1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и кибербезопасности
5. **Кафедра «Высшая школа кибербезопасности»**

**ЛАБОРАТОРНАЯ РАБОТА № 5**

1. «СИСТЕМНЫЕ ВЫЗОВЫ»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 5131001/30003 Шевчук Н.Е.

<*подпись*>

Преподаватель Огнёв Р.А.

<*подпись*>

1. Санкт-Петербург
2. 2025

**Оглавление**

[1 Цель работы 3](#_Toc187083112)

[2 Ход работы 3](#_Toc187083113)

[2.1 Блок-схема реализованного механизма обработки системных вызово 3](#_Toc187083114)

[2.2 Изменения в thread.h 3](#_Toc187083115)

[2.3 Изменения в process.c 4](#_Toc187083116)

[2.4 Изменения в syscall.c 5](#_Toc187083117)

[2.5 Таблица результатов анализа тестовых ситуаций 8](#_Toc187083118)

[2.6 Результаты тестов 11](#_Toc187083119)

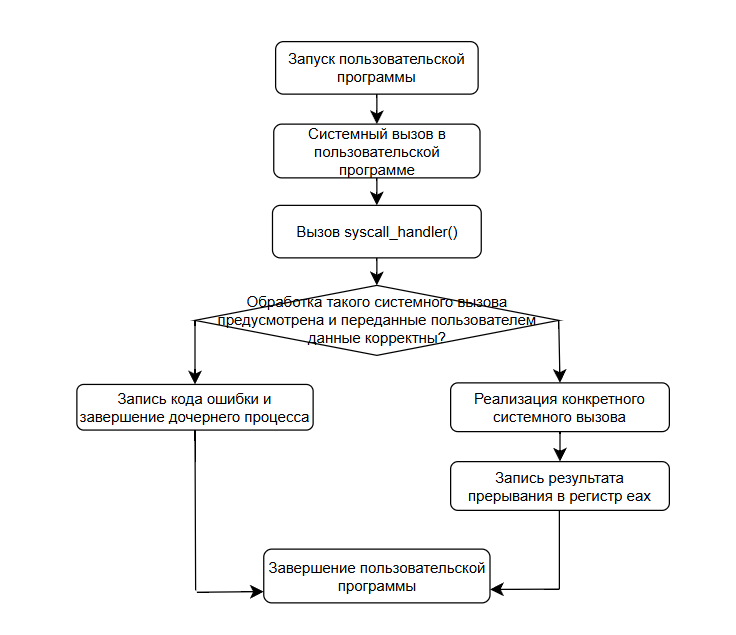
[3 Вывод 17](#_Toc187083120)

# Цель работы

Изучить механизм внутренних прерываний и системных вызовов, модифицировать обработчик системных вызовов и внедрить набор системных вызовов в учебную операционную систему Pintos.

# Ход работы

## Блок-схема реализованного механизма обработки системных вызово



## Изменения в thread.h

В струтуру thread были добавлены следующие поля (инициализированы, соответственно в функции init\_thread() директории thread.c):

struct thread

{

...

int exit\_status; //код завершения

struct list child\_list; //список дочерних потоков

struct list file\_list; //список файлов

struct semaphore sema; //семафор, сигнализирующий о завершении дочернего потока

struct list\_elem child\_elem; //элемент в списке дочрних потоков

int cur\_fd; //текущий дескриптор файла

... };

Добавлена структура thread\_info для использования функции thread\_foreach(), а также структура files, для сохранения информации о файлах дочерних потоков.

struct thread\_info {

tid\_t tid;

struct thread \*thread;

};

struct files {

struct list\_elem file\_elem;

struct file\* info\_file;

int info\_fd;

};

## Изменения в process.c

1. В функции process\_execute проверяется найден ли нужный файл (т.е. из командной строки извлекается имя файла). А также созданный поток добавляется в список дочерних потоков.

tid\_t

process\_execute (const char \*file\_name)

{

char \*fn\_copy;

tid\_t tid;

char \*name\_copy = palloc\_get\_page(0);

if (name\_copy == NULL)

return TID\_ERROR;

strlcpy(name\_copy, file\_name, PGSIZE);

char \*save\_ptr;

char \*name = strtok\_r(name\_copy, " ", &save\_ptr);

if (name == NULL) {

palloc\_free\_page(name\_copy);

return TID\_ERROR;

}

struct file \*file = filesys\_open(name);

if (file == NULL) {

palloc\_free\_page(name\_copy);

return -1;

}

file\_close(file);

/\* Make a copy of FILE\_NAME.

Otherwise there's a race between the caller and load(). \*/

fn\_copy = palloc\_get\_page (0);

if (fn\_copy == NULL)

return TID\_ERROR;

strlcpy (fn\_copy, file\_name, PGSIZE);

/\* Create a new thread to execute FILE\_NAME. \*/

tid = thread\_create (name, PRI\_DEFAULT, start\_process, fn\_copy);

if (tid == TID\_ERROR)

palloc\_free\_page (fn\_copy);

info.tid=tid;

info.thread = NULL;

enum intr\_level old\_level = intr\_disable();

thread\_foreach(\*get\_thread\_by\_tid, &info);

intr\_set\_level(old\_level);

list\_push\_front(&thread\_current()->child\_list, &info.thread->child\_elem);

return tid;

}

1. В функции process\_wait() проверяется, что у процесса есть дочерние потоки и необходимый дочерний поток ищется по своему tid. Также в ней опускается семафор, предназначенный для того, чтобы родительский процесс ожидал завершение дочернего (опускается, соответственно, в функции process\_exit()).

int

process\_wait (tid\_t child\_tid)

{

if (list\_empty(&thread\_current()->child\_list))

return -1;

info.tid=child\_tid;

info.thread = NULL;

enum intr\_level old\_level = intr\_disable();

thread\_foreach(\*get\_thread\_by\_tid, &info);

intr\_set\_level(old\_level);

if (info.thread == NULL)

return -1;

list\_remove(&info.thread->child\_elem);

sema\_down(&info.thread->sema);

return exit\_status;

}

## Изменения в syscall.c

1. Добавлена функция validate\_user\_pointer(), проверяющая корректность адреса, переданного пользовательской программой.

void validate\_user\_pointer(const void\* uaddr) {

if (!is\_user\_vaddr(uaddr) || !pagedir\_get\_page(thread\_current()->pagedir, uaddr)) {

thread\_current()->exit\_status = -1;

thread\_exit();

}

}

1. Добавлена функция validate\_user\_page(), проверяющая корректность адреса переданного пользователем (что страница, соответствующая указателю загружена в память), а также обновляет значение аргумента на физический адрес страницы.

void validate\_user\_page (int\* args) {

void\* phys\_page\_ptr = (void \*)pagedir\_get\_page(thread\_current()->pagedir, (const void\*)args[0]);

if (!phys\_page\_ptr) {

thread\_current()->exit\_status = -1;

thread\_exit();

}

args[0] = (int)phys\_page\_ptr;

}

1. В функции syscall\_handler() реализован требуемый набор системных вызовов. Корректность переданных пользователем адресов проверяется сразу при вызове функции.

static void

syscall\_handler (struct intr\_frame \*f)

{

validate\_user\_pointer((const void\*)f->esp);

int nr = \*(int\*)f->esp;

int \*ptr1 = (int\*)f->esp+1;

validate\_user\_pointer(ptr1);

int \*ptr2 = (int\*)f->esp+2;

validate\_user\_pointer(ptr2);

int \*ptr3 = (int\*)f->esp+3;

validate\_user\_pointer(ptr3);

int args[3] = {\*ptr1, \*ptr2, \*ptr3};

if (nr == SYS\_HALT)

shutdown\_power\_off();

else if (nr == SYS\_EXIT) {

int exit\_status = ((size\_t\*)f->esp)[1];

thread\_current()->exit\_status = exit\_status;

thread\_exit();

}

else if (nr == SYS\_EXEC) {

validate\_user\_page(&args[0]);

tid\_t child = process\_execute((const char\*)args[0]);

f->eax = child;

}

else if (nr == SYS\_WAIT) {

f->eax = process\_wait((tid\_t)args[0]);

}

else if (nr == SYS\_CREATE) {

validate\_user\_page(&args[0]);

f->eax = filesys\_create((const char\*)args[0], (unsigned int)args[1]);

}

else if (nr == SYS\_REMOVE) {

validate\_user\_page(&args[0]);

f->eax = filesys\_remove((const char\*)args[0]);

}

else if (nr == SYS\_OPEN) {

validate\_user\_page(&args[0]);

struct file\* file = filesys\_open((const char\*)args[0]);

if (file == NULL) {

f->eax=-1;

return;

}

else {

struct files \*new\_file = (struct files\*)malloc(sizeof(struct files));

new\_file->info\_file = file;

new\_file->info\_fd = thread\_current()->cur\_fd++;

list\_push\_back(&thread\_current()->file\_list, &new\_file->file\_elem);

f->eax = new\_file->info\_fd;

}

}

else if (nr == SYS\_CLOSE) {

int fd = args[0];

if (fd == STDOUT\_FILENO) {

thread\_current()->exit\_status = -1;

thread\_exit();

}

struct list\_elem \*e;

for (e = list\_begin(&thread\_current()->file\_list); e != list\_end(&thread\_current()->file\_list); e = list\_next(e)) {

struct files \*file = list\_entry(e, struct files, file\_elem);

if (file->info\_fd == fd) {

file\_close(file->info\_file);

list\_remove(e);

free(file);

f->eax = 0;

return;

}

}

f->eax = -1;

}

else if (nr == SYS\_FILESIZE) {

int fd = args[0];

struct list\_elem \*e;

for (e = list\_begin(&thread\_current()->file\_list); e != list\_end(&thread\_current()->file\_list); e = list\_next(e)) {

struct files \*file = list\_entry(e, struct files, file\_elem);

if (file->info\_fd == fd) {

int size = file\_length(file->info\_file);

f->eax = size;

return;

}

}

f->eax = -1;

}

else if (nr == SYS\_READ) {

int fd = args[0];

void \*buffer = (void \*)args[1];

void \*buffer\_copy = (void \*)args[1];

unsigned int size = (unsigned int)args[2];

for (unsigned int i = 0; i < size; i++) {

validate\_user\_pointer((const void\*)buffer\_copy);

buffer\_copy++;

}

validate\_user\_page(&args[1]);

int bytes\_read = -1;

if (fd == STDIN\_FILENO)

bytes\_read = input\_getc();

else {

struct list\_elem \*e;

for (e = list\_begin(&thread\_current()->file\_list); e != list\_end(&thread\_current()->file\_list); e = list\_next(e)) {

struct files \*file = list\_entry(e, struct files, file\_elem);

if (file->info\_fd == fd) {

bytes\_read = file\_read(file->info\_file, buffer, size);

break;

}

}

}

f->eax = bytes\_read;

}

else if (nr == SYS\_WRITE) {

int fd = args[0];

validate\_user\_page(&args[1]);

void \*buffer = (void \*)args[1];

unsigned int size = (unsigned int)args[2];

int bytes\_written = -1;

if (fd == STDOUT\_FILENO) {

putbuf(buffer, size);

bytes\_written = (int)size;

} else {

struct list\_elem \*e;

for (e = list\_begin(&thread\_current()->file\_list); e != list\_end(&thread\_current()->file\_list); e = list\_next(e)) {

struct files \*file = list\_entry(e, struct files, file\_elem);

if (file->info\_fd == fd) {

bytes\_written = file\_write(file->info\_file, buffer, size);

break;

}

}

}

f->eax = bytes\_written;

}

else {

printf("Unknown system call!\n");

thread\_current()->exit\_status = -1;

thread\_exit();

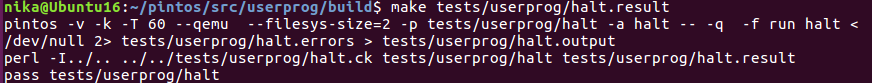
}

}

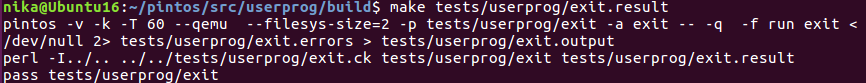
## Таблица результатов анализа тестовых ситуаций

|  |  |  |
| --- | --- | --- |
| Название теста: | Рассматриваемая проблема, описание теста: | Как была решена проблема: |
| halt | Завершение работы ОС Pintos | Использована функция shutdown\_power\_off() |
| exit | Завершение выполнения пользовательской программы | Выполняется с помощью thread\_exit(). Код завершения сохраняется в поле exit\_status структуры thread. |
| exec-once | Пользовательская программа создает и ожидает один дочерний процесс | Выполняется с помощью process\_execute(). В директории process.c добавлена логика работы со списком дочерних процессов: в process\_execute() дочерний процесс добавляется в список всех дочерних процессов родительского потока. В функции process\_wait() дочерний поток находится по его tid, и поднимается семафор для ожидания его завершения (опускается в process\_exit()). |
| exec-multiple | Пользовательская программа создает и ожидает четыре дочерних процесса |
| exec-arg | Пользовательская программа создает и ожидает один дочерний процесс с аргументами | Разбиение аргументов командной строки и их помещение в стек реализовано в предыдущей лабораторной в функции load() директории process.c. |
| exec-missing | Пользовательская программа пытается запустить несуществующую программу | В функции process\_execute() добалвено извлечение имени файла из командной строки и проверка на его корректность. |
| exec-bad-ptr | Пользовательская программа передает некорректный адрес | Проверка на корректность адресов переданных пользователем осуществляется с помощью validate\_user\_pointer() при вызове syscall\_handler(). |
| wait-simple | Ожидание завершения одного дочернего процесса | Выполняется с помощью process\_wait(). Ожидание происходит с помощью семафора sema, добавленного в струткуру thread. |
| wait-twice | Попытка дважды дождаться завершения одного дочернего процесса. Первое ожидание завершается с кодом выхода, второе - ошибкой | В process\_wait() первый дочерний поток удаляется из списка дочерних процессов родительского потока. Поэтому при повторном ожидании список оказывается пуст и программа завершается ошибкой. |
| wait-killed | Ожидание завершения процесса, который уже был завершен аварийно | У аварийно завершенного дочернего процесса будет некорректный адрес. Проверка на корректность адресов переданных пользователем осуществляется с помощью validate\_user\_pointer() при вызове syscall\_handler(). |
| wait-bad-pid | Ожидание завершения дочернего процесса с неверным идентификатором | В функции process\_wait() есть проверка, что список дочерних потоков не пуст. Поток с неверным идентификатором не попадет в этот список. |
| create-empty | Создание файла без имени и нулевым размером | Проверка на корректность адресов переданных пользователем осуществляется с помощью validate\_user\_pointer() при вызове syscall\_handler(). В функции filesys\_create() осуществляется проверка на корректность данных. |
| create-long | Попытка создания файла со слишком длинным именем | В функции filesys\_create() уже реализована необходимая проверка. |
| create-normal | Создание обычного файла | С помощью validate\_user\_page() значению args[0] присваивается физический адрес страницы. Затем файл создается с помощью filesys\_create(). |
| create-exists | Попытка создать файл, который уже существует | Проверка уже существует в функции filesys\_create(). |
| create-null | Попытка создать файл с нулевым указателем в качестве имени | С помощью validate\_user\_page() проверяется, что не передан нулевой указатель. Также проверка осуществляется в filesys\_create(). |
| create-bad-ptr | Передача некорректного адреса в качестве имени файла | Проверка на корректность адресов переданных пользователем осуществляется с помощью validate\_user\_pointer() при вызове syscall\_handler(). |
| create-bound | В качестве имени файла передается строка, находящаяся на разных страницах памяти | Проверка осуществляется с помощью validate\_user\_pointer(). |
| open-missing | Попытка открыть несуществующий файл | С помощью validate\_user\_page() значению args[0] присваивается физический адрес страницы. Затем файл открывается с помощью filesys\_open(). Найденный файл добавляется в список открытых файлов текущего потока. Его дескриптор сохраняется с помощью структуры files. В функции filesys\_open() уже реализована необходимая проверка. |
| open-normal | Открытие обычного файла |
| open-twice | Открытие одного и того же файла два раза. Должно успешно выполниться и вернуть разные дескрипторы. |
| open-boundary | В качестве имени файла передается строка, находящаяся на разных страницах памяти | Проверка осуществляется с помощью validate\_user\_pointer(). |
| open-empty | В качестве имени передается пустая строка | Проверка уже существует в функции filesys\_open(). |
| open-null | В качестве имени нулевой указатель |
| open-bad-ptr | Передача некорректного адреса в качестве имени файла | Проверка осуществляется с помощью validate\_user\_pointer(). |
| close-normal | Открытие и закрытие обычного файла | Файл ищется в списке открытых файлов текущего потока по своему дескриптору, затем закрывается с помощью c file\_close() и удаляется из списка открытых файлов. В функции thread\_exit реализовано закрытие всех открытых файлов потока при его завершении. |
| close-twice | Попытка закрыть файл дважды. Один раз это завершается успехом, а второй с ошибкой | При первом закрытии файл удаляется из списка открытых файлов потока. |
| close-stdin | Передача дескриптора потока ввода | Файла с таким дескриптором не будет в списке открытых файлов. |
| close-stdout | Передача дескриптора потока ввода | Проверяется равен ли дескриптор STDOUT\_FILENO, если да, то поток завершается с кодом -1. |
| close-bad-fd | Передача дескриптора, не соответствующего ни одному из открытых процессом файлов | Такого файла не будет в списке открытых файлов потока, поэтому функция вернет -1. |
| read-normal | Чтение обычного файла | Необходимый файл ищется в списке открытых файлов по своему дескриптору, затем происходит чтение из него с помощью функции filesys\_read(). |
| read-zero | Попытка считать из файла ноль байт | Проверка уже реализована в filesys\_read(). |
| read-bad-ptr | Попытка считать данные в некорректный буфер | Проверка осуществляется с помощью validate\_user\_pointer(). |
| read-boundary | Попытка считать файл в буфер,  который расположен на  разных страницах памяти | Проверка осуществляется с помощью validate\_user\_pointer(). |
| read-stdout | Попытка считать данные из устройства вывода | Файла с таким дескриптором не будет в списке открытых файлов. |
| read-bad-fd | Попытка считать из файла через некорректный дескриптор | Файла с таким дескриптором не будет в списке открытых файлов. Проверка на кореектность переданого адреса осущесвляется в функции validate\_user\_pointer(). |
| write-normal | Создание обычного файла и запись в него | Необходимый файл ищется в списке открытых файлов по своему дескриптору, затем происходит запись в него с помощью функции filesys\_write(). |
| write-zero | Запись нуля байт в файл | Проверка уже реализована в filesys\_write (). |
| write-bad-ptr | Запись данных из буфера с некорректным адресом | Проверка на корректность адресов переданных пользователем осуществляется с помощью validate\_user\_pointer() при вызове syscall\_handler(). |
| write-boundary | Чтение данных из буфера, который находится на разных страницах  памяти | Проверка осуществляется с помощью validate\_user\_pointer(). |
| write-stdin | Попытка записи в устройство ввода | Файла с таким дескриптором не будет в списке открытых файлов. |
| write-bad-fd | Попытка записи в файл с некорректным дескриптором | Файла с таким дескриптором не будет в списке открытых файлов. Проверка на кореектность переданого адреса осущесвляется в функции validate\_user\_pointer(). |

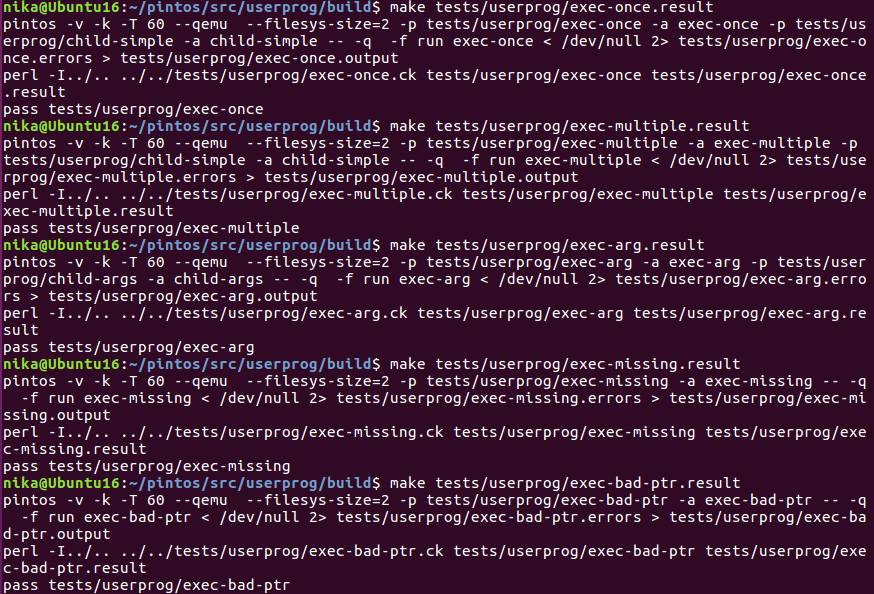
## Результаты тестов



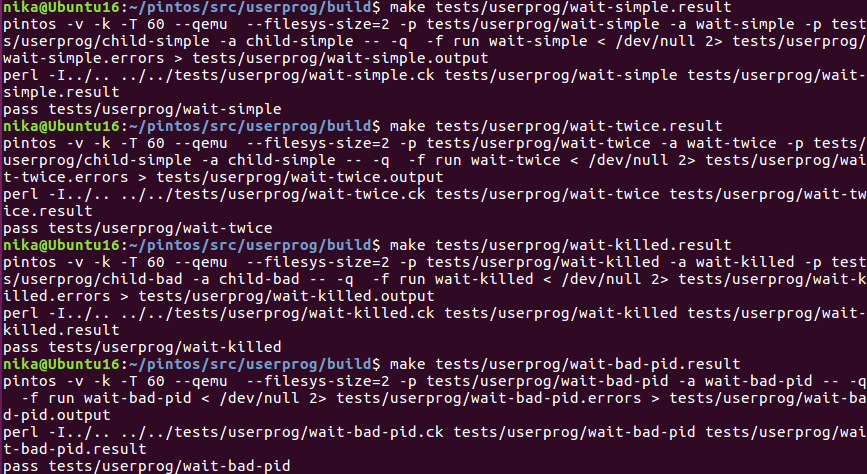
*Рисунок 1. Результат теста системного вызова halt*



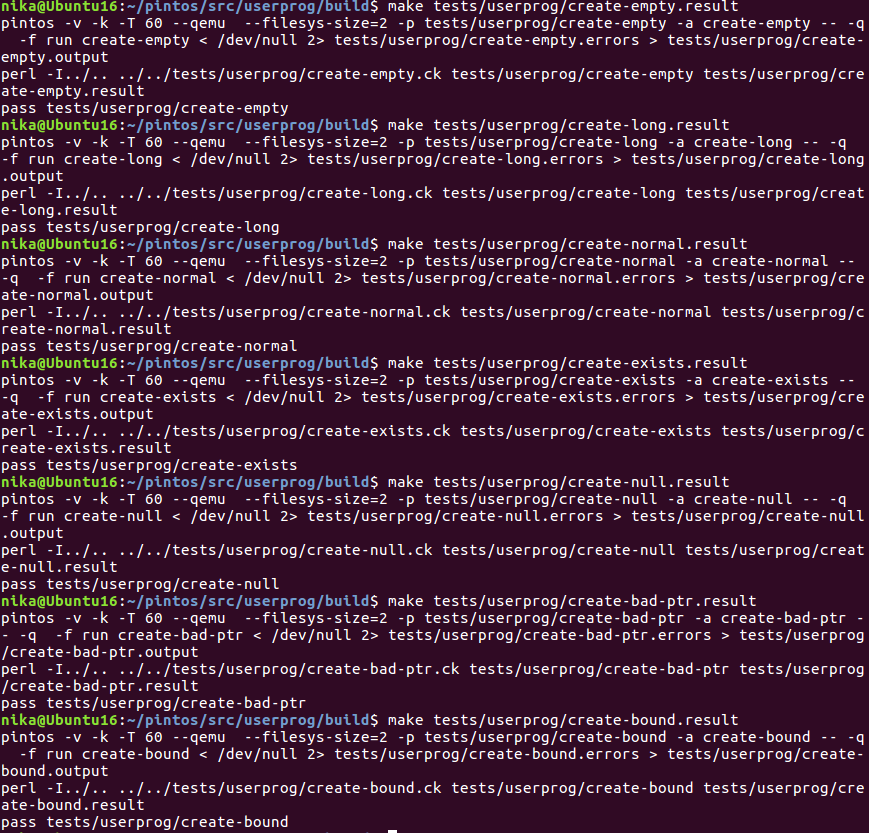
*Рисунок 2. Результат теста системного вызова exit*



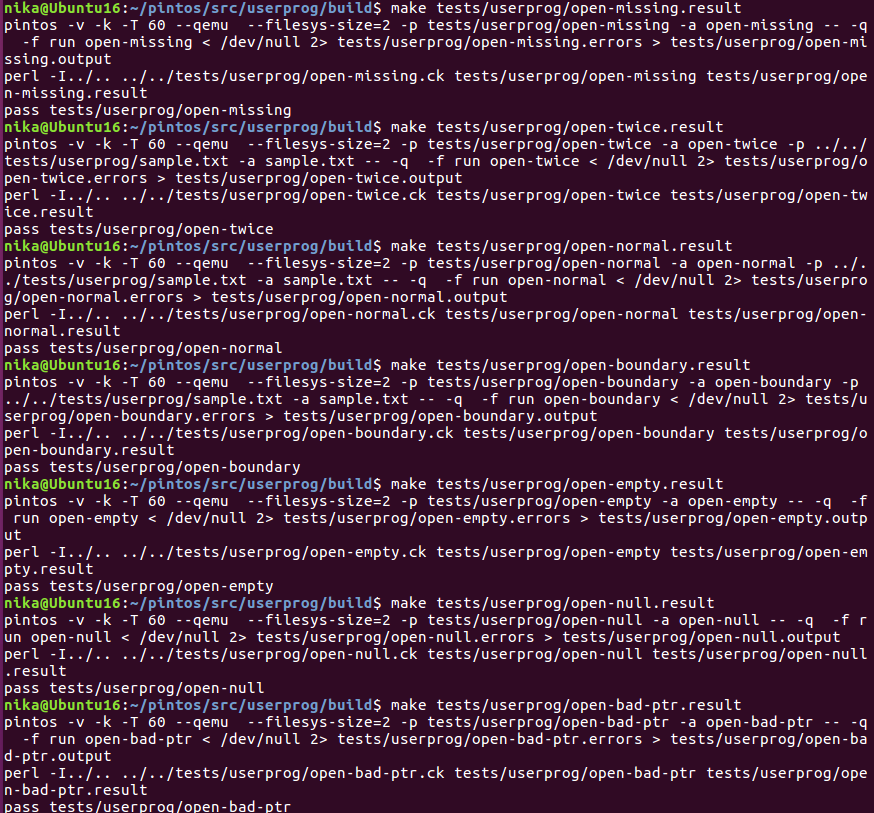
*Рисунок 3. Результаты тестов системного вызова exec*



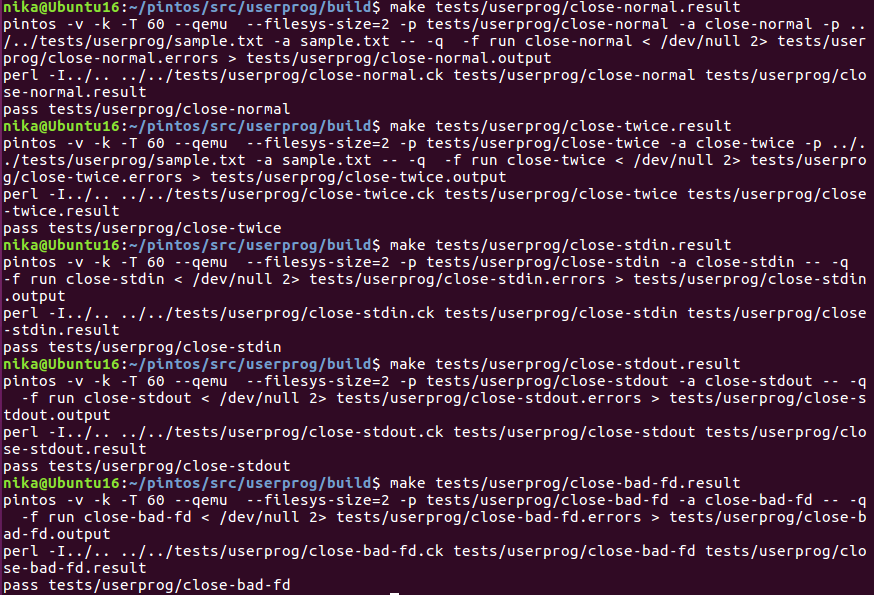
*Рисунок 4. Результаты тестов системного вызова wait*



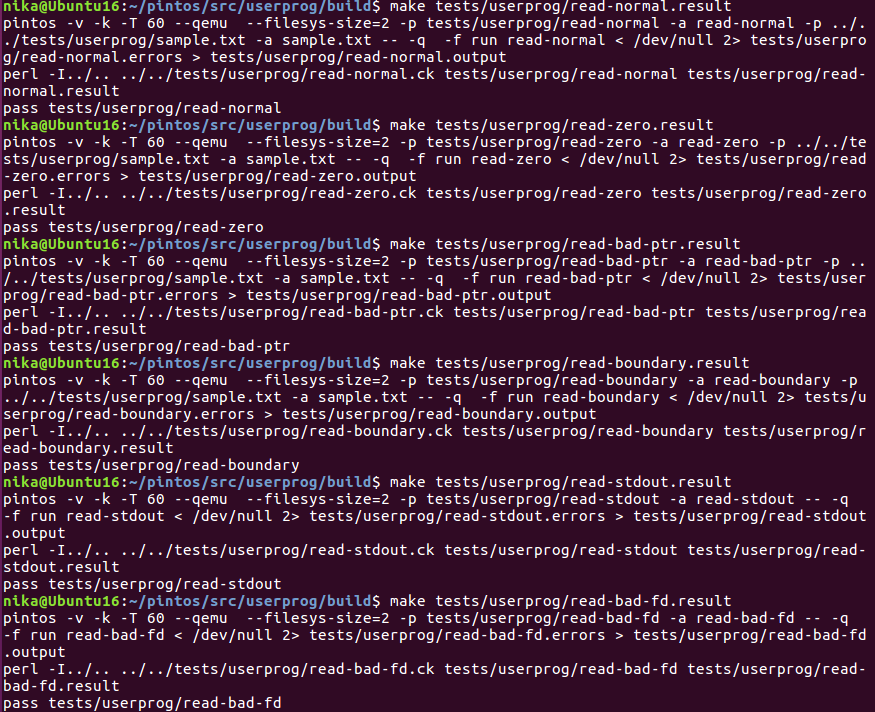
*Рисунок 5. Результаты тестов системного вызова create*



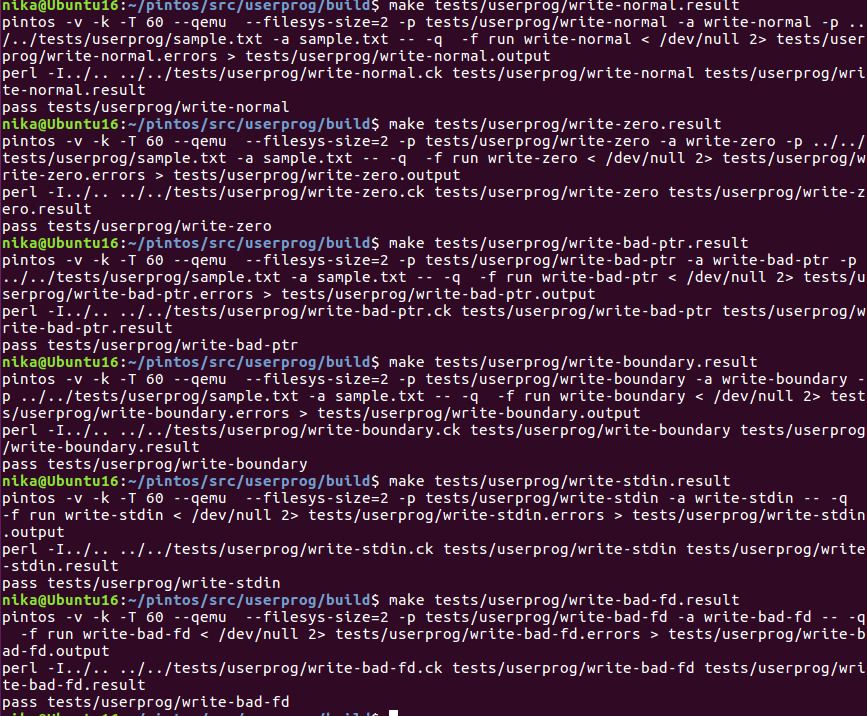
*Рисунок 6. Результаты тестов системного вызова open*



*Рисунок 7. Результаты тестов системного вызова close*



*Рисунок 8. Результаты тестов системного вызова read*



*Рисунок 9. Результаты тестов системного вызова write*

# Вывод

В ходе лабораторной работы были изучены механизмы внутренних прерываний и системных вызовов. Была модифицирована обработка системных вызовов и внедрен набор необходимыx системных вызовов в ОС Pintos.