1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. [Институт компьютерных наук и кибербезопасности](https://dl.spbstu.ru/)

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. **«Принципы разработки операционных систем»**
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 5151004/30001 Лясович С.Д.

<*подпись*>

Преподаватель Югай П.Э.

<*подпись*>

1. Санкт-Петербург
2. 2025
3. **Цель работы**

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

1. **Постановка задачи**

В ходе данной лабораторной работы необходимо реализовать операционную систему, выполняющую вычисление простых выражений. Для этого необходимо создать файлы “bootsect.asm”-файл загрузчика ядра на языке ассемблер и“kernel.cpp”-файл самого ядра.

Общие требования к загрузчику:

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.

Общие требования к ядру ОС:

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.

Ядро необходимо было реализовать в соответствии со следующими требованиями:

Функция загрузчика — запрос у пользователя цвета букв консоли, который будет использоваться после загрузки ядра ОС. Должны поддерживаться цвета: green, blue, red, yellow, gray, white. Загрузчик устанавливает каждому цвету в соответствие цифру и распечатывает эту информацию, например: "Press: 1-green,2-blue…". Пользователь нажимает цифровую клавишу и загрузчик немедленно, без подтверждений от пользователя, переходит к загрузке ОС.

Поддерживаемые ОС команды:

#info

Выводит информацию об авторе и средствах разработки ОС (транслятор ассемблера, компилятор) и цифровой кнопке, указанной пользователем при загрузке.

# expr

Вычисляет значение выражение с учетом приоритета операций и выводит его на экран.

Выражение содержит числа и знаки +-\*/ Должны поддерживаться унарные операции + и -. Деление является целочисленным (не вещественным).

В случае указания некорректного выражения выводится ошибка. В случае попытки делить на 0 — выводится ошибка. В случае возникновения переполнения целочисленного значения int — выводится ошибка. Перед числом может быть несколько унарных минусов.

# shutdown

Выключение компьютера.

1. **Описание алгоритма работы загрузчика**

При запуске команды из командной строки, происходит запуск кода загрузчика. В коде загрузчика реализовано работа с прерываниями, вывод строки с параметрами, переход в защищённый режим, передача управления ядру. На рисунках ниже представлена реализация основных функций загрузчика. Так же, на рисунке 3 представлена блок-схема реализованного кода загрузчика.

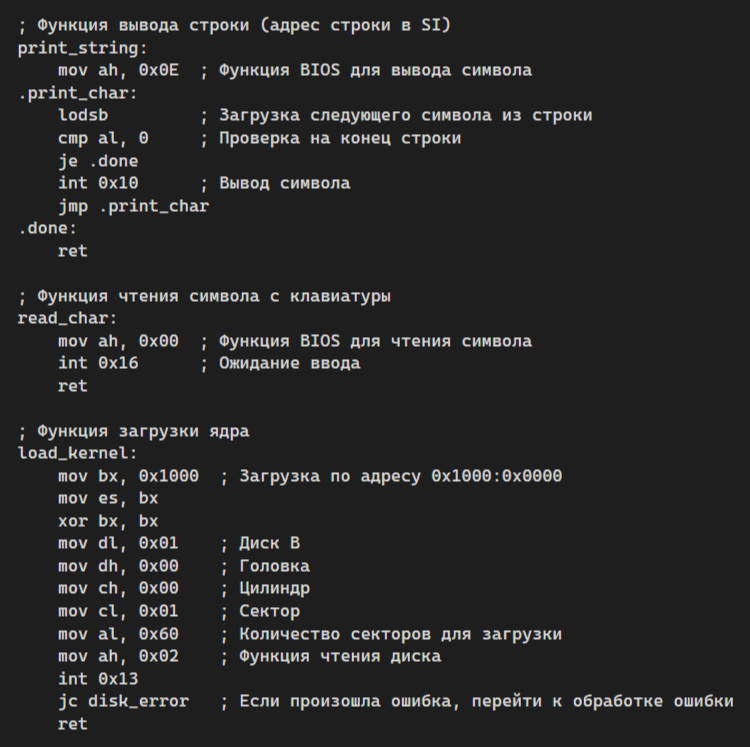


Рисунок 1 – Реализация функций выводящей строку, считывающей символ и загружающей ядро в файле ассемблера

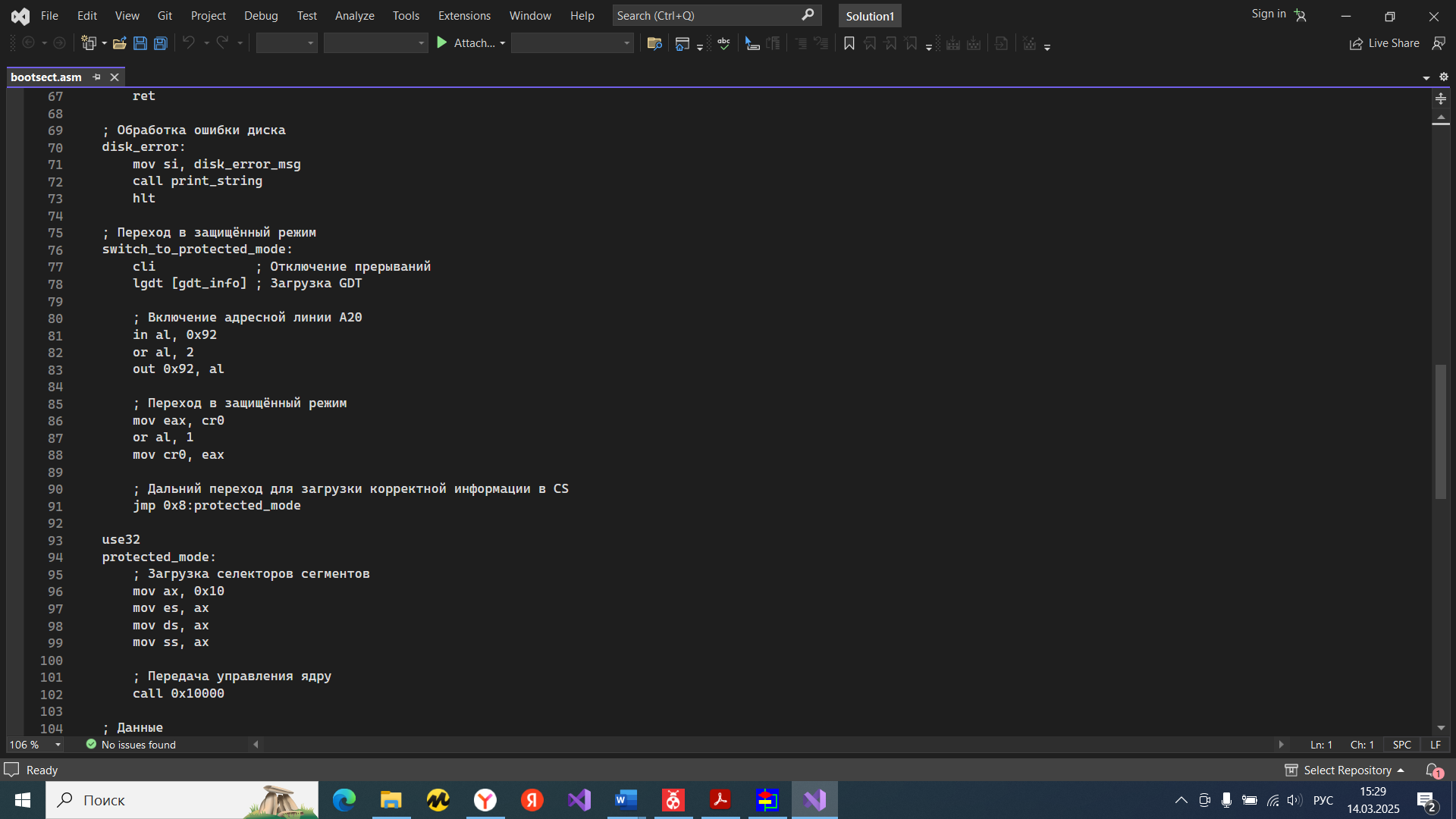


Рисунок 2 – Реализация функции, осуществляющей переход в защищённый режим

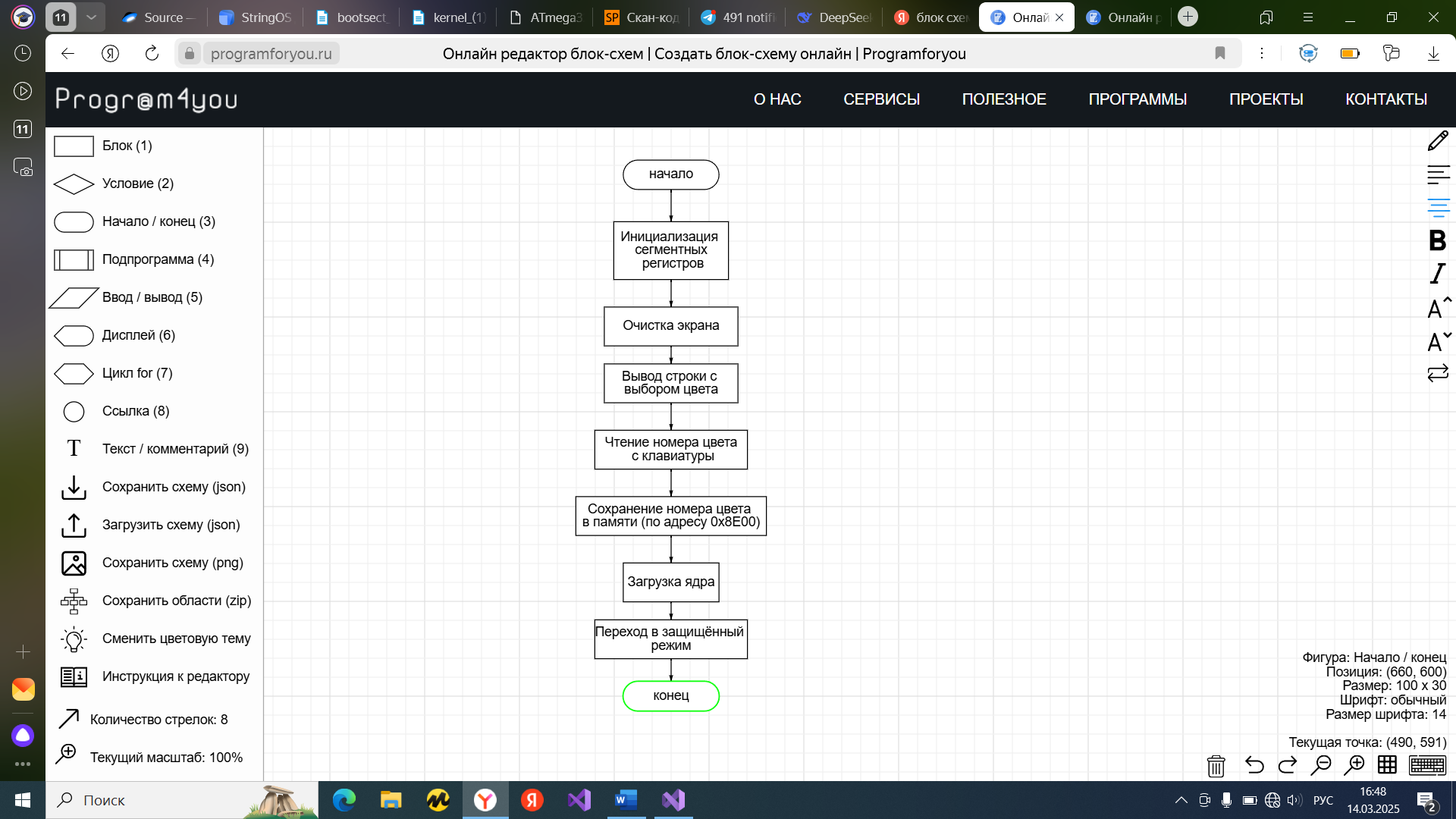


Рисунок 3 – Блок-схема кода загрузчика

1. **Описание алгоритма работы ядра**

Как только загрузчик считывает символ из строки, программа переходит к загрузке ядра. В ходе работы ядра происходит считывание команд из консоли, проверка команды на корректность и сравнение их с поддерживаемыми командами. Если считанная строка содержит в себе недопустимые сочетания символов, то выводится ошибка, иначе вызывается функция для выполнения этой команды. Ниже, в таблице 1, представлены основные функции обработки считанной строки.

Таблица 1 – Таблица функций, реализованных в коде ядра

|  |  |
| --- | --- |
| Название функции и её аргументы | Назначение функции |
| void on\_key(unsigned char code); | Обрабатывает скан-кода нажатых клавиш. |
| void same(const char\* string); | Сравнивает полученную команду с системными командами и в случае совпадения вызывает функции обрабатывающие конкретные случаи. |
| void expr(); | Перезаписывает строку убирая из неё часть, отвечающую за идентификацию команды. Передаёт строку в функцию проверки корректности. |
| int correct(char\* new\_str); | Проверяет строку, содержащую математическое выражение на корректность. При наличии ошибок, выводит соответствующие сообщения в консоль. |
| void calculator (char\* new\_str); | Обрабатывает арифметическое выражение в соответствии с приоритетами операций, получает ответ в виде строки. |
| int sr1(int array[10],char\* new\_str,int ww); | Преобразовывает часть строки левее знака в число. |
| int sr2(int array[10],char\* new\_str,int ww); | Преобразовывает часть строки правее знака в число. |
| void rezult\_in\_str(int symb, int array[10], char\* new\_str, int rezult,int full,int binflag); | Преобразовывает результат операции из числового вида в строку. |
| void shutdown(void); | Инициирует выключение системы. |

1. **Результаты тестирования программы**

В ходе тестирования были проведены 15 тстов. Их результаты представлены на риснках 4 и 5.

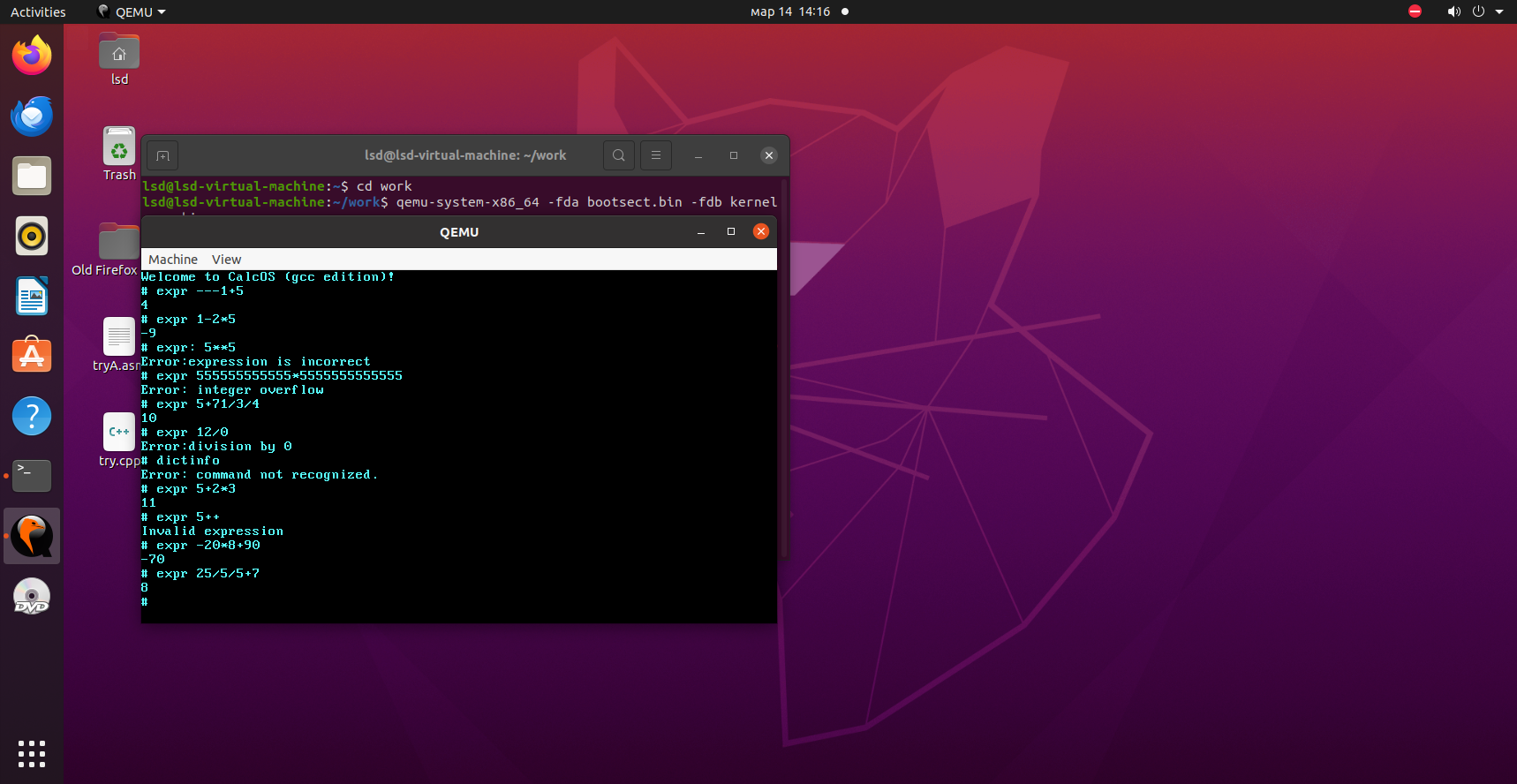


Рисунок 4 – Тестирование программы часть 1

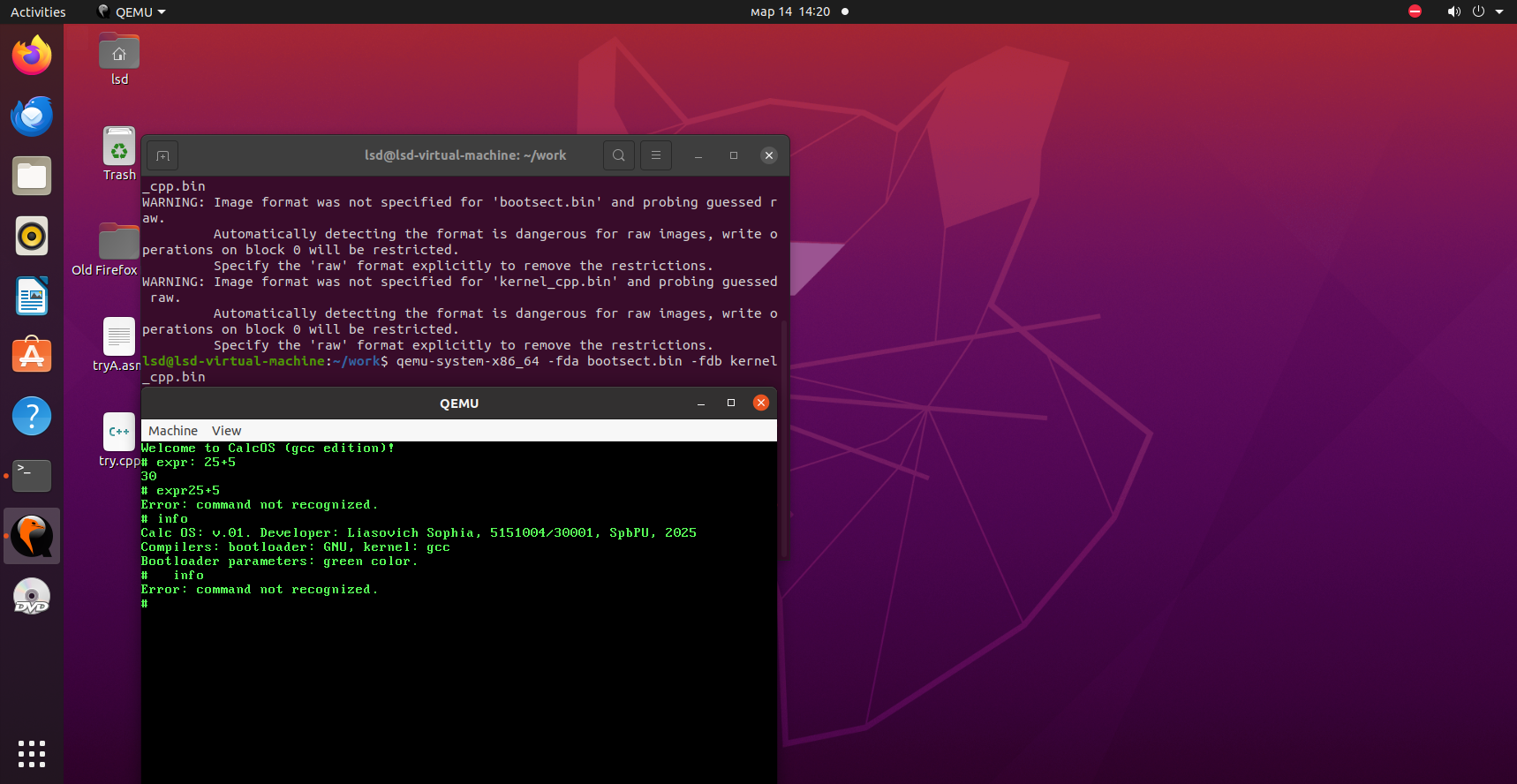


Рисунок 5 – Тестирование программы часть 2

1. **Выводы**

В ходе данной лабораторной работы были изучены основы разработки ОС, принципов низкоуровневого взаимодействия с аппаратным обеспечением, программирования системной функциональности и процесса загрузки системы.

**ПРИЛОЖЕНИЕ А**

Листинг программы «kernel.cpp»

\_\_asm("jmp kmain");

#define VIDEO\_BUF\_PTR (0xb8000)

#define IDT\_TYPE\_INTR (0x0E)

#define IDT\_TYPE\_TRAP (0x0F)

#define GDT\_CS (0x8)

#define CURSOR\_PORT (0x3D4)

#define VIDEO\_WIDTH (80)

#define PIC1\_PORT (0x20)

#define NULL 0

#define VIDEO\_HIGH 25

#include <stdint.h>

unsigned int pos=0;

unsigned int counter=0;

int c=0x0A;

int number = 0;

char str[41];

void calculator (char\* new\_str);

void cursor\_moveto(unsigned int strnum, unsigned int pos);

const char alphabet[58]={0 , 0, '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '+', 0, 0, 'q', 'w', 'e', 'r', 't', 'y', 'u', 'i','o', 'p', '[', ']', 0, 0, 'a', 's', 'd', 'f', 'g', 'h','j', 'k', 'l', ':', '\'', '`', 0, '\\', 'z', 'x', 'c','v', 'b', 'n', 'm', ',', '.', '/', 0,'\*',0,' '};

struct idt\_entry

{

unsigned short base\_lo;

unsigned short segm\_sel;

unsigned char always0;

unsigned char flags;

unsigned short base\_hi;

} \_\_attribute\_\_((packed));

struct idt\_ptr

{

unsigned short limit;

unsigned int base;

} \_\_attribute\_\_((packed));

struct idt\_entry g\_idt[256];

struct idt\_ptr g\_idtp;

extern "C" void keyb\_handler(void);

extern "C" void keyb\_process\_keys(void);

extern "C" void default\_intr\_handler(void);

\_\_asm\_\_(".global default\_intr\_handler\n"

"default\_intr\_handler:\n\t"

"iret");

/\* Define a keyboard handler stub that makes a call to a C function \*/

\_\_asm\_\_(".global keyb\_handler\n"

"keyb\_handler:\n\t"

"cld\n\t" /\* Set direction flag forward for C functions \*/

"pusha\n\t" /\* Save all the registers \*/

"call keyb\_process\_keys\n\t"

"popa\n\t" /\* Restore all the registers \*/

"iret");

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

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

}

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);

}

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 %0" : : "m" (g\_idtp));

}

void intr\_enable() {

asm("sti");

}

void intr\_disable() {

asm("cli");

}

void out\_str(int color, const char\* ptr, unsigned int strnum) {

unsigned char\* video\_buf = (unsigned char\*)VIDEO\_BUF\_PTR + 80 \* 2 \* strnum;

while (\*ptr)

{

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

video\_buf[1] = color;

video\_buf += 2;

ptr++;

}

if(counter>24){

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

for(int i = 0; i < VIDEO\_WIDTH\*25; i++) {

video\_buf[0]=' ';

video\_buf[1]=c;

video\_buf+=2;

//\*(video\_buf + i\*2) = '\0';

}

counter=0;

pos=0;

cursor\_moveto(counter,pos);

const char\* hello = "Welcome to CalcOS (gcc edition)!";

out\_str(c, hello, 0);

if (counter==25){counter=1;}

//const char\* new\_line = "# ";

//out\_str(c, new\_line, counter);

//cursor\_moveto(counter,pos);

}

}

void char\_out(const char\* ptr) {

unsigned char\* video\_buf = (unsigned char\*)VIDEO\_BUF\_PTR + 2 \* (80 \* counter) + pos \* 2;

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

video\_buf[1] = c;

}

static inline unsigned char inb(unsigned short port) {

unsigned char data;

asm volatile ("inb %w1, %b0" : "=a" (data) : "Nd" (port));

return data;

}

static inline void outb(unsigned short port, unsigned char data) {

asm volatile ("outb %b0, %w1" : : "a" (data), "Nd" (port));

}

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

asm volatile ("outw %w0, %w1" : : "a" (data), "Nd" (port));

}

void cursor\_moveto(unsigned int strnum, unsigned int pos) {

unsigned short new\_pos = (strnum \* VIDEO\_WIDTH) + pos;

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 shutdown(void) {

outw(0xB004, (unsigned short)0x2000); // qemu < 1.7, ex. 1.6.2

outw(0x604, (unsigned short)0x2000); // qemu >= 1.7

}

int size(const char\* size\_str){

int i=0;

while(size\_str[i]!='\0'|| size\_str[i]!=' '){

i++;

}

return i;

}

char i\_t\_c(int num){

if(num==1){

return '1';

}

else if(num==2){

return '2';

}

else if(num==3){

return '3';

}

else if(num==4){

return '4';

}

else if(num==5){

return '5';

}

else if(num==6){

return '6';

}

else if(num==7){

return '7';

}

else if(num==8){

return '8';

}

else if(num==9){

return '9';

}

else {return '0'; }

}

int c\_t\_i(char ch) {

if (ch == '1') { return 1; }

else if (ch == '2') { return 2; }

else if (ch == '3') { return 3; }

else if (ch == '4') { return 4; }

else if (ch == '5') { return 5; }

else if (ch == '6') { return 6; }

else if (ch == '7') { return 7; }

else if (ch == '8') { return 8; }

else if (ch == '9') { return 9; }

else { return 0; }

}

int char\_to\_int(int start, int end, char\* work\_str) {

int mn = 10;

int help = end - start-1 ;

if(help==-1){mn=1;}

while (help > 0) {

mn = mn \* 10;

help--;

}

int rezult = 0;

int number = 0;

while (start != end + 1) {

number = c\_t\_i(work\_str[start]);

rezult = rezult + (mn \* number);

if(mn>=10){mn = mn / 10;}

start++;

}

return rezult;

}

int over(int k, char\* new\_str){ //chek for overflow number

int number=char\_to\_int(k,k+9,new\_str);

if(number>2147483647){

counter++;

out\_str(c,"numbermore",counter);

return 0;}

else {return 1;}

}

int over1(int k, char\* new\_str){

if(c\_t\_i(new\_str[k])<2){return 1;}

else if(c\_t\_i(new\_str[k])>2){return 0;}

else if(c\_t\_i(new\_str[k+1])<1){return 1;}

else if(c\_t\_i(new\_str[k+1])>1){return 0;}

else if(c\_t\_i(new\_str[k+2])<4){return 1;}

else if(c\_t\_i(new\_str[k+2])>4){return 0;}

else if(c\_t\_i(new\_str[k+3])<7){return 1;}

else if(c\_t\_i(new\_str[k+3])>7){return 0;}

else if(c\_t\_i(new\_str[k+4])<4){return 1;}

else if(c\_t\_i(new\_str[k+4])>4){return 0;}

else if(c\_t\_i(new\_str[k+5])<8){return 1;}

else if(c\_t\_i(new\_str[k+5])>8){return 0;}

else if(c\_t\_i(new\_str[k+6])<3){return 1;}

else if(c\_t\_i(new\_str[k+6])>3){return 0;}

else if(c\_t\_i(new\_str[k+7])<6){return 1;}

else if(c\_t\_i(new\_str[k+7])>6){return 0;}

else if(c\_t\_i(new\_str[k+8])<4){return 1;}

else if(c\_t\_i(new\_str[k+8])>4){return 0;}

else if(c\_t\_i(new\_str[k+9])<7){return 1;}

else if(c\_t\_i(new\_str[k+9])>7){return 0;}

else {return 1;}

}

int correct(char\* new\_str){

int array[100];

int c\_ar=0;

int i=0;

int c\_numb=0;

for(i;new\_str[i]!='\0';i++){

if(new\_str[i]=='/' && new\_str[i+1]=='0'){ //situation /0

counter++;

out\_str(c,"Error:division by 0",counter);

return 0;

}

if(((new\_str[i]=='/' || new\_str[i]=='+' || new\_str[i]=='\*' || new\_str[i]=='-') && new\_str[i+1]=='\0')|| ((new\_str[i]=='/' || new\_str[i]=='\*') && i==0) ){ //situation where +-+/ first or last

counter++;

out\_str(c,"Invalid expression",counter);

return 0;

} //we rite index of numbers in array

if(new\_str[i]=='1' || new\_str[i]=='2' || new\_str[i]=='3' || new\_str[i]=='4' || new\_str[i]=='5' || new\_str[i]=='6' || new\_str[i]=='7' || new\_str[i]=='8' || new\_str[i]=='9' || new\_str[i]=='0'){

array[c\_ar]=i;

c\_ar++;

}

}

for(int j=0;j<c\_ar-1;j++){ //situation:5647++857574

if((array[j+1]-array[j])>2){

counter++;

out\_str(c,"Error:expression is incorrect",counter);

return 0;

}

if((array[j+1]-array[j])==1){ //situation overflow

if(c\_numb==8){

int k=j-8+1; //start of big number

if(k==1){k=0;}

int fl2=over1(k,new\_str);

if(fl2==0){

counter++;

out\_str(c,"Error: integer overflow",counter);

return 0;

}

}

c\_numb++;

}

else{c\_numb=0;}

}

return 1;

}

int sr2(int array[10],char\* new\_str,int ww){ //array with index if operation; str;number of operation

char second[21]={};

int start=0;

//counter++;

int midl=array[ww];

//char\_out(&new\_str[midl]);

midl++;

for(midl;new\_str[midl]!='+' && new\_str[midl]!='-'&& new\_str[midl]!='\*'&& new\_str[midl]!='/'&& new\_str[midl]!='\0';midl++){

}

int i=array[ww]+1;

int j=0;

for(i;i!=midl;i++){

second[j]=new\_str[i];

j++;

}

second[j]='\0';

int output2=char\_to\_int(0,j-1,second);

return output2; //i make second number from str tupe to int tupe

}

void binary (){counter++; out\_str(c,"10",counter);}

int sr1(int array[10],char\* new\_str,int ww){ //array with index if operation; str;number of operation

char first[21]={};

int start=0;

//counter++;

int midl=array[ww];

//char\_out(&new\_str[midl]);

midl--;

for(midl;new\_str[midl]!='+' && new\_str[midl]!='-'&& new\_str[midl]!='\*'&& new\_str[midl]!='/'&& midl!=0;midl--){

if(new\_str[midl]=='-'&& midl==0){midl++;}

}

int i=0;

for(midl;midl!=array[ww];midl++){

first[i]=new\_str[midl];

i++;

}

first[i]='\0';

int output1=char\_to\_int(0,i-1,first);

return output1; //i make first number from str tupe to int tupe

}

void rezult\_in\_str(int symb, int array[10], char\* new\_str, int rezult,int full,int binflag){

int midl=array[symb];

int i=midl-1;

//if(rezult==23){

// counter++;

//out\_str(c,"rez23",counter);

//}

char output\_str[40];

int j=0;

int tail=0;

int t=midl+1;

if(full>1){

for(i;new\_str[i]!='-' && new\_str[i]!='+' && new\_str[i]!='\*' && new\_str[i]!='/' && i!=0;i--){}

for(t;new\_str[t]!='-' && new\_str[t]!='+' && new\_str[t]!='\*' && new\_str[t]!='/' && new\_str[t]!='\0';t++){

if(new\_str[t]!='\0'){tail=1;}

}

for(j; j!=i+1 && i!=0;j++){

output\_str[j]=new\_str[j];

}

}

if(binflag==1){

output\_str[j]='-';

j++;

}

int k=10;

if(rezult/k>9){

k=k\*10;

}

if(rezult<10){k=1;}

while(k>9){

char help=i\_t\_c(rezult/k);

output\_str[j]=help;

rezult=rezult-((rezult/k)\*k);

j++;

k=k/10;

}

output\_str[j]=i\_t\_c(rezult);

j++;

if(tail==0){output\_str[j]='\0';}

else{

while(new\_str[t]!='\0'){

output\_str[j]=new\_str[t];

j++;

t++;

}

output\_str[j]='\0';

}

full--;

if(full>0){

//counter++;

//out\_str(c,"fuul>0",counter);

calculator(output\_str);

}

else {

counter++;

out\_str(c,output\_str,counter);

}

}

void calculator (char\* new\_str){

if((new\_str[0]=='-'&& new\_str[1]=='-')||(new\_str[0]=='+'&& new\_str[1]=='+')){

int i=0;

int j=0;

for(i;(new\_str[i]=='-')|| (new\_str[i]=='+');i++){}

if(i%2==0){

while(new\_str[i]!='\0'){

new\_str[j]=new\_str[i];

i++;

j++;

}

new\_str[j]='\0';

}

else{

i--;

if(new\_str[i]=='+'){i++;}

while(new\_str[i]!='\0'){

new\_str[j]=new\_str[i];

i++;

j++;

}

new\_str[j]='\0';

}

}

int mult[10];

int c\_m=0;

int sum[10];

int s\_m=0;

int del[10];

int d\_m=0;

int razn[10];

int r\_m=0;

for(int i=0;new\_str[i]!='\0';i++){

if(new\_str[i]=='\*'){

mult[c\_m]=i;

c\_m++;

}

if(new\_str[i]=='/'){

del[d\_m]=i;

d\_m++;

}

if(new\_str[i]=='+'){

sum[s\_m]=i;

s\_m++;

}

if(new\_str[i]=='-'&& i!=0){

razn[r\_m]=i;

r\_m++;

}

}

int full=r\_m+s\_m+d\_m+c\_m;

int num\_m=0;

int binflag=0;

if(c\_m>0){

if(new\_str[0]=='-'){ //vork with bin minus? if it is, we make flag ON and remember if; we work with as usual

binflag=1;

int j=0;

int i=1;

while(new\_str[i]!='\0'){

new\_str[j]=new\_str[i];

i++;

j++;

}

new\_str[j]='\0';

mult[num\_m]=mult[num\_m]-1;

}

int num1=sr1(mult,new\_str,num\_m); //how many;array;str;with whitch we work

int num2=sr2(mult,new\_str,num\_m);

int rezult=num1\*num2;

rezult\_in\_str(num\_m,mult,new\_str,rezult,full,binflag);

num\_m++;

c\_m--;

}

int dum\_m=0;

if (d\_m>0 && c\_m==0 && num\_m==0){

//counter++;

//out\_str(c,new\_str,counter);

int num1=sr1(del,new\_str,dum\_m);

int num2=sr2(del,new\_str,dum\_m);

int rezult=num1/num2;

//full=0;

rezult\_in\_str(dum\_m,del,new\_str,rezult,full,0);

dum\_m++;

d\_m--;

}

int sum\_m=0;

if(s\_m>0 && new\_str[0]!='-'&& d\_m==0 && dum\_m==0 && num\_m==0 && c\_m==0){

int num1=sr1(sum,new\_str,sum\_m);

int num2=sr2(sum,new\_str,sum\_m);

int rezult=num1+num2;

rezult\_in\_str(sum\_m,sum,new\_str,rezult,0,0);

sum\_m++;

s\_m--;

}

int ruzn\_m=0;

if((r\_m>0 || (s\_m>0 && new\_str[0]=='-'))&& d\_m==0 && dum\_m==0 && num\_m==0 && c\_m==0){

//counter++;

//out\_str(c,new\_str,counter);

if(s\_m>0 && new\_str[0]=='-'){

int num1=sr1(sum,new\_str,sum\_m);

int num2=sr2(sum,new\_str,sum\_m);

int rezult=0;

if(num1>num2){

rezult=num1-num2;

binflag=1;

}

else {

rezult=num2-num1;

binflag=0;

}

rezult\_in\_str(sum\_m,sum,new\_str,rezult,0,binflag);

sum\_m++;

s\_m--;

}

else {

int num1=sr1(razn,new\_str,ruzn\_m);

int num2=sr2(razn,new\_str,ruzn\_m);

int rezult=0;

if(num1>num2){

rezult=num1-num2;

binflag=0;

}

else {

rezult=num2-num1;

binflag=1;

}

rezult\_in\_str(ruzn\_m,razn,new\_str,rezult,0,binflag);

}

ruzn\_m++;

r\_m--;

}

}

void expr(){

char new\_str[37];

int h=0;

if(str[4]==' '){h=5;}

else{h=6;}

int l=0;

for(h;str[h]!='\0';h++){

new\_str[l]=str[h];

l++;

}

new\_str[l]='\0';

//out\_str(c,new\_str,counter);

int f=correct(new\_str); //str-good or we have mistakes

if (f==1){

//counter++;

//out\_str(c,"all is good",counter);

calculator(new\_str);

}

}

void same(const char\* string){

const char\* inf="info";

const char\* shut="shutdown";

const char\* exp="expr";

int i=0;

int flag=0;

if(string[0]=='i'){

for(i; inf[i]==string[i];i++)

if(i==3 && string[4]=='\0'){

counter++;

out\_str(c,"Calc OS: v.01. Developer: Liasovich Sophia, 5151004/30001, SpbPU, 2025",counter);

counter++;

out\_str(c,"Compilers: bootloader: GNU, kernel: gcc",counter);

counter++;

const char\* color\_info;

if(c==0x0A){

color\_info="Bootloader parameters: green color.";

}

else if(c==0x0B){

color\_info="Bootloader parameters: blue color.";

}

else if(c==0x0C){

color\_info="Bootloader parameters: red color.";

}

else if(c==0x0E){

color\_info="Bootloader parameters: yellow color.";

}

else if(c==0x07){

color\_info="Bootloader parameters: gray color.";

}

else if(c==0x0F){

color\_info="Bootloader parameters: white color.";

}

flag=1;

out\_str(c,color\_info,counter);

}

}

if(string[0]=='s'){

for(i=0; shut[i]==string[i];i++)

if(i==7 && string[8]=='\0'){

counter++;

flag=1;

out\_str(c,"shutdown",counter);

}

}

if(string[0]=='e'){

for(i=0; exp[i]==string[i];i++)

if(i==3 && (string[4]==':'||string[4]==' ')){

//counter++;

flag=1;

expr();

}

}

else if (string[0]!='i'&& string[0]!='s'&& string[0]!='e') {

counter++;

flag=1;

out\_str(c,"Error: command not recognized.",counter);

}

if(flag==0){

counter++;

out\_str(c,"Error: command not recognized.",counter);

}

}

void on\_key(unsigned char code) {

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

if (code == 14) {

if (number > 0) {

number--;

str[number] = '\0';

}

if (pos > 2) pos--;

char space[2] = " ";

char\_out(space);

cursor\_moveto(counter, pos);

}

else if (code == 28) { //enter

str[number] = '\0';

same(str);

counter++;

pos=2;

cursor\_moveto(counter,pos);

const char\* test= "# ";

out\_str(c, test, counter);

\*str={};

number=0;

return;

}

else {

char symbol;

if(code<58){

symbol=alphabet[code];

}

str[number]=symbol;

number++;

char\_out(&symbol);

pos++;

cursor\_moveto(counter,pos);

}

}

void keyb\_process\_keys() {

if (inb(0x64) & 0x01) {

unsigned char scan\_code;

unsigned char state;

scan\_code = inb(0x60);

if (scan\_code < 128)

on\_key(scan\_code);

}

outb(PIC1\_PORT, 0x20);

}

void keyb\_init() {

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

outb(PIC1\_PORT + 1, 0xFF ^ 0x02);

}

bool is\_valid\_char(uint8\_t symbol)

{

return (symbol >= '0' && symbol <= '9'); // Проверка, что символ — цифра

}

extern "C" int kmain() {

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

uint8\_t symbol = \*(uint8\_t\*)0x8e00;

// Проверяем корректность символа

if (is\_valid\_char(symbol))

{

if(symbol=='1'){

c=0x0A;

}

else if(symbol=='2'){

c=0x0B;

}

else if(symbol=='3'){

c=0x0C;

}

else if(symbol=='4'){

c=0x0E;

}

else if(symbol=='5'){

c=0x07;

}

else if(symbol=='6'){

c=0x0F;

}

}

for(int i = 0; i < VIDEO\_WIDTH\*25; i++)

\*(video\_buf + i\*2) = '\0';

const char\* hello = "Welcome to CalcOS (gcc edition)!";

out\_str(c, hello, 0);

counter=1;

pos=2;

const char\* new\_line = "# ";

out\_str(c, new\_line, counter);

cursor\_moveto(counter,pos);

intr\_disable();

intr\_init();

keyb\_init();

intr\_start();

intr\_enable();

/\*while (1) {

asm("hlt");

}\*/

return 0;

}

**ПРИЛОЖЕНИЕ B**

Листинг программы «bootsect.asm»

use16

org 0x7C00

start:

; Инициализация сегментных регистров

mov ax, cs

mov ds, ax

mov ss, ax

mov sp, start

; Очистка экрана

call clear\_screen

; Вывод строки с выбором цвета

mov si, color\_prompt

call print\_string

; Чтение номера цвета с клавиатуры

call read\_char

; Сохранение номера цвета в памяти (по адресу 0x8E00)

mov [0x8E00], al

; Загрузка ядра

call load\_kernel

; Переход в защищённый режим

jmp switch\_to\_protected\_mode

; Функция очистки экрана

clear\_screen:

mov ax, 0x03

int 0x10

ret

; Функция вывода строки (адрес строки в SI)

print\_string:

mov ah, 0x0E ; Функция BIOS для вывода символа

.print\_char:

lodsb ; Загрузка следующего символа из строки

cmp al, 0 ; Проверка на конец строки

je .done

int 0x10 ; Вывод символа

jmp .print\_char

.done:

ret

; Функция чтения символа с клавиатуры

read\_char:

mov ah, 0x00 ; Функция BIOS для чтения символа

int 0x16 ; Ожидание ввода

ret

; Функция загрузки ядра

load\_kernel:

mov bx, 0x1000 ; Загрузка по адресу 0x1000:0x0000

mov es, bx

xor bx, bx

mov dl, 0x01 ; Диск B

mov dh, 0x00 ; Головка

mov ch, 0x00 ; Цилиндр

mov cl, 0x01 ; Сектор

mov al, 0x60 ; Количество секторов для загрузки

mov ah, 0x02 ; Функция чтения диска

int 0x13

jc disk\_error ; Если произошла ошибка, перейти к обработке ошибки

ret

; Обработка ошибки диска

disk\_error:

mov si, disk\_error\_msg

call print\_string

hlt

; Переход в защищённый режим

switch\_to\_protected\_mode:

cli ; Отключение прерываний

lgdt [gdt\_info] ; Загрузка GDT

; Включение адресной линии A20

in al, 0x92

or al, 2

out 0x92, al

; Переход в защищённый режим

mov eax, cr0

or al, 1

mov cr0, eax

; Дальний переход для загрузки корректной информации в CS

jmp 0x8:protected\_mode

use32

protected\_mode:

; Загрузка селекторов сегментов

mov ax, 0x10

mov es, ax

mov ds, ax

mov ss, ax

; Передача управления ядру

call 0x10000

; Данные

color\_prompt:

db "Press: 1-green,2-blue,3-red,4-yellow,5-gray,6-white:", 0

disk\_error\_msg:

db "Disk error!", 0

; GDT

gdt:

; Нулевой дескриптор

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

; Сегмент кода

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

; Сегмент данных

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

gdt\_info:

dw gdt\_info - gdt - 1 ; Лимит GDT

dd gdt ; Адрес GDT

; Заполнение оставшейся части загрузочного сектора

times 510-($-$$) db 0

dw 0xAA55 ; Сигнатура загрузочного сектора