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

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. «ПРИНЦИПЫ РАЗРАБОТКИ ОПЕРАЦИОННЫХ СИСТЕМ»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 4851003/00002 Вологдин М.В.

<*подпись*>

1. Преподаватель
2. асс. преподавателя Крундышев В.М.

<*подпись*>

1. Санкт-Петербург
2. 2021
3. **Цель работы**

Изучение основ разработки ОС, принципов низкоуровневого взаимодействия с аппаратным обеспечением, программирования системной функциональности и процесса загрузки системы.

1. **Задачи работы**
2. Написать загрузчик, который:

1. Должен загружать ядро со второго гибкого диска по нужному адресу (0x10000 для плоского ядра, 0x11000+0x12000 для ядра PE).

2. Запрашивать у пользователя (либо BIOS) необходимые параметры и сохранять их в память (по произвольному адресу, определяется студентом).

3. Переводить процессор в защищенный режим и передавать управление ядру.

2)Написать ядро OC, которое:

1. ОС должна считать параметры, сохраненные загрузчиком (в п.2), и применить их.

2. Настроить прерывания и включить их.

3. Перейти в интерактивный режим: на экране должна быть строка приветствия. Пользователь вводит в нее команду, а ОС отвечает на нее. Команда состоит из ключевого слова и параметров (для некоторых команд) — указываются через пробел. Перечень поддерживаемых команд указан в варианте задания. Если команда не распознана — должны выводиться сведения об этом (ошибка).

4. ОС не позволяет вводить команды длиннее 40 символов. При попытке ввести команду длиннее лимита — ОС не будет реагировать на лишние нажатия. Должна поддерживаться клавиша Backspace.

5. ОС поддерживает ввод строчных букв латинского алфавита, пробела, цифр, знаков мат. операций + - / \*

6. Если все строки текущего экрана заполнены — экран очищается и взаимодействие с пользователем продолжается вновь, на чистом экране

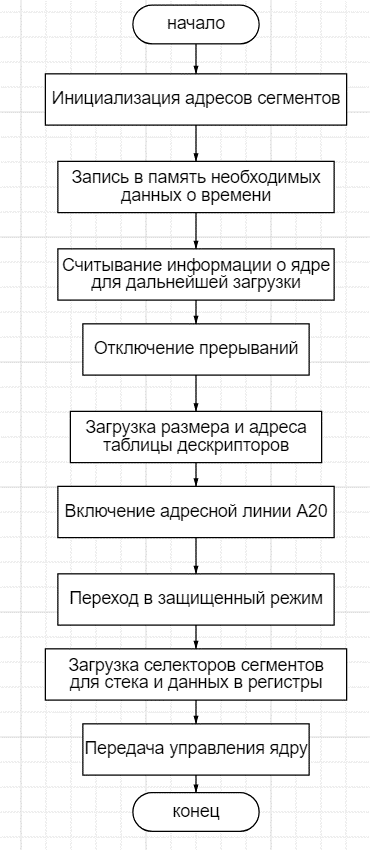
1. **Ход работы**
   1. **Загрузчик OC**

Рисунок 1 – Блок-схема загрузчика OC

Сначала происходит сохранение адреса сегмента кода в ax и сохранение этого адреса как начало сегмента данных и сегмента стека. Далее адрес стека сохраняется как адрес первой инструкции. Происходит работа с портами 0x70 и 0x71 для получения информации о времени загрузки ОС. Далее указываются адрес и регистры для считывания нужного количества секторов в адреса 0x11000 и 0x12000(код ядра). После этого отключаются прерывания и происходит загрузка размера и адреса таблицы дескрипторов. Включается адресная линия А20, устанавливается бит PE регистра CR0 - процессор перейдет в защищенный режим. И далее "дальний" переход (для загрузки корректной информации в cs) в инструкции, которые выполняются в защищённом режиме. Выполняется загрузка селекторов сегментов для стека и данных в регистры и функцией call выполняется передача управления ядру. Кроме того, инициализируется таблица дескрипторов, остальная часть файла заполняется нулями, а последние 2 байта 0x55 и 0xAA показывают, что этот файл является загрузочным.

* 1. **Ядро OC**

Ядро начинается с инструкции call kmain, которая вызывает основную функцию. Далее происходит очистка экрана, настройка прерываний, их включение и настройка прерываний клавиатуры. После этого программа находится в бесконечном цикле ожидания прерываний.

Нажатия клавиш выводят соответствующие символы на экран, обращение к дисплею происходит по адресу 0xB8000 + 2 \* (строка \* ширина\_экрана + столбец). При нажатии Enter программа проверяет соответствует ли ввод одной из доступных функций.

Прерывания с клавиатуры реализованы следующим образом: создан массив скан-кодов символов, каждый код соответствует своей клавише на клавиатуре. При вызывании кливиши символа, он выводится на экран, при нажатии несимвольных клавиш (например Enter) производится определённая функция, соответствующая назначению клавиши. Enter – перенос на новую строку + возможный вызов команды. Backspace – стирание предыдущего символа.

|  |  |  |
| --- | --- | --- |
| Имя функции | Описание параметров | Описание назначения |
| void call\_info() | void | Выводит информацию об авторе и средствах разработки OC |
| void call\_help() | void | Выводит информацию о доступных командах (все команды в таблице) |
| void clear\_scr() | void | Очищает экран |
| void call\_ticks() | void | Выводит количество тиков, прошедших с момента старта OC |
| void call\_loadtime() | void | Выводит время и дату, считанные с помощью BIOS на момент загрузки ядра ОС |
| void call\_curtime() | void | Выводит текущее время |
| void call\_uptime() | void | Выводит общее время работы системы |
| void call\_cpuid() | void | Определяет модель процессора (Vendor ID) и выводит на экран |
| void call\_shutdown() | void | Выключение компьютера |

Функция вывода строки на экран out\_str – посимвольная запись в адрес видеопамяти. Call\_info выводит нужную информацию с помощью двух вызовов out\_str. Call\_help аналогично выводит информацию о каждой функции с помощью вызовов out\_str. Clear\_src проходит циклом по всему экрану и заполняет каждую ячейку памяти пробелом, цвет символов светло-серый, а фона – черный.

Call\_ticks пользуется ассемблерной инструкцией rdtsc, которая в пару регистров edx:eax записывает 64-битное число – число тиков со старта системы. Регистры копируются в единое число, которое посимвольно выводится на экран с помощью функции out\_char – аналогично out\_str, только выводится всего один символ.

Call\_loadtime считывает из памяти (было записано в загрузчике) по адресам 0x1000, 0x1100 и 0x1200 записанные в bcd-формате соответственно часы, минуты и секунды. Далее преобразует их в десятичный вид и выводит посимвольно на экран в формате hh:mm:ss.

Call\_curtime с помощью взаимодействия с портами ввода-вывода 0x70 и 0x71 получает значение текущего времени.

В порт 0x70 записывается 4 (часы) и с порта 0x71 считывается значение часов, аналогично с минутами(2) и секундами(0)

Call\_uptime считает разность между curtime и loadtime и выводит на экран

Call\_cpuid с помощью ассемблерной инструкции cpuid помещает в регистры ebx:ecx:edx идентификатор производителя процессора, который расшифровывается при помощи сопоставления известных кодов и полученного. Имя производителя записывается на экран.

Call\_shutdown – для этой функции реализована вспомогательная функция outw для записи в порт 16 бит, система выключается при записи outw(0x0604, 0x2000);

Сборка OC:

Сборка загрузчика

yasm -f bin -o bootsect.bin bootsect.asm

Сборка Ядра

cl.exe /GS- /c kernel.cpp

link.exe /OUT:kernel.bin /BASE:0x10000 /FIXED /FILEALIGN:512 /MERGE:.rdata=.data /IGNORE:4254 /NODEFAULTLIB /ENTRY:startup /SUBSYSTEM:NATIVE kernel.obj

первая строка – компиляция ядра, вторая – линковщик, который из .obj файла делает .bin, нужный для запуска системы.

1. **Тестирование программы**

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Пять примеров входных данных

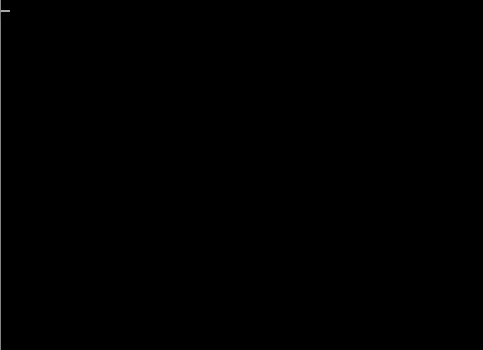


Рисунок 3 – Шестой пример(команда clear)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – С седьмого примера по двенадцатый

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – 13 и 14 примеры

Пятнадцатым является пример ввода команды shutdown, который выключает компьютер, поэтому на консоль ничего не выводится

1. **Выводы**

В ходе работы были изучены основы разработки ОС, принципы низкоуровневого взаимодействия с аппаратным обеспечением, а также была реализованы ядро и загрузчик своей операционной системы, которая выполняла команды в соответствии с вариантом. Была решена проблема вывода 64-битного числа в 32-битной архитектуре. Суммирование и умножение проводилось в строках, складывались символы с соответствующими индексами. Таким образом количество тиков со старта системы хранилось в строке и выводилось посимвольно.

ПРИЛОЖЕНИЕ А

Листинг программы «Bootsect.asm»

[BITS 16]

[ORG 0x7c00]

start:

mov ax, cs

mov ds, ax

mov ss, ax

mov sp, start

mov al, 4

out 0x70, al

in al, 0x71

mov [0x00001000], al

mov al, 2

out 0x70, al

in al, 0x71

mov [0x00001100], al

mov al, 0

out 0x70, al

in al, 0x71

mov [0x00001200], al

mov ax, 0x1000

mov es, ax

mov bx, 0x1000

mov dl, 1

mov dh, 0

mov ch, 0

mov cl, 2

mov al, 10

mov ah, 0x02

int 0x13

mov bx, 0x3000

mov dl, 1

mov dh, 0

mov ch, 0

mov cl, 12

mov al, 7

mov ah, 0x02

int 0x13

cli

lgdt [gdt\_info]

in al, 0x92

or al, 0x02

out 0x92, al

mov eax, cr0

or al, 0x01

mov cr0, eax

jmp 0x8: protected\_mode

[BITS 32]

protected\_mode:

mov ax, 0x10

mov es, ax

mov ds, ax

mov ss, ax

call 0x11000

gdt\_info:

dw gdt\_info - gdt

dw gdt, 0

gdt:

db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

db 0xff, 0xff, 0x00, 0x00, 0x00, 0x9A, 0xCF, 0x00

db 0xff, 0xff, 0x00, 0x00, 0x00, 0x92, 0xCF, 0x00

times (512 - ($ - start) - 2) db 0

db 0x55, 0xAA

ПРИЛОЖЕНИЕ Б

Листинг программы «kernel.cpp»

extern "C" int kmain();

\_\_declspec(naked) void startup()

{

\_\_asm call kmain;

}

void on\_key(int scan\_code);

void on\_key\_backspace();

void on\_key\_enter();

void call\_command();

void command\_handler(char\* str);

int strcmp(char\* s1, char\* s2);

void out\_char(char letter);

void out\_str(const char\* ptr);

void cursor\_moveto();

void clear\_scr();

void next\_line();

static inline void outw(unsigned short port, unsigned short data);

void print\_time(unsigned int time);

unsigned int get\_time(unsigned char mode);

unsigned int boot\_time(unsigned char\* address);

static inline unsigned int rdtsc(int\* r1, int\* r2);

void strcpy(char\* destination, char\* source);

char\* add(char\* str1, char\* str2);

char\* umn(char\* str, int pow);

//Основные функции

void call\_info();

void call\_help();

void call\_ticks();

void call\_loadtime();

void call\_curtime();

void call\_uptime();

void call\_cpuid();

void call\_shutdown();

#define TIMEZONE 3

int cur\_line = 0;

int cur\_col = 0;

int cur\_color = 0x07;

const char SCAN\_CODES[] = "\0\0001234567890-\0\0\0qwertyuiop\0\0\0\0asdfghjkl\0\0\0\0\0zxcvbnm\0\0/\0\*\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0+ ";

#define VIDEO\_BUF\_PTR 0xB8000

#define CURSOR\_PORT 0x3D4

#define VIDEO\_WIDTH 80

#define VIDEO\_HEIGHT 25

#define MAX\_STR 41

#define KEY\_BACKSPACE 14

#define KEY\_ENTER 28

#define IDT\_TYPE\_INTR (0x0E)

#define IDT\_TYPE\_TRAP (0x0F)

// Селектор секции кода, установленный загрузчиком ОС

#define GDT\_CS 0x8

#define PIC1\_PORT (0x20)

// Структура описывает данные об обработчике прерывания

#pragma pack(push, 1) // Выравнивание членов структуры запрещено

struct idt\_entry

{

unsigned short base\_lo; // Младшие биты адреса обработчика

unsigned short segm\_sel; // Селектор сегмента кода

unsigned char always0; // Этот байт всегда 0

unsigned char flags; // Флаги тип. Флаги: P, DPL, Типы - это константы - IDT\_TYPE...

unsigned short base\_hi; // Старшие биты адреса обработчика

};

// Структура, адрес которой передается как аргумент команды lidt

struct idt\_ptr

{

unsigned short limit;

unsigned int base;

};

#pragma pack(pop)

struct idt\_entry g\_idt[256]; // Реальная таблица IDT

struct idt\_ptr g\_idtp;

\_\_declspec(naked) void default\_intr\_handler()

{

\_\_asm {

pusha

}

\_\_asm {

popa

iretd

}

}

typedef void (\*intr\_handler)();

void intr\_reg\_handler(int num, unsigned short segm\_sel, unsigned short

flags, intr\_handler hndlr)

{

unsigned int hndlr\_addr = (unsigned int)hndlr;

g\_idt[num].base\_lo = (unsigned short)(hndlr\_addr & 0xFFFF);

g\_idt[num].segm\_sel = segm\_sel;

g\_idt[num].always0 = 0;

g\_idt[num].flags = flags;

g\_idt[num].base\_hi = (unsigned short)(hndlr\_addr >> 16);

}

// Функция инициализации системы прерываний: заполнение массива с адресами

//обработчиков

void intr\_init()

{

int i;

int idt\_count = sizeof(g\_idt) / sizeof(g\_idt[0]);

for (i = 0; i < idt\_count; i++)

intr\_reg\_handler(i, GDT\_CS, 0x80 | IDT\_TYPE\_INTR,

default\_intr\_handler); // segm\_sel=0x8, P=1, DPL=0, Type=Intr

}

void intr\_start()

{

int idt\_count = sizeof(g\_idt) / sizeof(g\_idt[0]);

g\_idtp.base = (unsigned int)(&g\_idt[0]);

g\_idtp.limit = (sizeof(struct idt\_entry) \* idt\_count) - 1;

\_\_asm {

lidt g\_idtp

}

//\_\_lidt(&g\_idtp);

}

void intr\_enable()

{

\_\_asm sti;

}

void intr\_disable()

{

\_\_asm cli;

}

\_\_inline unsigned char inb(unsigned short port)

{

unsigned char data;

\_\_asm {

push dx

mov dx, port

in al, dx

mov data, al

pop dx

}

return data;

}

\_\_inline void outb(unsigned short port, unsigned char data)

{

\_\_asm {

push dx

mov dx, port

mov al, data

out dx, al

pop dx

}

}

void keyb\_process\_keys()

{

// Проверка что буфер PS/2 клавиатуры не пуст (младший бит

// присутствует)

if (inb(0x64) & 0x01)

{

unsigned char scan\_code;

unsigned char state;

scan\_code = inb(0x60); // Считывание символа с PS/2 клавиатуры

if (scan\_code < 128) // Скан-коды выше 128 - это отпускание клавиши

on\_key(scan\_code);

}

}

\_\_declspec(naked) void keyb\_handler()

{

\_\_asm pusha;

// Обработка поступивших данных

keyb\_process\_keys();

// Отправка контроллеру 8259 нотификации о том, что прерывание

// обработано.Если не отправлять нотификацию, то контроллер не будет посылать

// новых сигналов о прерываниях до тех пор, пока ему не сообщать что

// прерывание обработано.

outb(PIC1\_PORT, 0x20);

\_\_asm {

popa

iretd

}

}

void keyb\_init()

{

// Регистрация обработчика прерывания

intr\_reg\_handler(0x09, GDT\_CS, 0x80 | IDT\_TYPE\_INTR, keyb\_handler);

// segm\_sel=0x8, P=1, DPL=0, Type=Intr

// Разрешение только прерываний клавиатуры от контроллера 8259

outb(PIC1\_PORT + 1, 0xFF ^ 0x02); // 0xFF - все прерывания, 0x02 - бит IRQ1(клавиатура).

// Разрешены будут только прерывания, чьи биты установлены в 0

}

void on\_key(int scan\_code)

{

switch (scan\_code)

{

case KEY\_BACKSPACE:

on\_key\_backspace();

break;

case KEY\_ENTER:

on\_key\_enter();

break;

default:

{

out\_char(SCAN\_CODES[scan\_code]);

}

}

}

void on\_key\_backspace()

{

if (cur\_col == 0)

return;

cur\_col--;

unsigned char\* video\_buf = (unsigned char\*)VIDEO\_BUF\_PTR;

video\_buf += 2 \* (VIDEO\_WIDTH \* cur\_line + cur\_col);

video\_buf[0] = ' ';

cursor\_moveto();

}

void on\_key\_enter()

{

if (cur\_line == 24)

{

clear\_scr();

cur\_col = 0;

cur\_line = 0;

cursor\_moveto();

call\_command();

return;

}

cur\_col = 0;

cur\_line++;

cursor\_moveto();

call\_command();

}

void call\_command()

{

unsigned char\* video\_buf = (unsigned char\*)VIDEO\_BUF\_PTR;

video\_buf += 2 \* (VIDEO\_WIDTH \* (cur\_line - 1));

char str[MAX\_STR];

for (int i = 0; i < 40; i++)

str[i] = video\_buf[2 \* i];

str[MAX\_STR - 1] = '\0';

command\_handler(str);

char info[MAX\_STR] = "info";

char help[MAX\_STR] = "help";

char clear[MAX\_STR] = "clear";

char ticks[MAX\_STR] = "ticks";

char loadtime[MAX\_STR] = "loadtime";

char curtime[MAX\_STR] = "curtime";

char uptime[MAX\_STR] = "uptime";

char cpuid[MAX\_STR] = "cpuid";

char shutdown[MAX\_STR] = "shutdown";

if (strcmp(str, info) == 1)

call\_info();

else if (strcmp(str, help) == 1)

call\_help();

else if (strcmp(str, clear) == 1)

clear\_scr();

else if (strcmp(str, ticks) == 1)

call\_ticks();

else if (strcmp(str, loadtime) == 1)

call\_loadtime();

else if (strcmp(str, curtime) == 1)

call\_curtime();

else if (strcmp(str, uptime) == 1)

call\_uptime();

else if (strcmp(str, cpuid) == 1)

call\_cpuid();

else if (strcmp(str, shutdown) == 1)

call\_shutdown();

else

out\_str("Invalid command");

}

void command\_handler(char\* str)

{

for (int i = MAX\_STR; i >= 1; i--)

{

if (str[i - 1] == ' ' && (str[i] == ' ' || str[i] == '\0'))

{

for (int j = i - 1; j < MAX\_STR; j++)

str[j] = str[j + 1];

}

}

while (str[0] == '\0' || str[0] == ' ')

for (int i = 0; i < MAX\_STR - 1; i++)

str[i] = str[i + 1];

}

int strcmp(char\* s1, char\* s2)

{

for (int i = 0; i < MAX\_STR; i++)

if (s1[i] != s2[i])

return 0;

return 1;

}

void call\_info()

{

out\_str("Vologdin Matvey 4851003/2 IKiZI SPbSTU 2022");

out\_str("Yasm, Intel syntax and Microsoft C compiler");

}

void call\_help()

{

out\_str("info - Information about author and development tools");

out\_str("help - Show information about available commands");

out\_str("clear - Clear the screen");

out\_str("ticks - Displays the number of ticks since system startup");

out\_str("loadtime - Show time, when OS started");

out\_str("curtime - Show the current time");

out\_str("uptime - Displays the total system uptime");

out\_str("cpuid - Specifies the processor model");

out\_str("shutdown - Shutdown computer");

}

void call\_ticks()

{

out\_str("Number of ticks:");

int edx, eax;

unsigned int bit = 0, offset, tmp, c\_flag = 0, k;

unsigned int ticks = rdtsc(&edx, &eax);

char str[21];

char res[21];

for (int i = 0; i < 19; i++)

str[i] = '0';

str[19] = '1';

str[20] = '\0';

for (int i = 0; i < 20; i++)

res[i] = '0';

res[20] = '\0';

while (bit < 64)

{

if (bit < 32)

{

offset = 1 << bit;

if (offset & eax)

{

strcpy(str, umn(str, bit));

strcpy(res, add(res, str));

}

}

else

{

offset = 1 << (bit - 32);

if (offset & edx)

{

strcpy(str, umn(str, bit));

strcpy(res, add(res, str));

}

}

bit++;

}

k = 0;

while (res[k] == '0')

k++;

while (k < 20)

{

out\_char(res[k]);

k++;

}

next\_line();

}

void strcpy(char\* destination, char\* source)

{

int i = 0;

while (source[i] != '\0')

{

destination[i] = source[i];

i++;

}

destination[i] = '\0';

}

char\* add(char\* str1, char\* str2)

{

int flag\_c = 0, t1, t2;

for (int i = 19; i >= 0; i--)

{

t1 = str1[i] - '0';

t2 = str2[i] - '0';

str1[i] = (t1 + t2 + flag\_c) % 10 + '0';

flag\_c = (t1 + t2 + flag\_c) / 10;

}

return str1;

}

char\* umn(char\* str, int pow)

{

char tmp[21], dop[21];

int cur\_dig, t, flag\_c = 0;

tmp[20] = '\0';

tmp[19] = '1';

for (int i = 0; i < 19; i++)

tmp[i] = '0';

while (pow > 0)

{

cur\_dig = 19;

flag\_c = 0;

while (cur\_dig >= 0)

{

t = tmp[cur\_dig] - '0';

tmp[cur\_dig] = (t + t) % 10 + '0' + flag\_c;

flag\_c = (t + t + flag\_c) / 10;

cur\_dig--;

}

pow--;

}

return tmp;

}

void call\_loadtime()

{

out\_str("Loadtime:");

unsigned int time = boot\_time((unsigned char\*)(0x00001000));

print\_time(time);

out\_char(':');

time = boot\_time((unsigned char\*)(0x00001100));

print\_time(time);

out\_char(':');

time = boot\_time((unsigned char\*)(0x00001200));

print\_time(time);

next\_line();

}

unsigned int boot\_time(unsigned char\* address)

{

unsigned char\* memory = (unsigned char\*)address;

unsigned char x = memory[0];

unsigned int answer = x - 6 \* (x >> 4);

if ((int)address == 0x00001000)

{

answer = answer + TIMEZONE;

answer = answer % 24;

}

return answer;

}

void call\_curtime()

{

out\_str("Current time:");

print\_time(get\_time(4));

out\_char(':');

print\_time(get\_time(2));

out\_char(':');

print\_time(get\_time(0));

next\_line();

}

void print\_time(unsigned int time)

{

unsigned int high = time / 10;

char c\_high = high + '0';

out\_char(c\_high);

unsigned int low = time % 10;

char c\_low = low + '0';

out\_char(c\_low);

}

unsigned int get\_time(unsigned char mode)

{

outb(0x70, mode);

unsigned char x = inb(0x71);

unsigned int answer = x - 6 \* (x >> 4);

if (mode == 4)

{

answer += TIMEZONE;

answer = answer % 24;

}

return answer;

}

void call\_uptime()

{

out\_str("Uptime:");

unsigned int time = boot\_time((unsigned char\*)(0x00001000));

print\_time(get\_time(4) - time);

out\_char(':');

time = boot\_time((unsigned char\*)(0x00001100));

print\_time(get\_time(2) - time);

out\_char(':');

time = boot\_time((unsigned char\*)(0x00001200));

print\_time(get\_time(0) - time);

next\_line();

}

void call\_cpuid()

{

int id;

\_\_asm {

xor eax, eax

cpuid

mov id, ebx

}

if (id == 0x756E6547)

out\_str("VendorId: GenuineIntel");

else if (id == 0x68747541)

out\_str("VendorId: AuthenticAMD");

else if (id == 0x69727943)

out\_str("VendorId: CyrixInstead");

else if (id == 0x746E6543)

out\_str("VendorId: CentaurHauls");

else if (id == 0x20536953)

out\_str("VendorId: SiS SiS SiS");

else if (id == 0x4778654E)

out\_str("VendorId: NexGenDriven");

else if (id == 0x756E6547)

out\_str("VendorId: GenuineTMx86");

else if (id == 0x65736952)

out\_str("VendorId: RiseRiseRise");

else if (id == 0x20434D55)

out\_str("VendorId: UMC UMC UMC");

else if (id == 0x646F6547)

out\_str("VendorId: Geode by NSC");

else out\_str("VendorId: Unknown");

}

void call\_shutdown()

{

outw(0x0604, 0x2000);

}

static inline unsigned int rdtsc(int\* r1, int\* r2)

{

int tmp1, tmp2;

\_\_asm {

rdtsc

mov tmp1, eax

mov tmp2, edx

}

\*(r1) = tmp1;

\*(r2) = tmp2;

return 1;

}

static inline void outw(unsigned short port, unsigned short data)

{

\_\_asm {

push dx

mov dx, port

mov ax, data

out dx, ax

pop dx

}

}

void out\_char(char letter)

{

if (cur\_col > 39)

return;

unsigned char\* video\_buf = (unsigned char\*)VIDEO\_BUF\_PTR;

video\_buf += 2 \* (VIDEO\_WIDTH \* cur\_line + cur\_col);

video\_buf[0] = letter;

video\_buf[1] = cur\_color;

cur\_col++;

cursor\_moveto();

}

void out\_str(const char\* ptr)

{

cur\_col = 0;

unsigned char\* video\_buf = (unsigned char\*)VIDEO\_BUF\_PTR;

video\_buf += 80 \* 2 \* cur\_line;

while (\*ptr)

{

video\_buf[0] = (unsigned char)\*ptr;

video\_buf[1] = cur\_color;

video\_buf += 2;

ptr++;

}

cur\_line++;

if (cur\_line >= 24)

clear\_scr();

cursor\_moveto();

}

void cursor\_moveto()

{

unsigned short new\_pos = (cur\_line \* VIDEO\_WIDTH) + cur\_col;

outb(CURSOR\_PORT, 0x0F);

outb(CURSOR\_PORT + 1, (unsigned char)(new\_pos & 0xFF));

outb(CURSOR\_PORT, 0x0E);

outb(CURSOR\_PORT + 1, (unsigned char)((new\_pos >> 8) & 0xFF));

}

void next\_line()

{

cur\_col = 0;

cur\_line++;

cursor\_moveto();

}

void clear\_scr()

{

unsigned char\* video = (unsigned char\*)VIDEO\_BUF\_PTR;

int count = 0;

while (count < VIDEO\_HEIGHT \* VIDEO\_WIDTH)

{

video[0] = ' ';

video[1] = cur\_color;

video += 2;

count++;

}

cur\_col = 0;

cur\_line = 0;

cursor\_moveto();

}

extern "C" int kmain()

{

clear\_scr();

out\_str("Welcome to InfoOS!");

intr\_init();

intr\_start();

intr\_enable();

keyb\_init();

while (1)

{

\_\_asm hlt;

}

return 0;

}