1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Высшая школа кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 5**

1. «**Системные вызовы**»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 5131001/20001 Маронова К.Д.

<*подпись*>

1. Преподаватель
2. Огнёв Р.А.

<*подпись*>

1. Санкт-Петербург
2. 2023

**1. Цель работы**

Изучение механизма внутренних прерываний и системных вызовов, модификация обработчика системных вызовов и внедрение набора системных вызовов в учебную операционную систему Pintos.

**2. Постановка задачи**

Основная задача лабораторной работы состоит из нескольких пунктов:

1. Модифицировать текущий механизм обработки системных вызовов;
2. Реализовать набор системных вызовов;
3. Проанализировать ситуации, предлагаемые встроенными тестами ОС Pintos по каждому из системных вызовов.

**3. Описание решения**

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Рассматриваемая проблема, описание теста:** | **Как была решена проблема:** |
| **SYS\_HALT** | | |
| halt | Выключение операционной системы Pintos | Использована функция shutdown\_power\_off (); |
| **SYS\_EXIT** | | |
| exit | Завершение выполнения пользовательской программы | Записываем код выхода из стека в поле для кода выхода (из предыдущей л.р) и исполняем функцию thread\_exit(). |
| **SYS\_EXEC** | | |
| exes-once | Запускает выполнение дочернего процесса и ожидает результата его выполнения | Произведены модификации функции process\_execute(). Все дочерние процессы, запущенные «родителем» закидываются в его список детей для дальнейшей работы с ними: |
| exec-multiple | Запускает и ожидает завершения нескольких дочерних процессов | То же решение, что и в предыдущем тесте. Формально, вызов exec-once несколько раз. |
| exec-arg | Запуск дочернего процесса с аргументами из пользовательской программы. | Решение данной проблемы было реализовано в предыдущей лабораторной работе. |
| exec-missing | Попытка запустить несуществующую программу. | Была реализована попытка открыть файл еще до запуска load. Если файл == NULL, то возвращается код выхода -1. В противном случае файл закрывается и выполнение идет дальше. |
| exec-bad-ptr | Передача некорректного адреса в системный вызов. | Была реализована функция validate\_user\_pointer(void\* addr), которая проверяет корректность адреса. В случае успешного прохождения проверки возвращала значение 1. Если вернула 0, то завершаем программу с кодом выхода -1. |
| **SYS\_WAIT** | | |
| wait-simple | Ожидание завершения дочернего процесса | Ищет «ребенка» с child\_tid и делает для него sema\_down для ожидания завершения выполнения. |
| wait-twice | Дважды ожидается завершения подпроцесса.  Второй вызов ожидания должен вернуть значение -1. | Реализована проверка на нахождение потока с заданным tid в списке детей. Если такой поток не найден (NULL), то возвращаем -1. |
| wait-killed | Подает недопустимый адрес и ожидает завершения. | Была реализована функция validate\_user\_pointer(void\* addr), которая проверяет корректность адреса. В случае успешного прохождения проверки возвращала значение 1. Если вернула 0, то завершаем программу с кодом выхода -1. |
| wait-bad-pid | Попытка запуска процесса с некорректным идентификатором. | Реализована проверка на нахождение потока с заданным tid в списке детей. Если такой поток не найден (NULL), то возвращаем -1. |
| **SYS\_CREATE** | | |
| create-empty | Попытка создать файл без имени и с нулевым размером. | Функция dir\_add()проверяет параметры, которые передаются для создания файла, с помощью условия (\*name == '\0' || strlen (name) > NAME\_MAX. |
| create-long | Попытка создание файла с длиной большей, чем максимально допустимая. | См. предыдущий пункт. |
| create-normal | Создание файла. | Реализован вызов функции filesys\_create(). |
| create-exists | Попытка создания файла, который уже существует в системе. | Функция dir\_lookup()проверяет параметры, которые передаются для создания файла. |
| create-null | Передача NULL в качестве имени создаваемого файла | Была реализована дополнительная проверка на то, что file\_name = NULL ((char\*\*)args[0]==NULL). |
| create-bad-ptr | Передача в качестве параметра некорректного участка памяти. | См. выше. |
| create-bound | В качестве имени файла передается строка, находящаяся на разных страницах памяти | См. выше. |
| **SYS\_OPEN** | | |
| open-missing | Попытка открыть файл, которого нет в файловой системе | Функция dir\_add()проверяет входные параметры. |
| open-normal | Открытие файла. | Реализован вызов функции filesys\_open(). Открытый файл со своими характеристиками (благодаря структуре file\_struct) записывается в список файлов, открытых этим потоком. |
| open-twice | Попытка открыть один и тот же файл дважды. Данное действие должно вернуть разные дескрипторы | Был реализован список открытых файлов(структур). Структура хранит в себе сам файл, и его дескриптор. |
| open-boundary | Попытка открытия файла, в качестве имени которого передается строка, расположенная на разных страницах памяти | См. выше. |
| open-empty | Попытка открыть файл, в качестве имени передается пустая строка | Реализована проверка имени файла на NULL. |
| open-bad-ptr | Попытка открыть файл, в качестве имени передается некорректный участок памяти | См. выше. |
| **SYS\_CLOSE** | | |
| close-normal | Закрывает файл, дескриптор которого подан на вход. | Как только в списке файлов, открытых данным потоком, находится файл с заданным дескриптором, то он закрывается. |
| close-twice | Попытка закрыть один и тот же файл дважды | Если данный файл не был найден в списке открытых файлов, то возвращается -1. |
| close-stdin | Попытка передачи дескриптора потока ввода | Реализована проверка (args[0] == STDIN\_FILENO || args[0] == STDOUT\_FILENO). |
| close-stdout | Передача дескриптора потока вывода | См. выше. |
| close-bad-fd | Передача дескриптора, не соответствующего ни одному из открытых процессом файлов. | См. выше. |
| **SYS\_READ** | | |
| read-normal | Открытие файла | Ищем файл с заданным дескриптором в списке открытых файлов, как только находим - чтение. |
| read-zero | Считывание 0 байт | В функции inode\_read\_at (struct inode \*inode, void \*buffer\_, off\_t size, off\_t offset), есть проверка на кол-во считываемых байт. |
| read-bad-ptr | Попытка считать данные в некорректный адрес. | См. выше. |
| read-boundary | Попытка считать файл в буфер, который расположен на разных страницах памяти. | См. выше. |
| read-stdout | Попытка считать данные из stdout | Реализована проверка if(args[0] == STDOUT\_FILENO). |
| read-bad-fd | Попытка считать из файла через некорректный дескриптор | Если файл с текущим дескриптором не будет найдет, то -1. |
| **SYS\_WRITE** | | |
| write-normal | Создание и запись в файл | Ищем файл с заданным дескриптором в списке открытых файлов, как только находим – запись. |
| write-zero | Запись 0 байт | В функции inode\_write\_at (struct inode \*inode, const void \*buffer\_, off\_t size, off\_t offset)есть проверка на кол-во записываемых байт. |
| write-bad-ptr | Запись данных из буфера, адрес которого является недопустимым | См. выше. |
| write-boundary | Чтение данных из буфера, который находится на разных страницах памяти | См. выше. |
| write-stdin | Попытка записи в поток ввода | Реализована проверка (args[0] == STDIN\_FILENO). |
| write-bad-fd | Попытка записи в файл, дескриптор, которого не является корректный | Если файл с текущим дескриптором не найдет, то -1. |

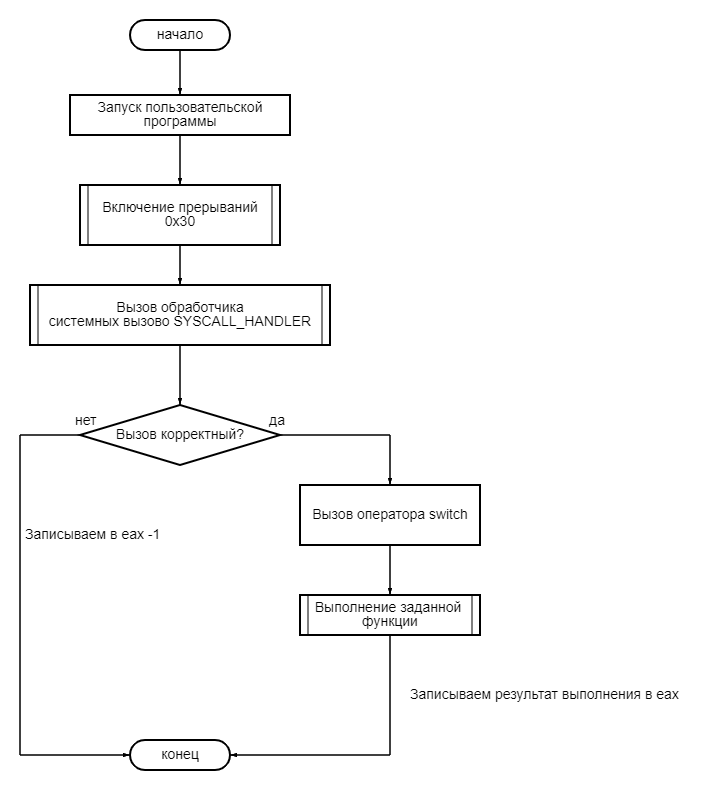


Рис.1 «Схема реализованного алгоритма».

**4. Выводы**

В ходе выполнения данной лабораторной работы были изучены механизм внутренних прерываний и системных вызовов, модифицирована системных вызовов обработчика и внедрен набор системных вызовов в учебную операционную систему Pintos.

Основные проблемы, встретившиеся в ходе работы заключались в следующем: 1) Без создания структуры file\_struct не получалось обращаться к файлам, лежащим в файловой системе (т.к. их характеристики нигде не хранились). Можно было бы использовать не список, а массив и тогда дескриптор был бы равен индексу в массиве, но было принято решение продолжить работу со списками. 2) Так же возникла проблема с хранение финального кода выхода потока для тестов типа wait-twice. Для этого в структуру файла был добавлен код выхода его потока. 3) Возникла ошибка при использовании функции list\_remove. Из-за подачи неверных аргументов в функцию дескрипторы файлом при закрытии подменялись на случайные числа.

Листинг программы:

int validate\_user\_pointer(void\* addr);

static void syscall\_handler(struct intr\_frame\*);

int\* args;

void syscall\_init(void)

{

intr\_register\_int(0x30, 3, INTR\_ON, syscall\_handler, "syscall");

}

static void

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

int\* nr;

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

if (!validate\_user\_pointer(nr)) {

thread\_current()->code\_for\_exiting = -1;

thread\_exit();

return;

}

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

struct list\_elem\* filesize\_el;

switch (\*nr) {

case SYS\_HALT:

shutdown\_power\_off();

break;

case SYS\_EXIT:

thread\_current()->code\_for\_exiting = args[0];

thread\_exit();

break;

case SYS\_CREATE:

if (!validate\_user\_pointer(args[0])) {

thread\_current()->code\_for\_exiting = -1;

f->eax = -1;

thread\_exit();

}

if ((char\*\*)args[0] == NULL) {

thread\_current()->code\_for\_exiting = -1;

thread\_exit();

}

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

break;

case SYS\_REMOVE:

if (!validate\_user\_pointer(args[0])) {

thread\_current()->code\_for\_exiting = -1;

thread\_exit();

f->eax = -1;

}

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

break;

case SYS\_OPEN:

f->eax=-1;

if (!validate\_user\_pointer(args[0])) {

thread\_current()->code\_for\_exiting = -1;

thread\_exit();

}

if ((char\*\*)args[0] == NULL) {

thread\_current()->code\_for\_exiting = -1;

thread\_exit();

}

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

if (FILE == NULL) return -1;

struct file\_struct\* new\_file = malloc(sizeof(struct file\_struct));

new\_file->fd = thread\_current()->cnt\_for\_des;

new\_file->current\_file = FILE;

f->eax = thread\_current()->cnt\_for\_des;

thread\_current()->cnt\_for\_des++;

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

break;

case SYS\_CLOSE:

if (list\_empty(&thread\_current()->open\_files)){

thread\_current()->code\_for\_exiting = -1;

thread\_exit();

break;

}

if (args[0] == STDIN\_FILENO || args[0] == STDOUT\_FILENO) {

thread\_current()->code\_for\_exiting = -1;

thread\_exit();

break;

}

struct list\_elem\* e = list\_begin(&thread\_current()->open\_files);

for (e; e != list\_end(&thread\_current()->open\_files); e = list\_next(e)) {

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

if (tmp->fd == args[0]) {

file\_close(tmp->current\_file);

list\_remove(e);

thread\_current()->cnt\_for\_des--;

break;

}

}

break;

case SYS\_FILESIZE:

filesize\_el = list\_begin(&thread\_current()->open\_files);

for (filesize\_el; filesize\_el != list\_end(&thread\_current()->open\_files); filesize\_el = list\_next(filesize\_el)) {

struct file\_struct\* tmp\_size = list\_entry(filesize\_el, struct file\_struct, file\_elem);

if (tmp\_size->fd == args[0]) {

f->eax = file\_length(tmp\_size->current\_file);

break;

//thread\_exit();

}

}

break;

case SYS\_READ:

if (!validate\_user\_pointer((char\*\*)args[1])) {

thread\_current()->code\_for\_exiting = -1;

thread\_exit();

break;

}

if(args[0] == STDIN\_FILENO){

f->eax=input\_getc();

}

if(!list\_empty(&thread\_current()->open\_files)){

struct list\_elem\* fileread\_el = list\_begin(&thread\_current()->open\_files);

for (fileread\_el; fileread\_el != list\_end(&thread\_current()->open\_files); fileread\_el = list\_next(fileread\_el)) {

struct file\_struct\* tmpr = list\_entry(fileread\_el, struct file\_struct, file\_elem);

if (tmpr->fd == args[0]) {

f->eax = file\_read(tmpr->current\_file, (char\*\*)args[1], args[2]);

break;

//thread\_exit();

}

}

}

break;

case SYS\_WRITE:

if (!validate\_user\_pointer((char\*\*)args[1])) {

thread\_current()->code\_for\_exiting = -1;

thread\_exit();

}

if(args[0] == STDOUT\_FILENO){

putbuf(((const char\*\*)f->esp)[2], ((size\_t\*)f->esp)[3]);

}

if(args[0] == STDIN\_FILENO){

return -1;

}

struct list\_elem\* filewrite\_el = list\_begin(&thread\_current()->open\_files);

for (filewrite\_el; filewrite\_el != list\_end(&thread\_current()->open\_files); filewrite\_el = list\_next(filewrite\_el)) {

struct file\_struct\* tmpr = list\_entry(filewrite\_el, struct file\_struct, file\_elem);

if (tmpr->fd == args[0]) {

f->eax = file\_write(tmpr->current\_file, (char\*\*)args[1], args[2]);

break;

//thread\_exit();

}

}

break;

case SYS\_WAIT:

if (!validate\_user\_pointer(nr)) return -1;

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

break;

case SYS\_EXEC:

if (!validate\_user\_pointer(args[0])) return -1;

f->eax = process\_execute((char\*\*)args[0]);

break;

}

}

int validate\_user\_pointer(void\* addr) {

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

return 0;

}

else {

return 1;

}

}

static thread\_func start\_process NO\_RETURN;

static bool load (const char \*cmdline, void (\*\*eip) (void), void \*\*esp);

struct semaphore sema\_for\_wait;

int gl\_ex\_code;

struct thread\* thread\_search(tid\_t tid);

/\* Начинает новый поток, выполняющий пользовательскую программу, загруженную из файла FILENAME.

Новый поток может быть запланирован (и даже завершен) до того, как process\_execute() вернется. В

озвращает идентификатор нового процесса (TID) или TID\_ERROR, если поток не может быть создан. \*/

tid\_t

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

{

char \*fn\_copy;

tid\_t tid;

struct thread\* cur = thread\_current();

sema\_init(&sema\_for\_wait, 0);

/\* Make a copy of FILE\_NAME.

В противном случае возникает гонка данных между вызывающим и функцией load().. \*/

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

if (fn\_copy == NULL)

return TID\_ERROR;

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

/////////////////NEW///////////////////

char\* token, \* save\_ptr, \* name\_copy;

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

if (name\_copy == NULL)

return TID\_ERROR;

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

// получение имени программы

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

struct file\* FILE = filesys\_open(token);

if (FILE == NULL)

{

file\_close(FILE);

gl\_ex\_code = -1;

return -1;

}

file\_close(FILE);

///////////////////////////////////////

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

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

if (tid == TID\_ERROR)

palloc\_free\_page (fn\_copy);

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

struct thread\* child\_thread = thread\_search(tid);

intr\_set\_level(old\_level);

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

return tid;

}

/\* A thread function that loads a user process and starts it

running. \*/

static void

start\_process (void \*file\_name\_)

{

char \*file\_name = file\_name\_;

struct intr\_frame if\_;

bool success;

/\* Initialize interrupt frame and load executable. \*/

memset (&if\_, 0, sizeof if\_);

if\_.gs = if\_.fs = if\_.es = if\_.ds = if\_.ss = SEL\_UDSEG;

if\_.cs = SEL\_UCSEG;

if\_.eflags = FLAG\_IF | FLAG\_MBS;

success = load (file\_name, &if\_.eip, &if\_.esp);

/\* If load failed, quit. \*/

palloc\_free\_page (file\_name);

if (!success){

thread\_current()->code\_for\_exiting=-1;

thread\_exit ();

}

/\*Начинает выполнение пользовательской программы за счет моделирования

возврата из прерывания, реализованного в функции `intr\_exit`

(в файле `threads/intr-stubs.S`). Поскольку `intr\_exit` принимает все свои

аргументы на стеке в виде структуры `intr\_frame`, мы просто устанавливаем

указатель стека (%esp) нашего стекового фрейма и переходим к нему. \*/

asm volatile ("movl %0, %%esp; jmp intr\_exit" : : "g" (&if\_) : "memory");

NOT\_REACHED ();

}

/\*Ожидает завершения потока с идентификатором TID и возвращает его код завершения.

Если поток был прерван ядром (т.е. убит из-за исключения), возвращает -1. Если TID

недопустим или он не является дочерним процессом вызывающего процесса, или если функция

process\_wait() уже была успешно вызвана для данного TID, то немедленно возвращает -1 без ожидания.

Эта функция будет реализована в задаче 2-2. В настоящее время она ничего не делает. \*/

int

process\_wait (tid\_t child\_tid)

{

struct thread\* child\_thread = NULL;

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

return -1;

for (struct list\_elem\* e = list\_begin(&thread\_current()->child\_list); e != NULL; e = list\_next(e))

{

child\_thread = list\_entry(e, struct thread, child\_elem);

if (child\_thread->tid == child\_tid)

break;

}

if (child\_thread == NULL)

return -1;

list\_remove(&child\_thread->child\_elem);

sema\_down(&child\_thread->sema\_for\_child);

return gl\_ex\_code;

}

/\* Free the current process's resources. \*/

void

process\_exit (void)

{

struct thread \*cur = thread\_current ();

uint32\_t \*pd;

/\* Destroy the current process's page directory and switch back

to the kernel-only page directory. \*/

pd = cur->pagedir;

if (pd != NULL)

{

/\* Correct ordering here is crucial. We must set

cur->pagedir to NULL before switching page directories,

so that a timer interrupt can't switch back to the

process page directory. We must activate the base page

directory before destroying the process's page

directory, or our active page directory will be one

that's been freed (and cleared). \*/

cur->pagedir = NULL;

pagedir\_activate (NULL);

pagedir\_destroy (pd);

//////////NEW//////////////////

char\* token, \* save\_ptr, \* name\_copy;

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

if (name\_copy == NULL)

return TID\_ERROR;

strlcpy(name\_copy, cur->name, PGSIZE);

// получение имени программы

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

printf("%s: exit(%d)\n", token, cur->code\_for\_exiting);

/\*printf("%s: exit(%d)\n", cur->name, cur->code\_for\_exiting);\*/

//sema\_up(&sema\_for\_wait);

sema\_up(&thread\_current()->sema\_for\_child);

gl\_ex\_code = cur->code\_for\_exiting;

///////////////////////////////

}

}

/\* Sets up the CPU for running user code in the current

thread.

This function is called on every context switch. \*/

void

process\_activate (void)

{

struct thread \*t = thread\_current ();

/\* Activate thread's page tables. \*/

pagedir\_activate (t->pagedir);

/\* Set thread's kernel stack for use in processing

interrupts. \*/

tss\_update ();

}

/\* We load ELF binaries. The following definitions are taken

from the ELF specification, [ELF1], more-or-less verbatim. \*/

/\* ELF types. See [ELF1] 1-2. \*/

typedef uint32\_t Elf32\_Word, Elf32\_Addr, Elf32\_Off;

typedef uint16\_t Elf32\_Half;

/\* For use with ELF types in printf(). \*/

#define PE32Wx PRIx32 /\* Print Elf32\_Word in hexadecimal. \*/

#define PE32Ax PRIx32 /\* Print Elf32\_Addr in hexadecimal. \*/

#define PE32Ox PRIx32 /\* Print Elf32\_Off in hexadecimal. \*/

#define PE32Hx PRIx16 /\* Print Elf32\_Half in hexadecimal. \*/

/\* Executable header. See [ELF1] 1-4 to 1-8.

This appears at the very beginning of an ELF binary. \*/

struct Elf32\_Ehdr

{

unsigned char e\_ident[16];

Elf32\_Half e\_type;

Elf32\_Half e\_machine;

Elf32\_Word e\_version;

Elf32\_Addr e\_entry;

Elf32\_Off e\_phoff;

Elf32\_Off e\_shoff;

Elf32\_Word e\_flags;

Elf32\_Half e\_ehsize;

Elf32\_Half e\_phentsize;

Elf32\_Half e\_phnum;

Elf32\_Half e\_shentsize;

Elf32\_Half e\_shnum;

Elf32\_Half e\_shstrndx;

};

/\* Program header. See [ELF1] 2-2 to 2-4.

There are e\_phnum of these, starting at file offset e\_phoff

(see [ELF1] 1-6). \*/

struct Elf32\_Phdr

{

Elf32\_Word p\_type;

Elf32\_Off p\_offset;

Elf32\_Addr p\_vaddr;

Elf32\_Addr p\_paddr;

Elf32\_Word p\_filesz;

Elf32\_Word p\_memsz;

Elf32\_Word p\_flags;

Elf32\_Word p\_align;

};

/\* Values for p\_type. See [ELF1] 2-3. \*/

#define PT\_NULL 0 /\* Ignore. \*/

#define PT\_LOAD 1 /\* Loadable segment. \*/

#define PT\_DYNAMIC 2 /\* Dynamic linking info. \*/

#define PT\_INTERP 3 /\* Name of dynamic loader. \*/

#define PT\_NOTE 4 /\* Auxiliary info. \*/

#define PT\_SHLIB 5 /\* Reserved. \*/

#define PT\_PHDR 6 /\* Program header table. \*/

#define PT\_STACK 0x6474e551 /\* Stack segment. \*/

/\* Flags for p\_flags. See [ELF3] 2-3 and 2-4. \*/

#define PF\_X 1 /\* Executable. \*/

#define PF\_W 2 /\* Writable. \*/

#define PF\_R 4 /\* Readable. \*/

static bool setup\_stack (void \*\*esp);

static bool validate\_segment (const struct Elf32\_Phdr \*, struct file \*);

static bool load\_segment (struct file \*file, off\_t ofs, uint8\_t \*upage,

uint32\_t read\_bytes, uint32\_t zero\_bytes,

bool writable);

/\* Loads an ELF executable from FILE\_NAME into the current thread.

Stores the executable's entry point into \*EIP

and its initial stack pointer into \*ESP.

Returns true if successful, false otherwise. \*/

bool

load (const char \*file\_name, void (\*\*eip) (void), void \*\*esp)

{

struct thread \*t = thread\_current ();

struct Elf32\_Ehdr ehdr;

struct file \*file = NULL;

off\_t file\_ofs;

bool success = false;

int i;

/\* Allocate and activate page directory. \*/

t->pagedir = pagedir\_create ();

if (t->pagedir == NULL)

goto done;

process\_activate ();

//////////////////NEW/////////////////////

char\* s\_ptr;

char\* tk;

char n\_copy[128];

//n\_copy = palloc\_get\_page(0);

if (n\_copy == NULL)

return TID\_ERROR;

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

// получение имени программы

tk = strtok\_r(n\_copy, " ", &s\_ptr);

file = filesys\_open(tk);

/////////////////////////////////////////

/\* Open executable file. \*/

/\*file = filesys\_open (file\_name);\*/

if (file == NULL)

{

printf ("load: %s: open failed\n", tk);

goto done;

}

/\* Read and verify executable header. \*/

if (file\_read (file, &ehdr, sizeof ehdr) != sizeof ehdr

|| memcmp (ehdr.e\_ident, "\177ELF\1\1\1", 7)

|| ehdr.e\_type != 2

|| ehdr.e\_machine != 3

|| ehdr.e\_version != 1

|| ehdr.e\_phentsize != sizeof (struct Elf32\_Phdr)

|| ehdr.e\_phnum > 1024)

{

printf ("load: %s: error loading executable\n", tk);

goto done;

}

/\* Read program headers. \*/

file\_ofs = ehdr.e\_phoff;

for (i = 0; i < ehdr.e\_phnum; i++)

{

struct Elf32\_Phdr phdr;

if (file\_ofs < 0 || file\_ofs > file\_length (file))

goto done;

file\_seek (file, file\_ofs);

if (file\_read (file, &phdr, sizeof phdr) != sizeof phdr)

goto done;

file\_ofs += sizeof phdr;

switch (phdr.p\_type)

{

case PT\_NULL:

case PT\_NOTE:

case PT\_PHDR:

case PT\_STACK:

default:

/\* Ignore this segment. \*/

break;

case PT\_DYNAMIC:

case PT\_INTERP:

case PT\_SHLIB:

goto done;

case PT\_LOAD:

if (validate\_segment (&phdr, file))

{

bool writable = (phdr.p\_flags & PF\_W) != 0;

uint32\_t file\_page = phdr.p\_offset & ~PGMASK;

uint32\_t mem\_page = phdr.p\_vaddr & ~PGMASK;

uint32\_t page\_offset = phdr.p\_vaddr & PGMASK;

uint32\_t read\_bytes, zero\_bytes;

if (phdr.p\_filesz > 0)

{

/\* Normal segment.

Read initial part from disk and zero the rest. \*/

read\_bytes = page\_offset + phdr.p\_filesz;

zero\_bytes = (ROUND\_UP (page\_offset + phdr.p\_memsz, PGSIZE)

- read\_bytes);

}

else

{

/\* Entirely zero.

Don't read anything from disk. \*/

read\_bytes = 0;

zero\_bytes = ROUND\_UP (page\_offset + phdr.p\_memsz, PGSIZE);

}

if (!load\_segment (file, file\_page, (void \*) mem\_page,

read\_bytes, zero\_bytes, writable))

goto done;

}

else

goto done;

break;

}

}

/\* Set up stack. \*/

if (!setup\_stack (esp))

goto done;

/\* Start address. \*/

\*eip = (void (\*) (void)) ehdr.e\_entry;

success = true;

//////////////NEW//////////////////

int argc = 0;

char \*argv[129];//массив для аргументов. первый индекс указывает на элемент в стеке, а второй индекс указывает на символ в строке.

char\* token\_for\_stack, \* save\_ptr;//переменные, необходимые для функции strtok

char \*array[128];

//Вспомогательная строка, т.к. исходную не стоит менять

char\* tmp\_filename = palloc\_get\_page(sizeof(char) \* (strlen(file\_name) + 1));

strlcpy(tmp\_filename, file\_name, (strlen(file\_name) + 1));

for (token\_for\_stack = strtok\_r(tmp\_filename, " ", &save\_ptr); token\_for\_stack != NULL; token\_for\_stack = strtok\_r(NULL, " ", &save\_ptr)) {//цикл по аргументам для их подсчёта

argv[argc]=token\_for\_stack;

argc++;

}//переменная, отвечающая за количестов аргументов

int len;

char\* set\_pointer;

void\* pointer;

//запись в есп наших аргументов в обратном порядке

for (int i = argc - 1; i >= 0; i--) {

len = strlen(argv[i]) + 1;

\*esp -= len;

strlcpy(\*esp, argv[i], len);

array[i] = \*esp; //тут происходит запись в массив указателей argv указатель на нашу строчку в стэке

}

set\_pointer = \*esp;

set\_pointer -= sizeof(char\*);

//Записываем адреса строк в стек

for (int i = argc - 1; i >= 0; i--) {

set\_pointer -= sizeof(char\*);

memcpy(set\_pointer, &array[i], sizeof(char\*));

}

void\*\* ptr = set\_pointer;

set\_pointer -= sizeof(char\*);

memcpy(set\_pointer, &ptr, sizeof(char\*));

set\_pointer -= sizeof(char\*);

\*set\_pointer = argc;//Запись количества аргументов

set\_pointer -= sizeof(char\*);

\*esp = set\_pointer;//адрес возврата

///////////////////////////////////////////////

done:

/\* We arrive here whether the load is successful or not. \*/

file\_close (file);

return success;

}

/\* load() helpers. \*/

static bool install\_page (void \*upage, void \*kpage, bool writable);

/\* Checks whether PHDR describes a valid, loadable segment in

FILE and returns true if so, false otherwise. \*/

static bool

validate\_segment (const struct Elf32\_Phdr \*phdr, struct file \*file)

{

/\* p\_offset and p\_vaddr must have the same page offset. \*/

if ((phdr->p\_offset & PGMASK) != (phdr->p\_vaddr & PGMASK))

return false;

/\* p\_offset must point within FILE. \*/

if (phdr->p\_offset > (Elf32\_Off) file\_length (file))

return false;

/\* p\_memsz must be at least as big as p\_filesz. \*/

if (phdr->p\_memsz < phdr->p\_filesz)

return false;

/\* The segment must not be empty. \*/

if (phdr->p\_memsz == 0)

return false;

/\* The virtual memory region must both start and end within the

user address space range. \*/

if (!is\_user\_vaddr ((void \*) phdr->p\_vaddr))

return false;

if (!is\_user\_vaddr ((void \*) (phdr->p\_vaddr + phdr->p\_memsz)))

return false;

/\* The region cannot "wrap around" across the kernel virtual

address space. \*/

if (phdr->p\_vaddr + phdr->p\_memsz < phdr->p\_vaddr)

return false;

/\* Disallow mapping page 0.

Not only is it a bad idea to map page 0, but if we allowed

it then user code that passed a null pointer to system calls

could quite likely panic the kernel by way of null pointer

assertions in memcpy(), etc. \*/

if (phdr->p\_vaddr < PGSIZE)

return false;

/\* It's okay. \*/

return true;

}

/\* Loads a segment starting at offset OFS in FILE at address

UPAGE. In total, READ\_BYTES + ZERO\_BYTES bytes of virtual

memory are initialized, as follows:

- READ\_BYTES bytes at UPAGE must be read from FILE

starting at offset OFS.

- ZERO\_BYTES bytes at UPAGE + READ\_BYTES must be zeroed.

The pages initialized by this function must be writable by the

user process if WRITABLE is true, read-only otherwise.

Return true if successful, false if a memory allocation error

or disk read error occurs. \*/

static bool

load\_segment (struct file \*file, off\_t ofs, uint8\_t \*upage,

uint32\_t read\_bytes, uint32\_t zero\_bytes, bool writable)

{

ASSERT ((read\_bytes + zero\_bytes) % PGSIZE == 0);

ASSERT (pg\_ofs (upage) == 0);

ASSERT (ofs % PGSIZE == 0);

file\_seek (file, ofs);

while (read\_bytes > 0 || zero\_bytes > 0)

{

/\* Calculate how to fill this page.

We will read PAGE\_READ\_BYTES bytes from FILE

and zero the final PAGE\_ZERO\_BYTES bytes. \*/

size\_t page\_read\_bytes = read\_bytes < PGSIZE ? read\_bytes : PGSIZE;

size\_t page\_zero\_bytes = PGSIZE - page\_read\_bytes;

/\* Get a page of memory. \*/

uint8\_t \*kpage = palloc\_get\_page (PAL\_USER);

if (kpage == NULL)

return false;

/\* Load this page. \*/

if (file\_read (file, kpage, page\_read\_bytes) != (int) page\_read\_bytes)

{

palloc\_free\_page (kpage);

return false;

}

memset (kpage + page\_read\_bytes, 0, page\_zero\_bytes);

/\* Add the page to the process's address space. \*/

if (!install\_page (upage, kpage, writable))

{

palloc\_free\_page (kpage);

return false;

}

/\* Advance. \*/

read\_bytes -= page\_read\_bytes;

zero\_bytes -= page\_zero\_bytes;

upage += PGSIZE;

}

return true;

}

/\* Create a minimal stack by mapping a zeroed page at the top of

user virtual memory. \*/

static bool

setup\_stack (void \*\*esp)

{

uint8\_t \*kpage;

bool success = false;

kpage = palloc\_get\_page (PAL\_USER | PAL\_ZERO);

if (kpage != NULL)

{

success = install\_page (((uint8\_t \*) PHYS\_BASE) - PGSIZE, kpage, true);

if (success)

\*esp = PHYS\_BASE;

else

palloc\_free\_page (kpage);

}

return success;

}

/\* Adds a mapping from user virtual address UPAGE to kernel

virtual address KPAGE to the page table.

If WRITABLE is true, the user process may modify the page;

otherwise, it is read-only.

UPAGE must not already be mapped.

KPAGE should probably be a page obtained from the user pool

with palloc\_get\_page().

Returns true on success, false if UPAGE is already mapped or

if memory allocation fails. \*/

static bool

install\_page (void \*upage, void \*kpage, bool writable)

{

struct thread \*t = thread\_current ();

/\* Verify that there's not already a page at that virtual

address, then map our page there. \*/

return (pagedir\_get\_page (t->pagedir, upage) == NULL

&& pagedir\_set\_page (t->pagedir, upage, kpage, writable));

}

struct thread\* thread\_search(tid\_t tid)

{

struct list\_elem\* e;

ASSERT(intr\_get\_level() == INTR\_OFF);

for (e = list\_begin(&all\_list); e != list\_end(&all\_list); e = list\_next(e))

{

struct thread\* t = list\_entry(e, struct thread, allelem);

if (t->tid == tid)

return t;

}

return NULL;

}