Математичка гимназија

МАТУРСКИ РАД

- из рачунарства и информатике -

Израда X86 32bit i686 језгра оперативног система

Ученик:
Алекса Вучковић IVд

Ментор:
Милош Арсић

Београд, јун 2021.

Садржај

1

Увод

Идеја за овај рад прозишла је из екстензивног коришћења GNU/Linux система, као и жеља за разумевањем рада рачунара на најнижем нивоу.
Цео код је писан у GNU Asembler-у и C-у и може се наћи на GitHub-u на страници `https://github.com/aleksav013/mykernel`. Сав код је доступан под GPLv3 лиценцом.

2

X86 архитектура

X86 архитектура је пробитно била осмобитна (садржала је регистре дужине 8 битова), 16битна, затим 32битна и на крају 64битна. Данас 64битну X86 архитектуру знамо као и AMD64, X86-64 или X86_64.

Заједно са ARM-ом једна од најкоришћенијих архитектура данашњице.

2.1 Регистри процесора

Постоји више врста регистара процесора[1, п. 75]. Неки од основних регистара које је потребно поменути дати су у тексту који следи. Разлог због којег су наведена и имена регистара претходних верзија X86 архитектуре је због тога сто је могуће адресирати првих x битова ако се користи име регистра за x-тобитну верзију те архитектуре. Наравно, ово важи само уколико је дужина регистра већа или једнака дужини регистра чију нотацију користимо.

Регистри опште намене:

8bit	al	bl	cl	dl	sil	dil	spl	bpl
16bit	ax	bx	cx	dx	si	di	sp	bp
32bit	eax	ebx	ecx	edx	esi	edi	esp	ebp
64bit	rax	rbx	rcx	rdx	rsi	rdi	rsp	rdp

Сегментни регистри:

cs	ds	ss	es	fs	gs

Контролни регистри:

cr0	cr2	cr3	cr4	cr8

Системски регистри који су показивачи на табеле:

gdtr	ldtr	idtr

Осим поменутих, почев од 16бит-не X86 архитектуре постоје и регистри ah, bh, ch, dh (h-higher) који представљају горњу половину (од 9. до 16. бита) ax,bx,cx,dx регистара редом. У даљем тексту биће приказан однос између регистара о коме је раније било речи, као и приказ и на то од ког до ког бита се односи дата нотација.

63-56	55-48	47-40	39-32	31-24	23-16	15-8	7-0
						ah	al
						ax	
				eax			
rax							

Приметимо да уколико нас интересује вредност другог бајта у 64битној X86 архитектури, до ње можемо доћи на 4 начина: ah, ax&0xFF00, eax&0x0000FF00 или rax&0x000000000000FF00.

2.2 Регистри опште намене

Регистри опште намене имају улогу у чувању операнди и показивача:

- Операнди за логичке и аритметицке операције
- Операнди за адресне калкулације
- Показиваце на меморијску локацију

Специфична улога регистара опште намене:

- eax - акумулатор за операнде и податке резултата
- ebx - показивач на податке у ds сегменту
- ecx - бројач за петље и операције над стринговима
- edx - показивач на У/И
- esi - показивач на податке на који показује ds регистар; почетни показивач за операције над стринговима

- edi - показивач на податке у сегменту на који показује es регистар; крајњи показивач за операције над стринговима
- esp - показивач на почетак стека
- ebp - показивач на податке у стеку

2.3 Сегментни регистри

Сегментни регистри садрже 16битне селекторе сегмента. Селектор сегмента је специјалан показивач који идентификује сегмент у меморији. Да би приступили одређеном сегменту у меморији, селектор сегмента који показује на тај сегмент мора бити доступан у одговарајућем сегментном регистру.
Специфична улога сегментних регистара:

- cs - code segment. cs регистар садржи селектор сегмента који показује на сегмент кода у коме се налазе инструкције које се извршавају.
- ds - data segment. Осим ds, сегментни регистри за сегменте података су и es, fs, као и gs.
- ss - stack segment ss регистар садржи селектор сегмента који показује на сегмент стека где се чува стек програма који се тренутно извршава. За разлику од регистра за сегмент кода, ss регистар се може експлицитно поставити што дозвољава апликацијама да поставе више стекова и да алтернирају између њих.

2.4 Real mode

Реални мод је стање процесора у којем нам је дозвољено адресирање само првих 20мб меморије. Прелазак из реалног у застицени мод постиже се далеким скоком "far jump".
`https://wiki.osdev.org/Real_Mode`

2.5 Сегментација

Сегментација је решење којим се омогућава адресирање више меморије него што је то хардверски предвиђено.
`https://wiki.osdev.org/Segmentation`

2.6 Protected mode

Заштићен мод је стање процесора у којем процесор има пун приступ целом опсегу меморије за разлику од реалног мода.
`https://wiki.osdev.org/Protected_Mode`

3

Boot

3.1 Редослед покретања

Од притиска дугмета за паљење рачунара, па до учитавања оперативног ситема постоји цео један процес. Након притиска дугмета рачунар прво извршава POST (Power On Self Test) који је једна од почетних фаза BIOS-a (Basic Input Output System). У POST-у рачунар покушава да инцијализује компоненте рачунарског система и проверава да ли оне испуњавају све услове за стартовање рачунара. Уколико је цео процес прошао без грешака наставља се даље извршавање BIOS-a. BIOS сада има улогу да пронађе медијум који садржи програм који ће учитати језгро оперативног система у рам меморију рачунара. Тај програм се назива Bootloader.
`https://wiki.osdev.org/Boot_Sequence`

3.2 Bootloader

Bootloader је програм који се налази у првих 512битова медијума, и његов задатак је да учита језгро оперативног система у рам меморију и преда му даље управљање.
`https://wiki.osdev.org/Bootloader`

3.3 ELF

ELF је формат бинарни фајл који се састоји од тачно одређених секција и који може да се покрене.
`https://wiki.osdev.org/ELF`

4

Коришћени алати

У даљем тексту се могу видети неки од алата коришћених у креирању овог рада. Сви коришћени алати поседују GPLv2 или GPLv3 лиценцу. GNU Public Licence је лиценца отвореног кода која дозвољава модификовање и дистрибуирање кода све док тај је тај код јавно доступан. Једини програм са листе који није GNU-ов је QEMU виртуална машина.

Оперативни систем коришћен у изради овог пројекта је Artix Linux. Artix Linux је GNU/Linux дистрибуција базирана на Arch Linux-у. Већина коришћених програма је већ компајлована и спремна за употребу и налази се у официјалним репозиторијима.

За програме који су морали бити мануелно компајловани дате су инструкције у њиховој подсекцији. Једини програми који су морали бити компајловани су binutils и gcc и то да не би користили стандардну библиотеку коју нам је обезбедио оперативни систем домаћин (онај на коме се компајлује овај пројекат). За остале програме који су коришћени препорука је користити оне који су доступни као спремни пакети у изворима одабране диструбуције GNU/Linux-а.

4.1 Binutils

Изворни код софтвера се може наћи на страници `https://www.gnu.org/software/binutils/`, заједно са упутством за компајловање и коришћење.
Овај софтверски пакет садржи програме неопходне за компајловање као што су асемблер и линкер.

4.1.1 Пре додавања C библиотеке

Из разлога што се не користи стандардна библиотека већ самостално написана специфицно за овај пројекат, потребно је мануелно компајловати GNU Binutils. Међутим, постоји могућност коришћења већ спремног пакета који се за дистрибуције базиране на Arch Linux-у може наћи на станици `https://aur.archlinux.org/packages/i686-elf-binutils/`. Поједине дистрибуције већ имају овај пакет компајлован, али је препорука мануелно компајловати да би се избегла некомпатибилност, а и просто из разлога што ће након формирања наше C библиотеке бити неопходно компајловати овај програм за сваки систем посебно.
За оне које желе сами да компајлују дат је део инструкција који се разликује од упутства датог на званичном сајту а тиче се конфигурисања пре компилације.

```
mkdir build
cd build

../configure \
    --target=i686-elf \
    --with-sysroot \
    --prefix=/usr \
    --bindir=/usr/bin \
    --libdir=/usr/lib/i686-elf \
    --disable-nls \
    --disable-werror

make
make install
```

4.1.2 Након додавања C библиотеке

Након додавања наше C библиотеке потребно је компајловати GNU Binutils тако да ту библиотеку и користи приликом компајловања нашег оперативног система. **Напомена:** Потребно је поставити $SYSROOT на локацију где се библиотека налази. То је могуће урадити на следећи начин:

```
export SYSROOT=/put/do/biblioteke
```

Инстукције за компајловање дате су у наставку:

```
../configure \
    --target=i686-elf \
    --with-sysroot=$SYSROOT \
    --prefix=/usr \
    --bindir=/usr/bin \
    --libdir=/usr/lib/i686-elf \
    --disable-nls \
    --disable-werror
```

4.1.3 GNU Asembler

Иако тренутно постоје много популарније алтернативе попут NASM (Netwide Assembler) и MASM (Microsoft Assembler) који користе новију Интелову синтаксу, аутор се ипак одлучио за GASM због компатибилности са GCC компајлером. GASM користи старију AT&T синтаксу коју карактерише: обрнут поредак параметара, префикс пре имена регистара и вредности константи, а и величина параметара мора бити дефинисана. Због тога ће можда неким читаоцима бити користан програм "intel2gas" који се за Arch Linux може наћи на станици `https://aur.archlinux.org/packages/intel2gas/`.

Овај програм је коришћен за компајловање дела кода написаног у асемблеру.

4.1.4 GNU Linker

Овај програм је коришћен за линковање, тј. "спајање" свог комапјлованог кода у једну бинарну датотеку типа ELF која представља кернел.

4.2 GCC

Изворни код софтвера се може наћи на страници `https://gcc.gnu.org/`, заједно са упутством за компајловање и коришћење.
`https://aur.archlinux.org/packages/i686-elf-gcc/`
GCC је GNU-ов сет компајлера.

4.2.1 Пре додавања LIBC

```
mkdir build
cd build

../configure \
    --target=i686-elf \
    --prefix=/usr \
    --disable-nls \
    --disable-plugin \
    --enable-languages=c,c++ \
    --without-headers

make all-gcc
make all-target-libgcc

make -k check || true

make install-gcc
make install-target-libgcc
```

4.2.2 После додавања LIBC

```
../configure \
    --target=i686-elf \
    --prefix=/usr \
    --with-sysroot=$SYSROOT \
    --disable-nls \
    --disable-plugin \
    --enable-languages=c,c++
```

4.3 GRUB

Изворни код софтвера се може наћи на страници `https://www.gnu.org/software/grub/`, заједно са упутством за компајловање и коришћење.
GRUB је bootloader који је коришћен на овом пројекту. План је да у будућности GRUB буде замењен са мојим bootloader-ом, и да се комплетан код буде мој.

4.4 QEMU

Изворни код софтвера се може наћи на страници `https://www.qemu.org/`, заједно са упутством за компајловање и коришћење.

QEMU је виртуална машина у којој ће језгро бити тестирано и приказано зарад практичних разлога. QEMU је одабран за овај пројекат јер за разлику од других виртуалних машина поседује cli (command line interface) из кога се лако може позивати из скрипти као што су Makefile-ови.

4.5 Make

Изворни код софтвера се може наћи на страници `https://www.gnu.org/software/make/` заједно са упутством за компајловање и коришћење. [2].
Make нам омогућава да са лакоћом одржавамо и манипулишемо изворним фајловима. Могуће је све компајловати, обрисати, креирати iso фајл као и покренути QEMU виртуелну машину са само једном кључном речи у терминалу. Креирани Makefile за потребе овог пројекта биће детаљно објашњен у даљем тексту.

4.6 Мање битни алати

4.6.1 NeoVim

NeoVim је текст едитор настао од Vim-a (Vi improved). [3].

4.6.2 git

Изворни код софтвера се може наћи на страници `https://git.kernel.org/pub/scm/git/git.git`.
Git је програм који нам помаже да одржавамо изводне фајлове и

4.6.3 xorriso(libisoburn)

`https://www.gnu.org/software/xorriso/`
Служи за креирање ISO фајлова који се могу "нарезати" на CD или USB флеш са којих се касније диже систем.

4.6.4 GDB

`https://www.sourceware.org/gdb/`

5

Језгро оперативног система

5.1 Почетак

`https://wiki.osdev.org/Bare_Bones`
as/boot.s:

```
.set ALIGN,    1<<0
.set MEMINFO,  1<<1
.set FLAGS,    ALIGN | MEMINFO
.set MAGIC,    0x1BADB002
.set CHECKSUM, -(MAGIC + FLAGS)
```

```
.section .multiboot
.align 4
.long MAGIC
.long FLAGS
.long CHECKSUM
```

```
.section .bss
.align 16
stack_bottom:
.skip 16384
stack_top:
```

```
.section .text
.global _start
.type _start, @function
_start:
        mov $stack_top, %esp
        call kernel_main
        cli
1:      hlt
        jmp 1b

.size _start, . - _start
```

5.2 Испис на екран - VGA

c/vga.c:

```
#include<types.h>
#include<string.h>
#include<asm.h>
#include<vga.h>
```

```
size_t terminal_row;
size_t terminal_column;
uint8_t terminal_color;
uint16_t* terminal_buffer;
```

```
void set_color(enum vga_color fg, enum vga_color bg)
{
    terminal_color = fg | bg << 4;
}
```

```
static inline uint16_t vga_entry(unsigned char uc, uint8_t color)
{
    return (uint16_t) uc | (uint16_t) color << 8;
}
```

```
void terminal_initialize()
{
    terminal_row=0;
    terminal_column=0;
    set_color(VGA_COLOR_LIGHT_GREY, VGA_COLOR_BLACK);
    terminal_buffer=(uint16_t*) 0xB8000;
    for(size_t y=0;y<VGA_HEIGHT;y++)
    {
        for(size_t x=0;x<VGA_WIDTH;x++)
        {
            const size_t index=y*VGA_WIDTH+x;
            terminal_buffer[index]=vga_entry(' ', terminal_color);
        }
    }
}
```

```
void terminal_putentryat(char c, uint8_t color, size_t x, size_t y)
{
    const size_t index=y*VGA_WIDTH+x;
    terminal_buffer[index]=vga_entry(c, color);
}
```

```
void movescreen()
{
    terminal_row--;
    for(size_t i=0;i<VGA_HEIGHT;i++) for(size_t j=0;j<VGA_WIDTH;j++) terminal_buffe
}
```

```
void next_field()
{
    if(++terminal_column==VGA_WIDTH) terminal_column=0,terminal_row++;
}
```

```
void previous_field()
{
    if(terminal_column) terminal_column--;
    else terminal_row--,terminal_column=VGA_WIDTH-1;
}
```

```
void terminal_putchar(char c)
{
    if(c=='\n') terminal_column=0,terminal_row++;
    else
    {
        terminal_putentryat(c, terminal_color, terminal_column, terminal_row);
        next_field();
    }
    if (terminal_row==VGA_HEIGHT) movescreen();
}
```

```
void terminal_writestring(char* data)
{
    for(int i=0;data[i]!='\0';i++) terminal_putchar(data[i]);
}
```

```
void clear()
{
    for(size_t i=0;i<VGA_HEIGHT;i++) for(size_t j=0;j<VGA_WIDTH;j++) terminal_putchar('
    terminal_column=0;
    terminal_row=0;
}
```

5.3 Global Desctiptor Table

c/gdt.c:

```
#include<types.h>
```

```
struct gdt_entry
{
    uint16_t limit;
    uint16_t base1;
    uint8_t base2;
    uint8_t access;
    uint8_t limit_flags;
    uint8_t base3;
} __attribute__((packed));
```

```
struct gdt_pointer
{
    uint16_t size;
    uint32_t offset;
} __attribute__((packed));
```

```
extern void load_gdt(struct gdt_pointer *gdtp);
struct gdt_entry gdt[5];
struct gdt_pointer gdtp;
```

```
void init_gdt_entry(size_t num, uint32_t limit, uint32_t base, uint8_t access, uint
{
    gdt[num].limit=limit;
    gdt[num].base1=(base & 0xffff);
    gdt[num].base2=(base & 0xff0000) >> 16;
    gdt[num].access=access;
    gdt[num].limit_flags=limit_flags;
    gdt[num].base3=(base & 0xff000000) >> 24;
}
```

```
void init_gdt_table()
{
    gdtp.size=sizeof(gdt)-1;
    gdtp.offset=(uint32_t)&gdt;

    init_gdt_entry(0,0,0,0,0);                                  // null segment
    init_gdt_entry(1,0xffffffff,0,0b10011010,0b11001111);       // code segment
    init_gdt_entry(2,0xffffffff,0,0b10010010,0b11001111);       // data segment
    init_gdt_entry(3,0xffffffff,0,0b11111010,0b11001111);       // user mode code s
    init_gdt_entry(4,0xffffffff,0,0b11110010,0b11001111);       // user mode data s

    load_gdt(&gdtp);
}
```

https://wiki.osdev.org/GDT

5.4 Interrupt Desctiptor Table

c/idt.c:

```
#include<types.h>
#include<irq.h>
#include<asm.h>
```

```
#define INTERRUPT_GATE_32 0x8E
#define KERNEL_CODE 0x08
#define KERNEL_DATA 0x10
#define PIC1_COMMAND_PORT 0x20
#define PIC1_DATA_PORT 0x21
#define PIC2_COMMAND_PORT 0xA0
#define PIC2_DATA_PORT 0xA1
```

```
struct idt_entry
{
    uint16_t offset1;
    uint16_t selector;
    uint8_t zero;
    uint8_t type_attr;
    uint16_t offset2;
} __attribute__((packed));
```

```
struct idt_pointer
{
    uint16_t size;
    uint32_t offset;
} __attribute__((packed));
```

```
extern void load_idt(struct idt_pointer *idtp);
extern void keyboard_irq();
struct idt_entry idt[256];
struct idt_pointer idtp;
```

```
void init_idt_entry(size_t num, uint32_t offset, uint16_t selector, uint8_t type_attr)
{
    idt[num].offset1=(offset & 0xffff);
    idt[num].selector=selector;
    idt[num].zero=0;
    idt[num].type_attr=type_attr;
    idt[num].offset2=(offset & 0xffff0000)>>16;
}
```

```
void add_idt_entry(size_t num,uint32_t offset)
{
    init_idt_entry(num,offset,KERNEL_CODE,INTERRUPT_GATE_32);
}
```

https://wiki.osdev.org/IDT

5.5 IRQ и PIC

c/idt.c:

```
void init_pic()
{
    ioport_out(PIC1_COMMAND_PORT, 0x11);
    ioport_out(PIC2_COMMAND_PORT, 0x11);
    ioport_out(PIC1_DATA_PORT, 0x20);
    ioport_out(PIC2_DATA_PORT, 0x28);
    ioport_out(PIC1_DATA_PORT, 0x04);
    ioport_out(PIC2_DATA_PORT, 0x02);
    ioport_out(PIC1_DATA_PORT, 0x01);
    ioport_out(PIC2_DATA_PORT, 0x01);
    ioport_out(PIC1_DATA_PORT, 0xff);
    ioport_out(PIC2_DATA_PORT, 0xff);

    ioport_out(PIC1_DATA_PORT, 0xFC);
}
```

```
void init_idt_table()
{
    init_pic();
    add_idt_entry(0,(uint32_t)irq0);
    add_idt_entry(1,(uint32_t)irq1);
    add_idt_entry(2,(uint32_t)irq2);
    add_idt_entry(3,(uint32_t)irq3);
    add_idt_entry(4,(uint32_t)irq4);
    add_idt_entry(5,(uint32_t)irq5);
    add_idt_entry(6,(uint32_t)irq6);
    add_idt_entry(7,(uint32_t)irq7);
    add_idt_entry(8,(uint32_t)irq8);
    add_idt_entry(9,(uint32_t)irq9);
    add_idt_entry(10,(uint32_t)irq10);
    add_idt_entry(11,(uint32_t)irq11);
    add_idt_entry(12,(uint32_t)irq12);
    add_idt_entry(13,(uint32_t)irq13);
    add_idt_entry(14,(uint32_t)irq14);
    add_idt_entry(15,(uint32_t)irq15);
    add_idt_entry(16,(uint32_t)irq16);
    add_idt_entry(17,(uint32_t)irq17);
    add_idt_entry(18,(uint32_t)irq18);
    add_idt_entry(19,(uint32_t)irq19);
    add_idt_entry(20,(uint32_t)irq20);
    add_idt_entry(21,(uint32_t)irq21);
    add_idt_entry(22,(uint32_t)irq22);
    add_idt_entry(23,(uint32_t)irq23);
    add_idt_entry(24,(uint32_t)irq24);
    add_idt_entry(25,(uint32_t)irq25);
    add_idt_entry(26,(uint32_t)irq26);
    add_idt_entry(27,(uint32_t)irq27);
    add_idt_entry(28,(uint32_t)irq28);
    add_idt_entry(29,(uint32_t)irq29);
    add_idt_entry(30,(uint32_t)irq30);
    add_idt_entry(31,(uint32_t)irq31);
    add_idt_entry(32,(uint32_t)timer_irq);
    add_idt_entry(33,(uint32_t)keyboard_irq);

    idtp.size=sizeof(struct idt_entry)*256-1;
    idtp.offset=(uint32_t)&idt;


    load_idt(&idtp);
}
```

https://wiki.osdev.org/IRQ https://wiki.osdev.org/PIC

5.6 Тастатура

c/keyboard.c:

```
#include<types.h>
#include<asm.h>
#include<stdio.h>
```

```
#define BUFFER_SIZE 200
#define BUFFER_LOG 200
char buffer[BUFFER_LOG][BUFFER_SIZE];
size_t buffer_size[BUFFER_LOG];
size_t buffer_current=0;
size_t buffer_all=0;
size_t buffer_index=0;
```

```
#define PIC1_COMMAND_PORT 0x20
#define PIC1_DATA_PORT 0x21
#define PIC2_COMMAND_PORT 0xA0
#define PIC2_DATA_PORT 0xA1
// IO Ports for Keyboard
#define KEYBOARD_DATA_PORT 0x60
#define KEYBOARD_STATUS_PORT 0x64
```

```
void previous_field(void);
void tty(char *buffer);
void prompt(void);
void clear();
void us_en(char keymap[]);
void us_en_shift(char keymap[]);
```

```
char charcode[256];
char shift_charcode[256];
bool ispressed[128];
#define lshift ispressed[0x2A]
#define rshift ispressed[0x36]
#define lctrl ispressed[0x1D]
```

```
void init_keyboard()
{
    us_en(charcode);
    us_en_shift(shift_charcode);
}
```

```
void deletelast()
{
    previous_field();
    printf(" ");
    previous_field();
}
```

```
void backspace()
{
    if(buffer_index<=0) return;

    deletelast();
    buffer[buffer_current][--buffer_index]='\0';
    return;
}
```

```
void enter()
{
    printf("\n");
    if(buffer_index>0)
    {
        tty(buffer[buffer_current]);
        buffer_size[buffer_current]=buffer_index;
        if(buffer_current==buffer_all) buffer_current=(++buffer_all);
        else
        {
            for(size_t i=0;i<BUFFER_SIZE;i++) buffer[buffer_all][i]='\0';
            buffer_current=buffer_all;
        }
        buffer_index=0;
    }
    prompt();
    return;
}
```

```
void space()
{
    buffer[buffer_current][buffer_index++]=' ';
    printf(" ");
}
```

```
void keyup()
{
    if(buffer_current>0)
    {
        buffer_size[buffer_current]=buffer_index;
        for(size_t i=0;i<buffer_index;i++) deletelast();
        buffer_current--;
        buffer_index=buffer_size[buffer_current];
        printf("%s",buffer[buffer_current]);
    }
}
```

```
void keydown()
{
    if(buffer_current<buffer_all)
    {
        buffer_size[buffer_current]=buffer_index;
        for(size_t i=0;i<buffer_index;i++) deletelast();
        buffer_current++;
        buffer_index=buffer_size[buffer_current];
        printf("%s",buffer[buffer_current]);
    }
}
```

```
void keyboard_handler()
{
    ioport_out(PIC1_COMMAND_PORT, 0x20);
    uint8_t status = ioport_in(KEYBOARD_STATUS_PORT);

    if (status & 0x1)
    {
        uint8_t keycode = ioport_in(KEYBOARD_DATA_PORT);
        if(keycode<0x80)
        {
            ispressed[keycode]=1;
            if(keycode==0x0E) backspace();
            else if(keycode==0x1C) enter();
            else if(keycode==0x39) space();
            else if(keycode==72) keyup();
            else if(keycode==80) keydown();
            else if(keycode==75) keyleft();
            else if(keycode==77) keyright();
            else
            {
                char c=charcode[keycode];
                if(c!=' ')
                {
                    if(lctrl)
                    {
                        if(c=='l')
                        {
                            clear();
                            prompt();
                            printf("%s",buffer[buffer_current]);
                            return;
                        }
                    }
                    if(lshift||rshift)
                    {
                        c=shift_charcode[keycode];
                    }
                    buffer[buffer_current][buffer_index++]=c;
                    printf("%c",c);
                }
            }
        }
        else
        {
            ispressed[keycode-0x80]=0;
        }
    }
}
```

5.7 PIT - Programmable Interval Timer

c/timer.c:

```
#include<types.h>
#include<asm.h>
#include<stdio.h>
```

```
void add_idt_entry(size_t num,uint32_t offset);
uint32_t tick=0;
const uint32_t TICKS_PER_SECOND=50;
extern uint32_t time;
uint32_t time=0;
```

```
void timer_handler()
{
    tick++;
    if(tick==TICKS_PER_SECOND)
    {
        //printf("%d seconds passed\n",time);
        tick=0;
        time++;
    }

    ioport_out(0x20, 0x20);
    ioport_out(0xa0,0x20);
}
```

```
void init_timer(uint32_t frequency)
{
    // Firstly, register our timer callback.

    // The value we send to the PIT is the value to divide it's input clock
    // (1193180 Hz) by, to get our required frequency. Important to note is
    // that the divisor must be small enough to fit into 16-bits.
    uint32_t divisor = 1193180 / frequency;

    // Send the command byte.
    ioport_out(0x43, 0x36);

    // Divisor has to be sent byte-wise, so split here into upper/lower bytes.
    uint8_t l = (uint8_t)(divisor & 0xFF);
    uint8_t h = (uint8_t)( (divisor>>8) & 0xFF );

    // Send the frequency divisor.
    ioport_out(0x40, l);
    ioport_out(0x40, h);
}
```

https://wiki.osdev.org/PIT

5.8 Heap

c/heap.c:

```
#include<types.h>
```

```
typedef struct _KHEAPBLOCKBM {
        struct _KHEAPBLOCKBM                    *next;
        uint32_t                                size;
        uint32_t                                used;
        uint32_t                                bsize;
        uint32_t                                lfb;
} KHEAPBLOCKBM;
```

```
typedef struct _KHEAPBM {
        KHEAPBLOCKBM                    *fblock;
} KHEAPBM;
```

```
void k_heapBMInit(KHEAPBM *heap) {
        heap->fblock = 0;
}
```

```
int k_heapBMAddBlock(KHEAPBM *heap, uintptr_t addr, uint32_t size, uint32_t bsize)
        KHEAPBLOCKBM            *b;
        uint32_t                bcnt;
        uint32_t                x;
        uint8_t                 *bm;

        b = (KHEAPBLOCKBM*)addr;
        b->size = size - sizeof(KHEAPBLOCKBM);
        b->bsize = bsize;

        b->next = heap->fblock;
        heap->fblock = b;

        bcnt = b->size / b->bsize;
        bm = (uint8_t*)&b[1];

        /* clear bitmap */
        for (x = 0; x < bcnt; ++x) {
                        bm[x] = 0;
        }

        /* reserve room for bitmap */
        bcnt = (bcnt / bsize) * bsize < bcnt ? bcnt / bsize + 1 : bcnt / bsize;
        for (x = 0; x < bcnt; ++x) {
                        bm[x] = 5;
        }

        b->lfb = bcnt - 1;

        b->used = bcnt;

        return 1;
}
```

```
static uint8_t k_heapBMGetNID(uint8_t a, uint8_t b) {
        uint8_t         c;
        for (c = a + 1; c == b || c == 0; ++c);
        return c;
}
```

```
void *k_heapBMAlloc(KHEAPBM *heap, uint32_t size) {
        KHEAPBLOCKBM                    *b;
        uint8_t                         *bm;
        uint32_t                        bcnt;
        uint32_t                        x, y, z;
        uint32_t                        bneed;
        uint8_t                         nid;

        /* iterate blocks */
        for (b = heap->fblock; b; b = b->next) {
            //printf("size:%d, used:%d, bsize:%d, lfb:%d\n",b->size,b->used,b->bsize,b->lfb
                /* check if block has enough room */
                if (b->size - (b->used * b->bsize) >= size) {

                        bcnt = b->size / b->bsize;
                        bneed = (size / b->bsize) * b->bsize < size ? size / b->bsize +
                        bm = (uint8_t*)&b[1];
                        //printf("bcnt:%d, bneed:%d, bm:%d\n",bcnt,bneed,bm);

                        for (x = (b->lfb + 1 >= bcnt ? 0 : b->lfb + 1); x != b->lfb; ++x
                                /* just wrap around */
                                if (x >= bcnt) {
                                        x = 0;
                                }

                                if (bm[x] == 0) {
                                        /* count free blocks */
                                        for (y = 0; bm[x + y] == 0 && y < bneed && (x +

                                        /* we have enough, now allocate them */
                                        if (y == bneed) {
                                                /* find ID that does not match left or
                                                nid = k_heapBMGetNID(bm[x - 1], bm[x + y

                                                /* allocate by setting id */
                                                for (z = 0; z < y; ++z) {
                                                        bm[x + z] = nid;
                                                }

                                                /* optimization */
                                                b->lfb = (x + bneed) - 2;

                                                /* count used blocks NOT bytes */
                                                b->used += y;

                                                return (void*)(x * b->bsize + (uintptr_t

                                        }

                                        /* x will be incremented by one ONCE more in our
                                        x += (y - 1);
                                        continue;
                                }
                        }
                }
        }

        return 0;
}
```

```
void k_heapBMFree(KHEAPBM *heap, void *ptr) {
        KHEAPBLOCKBM            *b;
        uintptr_t               ptroff;
        uint32_t                bi, x;
        uint8_t                 *bm;
        uint8_t                 id;
        uint32_t                max;

        for (b = heap->fblock; b; b = b->next) {
                if ((uintptr_t)ptr > (uintptr_t)b && (uintptr_t)ptr < (uintptr_t)b
                        /* found block */
                        ptroff = (uintptr_t)ptr - (uintptr_t)&b[1];
/* get offset to get block */
                        /* block offset in BM */
                        bi = ptroff / b->bsize;
                        /* .. */
                        bm = (uint8_t*)&b[1];
                        /* clear allocation */
                        id = bm[bi];
                        /* oddly.. GCC did not optimize this */
                        max = b->size / b->bsize;
                        for (x = bi; bm[x] == id && x < max; ++x) {
                                bm[x] = 0;
                        }
                        /* update free block count */
                        b->used -= x - bi;
                        return;
                }
        }

        /* this error needs to be raised or reported somehow */
        return;
}
```

```
KHEAPBM kheap;
void kheapinit()
{
    k_heapBMInit(&kheap);
}
int kheapaddblock(uintptr_t addr,uint32_t size,uint32_t bsize)
{
    return k_heapBMAddBlock(&kheap,addr,size,bsize);
}
```

```
void *kmalloc(uint32_t size)
{
    return k_heapBMAlloc(&kheap, size);

}
void kfree(void *ptr)
{
    k_heapBMFree(&kheap, ptr);
}
```

https://wiki.osdev.org/Heap

5.9 Paging

c/paging.c:

```
#include<types.h>
```

```
extern void loadPageDirectory(uint32_t*);
extern void enablePaging();
uint32_t page_directory[1024] __attribute__((aligned(4096)));
```

```
void set_pd()
{
    //set each entry to not present
    for(size_t i=0;i<1024;i++)
    {
        // This sets the following flags to the pages:
        //   Supervisor: Only kernel-mode can access them
        //   Write Enabled: It can be both read from and written to
        //   Not Present: The page table is not present
        page_directory[i] = 0x00000002;
    }
}
```

```
uint32_t page_table[1024][1024] __attribute__((aligned(4096)));
```

```
void set_pt(size_t num,uint32_t address)
{
    // holds the physical address where we want to start mapping these pages to.
    // in this case, we want to map these pages to the very beginning of memory.
    //we will fill all 1024 entries in the table, mapping 4 megabytes
    for(size_t i=0;i<1024;i++)
    {
        // As the address is page aligned, it will always leave 12 bits zeroed.
        // Those bits are used by the attributes ;)
        page_table[num][i] = (address + i * 0x1000) | 3; // attributes: supervisor
    }

    page_directory[num] = ((uint32_t)page_table[num]) | 3;
    // attributes: supervisor level, read/write, present
}
```

```
void set_paging()
{
    set_pd();
    for(size_t i=0;i<1024;i++) set_pt(i,0x00400000 * i); // all 4GB mapped
    loadPageDirectory(page_directory);
    enablePaging();
}
```

https://wiki.osdev.org/Paging

5.10 Moj LIBC

https://wiki.osdev.org/Creating_a_C_Library
include/asm.h:

```
#ifndef ASM_H
#define ASM_H

#include<types.h>

extern uint8_t ioport_in(uint8_t port);
extern void ioport_out(uint8_t port, char data);

#endif
```

include/errno.h:

include/heap.h:

```
#ifndef HEAP_H
#define HEAP_H

#include<types.h>

void kheapinit();
int kheapaddblock(uintptr_t addr,uint32_t size,uint32_t bsize);
void *kmalloc(uint32_t size);
void kfree(void *ptr);

#endif
```

include/irq.h:

```
#ifndef IRQ_H
#define IRQ_H

extern void irq0();
extern void irq1();
extern void irq2();
extern void irq3();
extern void irq4();
extern void irq5();
extern void irq6();
extern void irq7();
extern void irq8();
extern void irq9();
extern void irq10();
extern void irq11();
extern void irq12();
extern void irq13();
extern void irq14();
extern void irq15();
extern void irq16();
extern void irq17();
extern void irq18();
extern void irq19();
extern void irq20();
extern void irq21();
extern void irq22();
extern void irq23();
extern void irq24();
extern void irq25();
extern void irq26();
extern void irq27();
extern void irq28();
extern void irq29();
extern void irq30();
extern void irq31();
extern void timer_irq();
extern void keyboard_irq();

#endif
```

include/stdio.h:

```
#ifndef _STDIO_H
#define _STDIO_H
#include <stdarg.h>
#include <stddef.h>
#define SEEK_SET 0
typedef struct { int unused; } FILE;
#ifdef __cplusplus
extern "C" {
#endif
extern FILE* stderr;
#define stderr stderr
int fclose(FILE*);
int fflush(FILE*);
FILE* fopen(const char*, const char*);
int fprintf(FILE*, const char*, ...);
size_t fread(void*, size_t, size_t, FILE*);
int fseek(FILE*, long, int);
long ftell(FILE*);
size_t fwrite(const void*, size_t, size_t, FILE*);
void setbuf(FILE*, char*);
int vfprintf(FILE*, const char*, va_list);

void printf(char *str, ...);

#ifdef __cplusplus
}
#endif
#endif
```

include/stdlib.h:

```
#ifndef _STDLIB_H
#define _STDLIB_H
#ifdef __cplusplus
extern "C" {
#endif
void abort(void);
int atexit(void (*)(void));
int atoi(const char*);
void free(void*);
char* getenv(const char*);
void* malloc(size_t);
#ifdef __cplusplus
}
#endif
#endif
```

include/string.h:

```
#ifndef _STRING_H
#define _STRING_H
#include <stddef.h>
#ifdef __cplusplus
extern "C" {
#endif
void* memcpy(void*, const void*, size_t);
void* memset(void*, int, size_t);
char* strcpy(char*, const char*);
size_t strlen(const char*);
#ifdef __cplusplus
}
#endif

#include<types.h>

size_t stringlen(char *str);
bool stringcmp(char *str1,char *str2);
void stringcat(char *str1,char *str2);
void stringrev(char *str);
void itos(uint32_t num,char *str);
uint32_t stoi(const char *str);
double stof(const char *str);
void ftos(double num, char *str);

#endif
```

include/time.h:

include/types.h:

```
#ifndef TYPES_H
#define TYPES_H

#include<stdbool.h>
#include<stddef.h>
#include<stdint.h>

#endif
```

include/unistd.h:

```
#ifndef _UNISTD_H
#define _UNISTD_H
#include <sys/types.h>
#ifdef __cplusplus
extern "C" {
#endif
int execv(const char*, char* const[]);
int execve(const char*, char* const[], char* const[]);
int execvp(const char*, char* const[]);
pid_t fork(void);
#ifdef __cplusplus
}
#endif
#endif
```

include/vga.h:

```
#ifndef VGA_H
#define VGA_H

#include<types.h>

static const size_t VGA_WIDTH = 80;
static const size_t VGA_HEIGHT = 25;

enum vga_color {
    VGA_COLOR_BLACK = 0,
    VGA_COLOR_DARK_BLUE = 1,
    VGA_COLOR_GREEN = 2,
    VGA_COLOR_TURQUOISE = 3,
    VGA_COLOR_RED = 4,
    VGA_COLOR_PURPLE = 5,
    VGA_COLOR_BROWN = 6,
    VGA_COLOR_LIGHT_GREY = 7,
    VGA_COLOR_DARK_GREY = 8,
    VGA_COLOR_BLUE = 9,
    VGA_COLOR_LIGHT_GREEN = 10,
    VGA_COLOR_LIGHT_BLUE = 11,
    VGA_COLOR_LIGHT_RED = 12,
    VGA_COLOR_PINK = 13,
    VGA_COLOR_YELLOW = 14,
    VGA_COLOR_WHITE = 15,
};
void set_color(enum vga_color fg, enum vga_color bg);

#endif
```

include/sys/types.h:

```
#ifndef _SYS_TYPES_H
#define _SYS_TYPES_H
typedef int pid_t;
#endif
```

6

Закључак

Овај пројекат је био сјајан показатељ колико је заправо комплексно.

Литература

[1] Intel. *Intel® 64 and IA-32 architectures software developer's manual combined volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4.* URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

[2] Robert Mecklenburg. *Managing Projects with GNU Make.* 3rd ed. Nutshell Handbooks. O'Reilly Media, 2004. ISBN: 0596006101,9780596006105. URL: http://gen.lib.rus.ec/book/index.php?md5=6754a2b1ac7071ea1ccedc05801f5424.

[3] Arnold Robbins, Elbert Hannah, and Linda Lamb. *Learning the Vi and Vim Editors.* 7th ed. O'Reilly Media, 2008. ISBN: 9780596529833,059652983X. URL: http://gen.lib.rus.ec/book/index.php?md5=bc6fb75f968bcc39e4446c29bf04d2d1.

[4] *OsDev Wiki.* URL: https://wiki.osdev.org/Expanded_Main_Page.

[5] Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani. *Algorithms.* 1st ed. McGraw-Hill Science/Engineering/Math, 2006. ISBN: 0073523402,9780073523408. URL: http://gen.lib.rus.ec/book/index.php?md5=529078edeb67e6ca26edc683ad3f6a51.

[6] David Salomon. *Assemblers and loaders.* Ellis Horwood series in computers and their applications. Ellis Horwood, 1992. ISBN: 9780130525642,0130525642. URL: http://gen.lib.rus.ec/book/index.php?md5=A5065F7D488058B77DD8FBF85DF7490F.

[7] Alfred V. Aho et al. *Compilers - Principles, Techniques, and Tools.* 2nd ed. Pearson/Addison Wesley, 2006. ISBN: 0321486811. URL: http://gen.lib.rus.ec/book/index.php?md5=90db32d070cfb70ca617e655d5c35529.

[8] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach.* 5th ed. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 2011. ISBN: 012383872X,9780123838728.

URL: http://gen.lib.rus.ec/book/index.php?md5=041f6d8fb8e6b6d1ed26a824775b5a0d.

[9] Bryant Randal E. and O'Hallaron David Richard. *Computer Systems: A Programmer's Perspective*. 2nd ed. Pearson, 2010. ISBN: 0136108040,9780136108047. URL: http://gen.lib.rus.ec/book/index.php?md5=3E84B730EC874FFDB1A069FF482D112C.

[10] Andrew S. Tanenbaum and Todd Austin. *Structured Computer Organization*. 6th ed. Prentice Hall, 2012. ISBN: 0132916525,9780132916523. URL: http://gen.lib.rus.ec/book/index.php?md5=f6fc02a547e862360e743754fc06375b.

[11] Benjamin Lunt. *USB: The Universal Serial Bus (FYSOS: Operating System Design Book 8)*. 2013. ISBN: 1717425364. URL: http://gen.lib.rus.ec/book/index.php?md5=D4A70ECFDC3F4EB7DC62723D7BCB222C.

[12] Andrew S. Tanenbaum. *Modern Operating Systems*. 3rd ed. Pearson Prentice Hall, 2008. ISBN: 0136006639,9780136006633. URL: http://gen.lib.rus.ec/book/index.php?md5=ac8cd7d4fd0467e923a0c04b7a939f84.

[13] Abraham Silberschatz, Peter B Galvin, and Greg Gagne. *Operating system concepts*. 9th ed. Wiley, 2012. ISBN: 9781118063330,1118063333. URL: http://gen.lib.rus.ec/book/index.php?md5=be3559401fec4b4ba93cc8e2f2e05601.

[14] William Stallings. *Operating systems: internals and design principles*. 7th ed. Prentice Hall, 2011. ISBN: 013230998X, 9780132309981, 0273751506, 9780273751502, 1299318266, 9781299318267. URL: http://gen.lib.rus.ec/book/index.php?md5=6990c44cecea007aa328a9a86c3027d1.

[15] Remzi H. Arpaci-Dusseau and Andrea C Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Paperback. Createspace Independent Publishing Platform, 2018. ISBN: 198508659X,9781985086593. URL: http://gen.lib.rus.ec/book/index.php?md5=aa2c64b8cee2819de114afdebd113e7a.

[16] Richard A. Burgess. *Mmurtl V1.0*. Ip Data Corp, 2000. ISBN: 9781588530004,1588530000. URL: http://gen.lib.rus.ec/book/index.php?md5=eb2ae29c03308f4f95a57a2a1dadab22.

[17] Andrew S. Tanenbaum and Albert S. Woodhull. *Operating Systems: Design and Implementation*. 3rd ed. Pearson, 2006. ISBN: 0131429388, 9780131429383. URL: http://gen.lib.rus.ec/book/index.php?md5=fdafa32272838b3eff5583d668e3192b.

[18] Marshall Kirk McKusick, George V. Neville-Neil, and Robert N.M. Watson. *The Design and Implementation of the FreeBSD Operating System*. 2nd ed. Addison-Wesley Professional, 2014. ISBN: 0321968972,9780321968975. URL: http://gen.lib.rus.ec/book/index.php?md5=bc62cc0f2ad546d5a7b2d2c610ee14ae.

[19] Samuel P. Harbison and Guy L. Steele. *C: A Reference Manual*. 5th ed. Prentice Hall, 2002. ISBN: 013089592X,9780130895929. URL: http://gen.lib.rus.ec/book/index.php?md5=fe81d808c24e1a0c0d479520db57cd86.

[20] Brian W. Kernighan and Dennis M. Ritchie. *C Programming Language*. 2nd ed. Prentice Hall, 1988. ISBN: 0131103628,9780131103627. URL: http://gen.lib.rus.ec/book/index.php?md5=5013a19e2b20b82d104bae34ac7e5320.

[21] Bill Rosenblatt and Cameron Newham. *Learning the bash shell*. 3rd ed. Nutshell handbook. O'Reilly, 2005. ISBN: 9780596009656,0596009658. URL: http://gen.lib.rus.ec/book/index.php?md5=F0B2387CFA911873144412D2DF51E16C.

[22] Stephen A. Rago and W. Richard Stevens. *Advanced Programming in the UNIX Environment*. 3rd ed. Addison-Wesley professional computing series. Addison-Wesley Professional, 2013. ISBN: 0321637739,9780321637734. URL: http://gen.lib.rus.ec/book/index.php?md5=8f4dd448cc992b8ab4a38dd056b09478.