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»**

Группа: М8О-211Б-23

Студент: Акимов К.К.

Преподаватель: Бахарев В.Д.

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

Дата: 30.11.24

Москва, 2024

**Постановка задачи**

**Вариант 6.**

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

**Общий метод и алгоритм решения**

1. ssize\_t write(int \_\_fd, const void \*\_\_buf, size\_t \_\_n);

Записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.

1. exit(int \_\_status);  
   Завершает выполнение программы с указанным статусом. Используется в случае ошибок для завершения работы программы.
2. fopen(const char \*\_\_filename, const char \*\_\_mode);  
   Открывает файл, имя которого указано в строке, на которую указывает filename, и связывает с ним поток, mode указывает на режим работы с файлом.
3. Int pipe(int \*fd);

Создает канал и помещает дескрипторы файла для чтения и записи в fd[0] и fd[1].

1. fork(void);  
   Создает дочерний процесс.
2. Int close(int \_\_fd);   
   Сообщает операционной системе об окончании работы с файловым дескриптором, и закрывает файл (FD).
3. Int dup2(int \_\_fd, int \_\_fd2);   
   Копирует FD в FD2, закрыв FD2 если это требуется.
4. Int execl(const char \*\_\_path, char \*const \*\_\_argv);

Заменяет образ текущего процесса на образ нового процесса, определённого в пути path.

1. ssize\_t read(int \_\_fd, void \*\_\_buf, size\_t \_\_nbytes);

Считывает указанное количество байт из файла (FD) в буфер (BUF).

1. pid\_t wait(int \*\_\_stat\_loc);

Используются для ожидания изменения состояния процесса-потомка вызвавшего процесса и получения информации о потомке, чьё состояние изменилось.

1. fgets(char \*\_\_s, int \_\_n, FILE \*\_\_stream);  
   Читает строку из потока.
2. strtok(char \*\_\_str, const char \*\_\_delim);  
   Разделяет строку на токены.
3. strcspn(const char \*\_\_s, const char \*\_\_reject);  
   Находит позицию первого символа в строке, который не содержится в указанном наборе.

Для выполнения данной лабораторной работы я изучил указанные выше системные вызовы, а также пример выполнения подобного задания.

Программа parent.c на вход в качестве аргумента командной строки получает путь к файлу, в котором записаны числа типа int. С помощью fopen() мы открываем файл с режимом “r”. Затем создаем pipe() для общения дочернего процесса и родительского. С помощью fork() создаем дочерний процесс и дальше по pid процесса определяем кто он, родитель или ребенок. Если процесс ребенок мы закрываем канал для чтения в pipe() и зная файловый дескриптор файла, полученный с помощь fileno(), переопределяем поток ввода на открытый файл, а поток вывода на канал для записи в pipe. Дальше используя execl() заменяем образ текущего процесса на образ нового процесса, определённого в пути path, который является файлом child.c и потом закрываем наш file.

Программа child.c начинает считывать строки из файла ( который является переопределенным потоком ввода) с помощью fgets() пока мы не достигнем конца файла. Дальше так как файл находится на Windows, а программа запускается через терминал Ubuntu нам необходимо убрать символ “\r” и символ “\n” для корректной работы strtok, которая разбивает строку на токены по указанному разделителю. Как только будет достигнут конец строки будет возвращено NULL. А пока мы не дошли до конца файла мы будем брать каждое число в строковом представлении и проверяя, что это действительно число типа int, превращаем его из строки в число и прибавляем к сумме строки. Если получили не число просто выводим сообщение о том , что в строке n имеется невалидное значение. Это мы проверяем с помощью переменой valid\_line. Как только дошли до конца файла проверяем, что цикл прервался не из-за ошибки, а из-за конца файла.

Если pid > 0 значит мы в родителе. Мы закрываем канал для записи в pipe. И затем считывая из pipe результаты программы child.c с помощью read() выводим их в терминал. Потом закрываем файл и с помощью wait() дожидаемся окончания выполнения дочернего процесса, параллельно проверяя, что процесс завершился без ошибок с помощью статуса завершения процесса и макроса WIFEXITED.

Каждый системный вызов проверяется на корректность выполнения и в случае ошибки происходит обработка соответствующего случая.

**Код программы**

**parent.c:**

#include <stdio.h>  
#include <unistd.h>  
#include <sys/wait.h>  
#include <stdlib.h>  
#include <string.h>  
  
  
int main(int argc, char \*argv[]){  
 if(argc != 2){  
 write(STDERR\_FILENO, "Error, there should be 2 arguments here\n", 40);

exit(EXIT\_FAILURE);  
  
 }  
  
 FILE \*file = fopen(argv[1], "r");  
 if(!file){  
 write(STDERR\_FILENO, "Error with open file\n", 21);  
 exit(EXIT\_FAILURE);  
 }  
  
 int channel[2];  
 if(pipe(channel) == -1){  
 fclose(file);  
 write(STDERR\_FILENO, "Error with create pipe\n", 24);  
 exit(EXIT\_FAILURE);  
 }  
  
 pid\_t pid = fork();  
  
 if(pid == 0){  
  
 close(channel[0]); // close read  
 int file\_fd = fileno(file);  
 if(dup2(file\_fd, STDIN\_FILENO) == -1){

write(STDERR\_FILENO, "Error redirecting stdin\n", 24);  
 exit(EXIT\_FAILURE);  
 }  
  
  
 if(dup2(channel[1], STDOUT\_FILENO) == -1){  
 write(STDERR\_FILENO, "Error redirecting stdout\n", 25);  
 exit(EXIT\_FAILURE);  
 }  
 close(channel[1]); // close write  
  
 if(execl("./child", "", NULL) == -1){  
 write(STDERR\_FILENO, "Error execl\n", 12);  
 exit(EXIT\_FAILURE);  
 }  
 fclose(file);  
 }  
 else if(pid > 0){  
  
 close(channel[1]); // close write  
 ssize\_t bytes\_read;  
  
 char buf[BUFSIZ];  
 while((bytes\_read = read(channel[0], buf, sizeof(buf))) > 0){  
 write(STDOUT\_FILENO, buf, bytes\_read);  
  
 }  
 // Check  
 if (bytes\_read == -1) {  
 write(STDERR\_FILENO, "Error reading from pipe\n", 24);  
 exit(EXIT\_FAILURE);  
 write(STDOUT\_FILENO, buf, bytes\_read);  
 }  
  
 close(channel[0]); // close read  
 fclose(file);  
  
  
 // Ждем завершения дочернего процесса  
 int status;  
 if (wait(&status) == -1) {  
 write(STDERR\_FILENO, "Error waiting for child process\n", 32);  
 exit(EXIT\_FAILURE);  
 } else if (!(WIFEXITED(status))) {  
 write(STDERR\_FILENO, "Child process terminated with error\n", 36);  
  
 }  
 }  
 else{  
 fclose(file);  
 close(channel[0]);  
 close(channel[1]);  
 write(STDERR\_FILENO, "Error with fork\n", 16);  
 exit(EXIT\_FAILURE);  
  
 }  
 return 0;  
}

**child.c:**

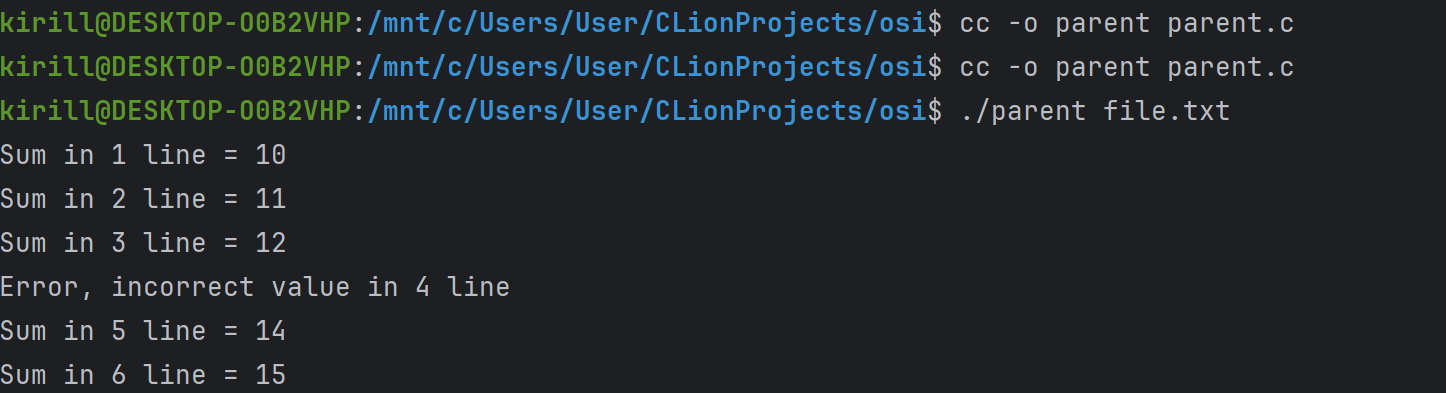
#include <unistd.h>  
#include <string.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <limits.h>  
  
enum Errors{  
 *OK*,  
 *ERROR*  
};  
  
enum Errors Str\_to\_int(char \*str, int \*answer, int line){  
 char \*endptr;  
 long number = strtol(str, &endptr, 10);  
  
 if (number == LONG\_MAX || number == LONG\_MIN){  
 char bufferic[BUFSIZ];  
 int length = snprintf(bufferic, sizeof(bufferic), "Error, incorrect value in %d line\n", line);  
 write(STDOUT\_FILENO, bufferic, length);  
 return *ERROR*;  
 }  
 else if (\*endptr != '\0' || number > INT\_MAX || number < INT\_MIN) {  
 char bufferic[BUFSIZ];  
 int length = snprintf(bufferic, sizeof(bufferic), "Error, incorrect value in %d line\n", line);  
 write(STDOUT\_FILENO, bufferic, length);  
 return *ERROR*;  
 }  
 \*answer = number;  
 return *OK*;  
  
}

int main(){  
  
 char buffer[BUFSIZ];  
 int line\_sum = 0, line = 1;  
  
 while(fgets(buffer, sizeof(buffer), stdin) != NULL){  
  
  
 int valid\_line = 1;  
 buffer[strcspn(buffer, "\n")] = '\0';  
 buffer[strcspn(buffer, "\r")] = '\0';  
  
 char \*token = strtok(buffer, " ");  
  
  
 while (token != NULL) {  
 int num;  
 if (Str\_to\_int(token, &num, line) != *OK*) {  
 valid\_line = 0;  
  
 line++;  
 line\_sum = 0;  
 break;  
  
 } else {  
 line\_sum += num;  
 }  
 token = strtok(NULL, " ");  
 }  
  
 if (valid\_line) {  
 char buf[BUFSIZ];  
 int length = snprintf(buf, sizeof(buf), "Sum in %d line = %d\n", line, line\_sum);  
 ssize\_t bytes\_written = write(STDOUT\_FILENO, buf, length);  
  
 // Check  
 if (bytes\_written == -1) {  
 write(STDERR\_FILENO, "Error writing to stdout\n", 24);  
 } else if (bytes\_written != length) {  
 write(STDERR\_FILENO, "Warning: partial write\n", 23);  
 }  
 line\_sum = 0;  
 line++;  
 }  
  
 }  
 if (ferror(stdin)) {  
 write(STDERR\_FILENO, "Error reading input\n", 20);  
 exit(EXIT\_FAILURE);  
 }  
  
 return 0;  
  
}

**Протокол работы программы**

**Тесты:**

**file.txt:**  
2 3 5  
1 1 9  
12 0 0  
16 -2 -1 Ded\_Makar  
28 -10 -5 1  
12 23 -23 3



**Strace:**

kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/CLionProjects/osi$ strace -f ./parent file.txt

execve("./parent", ["./parent", "file.txt"], 0x7ffcfe6f99f0 /\* 27 vars \*/) = 0

brk(NULL) = 0x559ced059000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f6cfd9ba000

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=20231, ...}) = 0

mmap(NULL, 20231, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f6cfd9b5000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f6cfd7a3000

mmap(0x7f6cfd7cb000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f6cfd7cb000

mmap(0x7f6cfd953000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7f6cfd953000

mmap(0x7f6cfd9a2000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7f6cfd9a2000

mmap(0x7f6cfd9a8000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f6cfd9a8000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f6cfd7a0000

arch\_prctl(ARCH\_SET\_FS, 0x7f6cfd7a0740) = 0

set\_tid\_address(0x7f6cfd7a0a10) = 1066

set\_robust\_list(0x7f6cfd7a0a20, 24) = 0

rseq(0x7f6cfd7a1060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f6cfd9a2000, 16384, PROT\_READ) = 0

mprotect(0x559ceba9c000, 4096, PROT\_READ) = 0

mprotect(0x7f6cfd9f2000, 8192, PROT\_READ) = 0

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

munmap(0x7f6cfd9b5000, 20231) = 0

getrandom("\x90\x62\x2f\x8f\x23\xba\xf7\x23", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x559ced059000

brk(0x559ced07a000) = 0x559ced07a000

openat(AT\_FDCWD, "file.txt", O\_RDONLY) = 3

pipe2([4, 5], 0) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 1067 attached

, child\_tidptr=0x7f6cfd7a0a10) = 1067

[pid 1067] set\_robust\_list(0x7f6cfd7a0a20, 24 <unfinished ...>

[pid 1066] close(5 <unfinished ...>

[pid 1067] <... set\_robust\_list resumed>) = 0

[pid 1066] <... close resumed>) = 0

[pid 1066] read(4, <unfinished ...>

[pid 1067] close(4) = 0

[pid 1067] dup2(3, 0) = 0

[pid 1067] dup2(5, 1) = 1

[pid 1067] close(5) = 0

[pid 1067] execve("./child", [""], 0x7ffedce36bc0 /\* 27 vars \*/) = 0

[pid 1067] brk(NULL) = 0x55e251377000

[pid 1067] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fe03929e000

[pid 1067] access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

[pid 1067] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 4

[pid 1067] fstat(4, {st\_mode=S\_IFREG|0644, st\_size=20231, ...}) = 0

[pid 1067] mmap(NULL, 20231, PROT\_READ, MAP\_PRIVATE, 4, 0) = 0x7fe039299000

[pid 1067] close(4) = 0

[pid 1067] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 4

[pid 1067] read(4, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

[pid 1067] 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 1067] fstat(4, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 0

[pid 1067] 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 1067] mmap(NULL, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 4, 0) = 0x7fe039087000

[pid 1067] mmap(0x7fe0390af000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x28000) = 0x7fe0390af000

[pid 1067] mmap(0x7fe039237000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x1b0000) = 0x7fe039237000

[pid 1067] mmap(0x7fe039286000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x1fe000) = 0x7fe039286000

[pid 1067] mmap(0x7fe03928c000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fe03928c000

[pid 1067] close(4) = 0

[pid 1067] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fe039084000

[pid 1067] arch\_prctl(ARCH\_SET\_FS, 0x7fe039084740) = 0

[pid 1067] set\_tid\_address(0x7fe039084a10) = 1067

[pid 1067] set\_robust\_list(0x7fe039084a20, 24) = 0

[pid 1067] rseq(0x7fe039085060, 0x20, 0, 0x53053053) = 0

[pid 1067] mprotect(0x7fe039286000, 16384, PROT\_READ) = 0

[pid 1067] mprotect(0x55e250ddc000, 4096, PROT\_READ) = 0

[pid 1067] mprotect(0x7fe0392d6000, 8192, PROT\_READ) = 0

[pid 1067] prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 1067] munmap(0x7fe039299000, 20231) = 0

[pid 1067] fstat(0, {st\_mode=S\_IFREG|0777, st\_size=68, ...}) = 0

[pid 1067] getrandom("\x37\xe9\xd6\xab\x96\xc2\x97\x58", 8, GRND\_NONBLOCK) = 8

[pid 1067] brk(NULL) = 0x55e251377000

[pid 1067] brk(0x55e251398000) = 0x55e251398000

[pid 1067] read(0, "2 3 5\r\n1 1 9\r\n12 0 0\r\n16 -2 -1 D"..., 4096) = 68

[pid 1067] write(1, "Sum in 1 line = 10\n", 19) = 19

[pid 1066] <... read resumed>"Sum in 1 line = 10\n", 8192) = 19

[pid 1067] write(1, "Sum in 2 line = 11\n", 19 <unfinished ...>

[pid 1066] write(1, "Sum in 1 line = 10\n", 19 <unfinished ...>

Sum in 1 line = 10

[pid 1067] <... write resumed>) = 19

[pid 1066] <... write resumed>) = 19

[pid 1067] write(1, "Sum in 3 line = 12\n", 19 <unfinished ...>

[pid 1066] read(4, <unfinished ...>

[pid 1067] <... write resumed>) = 19

[pid 1066] <... read resumed>"Sum in 2 line = 11\nSum in 3 line"..., 8192) = 38

[pid 1067] write(1, "Error, incorrect value in 4 line"..., 33 <unfinished ...>

[pid 1066] write(1, "Sum in 2 line = 11\nSum in 3 line"..., 38 <unfinished ...>

Sum in 2 line = 11

Sum in 3 line = 12

[pid 1067] <... write resumed>) = 33

[pid 1066] <... write resumed>) = 38

[pid 1067] write(1, "Sum in 5 line = 14\n", 19 <unfinished ...>

[pid 1066] read(4, <unfinished ...>

[pid 1067] <... write resumed>) = 19

[pid 1066] <... read resumed>"Error, incorrect value in 4 line"..., 8192) = 52

[pid 1067] write(1, "Sum in 6 line = 15\n", 19 <unfinished ...>

[pid 1066] write(1, "Error, incorrect value in 4 line"..., 52 <unfinished ...>

Error, incorrect value in 4 line

Sum in 5 line = 14

[pid 1067] <... write resumed>) = 19

[pid 1066] <... write resumed>) = 52

[pid 1066] read(4, <unfinished ...>

[pid 1067] read(0, <unfinished ...>

[pid 1066] <... read resumed>"Sum in 6 line = 15\n", 8192) = 19

[pid 1066] write(1, "Sum in 6 line = 15\n", 19Sum in 6 line = 15

) = 19

[pid 1066] read(4, <unfinished ...>

[pid 1067] <... read resumed>"", 4096) = 0

[pid 1067] exit\_group(0) = ?

[pid 1066] <... read resumed>"", 8192) = 0

[pid 1066] close(4) = 0

[pid 1066] close(3 <unfinished ...>

[pid 1067] +++ exited with 0 +++

<... close resumed>) = 0

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=1067, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 1067

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

Входе написания данной лабораторной работы я научился работать с системными вызовами в СИ. Научился создавать программы, состоящие из нескольких процессов, и передавать данные между процессами по каналу. Во время отладки программы я познакомился с утилитой strace. Лабораторная работа была довольно интересна, так как я раньше не создавал программы на СИ, которые запускают несколько процессов параллельно.