Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-213Б-23

Студент: Марьин Д. А.

Преподаватель: Бахарев В.Д.

Оценка:

_____

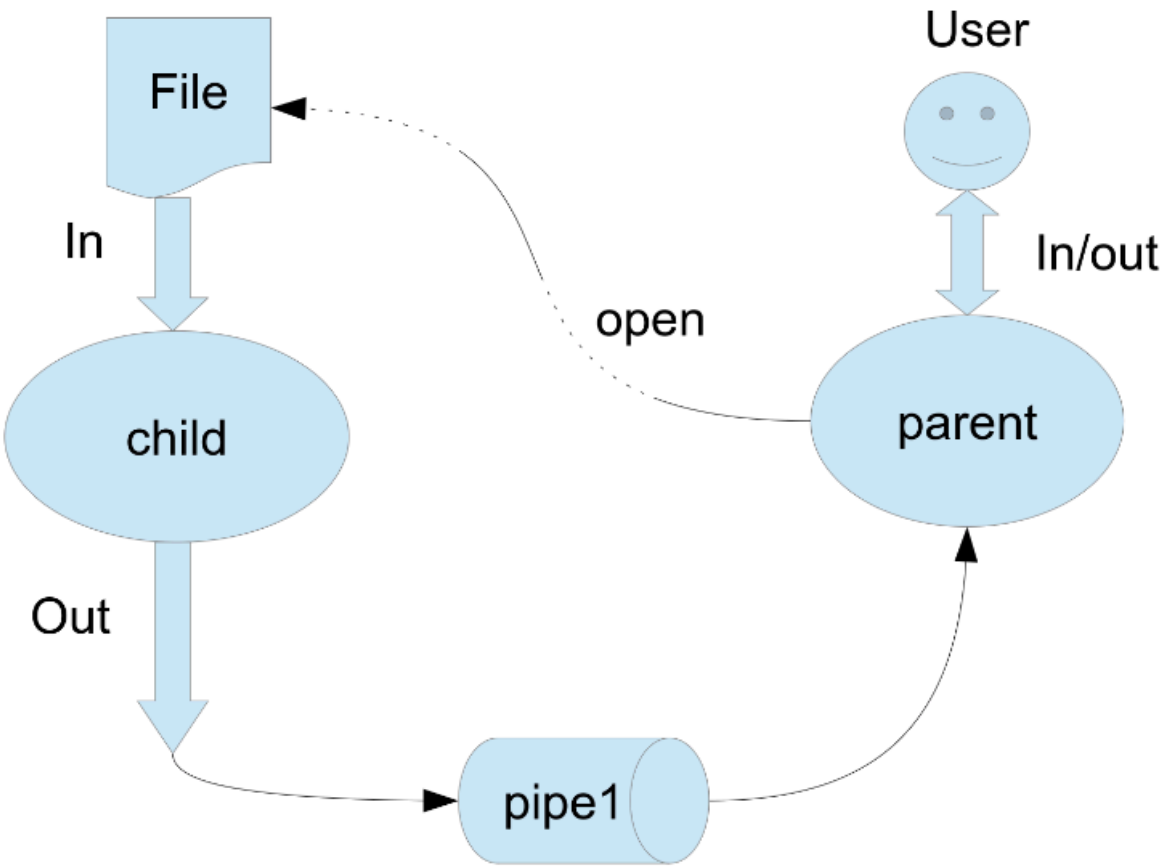
Дата: 25.11.24

Москва, 2024

Постановка задачи

Вариант 9.

Группа вариантов 2



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.

9 вариант) В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. 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.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork() - создание дочернего процесса
- int execve(const char *filename, char *const argv[], char *const envp[]) (и другие вариации exec) - замена образа памяти процесса
- pid_t waitpid(pid_t pid, int *status, int options) - Ожидание завершения дочернего процесса
- void exit(int status) - завершения выполнения процесса и возвращение статуса
- int shm_open(const char *name, int oflag, mode_t mode) – создание/открытие именованной области разделенной памяти
- int shm_unlink(const char *name) – уничтожение именованной области памяти
- int ftruncate(int fd, off_t length) – устанавливает размер файла
- mmap
- sem_t *sem_open(const char *, int, ...) – создание/открытие именованного семефора
- int sem_wait(sem_t *) – уменьшить семафор (ожидание если 0)
- int sem_post(sem_t *) – увеличить семафор
- int sem_unlink(const char *) – уничтожить именованный семафор
- int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
- int open(const char *pathname, int flags, mode_t mode) – открытие/создание файла
- int close(int fd) - закрыть файл

Программа считывает из стандартного ввода имя файла, создает shared memory и 2 семафора: один для дочернего потока, второй для родительского. Shared memory устанавливается размер достаточный для хранения 2 значений типа int: одно - передаваемое число, второе - флаг ошибки . Далее порождается дочерний процесс, у которого с помощью функции dup2 стандартный ввод перенаправлен на открытый файл. Образ дочернего процесса заменяется на child.out с помощью функции execl.

Дочерний процесс читает и валидирует числа, затем записывает результат в shared memory, организуя работу с данными с помощью семафора.

Результат работы read_line_of_int записывается по индексу 0 в shared memory, пока функция не вернёт 0. После этого по индексу 1 записывается 0 и процесс завершается.

Код программы

parent.c

```
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <semaphore.h>
#include <string.h>

#define SHM_NAME "/shared_memory"
#define SEM_CHILD "/sem_child"
#define SEM_PARENT "/sem_parent"
#define BUFSIZE 1024

int reverse(char* str, int length) {
int start = 0;
int end = length - 1;
while (start < end) {
char temp = str[start];
str[start] = str[end];
str[end] = temp;
start++;
end--;
}
return 0;
}

float my_pow(float base, int exp) {
float result = 1.0;
int is_negative = 0;

if (exp < 0) {
exp = -exp;
is_negative = 1;
}

while (exp > 0) {
if (exp % 2 == 1) {
result *= base;
}
base *= base;
exp /= 2;
}
```

```
if (is_negative) {
result = 1.0 / result;
}
return result;
}

int int_to_str(int num, char* str, int d) {
int i = 0;
int is_negative = 1 ? num < 0: 0;
if (is_negative) {
num *= -1;
}
if (num == 0) {
str[i++] = '0';
} else {
while (num != 0) {
str[i++] = (num % 10) + '0';
num /= 10;
}
}

while (i < d) {
str[i++] = '0'; // добавляем нули, если нужно
}

if (is_negative) {
str[i++] = '-';
}
reverse(str, i);
str[i] = '\0';
return i;
}

int float_to_str(float n, char* res, int afterpoint) {
int ipart = (int)n;

float fpart = n - (float)ipart;
int i = int_to_str(ipart, res, 0);

if (afterpoint != 0) {
res[i] = '.';
i++;

fpart = fpart * my_pow(10, afterpoint);

int_to_str((int)fpart, res + i, afterpoint);
}
return 0;
}

int main() {
```

```
const char start_msg[] = "Type name of your file: ";
write(STDOUT_FILENO, start_msg, sizeof(start_msg));

char filename[BUFSIZE];
int n = read(STDIN_FILENO, filename, sizeof(filename) - 1);

if (n > 0 && n < sizeof(filename)) {
filename[n - 1] = '\0';
} else {
filename[sizeof(filename) - 1] = '\0';
}

int shm = shm_open(SHM_NAME, O_RDWR | O_CREAT, 0666);
if (shm == -1) {
char* msg = "ERROR: fail to create shared memory\n";
write(STDOUT_FILENO, msg, strlen(msg));
exit(-1);
}


if (ftruncate(shm, sizeof(char) * 32) == -1) {
char* msg = "ERROR: fail to set size of shared memory\n";
write(STDOUT_FILENO, msg, strlen(msg));
exit(-1);
}

sem_t* sem_child = sem_open(SEM_CHILD, O_CREAT, 0666, 0);
if (sem_child == SEM_FAILED) {
char* msg = "ERROR: fail to open semaphore\n";
write(STDOUT_FILENO, msg, strlen(msg));
exit(-1);
}

sem_t* sem_parent = sem_open(SEM_PARENT, O_CREAT, 0666, 0);
if (sem_parent == SEM_FAILED) {
char* msg = "fail to open semaphore\n";
write(STDOUT_FILENO, msg, strlen(msg));
exit(-1);
}

int fd[2];

pid_t pid = fork();

float* res = (float*)mmap(0, sizeof(float) * 2, PROT_READ|PROT_WRITE, MAP_SHARED, shm,
0);
switch (pid)
{
case -1:
const char msg[] = "ERROR: new process has not been created\n";
write(STDOUT_FILENO, msg, sizeof(msg));
exit(EXIT_FAILURE);
```

```
break;
case 0:

execlp("./child", "./child", filename, (char*)NULL);

const char exec_msg[] = "ERROR: process has not started\n";
write(STDERR_FILENO, exec_msg, sizeof(exec_msg));
exit(EXIT_FAILURE);
break;
default:

sem_post(sem_parent);
sem_wait(sem_child);

if (res[1] != -1.) {
char str[BUFSIZE];
float_to_str(*res, str, 5);
size_t size_msg = strlen(str);
str[size_msg] = '\n';
write(STDOUT_FILENO, str, ++size_msg);
}

break;
}

shm_unlink(SHM_NAME);
sem_unlink(SEM_CHILD);
sem_unlink(SEM_PARENT);
munmap(res, sizeof(float) * 2);
sem_close(sem_child);
sem_close(sem_parent);

return 0;
}
```

child.c

```
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <semaphore.h>

#define SHM_NAME "/shared_memory"
#define SEM_CHILD "/sem_child"
#define SEM_PARENT "/sem_parent"
#define BUFSIZE 1024

int is_separator(const char c) {
return (c == ' ') || (c == '\n') || (c == '\t') || (c == '\0');
```

```
}

int validate_float_number(const char* str) {
int len_str = strlen(str);
if (len_str == 0) {
return 0;
}
int num = 0;
int was_sep = 0;
for (int i = 0; i < len_str; ++i) {
if (str[i] == '-') {
if (i == 0) {
continue;
} else {
return 0;
}
}
if (str[i] == '.' || str[i] == ','){
if (was_sep) {
return 0;
}
was_sep = 1;
} else if ((('0' > str[i] || str[i] > '9') && str[i] != '.' && str[i] != ',' ) ||
is_separator(str[i])) {
return 0;
}
}
return 1;
}

int main(int argc, char* argv[]) {
int shm = shm_open(SHM_NAME, O_RDWR, 0666);

sem_t* sem_child = sem_open(SEM_CHILD, O_EXCL);
if (sem_child == SEM_FAILED) {
char* msg = "fail to open semaphore\n";
write(STDOUT_FILENO, msg, strlen(msg));
exit(-1);
}

sem_t* sem_parent = sem_open(SEM_PARENT, O_EXCL);
if (sem_parent == SEM_FAILED) {
char* msg = "fail to open semaphore\n";
write(STDOUT_FILENO, msg, strlen(msg));
exit(-1);
}

if (argc != 2) {
const char msg[] = "Usage: ./child <filename>\n";
write(STDERR_FILENO, msg, sizeof(msg));
exit(EXIT_FAILURE);
```

```
}

int file = open(argv[1], O_RDONLY);

if (file == -1) {
const char msg[] = "ERROR: wrong file\n";
write(STDOUT_FILENO, msg, sizeof(msg));
exit(EXIT_FAILURE);
}

dup2(file, STDIN_FILENO);
close(file);


char buffer[BUFSIZE];
char msg[BUFSIZE];
char ch[BUFSIZE];
int index = 0;
float current_number = 0;
float first_number = 0;
int is_first_number = 1;
int n = 0;

float* res = (float*)mmap(0, sizeof(float) * 2, PROT_READ|PROT_WRITE, MAP_SHARED, shm,
0);

while ((n = read(STDIN_FILENO, ch, sizeof(char))) > 0)
{
if (is_separator(*ch)) {
buffer[index] = '\0';
if (strlen(buffer) == 0) {
continue;
}
if (validate_float_number(buffer)) {
current_number = atof(buffer);
} else {
sem_wait(sem_parent);

res[1] = -1.;
res[0] = 0.;
const char msg[] = "ERROR: wrong number\n";
write(STDOUT_FILENO, msg, sizeof(msg));

sem_post(sem_child);
exit(EXIT_FAILURE);
}
if (is_first_number) {
first_number = current_number;
is_first_number = 0;
} else {
if (current_number != 0) {
first_number /= current_number;
```

```
} else {
sem_wait(sem_parent);

res[0] = 0.;
res[1] = -1.;
const char msg[] = "ERROR: division by zero\n";
write(STDOUT_FILENO, msg, sizeof(msg));

sem_post(sem_child);
exit(EXIT_FAILURE);
}
}
index = 0;
} else {
buffer[index++] = *ch;
}
}
// Обработка последнего числа в файле, если не было разделителя
if (index > 0) {
buffer[index] = '\0';
if (validate_float_number(buffer)) {
current_number = atof(buffer);
} else {
sem_wait(sem_parent);

res[1] = -1.;
res[0] = 0.;
const char msg[] = "ERROR: wrong number\n";
write(STDOUT_FILENO, msg, sizeof(msg));
sem_post(sem_child);

exit(EXIT_FAILURE);
}
if (is_first_number) {
first_number = current_number;
} else {
if (current_number != 0) {
first_number /= current_number;
} else {
sem_wait(sem_parent);

res[0] = 0.;
res[1] = -1.;
const char msg[] = "ERROR: division by zero\n";
write(STDOUT_FILENO, msg, sizeof(msg));

sem_post(sem_child);
exit(EXIT_FAILURE);
}
}
}
sem_wait(sem_parent);
```

```
res[0] = first_number;
sem_post(sem_child);

close(STDIN_FILENO);
munmap(res, sizeof(float) * 2);
sem_close(sem_child);
sem_close(sem_parent);

return 0;
}
```

Протокол работы программы

$ Type name of your file: input.txt
20.00000

```
1 -2 -3
20 -14
1 2 3 4 5
$ cat > run.txt
a.txt
$ ./main.out < run.txt
3
0
-4
6
15
```

Strace:

```
execve("./a.out", ["./a.out"], 0x7fff39388b30 /* 81 vars */) = 0
brk(NULL)                               = 0x651c87087000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x78914b843000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/usr/local/cuda-12.6/lib64/glibc-hwcaps/x86-64-v3/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (Нет такого файла или каталога)
newfstatat(AT_FDCWD, "/usr/local/cuda-12.6/lib64/glibc-hwcaps/x86-64-v3/", 0x7ffd509aaff0, 0) = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/usr/local/cuda-12.6/lib64/glibc-hwcaps/x86-64-v2/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (Нет такого файла или каталога)
newfstatat(AT_FDCWD, "/usr/local/cuda-12.6/lib64/glibc-hwcaps/x86-64-v2/", 0x7ffd509aaff0, 0) = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/usr/local/cuda-12.6/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (Нет такого файла или каталога)
newfstatat(AT_FDCWD, "/usr/local/cuda-12.6/lib64/", {st_mode=S_IFDIR|0755, st_size=4096, ...}, 0) = 0
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=79571, ...}) = 0
mmap(NULL, 79571, PROT_READ, MAP_PRIVATE, 3, 0) = 0x78914b82f000
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) = 0x78914b600000
mmap(0x78914b628000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x78914b628000
mmap(0x78914b7b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x78914b7b0000
mmap(0x78914b7ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x78914b7ff000
mmap(0x78914b805000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x78914b805000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x78914b82c000
arch_prctl(ARCH_SET_FS, 0x78914b82c740) = 0
set_tid_address(0x78914b82ca10)         = 28327
set_robust_list(0x78914b82ca20, 24)     = 0
rseq(0x78914b82d060, 0x20, 0, 0x53053053) = 0
mprotect(0x78914b7ff000, 16384, PROT_READ) = 0
mprotect(0x651c85fdb000, 4096, PROT_READ) = 0
mprotect(0x78914b87b000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x78914b82f000, 79571)           = 0
write(1, "Type name of your file: \0", 25Type name of your file: ) = 25
read(0, input.txt
"input.txt\n", 1023)            = 10
openat(AT_FDCWD, "/dev/shm/shared_memory", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 32)                        = 0
openat(AT_FDCWD, "/dev/shm/sem.sem_child", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (Нет такого файла или каталога)
getrandom("\xef\xe4\xf9\x7f\x0f\xb9\x08\x0f", 8, GRND_NONBLOCK) = 8
```

```
newfstatat(AT_FDCWD, "/dev/shm/sem.dEirTW", 0x7ffd509ab210, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/dev/shm/sem.dEirTW", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 4
write(4, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x78914b842000
link("/dev/shm/sem.dEirTW", "/dev/shm/sem.sem_child") = 0
fstat(4, {st_mode=S_IFREG|0664, st_size=32, ...}) = 0
getrandom("\x8e\xbd\x11\xd2\xc9\x14\xb5\xa1", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x651c87087000
brk(0x651c870a8000)                     = 0x651c870a8000
unlink("/dev/shm/sem.dEirTW")           = 0
close(4)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.sem_parent", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (Нет такого файла или каталога)
getrandom("\xe7\x0d\x96\x00\x1d\x96\x99\x86", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.LhOKLo", 0x7ffd509ab210, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/dev/shm/sem.LhOKLo", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 4
write(4, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x78914b841000
link("/dev/shm/sem.LhOKLo", "/dev/shm/sem.sem_parent") = 0
fstat(4, {st_mode=S_IFREG|0664, st_size=32, ...}) = 0
unlink("/dev/shm/sem.LhOKLo")           = 0
close(4)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x78914b82ca10) = 28563
mmap(NULL, 8, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x78914b840000
futex(0x78914b842000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY) = 0
write(1, "20.00000\n", 920.00000
)             = 9
unlink("/dev/shm/shared_memory")        = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=28563, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
unlink("/dev/shm/sem.sem_child")        = 0
unlink("/dev/shm/sem.sem_parent")       = 0
munmap(0x78914b840000, 8)               = 0
munmap(0x78914b842000, 32)              = 0
munmap(0x78914b841000, 32)              = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Разработка данной программы основывалась на глубоком понимании принципов взаимодействия родительского и дочернего процессов в контексте языка программирования C. Ключевым аспектом стало создание надежного механизма обмена данными, реализованного с применением разделяемой памяти и семафоров. Особое внимание было уделено разработке эффективных функций, предназначенных для манипулирования целочисленными данными.

Результатом проделанной работы стала программа, которая наглядно демонстрирует высокий уровень владения ключевыми концепциями межпроцессного взаимодействия и обработки данных в рамках системного программирования на языке C. Программа является свидетельством глубокого понимания принципов параллельного выполнения процессов, синхронизации доступа к общим ресурсам и обеспечения целостности данных при многопоточном взаимодействии. Она также подчеркивает умение разработчика создавать эффективные и надежные программные решения, способные корректно функционировать в сложных системных средах.