Create Your Own K

Pritam Zope
13 Jan 2018 CPOL

In this article we will create a simple kernel such as printing HelloWorld first and then writing functions for printing numbers, Keyboard I/O, Box Drawing GUI, and Tic-Tac-Toe game in kernel in C

Download kernel_images_2.zip - 7.8 MB

Download kernel_images_1.zip - 5.8 MB

Download kernel_source.zip - 617.6 KB

Introduction

Ok, You already know what kernel is https://en.wikipedia.org/wiki/Kernel_(operating_system)

The first part of writing an operating system is to write a bootloader in 16 bit assembly (real mode).
Bootloader is a piece of program that runs before any operating system is running.
it is used to boot other operating systems, usually each operating system has a set of bootloaders specific for it.
Go to following link to create your own bootloader in 16 bit assembly

https://createyourownos.blogspot.in/

Bootloaders generally select a specififc operating system and starts it's process and then operating system loads itself into memory.
If you are writing your own bootloader for loading a kernel you need to know the overall addressing/interrupts of memory as well as BIOS.
Mostly each operating system has specific bootloader for it.
There are lots of bootloaders available out there in online market.

But there are some proprietary bootloaders such as Windows Boot Manager for Windows operating systems or BootX for Apple's ope
But there are lots of free and open source b

https://en.wikipedia.org/wiki/Co

Among most famous is GNU GRUB - GNU (
project for Unix like systems.

https://en.wikipedia.org/wiki/GN

We will use GNU GRUB to load our kernel b
systems.

Requirements

GNU/Linux :- Any distribution(Ubuntu/Debian/RedHat etc.).
Assembler :- GNU Assembler(**gas**) to assemble the assembly laguage file.
GCC :- GNU Compiler Collection, C compiler. Any version 4, 5, 6, 7, 8 etc.
Xorriso :- A package that creates, loads, manipulates ISO 9660 filesystem images.(man xorriso)
grub-mkrescue :- Make a GRUB rescue image, this package internally calls the xorriso functionality to build an iso image.
QEMU :- Quick EMUlator to boot our kernel in virtual machine without rebooting the main system.

Using the code

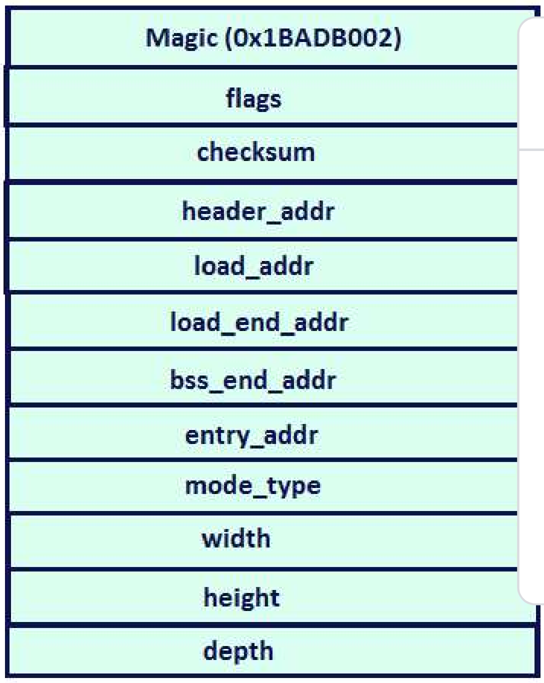
Alright, writing a kernel from scratch is to print something on screen.
So we have a VGA(Visual Graphics Array), a hardware system that controls the display.

https://en.wikipedia.org/wiki/Video_Graphics_Array

VGA has a fixed amount of memory and addresssing is **0xA0000** to **0xBFFFF**.

0xA0000 for EGA/VGA graphics modes (64 KB)
0xB0000 for monochrome text mode (32 KB)
0xB8000 for color text mode and CGA-compatible graphics modes (32 KB)

First you need a multiboot bootloader file that instruct the GRUB to load it.
Following fields must be define.



Multiboot header

Magic :- A fixed hexadecimal number identified by the bootloader as the header(starting point) of the kernel to be loaded.
flags :- If bit 0 in the flags word is set, then all boot modules loaded along with the operating system must be aligned on page (4KB) boundaries.
checksum :- which is used by special purpose by bootloader and its value must be the sum of magic no and flags.

We don't need other information,
but for more details https://www.gnu.org/software/grub/manual/multiboot/multiboot.pdf

Ok lets write a GAS assembly code for above information.
we dont need some fields as shown in above image.

boot.S

Python Shrink ▲

```
# set magic number to 0x1BADB002 to identified by bootloader
.set MAGIC,    0x1BADB002

# set flags to 0
.set FLAGS,    0

# set the checksum
.set CHECKSUM, -(MAGIC + FLAGS)

# set multiboot enabled
.section .multiboot

# define type to long for each data defined as above
.long MAGIC
```

```
.long FLAGS
.long CHECKSUM


# set the stack bottom
stackBottom:

# define the maximum size of stack to 5
.skip 1024


# set the stack top which grows from hi
stackTop:

.section .text
.global _start
.type _start, @function


_start:

  # assign current stack pointer location to stackTop
    mov $stackTop, %esp

  # call the kernel main source
    call kernel_entry

    cli


# put system in infinite loop
hltLoop:

    hlt
    jmp hltLoop

.size _start, . - _start
```

We have defined a stack of size 1024 bytes and managed by stackBottom and stackTop identifiers. Then in _start, we are storing a current stack pointer, and calling the main function of a kernel(kernel_entry).

As you know, every process consists of different sections such as data, bss, rodata and text. You can see the each sections by compiling the source code without assembling it.

e.g.: Run the following command

gcc -S kernel.c

and see the kernel.S file.

And this sections requires a memory to store them, this memory size is provided by the linker image file.
Each memory is aligned with the size of each block.
It mostly require to link all the object files together to form a final kernel image.
Linker image file provides how much size should be allocated to each of the sections.

The information is stored in the final kernel image.
If you open the final kernel image(.bin file) i
the linker image file consists of an entry po
sections with size defined in the BLOCK key

Se connecter à codeproject.com avec Google

abdo sa
asaqim@gmail.com

Continuer en tant que abdo

Pour créer votre compte, Google partagera votre nom, votre adresse e-mail et votre photo de profil avec codeproject.com. Consultez les Règles de confidentialité et les Conditions d'utilisation de l'application codeproject.com.

linker.ld

C++

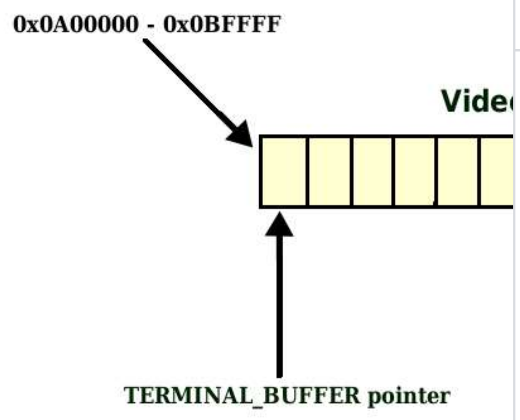
```
/* entry point of our kernel */
ENTRY(_start)

SECTIONS
{
    /* we need 1MB of space atleast */
    . = 1M;

    /* text section */
    .text BLOCK(4K) : ALIGN(4K)
    {
        *(.multiboot)
        *(.text)
    }

    /* read only data section */
    .rodata BLOCK(4K) : ALIGN(4K)
    {
        *(.rodata)
    }

    /* data section */
    .data BLOCK(4K) : ALIGN(4K)
    {
        *(.data)
    }

    /* bss section */
    .bss BLOCK(4K) : ALIGN(4K)
    {
        *(COMMON)
        *(.bss)
    }

}
```

Now you need a configuration file that instruct the grub to load menu with associated image file
grub.cfg

C++

```
menuentry "MyOS" {
    multiboot /boot/MyOS.bin
}
```

Now let's write a simple HelloWorld kernel code.



Simple :-

kernel_1 :-

kernel.h

C++ Shrink ▲

```
#ifndef KERNEL_H
#define KERNEL_H

typedef unsigned char uint8;
typedef unsigned short uint16;
typedef unsigned int uint32;


#define VGA_ADDRESS 0xB8000
#define BUFSIZE 2200

uint16* vga_buffer;

#define NULL 0

enum vga_color {
    BLACK,
    BLUE,
    GREEN,
    CYAN,
    RED,
    MAGENTA,
    BROWN,
    GREY,
    DARK_GREY,
    BRIGHT_BLUE,
    BRIGHT_GREEN,
    BRIGHT_CYAN,
    BRIGHT_RED,
    BRIGHT_MAGENTA,
    YELLOW,
    WHITE,
};

#endif
```

Se connecter à codeproject.com avec Google

abdo sa
asaqim@gmail.com

Continuer en tant que abdo

Pour créer votre compte, Google partagera votre nom, votre adresse e-mail et votre photo de profil avec codeproject.com. Consultez les Règles de confidentialité et les Conditions d'utilisation de l'application codeproject.com.

Here we are using 16 bit vga buffer, on my
bit starts at **0xA0000**.
An unsigned 16 bit type terminal buffer poi
It has 8*16 pixel font size.
see above image.

kernel.c

C++

```
#include "kernel.h"

/*
16 bit video buffer elements(register ax)
8 bits(ah) higher :
  lower 4 bits - forec olor
  higher 4 bits - back color

8 bits(al) lower :
  8 bits : ASCII character to print
*/
uint16 vga_entry(unsigned char ch, uint8 fore_color, uint8 back_color)
{
  uint16 ax = 0;
  uint8 ah = 0, al = 0;

  ah = back_color;
  ah <<= 4;
  ah |= fore_color;
  ax = ah;
  ax <<= 8;
  al = ch;
  ax |= al;

  return ax;
}

//clear video buffer array
void clear_vga_buffer(uint16 **buffer, uint8 fore_color, uint8 back_color)
{
  uint32 i;
  for(i = 0; i < BUFSIZE; i++){
    (*buffer)[i] = vga_entry(NULL, fore_color, back_color);
  }
}

//initialize vga buffer
void init_vga(uint8 fore_color, uint8 back_color)
{
  vga_buffer = (uint16*)VGA_ADDRESS;  //point vga_buffer pointer to VGA_ADDRESS
  clear_vga_buffer(&vga_buffer, fore_color, back_color);  //clear buffer
}
```

```
void kernel_entry()
{
  //first init vga with fore & back col
  init_vga(WHITE, BLACK);

  //assign each ASCII character to vide
  //you can change colors here
  vga_buffer[0] = vga_entry('H', WHITE,
  vga_buffer[1] = vga_entry('e', WHITE,
  vga_buffer[2] = vga_entry('l', WHITE,
  vga_buffer[3] = vga_entry('l', WHITE,
  vga_buffer[4] = vga_entry('o', WHITE,
  vga_buffer[5] = vga_entry(' ', WHITE,
  vga_buffer[6] = vga_entry('W', WHITE,
  vga_buffer[7] = vga_entry('o', WHITE,
  vga_buffer[8] = vga_entry('r', WHITE,
  vga_buffer[9] = vga_entry('l', WHITE,
  vga_buffer[10] = vga_entry('d', WHITE
}
```

The value returned by **vga_entry()** function is the **uint16** type with highlighting the character to print it with color.
The value is stored in the buffer to display the characters on a screen.
First lets point our pointer **vga_buffer** to VGA address **0xB8000**.

Segment : 0xB800 & Offset : 0(our index variable(vga_index))
Now you have an array of VGA, you just need to assign specific value to each index of array according to what to print on a screen as we usually do in assigning the value to array.
See the above code that prints each character of HelloWorld on a screen.

Ok lets compile the source.
type sh run.sh command on terminal.

run.sh

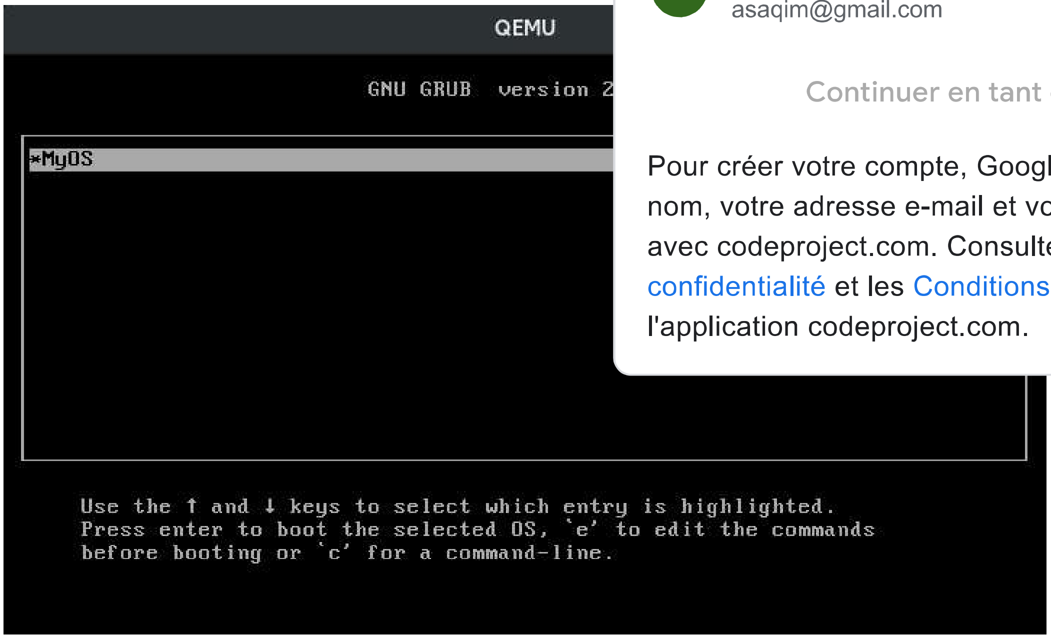
Python

```
#assemble boot.s file
as --32 boot.s -o boot.o

#compile kernel.c file
gcc -m32 -c kernel.c -o kernel.o -std=gnu99 -ffreestanding -O2 -Wall -Wextra

#linking the kernel with kernel.o and boot.o files
ld -m elf_i386 -T linker.ld kernel.o boot.o -o MyOS.bin -nostdlib

#check MyOS.bin file is x86 multiboot file or not
grub-file --is-x86-multiboot MyOS.bin

#building the iso file
mkdir -p isodir/boot/grub
cp MyOS.bin isodir/boot/MyOS.bin
cp grub.cfg isodir/boot/grub/grub.cfg
grub-mkrescue -o MyOS.iso isodir
```

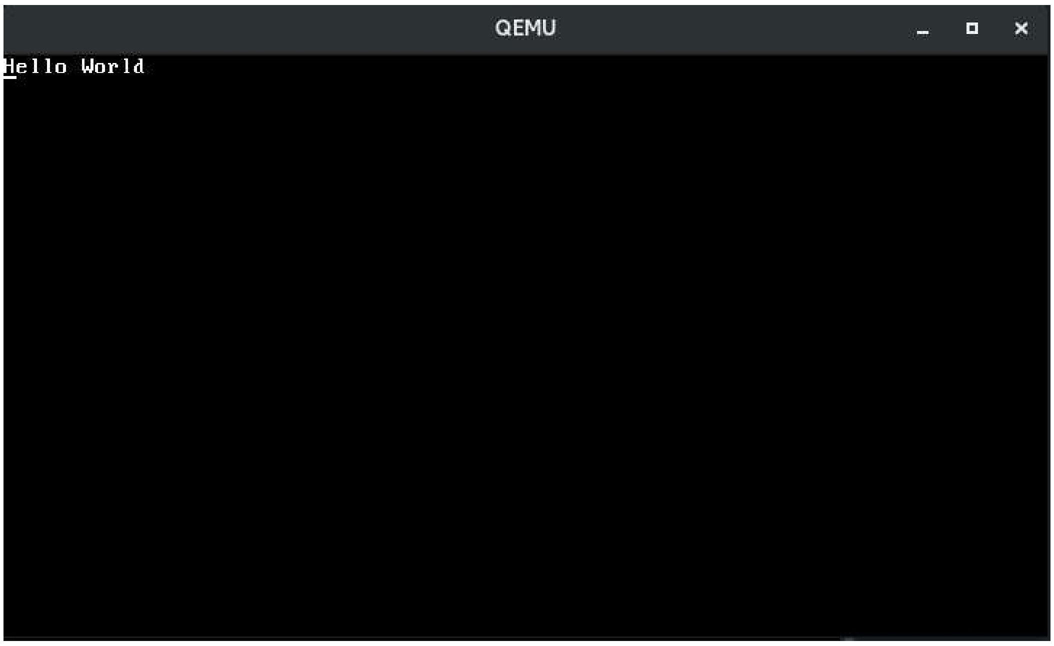
```
#run it in qemu
qemu-system-x86_64 -cdrom MyOS.iso
```

Make sure you have installed all the packag

the output is



```
GNU GRUB  version 2

*MyOS

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, `e' to edit the commands
before booting or `c' for a command-line.
```



```
Hello World
```

As you can see, it is a overhead to assign each and every value to VGA buffer, so we can write a function for that, which can print our string on a screen (means assigning each character value to VGA buffer from a string).

kernel_2 :-

kernel.h

C++ Shrink ▲

```
#ifndef KERNEL_H
#define KERNEL_H

typedef unsigned char uint8;
typedef unsigned short uint16;
typedef unsigned int uint32;


#define VGA_ADDRESS 0xB8000
#define BUFSIZE 2200

uint16* vga_buffer;

#define NULL 0

enum vga_color {
    BLACK,
    BLUE,
    GREEN,
    CYAN,
    RED,
    MAGENTA,
    BROWN,
    GREY,
    DARK_GREY,
    BRIGHT_BLUE,
    BRIGHT_GREEN,
    BRIGHT_CYAN,
    BRIGHT_RED,
    BRIGHT_MAGENTA,
    YELLOW,
    WHITE,
};

#endif
```

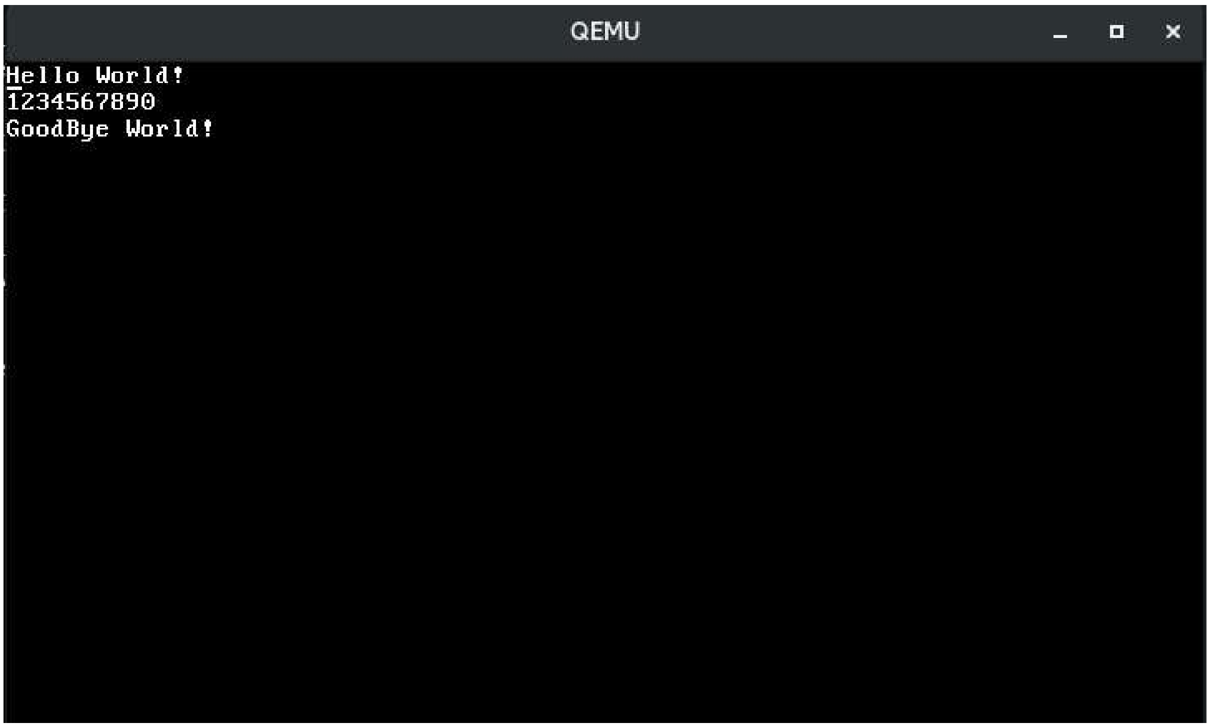
digit_ascii_codes are hexadecimal values of characters 0 to 9. we need them when we want to print them on a screen.vga_index is the our VGA array index. vga_index is increased when value is assigned to that index.To print an 32 bit integer, first you need to convert it into a string and then print a string.
BUFSIZE is the limit of our VGA. To print a new line, you have to skip some bytes in VGA pointer(vga_buffer) according to the pixel font size.
For this we need another variable that stores the current line index(next_line_index).

C++ Shrink ▲

```
#include "kernel.h"

//index for video buffer array
uint32 vga_index;
//counter to store new lines
static uint32 next_line_index = 1;
//fore & back color values
uint8 g_fore_color = WHITE, g_back_color = BLUE;
//digit ascii code for printing integers
int digit_ascii_codes[10] = {0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38,
0x39};
```

```
/*
16 bit video buffer elements(register ax)
8 bits(ah) higher :
  lower 4 bits - forec olor
  higher 4 bits - back color

8 bits(al) lower :
  8 bits : ASCII character to print
*/
uint16 vga_entry(unsigned char ch, uint
{
  uint16 ax = 0;
  uint8 ah = 0, al = 0;

  ah = back_color;
  ah <<= 4;
  ah |= fore_color;
  ax = ah;
  ax <<= 8;
  al = ch;
  ax |= al;

  return ax;
}

//clear video buffer array
void clear_vga_buffer(uint16 **buffer, uint8 fore_color, uint8 back_color)
{
  uint32 i;
  for(i = 0; i < BUFSIZE; i++){
    (*buffer)[i] = vga_entry(NULL, fore_color, back_color);
  }
  next_line_index = 1;
  vga_index = 0;
}

//initialize vga buffer
void init_vga(uint8 fore_color, uint8 back_color)
{
  vga_buffer = (uint16*)VGA_ADDRESS;
  clear_vga_buffer(&vga_buffer, fore_color, back_color);
  g_fore_color = fore_color;
  g_back_color = back_color;
}

/*
increase vga_index by width of row(80)
*/
void print_new_line()
{
  if(next_line_index >= 55){
    next_line_index = 0;
    clear_vga_buffer(&vga_buffer, g_fore_color, g_back_color);
  }
  vga_index = 80*next_line_index;
  next_line_index++;
}

//assign ascii character to video buffer
void print_char(char ch)
{
  vga_buffer[vga_index] = vga_entry(ch, g_fore_color, g_back_color);
  vga_index++;
}


uint32 strlen(const char* str)
{
  uint32 length = 0;
  while(str[length])
    length++;
  return length;
}

uint32 digit_count(int num)
{
  uint32 count = 0;
  if(num == 0)
    return 1;
  while(num > 0){
    count++;
    num = num/10;
  }
  return count;
}

void itoa(int num, char *number)
{
  int dgcount = digit_count(num);
  int index = dgcount - 1;
  char x;
  if(num == 0 && dgcount == 1){
    number[0] = '0';
    number[1] = '\0';
  }else{
    while(num != 0){
      x = num % 10;
      number[index] = x + '0';
      index--;
      num = num / 10;
    }
    number[dgcount] = '\0';
  }
}

//print string by calling print_char
void print_string(char *str)
{
  uint32 index = 0;
  while(str[index]){
    print_char(str[index]);
    index++;
  }
}

//print int by converting it into string
//& then printing string
void print_int(int num)
{
  char str_num[digit_count(num)+1];
  itoa(num, str_num);
  print_string(str_num);
```

```
}


void kernel_entry()
{
  //first init vga with fore & back col
  init_vga(WHITE, BLACK);

  /*call above function to print someth
    here to change the fore & back colc
    assign g_fore_color & g_back_color
    g_fore_color = BRIGHT_RED;
  */
  print_string("Hello World!");
  print_new_line();
  print_int(123456789);
  print_new_line();
  print_string("Goodbye World!");

}
```



```
Hello World!
1234567890
GoodBye World!
```

As you can see it is the overhead to call each and every function for displaying the values, that's why C programming provides a **printf()** function with format specifiers which print/set specific value to standard output device with each specifier with literals such as \n, \t, \r etc.

Keyboard :-

For keyboard I/O, use port number 0x60 with instructions in/out.Download kernel_source code fro keyboard. It reads keystrokes from user and displays them on screen.

C++ Shrink ▲

```
#ifndef KEYBOARD_H
#define KEYBOARD_H

#define KEYBOARD_PORT 0x60


#define KEY_A 0x1E
#define KEY_B 0x30
#define KEY_C 0x2E
#define KEY_D 0x20
#define KEY_E 0x12
#define KEY_F 0x21
#define KEY_G 0x22
#define KEY_H 0x23
#define KEY_I 0x17
#define KEY_J 0x24
#define KEY_K 0x25
#define KEY_L 0x26
#define KEY_M 0x32
#define KEY_N 0x31
#define KEY_O 0x18
#define KEY_P 0x19
#define KEY_Q 0x10
#define KEY_R 0x13
#define KEY_S 0x1F
#define KEY_T 0x14
#define KEY_U 0x16
#define KEY_V 0x2F
#define KEY_W 0x11
#define KEY_X 0x2D
#define KEY_Y 0x15
#define KEY_Z 0x2C
#define KEY_1 0x02
#define KEY_2 0x03
#define KEY_3 0x04
#define KEY_4 0x05
#define KEY_5 0x06
#define KEY_6 0x07
#define KEY_7 0x08
#define KEY_8 0x09
#define KEY_9 0x0A
#define KEY_0 0x0B
#define KEY_MINUS 0x0C
#define KEY_EQUAL 0x0D
#define KEY_SQUARE_OPEN_BRACKET 0x1A
#define KEY_SQUARE_CLOSE_BRACKET 0x1B
#define KEY_SEMICOLON 0x27
#define KEY_BACKSLASH 0x2B
#define KEY_COMMA 0x33
#define KEY_DOT 0x34
#define KEY_FORESLHASH 0x35
#define KEY_F1 0x3B
#define KEY_F2 0x3C
#define KEY_F3 0x3D
#define KEY_F4 0x3E
#define KEY_F5 0x3F
#define KEY_F6 0x40
#define KEY_F7 0x41
#define KEY_F8 0x42
```

```
#define KEY_F9 0x43
#define KEY_F10 0x44
#define KEY_F11 0x85
#define KEY_F12 0x86
#define KEY_BACKSPACE 0x0E
#define KEY_DELETE 0x53
#define KEY_DOWN 0x50
#define KEY_END 0x4F
#define KEY_ENTER 0x1C
#define KEY_ESC 0x01
#define KEY_HOME 0x47
#define KEY_INSERT 0x52
#define KEY_KEYPAD_5 0x4C
#define KEY_KEYPAD_MUL 0x37
#define KEY_KEYPAD_Minus 0x4A
#define KEY_KEYPAD_PLUS 0x4E
#define KEY_KEYPAD_DIV 0x35
#define KEY_LEFT 0x4B
#define KEY_PAGE_DOWN 0x51
#define KEY_PAGE_UP 0x49
#define KEY_PRINT_SCREEN 0x37
#define KEY_RIGHT 0x4D
#define KEY_SPACE 0x39
#define KEY_TAB 0x0F
#define KEY_UP 0x48


#endif
```

inb() receives byte from specified port and returns it.

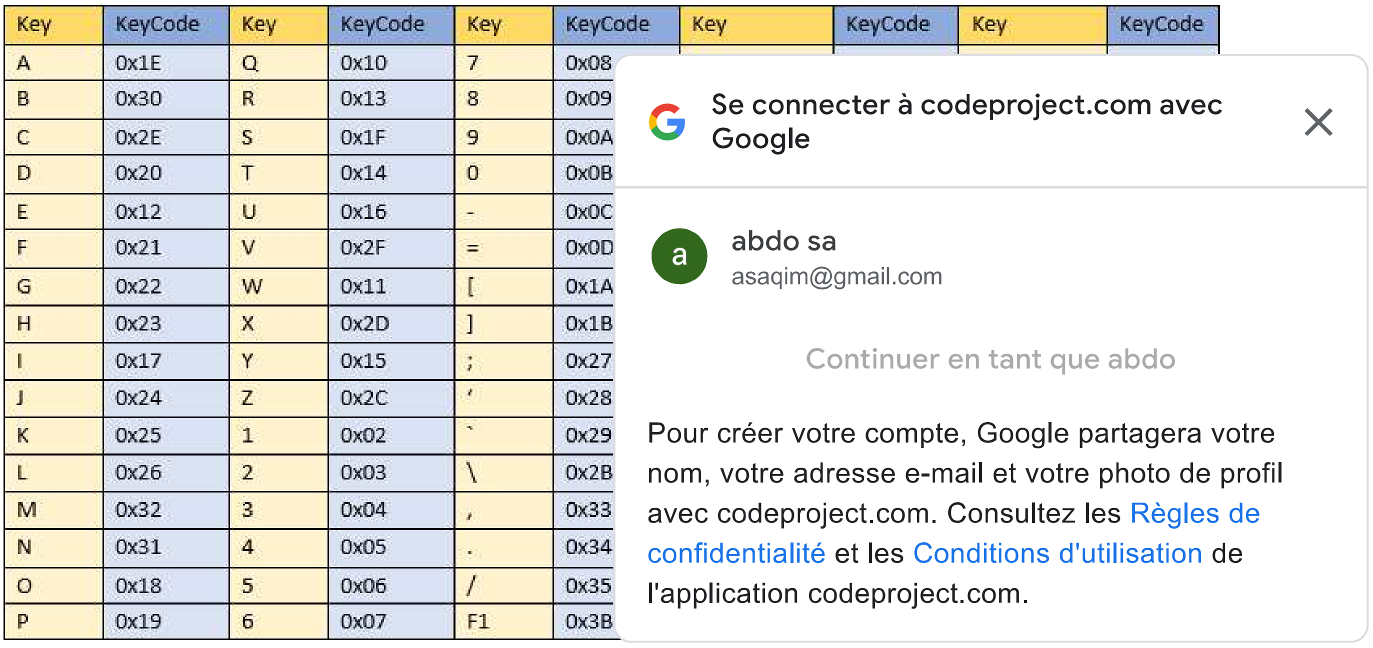
outb() send byte to specified port.

C++ Shrink ▲

```
uint8 inb(uint16 port)
{
  uint8 ret;
  asm volatile("inb %1, %0" : "=a"(ret) : "d"(port));
  return ret;
}

void outb(uint16 port, uint8 data)
{
  asm volatile("outb %0, %1" : "=a"(data) : "d"(port));
}

char get_input_keycode()
{
  char ch = 0;
  while((ch = inb(KEYBOARD_PORT)) != 0){
    if(ch > 0)
      return ch;
  }
  return ch;
}

/*
keep the cpu busy for doing nothing(nop)
so that io port will not be processed by cpu
```

```
here timer can also be used, but lets do this in looping counter
*/
void wait_for_io(uint32 timer_count)
{
  while(1){
    asm volatile("nop");
    timer_count--;
    if(timer_count <= 0)
      break;
    }
}

void sleep(uint32 timer_count)
{
  wait_for_io(timer_count);
}

void test_input()
{
  char ch = 0;
  char keycode = 0;
  do{
    keycode = get_input_keycode();
    if(keycode == KEY_ENTER){
      print_new_line();
    }else{
      ch = get_ascii_char(keycode);
      print_char(ch);
    }
    sleep(0x02FFFFFF);
  }while(ch > 0);
}

void kernel_entry()
{
  init_vga(WHITE, BLUE);
  print_string("Type here, one key per second, ENTER to go to next line");
  print_new_line();
  test_input();

}
```

Each keycode is being converted into its ASCII character by function **get_ascii_char()**.



Key	KeyCode	Key	KeyCode	Key	KeyCode	Key	KeyCode	Key	KeyCode
A	0x1E	Q	0x10	7	0x08				
B	0x30	R	0x13	8	0x09				
C	0x2E	S	0x1F	9	0x0A				
D	0x20	T	0x14	0	0x0B				
E	0x12	U	0x16	-	0x0C				
F	0x21	V	0x2F	=	0x0D				
G	0x22	W	0x11	[	0x1A				
H	0x23	X	0x2D	]	0x1B				
I	0x17	Y	0x15	;	0x27				
J	0x24	Z	0x2C	'	0x28				
K	0x25	1	0x02	`	0x29				
L	0x26	2	0x03	\	0x2B				
M	0x32	3	0x04	,	0x33				
N	0x31	4	0x05	.	0x34				
O	0x18	5	0x06	/	0x35				
P	0x19	6	0x07	F1	0x3B				

Se connecter à codeproject.com avec Google

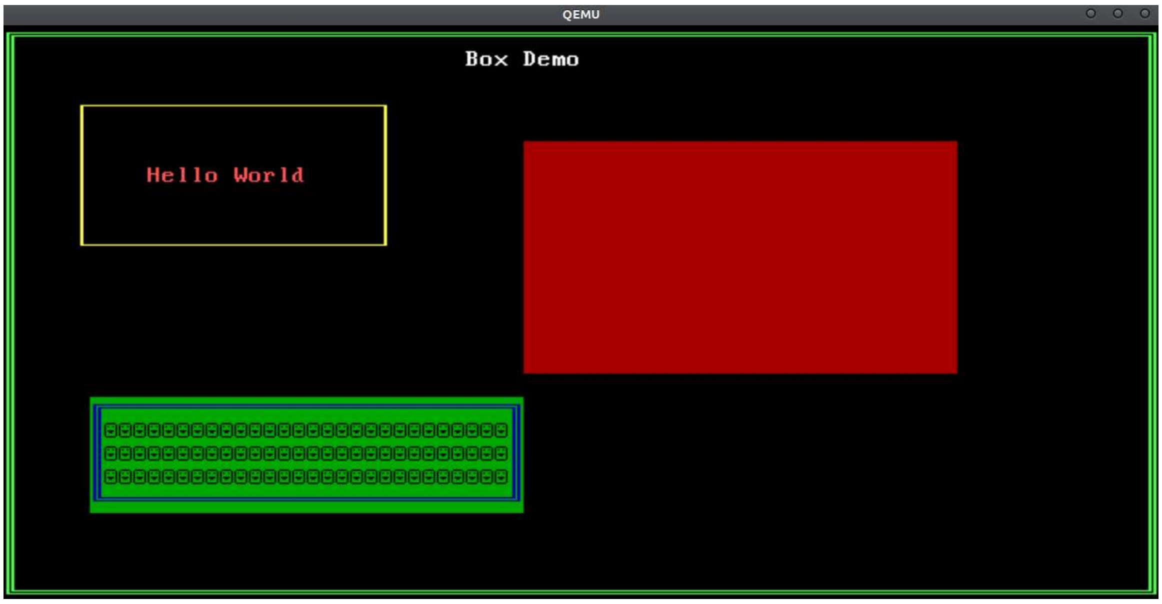
abdo sa
asaqim@gmail.com

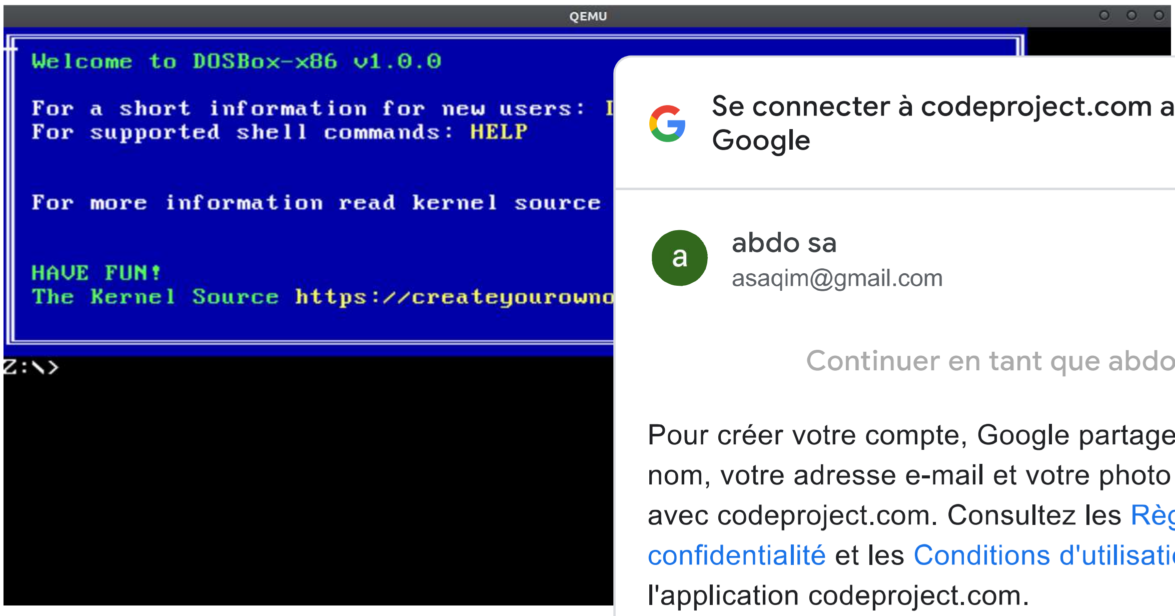
Continuer en tant que abdo

Pour créer votre compte, Google partagera votre nom, votre adresse e-mail et votre photo de profil avec codeproject.com. Consultez les Règles de confidentialité et les Conditions d'utilisation de l'application codeproject.com.

Box-drawing GUI :-

Download the kernel_source for drawing boxes used in old systems such as DOSBox etc. (kernel_source/GUI/)



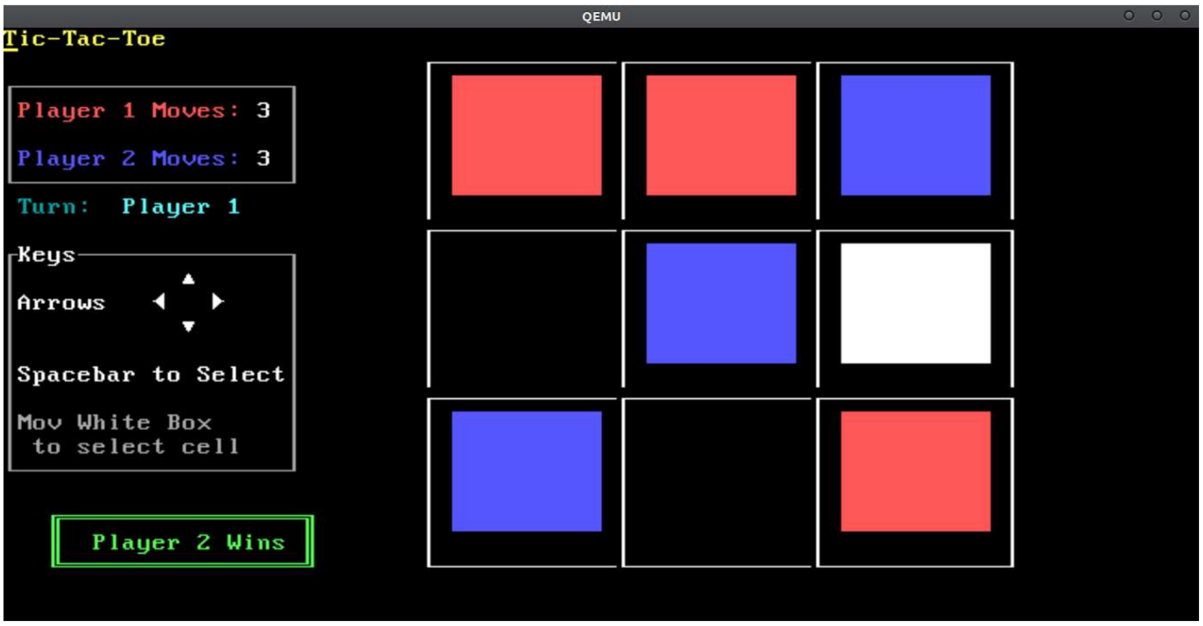


```
Welcome to DOSBox-x86 v1.0.0

For a short information for new users: I
For supported shell commands: HELP


For more information read kernel source


HAVE FUN!
The Kernel Source https://createyourowno
Z:\>
```

Tic-Tac-Toe Game :-

We have printing code, keyboard I/O handling and GUI using box drawing characters.So lets write a simple Tic-Tac-Toe game in kernel that can be run on any PC.

Download the kernel_source code, kernel_source/Tic-Tac-Toe.



How to Play :

Use arrow keys(UP, DOWN, LEFT, RIGHT) to move white box between cells and press SPACEBAR to select that cell.

RED color for player 1 Box and BLUE color for player 2 box.

See Turn for which player has a turn to select cell.(Turn: Player 1)

If you are running this on actual hardware then increase the value of **sleep()** function in **launch_game()** in **tic_tac_toe.c** and in **kerr**
not too fast. I used 0x2FFFFFFF.

For more about os from scratch, os calculat s
link.

Se connecter à codeproject.com avec Google ✕

abdo sa
asaqim@gmail.com

Continuer en tant que abdo

Pour créer votre compte, Google partagera votre nom, votre adresse e-mail et votre photo de profil avec codeproject.com. Consultez les Règles de confidentialité et les Conditions d'utilisation de l'application codeproject.com.

References

- http://wiki.osdev.org/Expanded_Main
- http://www.brokenthorn.com/
- http://mikeos.sourceforge.net/
- https://github.com/pritamzope/OS

License

Written By
Pritam Zope
Software Developer
India

Software Engineer

Comments and Discussions

You must Sign In to use this message board.

Search Comments

Se connecter à codeproject.com avec Google

abdo sa
asaqim@gmail.com

Continuer en tant que abdo

Pour créer votre compte, Google partagera votre nom, votre adresse e-mail et votre photo de profil avec codeproject.com. Consultez les Règles de confidentialité et les Conditions d'utilisation de l'application codeproject.com.

Amazing article! I'd love to see a follo memory/protected mode
Alexandr Savochkin 17-Feb-22 6:06

not exiting. also tried removing "hltl
Member 15369112 29-Sep-21 22:34

Re: not exiting. also tried removing "h
Member 15369112 20-Oct-21 19:4

says "qemu-system-x86_64" is not fo
Member 15369112 23-Sep-21 19:49

Re: says "qemu-system-x86_64" is no
Raj P Sep2021 26-Sep-21 2:09

Re: says "qemu-system-x86_64" is not found right after installing
Member 15369112 29-Sep-21 20:11

Re: says "qemu-system-x86_64" is not found right after installing
Pritam Zope 2-Oct-21 21:39

File System
Member 14893727 19-Jul-20 20:02

backspace
Member 14683940 10-Dec-19 5:53

Re: backspace
Pritam Zope 2-Oct-21 21:40

Can't execute with qemu
Member 14123725 26-May-19 12:29

Boot.s won't compile. How do I fix it?
Member 14364916 12-May-19 7:09

Compile with GCC 5.4
Member 14024307 8-Nov-18 13:43

Re: Compile with GCC 5.4
Pritam Zope 16-Nov-18 7:03

error multiboot header not found
Member 12276590 26-Sep-18 0:02

Re: error multiboot header not found
Member 14024307 8-Nov-18 13:45

Cannot Compile, this is my output
Chris Long 27-Jul-18 10:51

Re: Cannot Compile, this is my output
Member 14024307 8-Nov-18 13:5

Re: Cannot Compile, this is my out
quadbyt 19-Dec-18 11:28

Re: Cannot Compile, this is my o
Member 14232945 22-Jul-19

About the mentioned article
xxmarcxx 1-Mar-18 23:49

My vote of 5
Jose A Pascoa 12-Feb-18 21:58

Se connecter à codeproject.com avec Google

abdo sa
asaqim@gmail.com

Continuer en tant que abdo

Pour créer votre compte, Google partagera votre nom, votre adresse e-mail et votre photo de profil avec codeproject.com. Consultez les Règles de confidentialité et les Conditions d'utilisation de l'application codeproject.com.

Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.