в том числе дочерним процессом. Функция возвращает файловый дескриптор для ссылки на этот объект общей памяти. Этот дескриптор в дальнейшем используется функцией **mmap** для сопоставления адресного пространства процесса с файлами или устройствами.

После создания общего объекта памяти задается размер разделяемой памяти вызовом **ftruncate**. На входе у функции передаётся файловый дескриптор объекта и необходимый размер. Когда файл сопоставлен с адресным пространством памяти, к нему можно обращаться как к массиву в программе из любого процесса, который знает этот адрес.

Программа преобразует буфер данных, которые вводит пользователь в консоли, в массив чисел типа float и копирует этот массив в выделенную ранее область разделяемой памяти. В первом элементе этого массива будет хранится число его элементов. Название этой области памяти затем будет передаваться в дочерний процесс.

Ввод и открытие файлов: Программа запрашивает у пользователя имя файла, в который будут записаны выходные данные дочернего процесса. Открывает эти файлы с режимом O\_WRONLY | O\_CREAT | O\_TRUNC | O\_APPEND, что позволяет создавать новый файл или перезаписывать существующий.

Создание дочерних процессов:

Дочерний процесс (child): Порождается с помощью вызова fork() и перенастраивает стандартный ввод на чтение из pipe[1] и стандартный вывод на запись в file. Запускает внешний исполняемый файл ./client\_prog с помощью execv().

Программа через аргумент вызова получает название области разделяемой памяти, в которой родительский процесс сохранил данные для обработки. С помощью функций shm\_open() и mmap() эти данные выводятся в массив чисел float и производится деление первого введенного числа на все остальные. Останавливает работу при попытке деления на 0 или после последнего числа. Результат записывает в файл file, который указан как аргумент программы.

Родительский процесс: ожидает завершения дочернего процесса и также завершает работу.

Ожидание завершения дочернего процесса: Родительский процесс использует wait(), чтобы дождаться завершения дочернего процесса.

**Код программы**

**serv\_prog.c**

#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <stdlib.h>

#include <stdio.h>

#define **SHARED\_MEMORY\_OBJECT\_NAME** "my\_shared\_memory"

static char CLIENT\_PROGRAM\_NAME[] = "client\_prog";

*// make float array from char's buffer*

float\* **make\_array**(char\* buf)

{

    float\* arr = **NULL**;

    int i = 0;

    char\* nstr = buf;

    while (++i)

    {

        float d = 0;

        d = **strtof**(nstr, &nstr);

        if (d==0) break;

        float\* new\_arr = (float\*)**realloc**(arr, (i+1)\*sizeof(float));

        if (new\_arr != **NULL**)

        {

            arr = new\_arr;

            arr[0] = i; *// 1-st number stores length of array*

            arr[i] = d; *// add new array member*

        }

    }

    return arr;

}

int **main**(int argc, char \*\*argv) {

    if (argc == 1) {

        char msg[1024];

**uint32\_t** len = **snprintf**(msg, sizeof(msg) - 1, "usage: %s filename\n", argv[0]);

**write**(STDERR\_FILENO, msg, len);

**exit**(**EXIT\_SUCCESS**);

    }

    pid\_t ppid = **getpid**(); *// NOTE: Get parent PID*

    {

        char msg[128];

**int32\_t** len = **snprintf**(msg, sizeof(msg) - 1, "%d: Start typing row of number. Press 'Ctrl-D' or 'Enter' with no input to exit\n", ppid);

**write**(STDOUT\_FILENO, msg, len);

    }

*// console input data*

    char buf[4096];

    ssize\_t bytes;

    while (bytes = **read**(STDIN\_FILENO, buf, sizeof(buf))) {

        if (bytes < 0) {

            const char msg[] = "error: failed to read from stdin\n";

**write**(STDERR\_FILENO, msg, sizeof(msg));

**exit**(**EXIT\_FAILURE**);

        } else if (buf[0] == '\n' || buf[bytes-1] == '\n') {

*// NOTE: When Enter is pressed with no input, then exit*

            break;

        }

    }

*// create an array from input data*

**uint32\_t** ArrSize, i;

    float\* numbers = **make\_array**(buf);

    if (numbers != **NULL**)

    {

        ArrSize = (int)numbers[0];

    }

*// create a shared memory object to store data*

**uint32\_t** shm;

    if ( (shm = **shm\_open**(**SHARED\_MEMORY\_OBJECT\_NAME**, **O\_CREAT**|**O\_RDWR**, 0777)) == -1 ) {

**perror**("shm\_open");

        return 1;

    }

*// trunc file to size N+1 bytes*

    if ( **ftruncate**(shm, ArrSize+1) == -1 ) {

**perror**("ftruncate");

        return 1;

    }

*// data mapping to memory*

    float \*addr = (float\*)**mmap**(**NULL**, (ArrSize+1) \* sizeof(float), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm, 0);

    if (addr == MAP\_FAILED) {

**perror**("Mapping Failed\n");

        return 1;

    }

    for (i = 0; i <= ArrSize; i++) addr[i] = (float)numbers[i];

    int err = **munmap**(addr, (ArrSize+1) \* sizeof(float));

**close**(shm);

    if (err != 0) {

**perror**("Unmapping Failed\n");

        return 1;

    }

*// NOTE: Get full path to the directory, where program resides*

    char progpath[1024];

    {

#ifdef **\_\_APPLE\_\_**

        uint32\_t len = **proc\_pidpath**(ppid, progpath, sizeof(progpath));

#else *// not \_\_APPLE\_\_*

*// Read the symbolic link '/proc/self/exe'.*

*// определить каталог с программой*

        const char \*linkName = "/proc/self/exe";

**uint32\_t** len = **readlink**(linkName, progpath, sizeof(progpath) - 1);

#endif *// else not \_\_APPLE\_\_*

        if (len <= 0)

        {

            const char msg[] = "error: failed to read full program path\n";

**write**(STDERR\_FILENO, msg, sizeof(msg));

**exit**(**EXIT\_FAILURE**);

        }

        while (progpath[len] != '/')

            --len;

        progpath[len] = 0;

    }

*// NOTE: Open pipe*

    int channel[2];

    if (**pipe**(channel) == -1) {

        const char msg[] = "error: failed to create pipe\n";

**write**(STDERR\_FILENO, msg, sizeof(msg));

**exit**(**EXIT\_FAILURE**);

    }

*// NOTE: Spawn a new process*

    const pid\_t child = **fork**();

    switch (child) {

        case -1: { *// NOTE: Kernel fails to create another process*

            const char msg[] = "error: failed to spawn new process\n";

**write**(STDERR\_FILENO, msg, sizeof(msg));

**exit**(**EXIT\_FAILURE**);

        } break;

        case 0: { *// NOTE: We're a child, child doesn't know its pid after fork*

            pid\_t pid = **getpid**(); *// NOTE: Get child PID*

**dup2**(STDIN\_FILENO, channel[STDIN\_FILENO]);

**close**(channel[STDOUT\_FILENO]);

            {

                char msg[64];

                const **int32\_t** length = **snprintf**(msg, sizeof(msg), "%d: I'm a child\n", pid);

**write**(STDOUT\_FILENO, msg, length);

            }

            {

                char path[1024];

**snprintf**(path, sizeof(path) - 1, "%s/%s", progpath, CLIENT\_PROGRAM\_NAME) < 0 ? **abort**() : (void)0;

*// NOTE: args[0] must be a program name, next the actual arguments*

*// NOTE: `NULL` at the end is mandatory, because `exec\*`*

*//       expects a NULL-terminated list of C-strings*

                char \*const args[] = {CLIENT\_PROGRAM\_NAME, argv[1], **SHARED\_MEMORY\_OBJECT\_NAME**, **NULL**};

**int32\_t** status = **execv**(path, args);

                if (status == -1) {

                    const char msg[] = "error: failed to exec into new exectuable image\n";

**write**(STDERR\_FILENO, msg, sizeof(msg));

**exit**(**EXIT\_FAILURE**);

                }

            }

        } break;

        default: { *// NOTE: We're a parent, parent knows PID of child after fork*

            pid\_t pid = **getpid**(); *// NOTE: Get parent PID*

            {

                char msg[64];

                const **int32\_t** length = **snprintf**(msg, sizeof(msg),

                    "%d: I'm a parent, my child has PID %d\n", pid, child);

**write**(STDOUT\_FILENO, msg, length);

            }

*// NOTE: `wait` blocks the parent until child exits*

            int child\_status;

**wait**(&child\_status);

            if (child\_status != **EXIT\_SUCCESS**) {

                const char msg[] = "error: child exited with error\n";

**write**(STDERR\_FILENO, msg, sizeof(msg));

**exit**(child\_status);

            }

        } break;

    }

}

**client\_prog.c**

#include <stdint.h>

#include <stdbool.h>

#include <sys/mman.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdio.h>

int **main**(int argc, char \*\*argv) {

    char buf[4096];

    ssize\_t bytes;

**int32\_t** file = **open**(argv[1], **O\_WRONLY** | **O\_CREAT** | **O\_TRUNC** | **O\_APPEND**, 0600);

    if (file == -1) {

        const char msg[] = "error: failed to open requested file\n";

**write**(STDERR\_FILENO, msg, sizeof(msg));

**exit**(**EXIT\_FAILURE**);

    }

    pid\_t pid = **getpid**();

    char messg[300];

**int32\_t** len = **snprintf**(messg, sizeof(messg), "%d: Solution of expression (", pid);

**write**(STDERR\_FILENO, messg, len);

**write**(file, messg, len);

    char\* SHARED\_MEMORY\_OBJECT\_NAME = argv[2];

    int i, shm, N = 0;

    float \*addr;

    float d, res;

    if ( (shm = **shm\_open**(SHARED\_MEMORY\_OBJECT\_NAME, 0|**O\_RDWR**, 0777)) == -1 ) {

**perror**("shm\_open");

        return 1;

    }

    for (i=0; i<2; i++)

    {

        addr = (float\*)**mmap**(**NULL**, (N+1)\*sizeof(float), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm, 0);

        if (addr == MAP\_FAILED) {

**perror**("Mapping Failed\n");

            return 1;

        }

        if (i==0)

        {

            N = (int)addr[0];

        }

    }

    res = 0;

    for (i = 1; i <= N; i++) {

        d = addr[i];

        if (res == 0)

           res = d;

        else

           res = res / d;

*// Выводим результат преобразования на экран*

        char msg[32];

**int32\_t** len;

        if (i==1)

            len = **snprintf**(msg, sizeof(msg) - 1, "%0.3f", d);

        else

            len = **snprintf**(msg, sizeof(msg) - 1, " / %0.3f", d);

**write**(STDERR\_FILENO, msg, len);

**write**(file, msg, len);

    }

    len = **snprintf**(messg, sizeof(messg) - 1, ") is %0.3f\n",res);

**write**(STDERR\_FILENO, messg, len);

**int32\_t** written = **write**(file, messg, len);

    if (written != len) {

        const char msg[] = "error: failed to write to file\n";

**write**(STDERR\_FILENO, msg, sizeof(msg));

**exit**(**EXIT\_FAILURE**);

    }

    int err = **munmap**(addr, (N+1) \* sizeof(float));

    if (err != 0) {

**perror**("Unmapping Failed\n");

        return 1;

    }

**shm\_unlink**(SHARED\_MEMORY\_OBJECT\_NAME);

}

**Протокол работы программы**

**Тестирование:**

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab03$ gcc serv\_prog.c

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab03$ gcc client\_prog.c -o client\_prog

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab03$ ./a.out result.txt

3186: Start typing row of number. Press 'Ctrl-D' or 'Enter' with no input to exit

27.9 1.25 3.57 -4.84 0.02

3186: I'm a parent, my child has PID 3187

3187: I'm a child

3187: Solution of **expression** (27.900 / 1.250 / 3.570 / -4.840 / 0.020) is -64.588

**Strace**

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab03$ strace -f ./a.out result.txt

**execve**("./a.out", ["./a.out", "result.txt"], 0x7ffed17d9f30 */\* 46 vars \*/*) = 0

**brk**(NULL)                               = 0x62466fd9a000

**arch\_prctl**(0x3001 */\* ARCH\_??? \*/*, 0x7ffd688a7710) = -1 **EINVAL** (Недопустимый аргумент)

**mmap**(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x759a20742000

**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

**newfstatat**(3, "", {st\_mode=S\_IFREG|0644, st\_size=64071, ...}, AT\_EMPTY\_PATH) = 0

**mmap**(NULL, 64071, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x759a20732000

**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\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 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) = 0x759a20400000

**mprotect**(0x759a20428000, 2023424, PROT\_NONE) = 0

**mmap**(0x759a20428000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x759a20428000

**mmap**(0x759a205bd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x759a205bd000

**mmap**(0x759a20616000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x759a20616000

**mmap**(0x759a2061c000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x759a2061c000

**close**(3)                                = 0

**mmap**(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x759a2072f000

**arch\_prctl**(ARCH\_SET\_FS, 0x759a2072f740) = 0

**set\_tid\_address**(0x759a2072fa10)         = 3186

**set\_robust\_list**(0x759a2072fa20, 24)     = 0

**rseq**(0x759a207300e0, 0x20, 0, 0x53053053) = 0

**mprotect**(0x759a20616000, 16384, PROT\_READ) = 0

**mprotect**(0x62466e6c8000, 4096, PROT\_READ) = 0

**mprotect**(0x759a2077c000, 8192, PROT\_READ) = 0

**prlimit64**(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

**munmap**(0x759a20732000, 64071)           = 0

**getpid**()                                = 3186

**write**(1, "3186: Start typing row of number"..., 823186: Start typing row of number. Press 'Ctrl-D' or 'Enter' with no input to exit

) = 82

**read**(0, 27.9 1.25 3.57 -4.84 0.02

"27.9 1.25 3.57 -4.84 0.02\n", 4096) = 26

**getrandom**("\x84\x33\x16\xda\x4e\x84\x9b\x8e", 8, GRND\_NONBLOCK) = 8

**brk**(NULL)                               = 0x62466fd9a000

**brk**(0x62466fdbb000)                     = 0x62466fdbb000

**openat**(AT\_FDCWD, "/dev/shm/my\_shared\_memory", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0777) = 3

**ftruncate**(3, 6)                         = 0

**mmap**(NULL, 24, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x759a2077b000

**munmap**(0x759a2077b000, 24)              = 0

**close**(3)                                = 0

**readlink**("/proc/self/exe", "/home/irina/Prog/Prog\_C/Kurs2/La"..., 1023) = 47

**pipe2**([3, 4], 0)                        = 0

**clone**(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 3187 attached

, child\_tidptr=0x759a2072fa10) = 3187

[pid  3186] **getpid**( <unfinished ...>

[pid  3187] **set\_robust\_list**(0x759a2072fa20, 24 <unfinished ...>

[pid  3186] <... getpid resumed>)       = 3186

[pid  3187] <... set\_robust\_list resumed>) = 0

[pid  3186] **write**(1, "3186: I'm a parent, my child has"..., 423186: I'm a parent, my child has PID 3187

) = 42

[pid  3187] getpid( <unfinished ...>

[pid  3186] wait4(-1,  <unfinished ...>

[pid  3187] <... getpid resumed>)       = 3187

[pid  3187] dup2(0, 3)                  = 3

[pid  3187] close(4)                    = 0

[pid  3187] write(1, "3187: I'm a child\n", 183187: I'm a child

) = 18

[pid  3187] execve("/home/irina/Prog/Prog\_C/Kurs2/LabOS/Lab03/client\_prog", ["cli-ent\_prog", "result.txt", "my\_shared\_memory"], 0x7ffd688a78f0 /\* 46 vars \*/) = 0

[pid  3187] brk(NULL)                   = 0x5b80560f1000

[pid  3187] arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffbec171a0) = -1 EINVAL (Недопустимый аргумент)

[pid  3187] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ae292232000

[pid  3187] access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (Нет такого файла или ката-лога)

[pid  3187] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 4

[pid  3187] newfstatat(4, "", {st\_mode=S\_IFREG|0644, st\_size=64071, ...}, AT\_EMPTY\_PATH) = 0

[pid  3187] mmap(NULL, 64071, PROT\_READ, MAP\_PRIVATE, 4, 0) = 0x7ae292222000

[pid  3187] close(4)                    = 0

[pid  3187] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 4

[pid  3187] read(4, "\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

[pid  3187] pread64(4, "\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

[pid  3187] pread64(4, "\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

[pid  3187] pread64(4, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

[pid  3187] newfstatat(4, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...}, AT\_EMPTY\_PATH) = 0

[pid  3187] pread64(4, "\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

[pid  3187] mmap(NULL, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 4, 0) = 0x7ae291e00000

[pid  3187] mprotect(0x7ae291e28000, 2023424, PROT\_NONE) = 0

[pid  3187] mmap(0x7ae291e28000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x28000) = 0x7ae291e28000

[pid  3187] mmap(0x7ae291fbd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x1bd000) = 0x7ae291fbd000

[pid  3187] mmap(0x7ae292016000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x215000) = 0x7ae292016000

[pid  3187] mmap(0x7ae29201c000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ae29201c000

[pid  3187] close(4)                    = 0

[pid  3187] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ae29221f000

[pid  3187] arch\_prctl(ARCH\_SET\_FS, 0x7ae29221f740) = 0

[pid  3187] set\_tid\_address(0x7ae29221fa10) = 3187

[pid  3187] set\_robust\_list(0x7ae29221fa20, 24) = 0

[pid  3187] rseq(0x7ae2922200e0, 0x20, 0, 0x53053053) = 0

[pid  3187] mprotect(0x7ae292016000, 16384, PROT\_READ) = 0

[pid  3187] mprotect(0x5b80553eb000, 4096, PROT\_READ) = 0

[pid  3187] mprotect(0x7ae29226c000, 8192, PROT\_READ) = 0

[pid  3187] prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid  3187] munmap(0x7ae292222000, 64071) = 0

[pid  3187] openat(AT\_FDCWD, "result.txt", O\_WRONLY|O\_CREAT|O\_TRUNC|O\_APPEND, 0600) = 4

[pid  3187] getpid()                    = 3187

[pid  3187] write(2, "3187: Solution of **expression** (", 303187: Solution of expression () = 30

[pid  3187] write(4, "3187: Solution of **expression** (", 30) = 30

[pid  3187] openat(AT\_FDCWD, "/dev/shm/my\_shared\_memory", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = 5

[pid  3187] mmap(NULL, 4, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7ae29226b000

[pid  3187] mmap(NULL, 24, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7ae292231000

[pid  3187] write(2, "27.900", 627.900)       = 6

[pid  3187] write(4, "27.900", 6)       = 6

[pid  3187] write(2, " / 1.250", 8 / 1.250)     = 8

[pid  3187] write(4, " / 1.250", 8)     = 8

[pid  3187] write(2, " / 3.570", 8 / 3.570)     = 8

[pid  3187] write(4, " / 3.570", 8)     = 8

[pid  3187] write(2, " / -4.840", 9 / -4.840)    = 9

[pid  3187] write(4, " / -4.840", 9)    = 9

[pid  3187] write(2, " / 0.020", 8 / 0.020)     = 8

[pid  3187] write(4, " / 0.020", 8)     = 8

[pid  3187] write(2, ") is -64.588\n", 13) is -64.588

) = 13

[pid  3187] write(4, ") is -64.588\n", 13) = 13

[pid  3187] munmap(0x7ae292231000, 24)  = 0

[pid  3187] unlink("/dev/shm/my\_shared\_memory") = 0

[pid  3187] exit\_group(0)               = ?

[pid  3187] +++ exited with 0 +++

<... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 3187

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=3187, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

exit\_group(0)                           = ?

+++ exited with 0 +++

**Вывод**

**В ходе лабораторной работы я научилась создавать процессы, каналы и другие конструкции. Таким образом, программа иллюстрирует использование базовых системных вызовов для создания процессов и организации их работы с файлами и межпроцессным взаимодействием с помощью каналов и разделяемой памяти.**