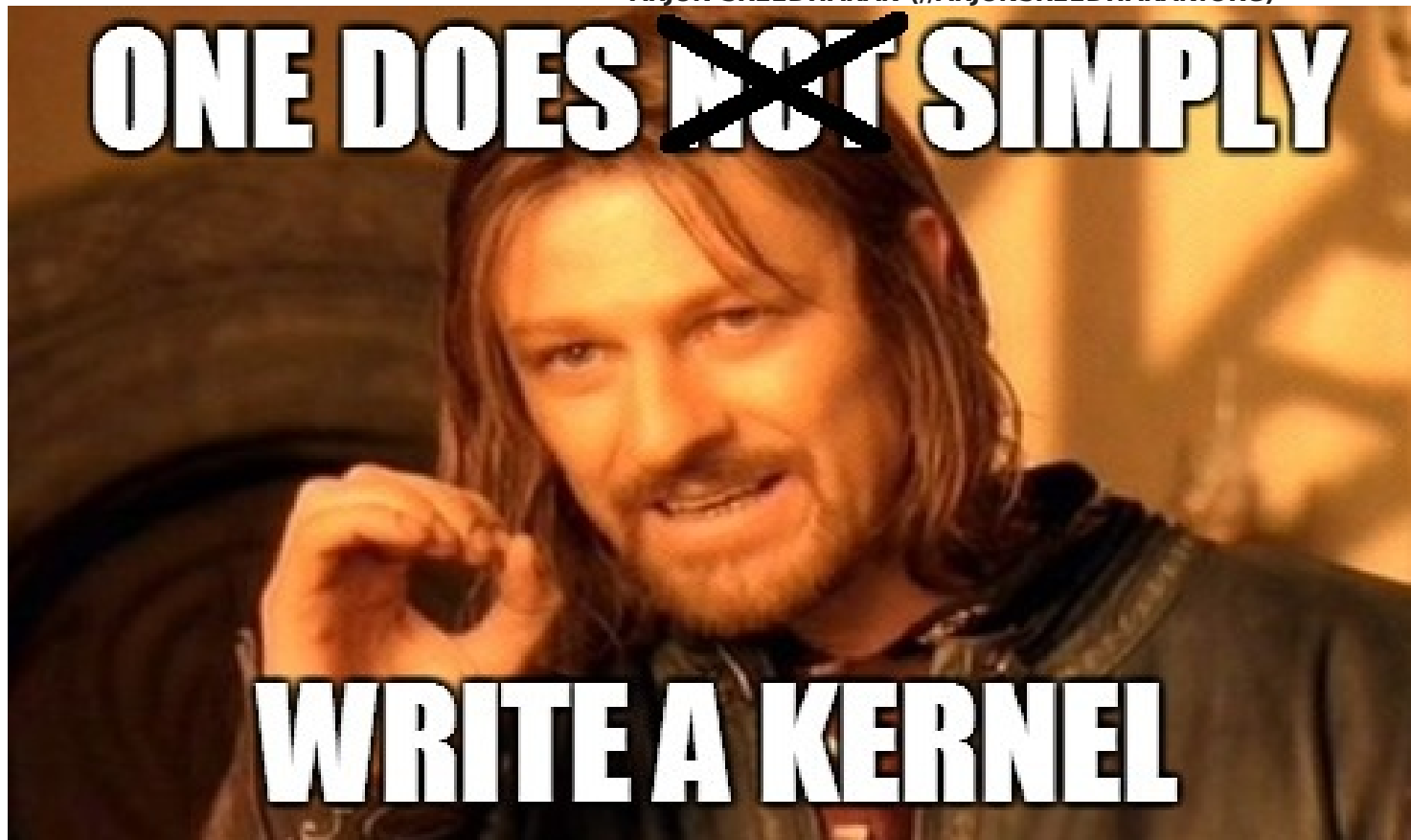




Kernel 101 - Let's write a Kernel (<http://arjunsreedharan.org/post/82710718100/kernel-101-lets-write-a-kernel>)

Hello World,

Let us write a simple kernel which could be loaded with the GRUB boot loader on an x86 system. This kernel will display a message on the screen and then hang.



How does an x86 machine boot

Before we think about writing a kernel, let's see how the machine boots up and transfers control to the kernel:

The x86 CPU is hardwired to begin execution at the physical address [**0xFFFFFFFF0**]. It is in fact, the last 16 bytes of the 32-bit address space. This address just contains a jump instruction to the address in memory where BIOS has copied itself.

Thus, the BIOS code starts its execution. BIOS first searches for a bootable device in the configured boot device order. It checks for a certain magic number to determine if the device is bootable or not.

Once the BIOS has found a bootable device, it copies the contents of the device's first sector into RAM starting from physical address [**0x7c00**]; and then jumps into the address and executes the code just loaded. This code is called the **bootloader**.

The bootloader then loads the kernel at the physical address [**0x100000**]. The address [0x100000] is used as the start-address for all big kernels on x86 machines.

What all do we need?

- * An x86 computer (of course)
- * Linux
- * [NASM assembler](#)
- * [gcc](#)
- * [ld \(GNU Linker\)](#)
- * [grub \(//www.nasm.us/\)](http://www.nasm.us/)

Source Code

Source code is available at my [Github repository - mkernel \(//github.com/arjun024/mkernel\)](https://github.com/arjun024/mkernel)

The entry point using assembly

We like to write everything in C, but we cannot avoid a little bit of assembly. We will write a small file in x86 assembly-language that serves as the starting point for our kernel. All our assembly file will do is invoke an external function which we will write in C, and then halt the program flow.

How do we make sure that this assembly code will serve as the starting point of the kernel?

We will use a linker script that links the object files to produce the final kernel executable. (more explained later) In this linker script, we will explicitly specify that we want our binary to be loaded at the address [0x100000]. This address, as I have said earlier, is where the kernel is expected to be. Thus, the bootloader will take care of firing the kernel's entry point.

Here's the assembly code:

```
;;kernel.asm
bits 32                ;nasm directive - 32 bit
section .text

global start
extern kmain           ;kmain is defined in the c file

start:
    cli                ;block interrupts
    mov esp, stack_space ;set stack pointer
    call kmain
```

```

    hlt                ;halt the CPU

section .bss
resb 8192              ;8KB for stack
stack_space:

```

The first instruction `bits 32` is not an x86 assembly instruction. It's a directive to the NASM assembler that specifies it should generate code to run on a processor operating in 32 bit mode. It is not mandatorily required in our example, however is included here as it's good practice to be explicit.

The second line begins the text section (aka code section). This is where we put all our code.

`global` is another NASM directive to set symbols from source code as global. By doing so, the linker knows where the symbol `start` is; which happens to be our entry point.

`kmain` is our function that will be defined in our `kernel.c` file. `extern` declares that the function is declared elsewhere.

Then, we have the `start` function, which calls the `kmain` function and halts the CPU using the `hlt` instruction. Interrupts can awake the CPU from an `hlt` instruction. So we disable interrupts beforehand using `cli` instruction. `cli` is short for clear-interrupts.

We should ideally set aside some memory for the stack and point the stack pointer (`esp`) to it. However, it seems like GRUB does this for us and the stack pointer is already set at this point. But, just to be sure, we will allocate some space in the BSS section and point the stack pointer to the beginning of the allocated memory. We use the `resb` instruction which reserves memory given in bytes. After it, a label is left which will point to the edge of the reserved piece of memory. Just before the `kmain` is called, the stack pointer (`esp`) is made to point to this space using the `mov` instruction.

The kernel in C

In `kernel.asm`, we made a call to the function `kmain()`. So our C code will start executing at `kmain()`:

```

/*
 * kernel.c
 */
void kmain(void)
{
    const char *str = "my first kernel";
    char *vidptr = (char*)0xb8000; //video mem begins here.
    unsigned int i = 0;
    unsigned int j = 0;

    /* this loops clears the screen
     * there are 25 lines each of 80 columns; each element takes 2 bytes */
    while(j < 80 * 25 * 2) {
        /* blank character */
        vidptr[j] = ' ';
        /* attribute-byte - light grey on black screen */
        vidptr[j+1] = 0x07;
    }
}

```

```

        j = j + 2;
    }

    j = 0;

    /* this loop writes the string to video memory */
    while(str[j] != '\0') {
        /* the character's ascii */
        vidptr[i] = str[j];
        /* attribute-byte: give character black bg and light grey fg */
        vidptr[i+1] = 0x07;
        ++j;
        i = i + 2;
    }
    return;
}

```

All our kernel will do is clear the screen and write to it the string “my first kernel”.

First we make a pointer `vidptr` that points to the address **[0xb8000]**. This address is the start of video memory in protected mode. The screen’s text memory is simply a chunk of memory in our address space. The memory mapped input/output for the screen starts at [0xb8000] and supports 25 lines, each line contain 80 ascii characters.

Each character element in this text memory is represented by 16 bits (2 bytes), rather than 8 bits (1 byte) which we are used to. The first byte should have the representation of the character as in ASCII. The second byte is the `attribute-byte`. This describes the formatting of the character including attributes such as color.

To print the character `s` in green color on black background, we will store the character `s` in the first byte of the video memory address and the value [0x02] in the second byte. `0` represents black background and `2` represents green foreground.

Have a look at table below for different colors:

0 - Black, 1 - Blue, 2 - Green, 3 - Cyan, 4 - Red, 5 - Magenta, 6 - Brown, 7 - Light Grey, 8 - Dark Grey, 9 - Light Blue, 10/a - Light Green, 11/b - Light Cyan, 12/c - Light Red, 13/d - Light Magenta, 14/e - Light Brown, 15/f – White.

In our kernel, we will use light grey character on a black background. So our attribute-byte must have the value [0x07].

In the first while loop, the program writes the blank character with [0x07] attribute all over the 80 columns of the 25 lines. This thus clears the screen.

In the second while loop, characters of the null terminated string “my first kernel” are written to the chunk of video memory with each character holding an attribute-byte of [0x07].

This should display the string on the screen.

The linking part

We will assemble `kernel.asm` with NASM to an object file; and then using GCC we will compile `kernel.c` to another object file. Now, our job is to get these objects linked to an executable bootable kernel.

For that, we use an explicit linker script, which can be passed as an argument to `ld` (our linker).

```

/*
 * link.ld
 */
OUTPUT_FORMAT(elf32-i386)
ENTRY(start)
SECTIONS
{
    . = 0x100000;
    .text : { *(.text) }
    .data : { *(.data) }
    .bss  : { *(.bss) }
}

```

First, we set the **output format** of our output executable to be 32 bit [Executable and Linkable Format \(//elinux.org/Executable_and_Linkable_Format_\(ELF\)\)](http://elinux.org/Executable_and_Linkable_Format_(ELF)) (ELF). ELF is the standard binary file format for Unix-like systems on x86 architecture.

ENTRY takes one argument. It specifies the symbol name that should be the entry point of our executable.

SECTIONS is the most important part for us. Here, we define the layout of our executable. We could specify how the different sections are to be merged and at what location each of these is to be placed.

Within the braces that follow the SECTIONS statement, the period character (.) represents the **location counter**.

The location counter is always initialized to [0x0] at beginning of the SECTIONS block. It can be modified by assigning a new value to it.

Remember, earlier I told you that kernel's code should start at the address [0x100000]. So, we set the location counter to [0x100000].

Have look at the next line **.text : { *(.text) }**

The asterisk (*) is a wildcard character that matches any file name. The expression `*(.text)` thus means all `.text` input sections from all input files.

So, the linker merges all text sections of the object files to the executable's text section, at the address stored in the location counter. Thus, the code section of our executable begins at [0x100000].

After the linker places the text output section, the value of the location counter will become 0x1000000 + the size of the text output section.

Similarly, the data and bss sections are merged and placed at the then values of location-counter.

Grub and Multiboot

Now, we have all our files ready to build the kernel. But, since we like to boot our kernel with the [GRUB bootloader \(//www.gnu.org/software/grub/\)](http://www.gnu.org/software/grub/), there is one step left.

There is a standard for loading various x86 kernels using a boot loader; called as **Multiboot specification**.

GRUB will only load our kernel if it complies with the [Multiboot spec \(//www.gnu.org/software/grub/manual/multiboot/multiboot.html\)](http://www.gnu.org/software/grub/manual/multiboot/multiboot.html).

According to the spec, the kernel must contain a header (known as Multiboot header) within its first 8 KiloBytes.

Further, This Multiboot header must contain 3 fields that are 4 byte aligned namely:

- a **magic** field: containing the magic number **[0x1BADB002]**, to identify the header.
- a **flags** field: We will not care about this field. We will simply set it to zero.
- a **checksum** field: the checksum field when added to the fields 'magic' and 'flags' must give zero.

So our kernel.asm will become:

```
;;kernel.asm

;nasm directive - 32 bit
bits 32
section .text
    ;multiboot spec
    align 4
    dd 0x1BADB002        ;magic
    dd 0x00              ;flags
    dd - (0x1BADB002 + 0x00) ;checksum. m+f+c should be zero

global start
extern kmain             ;kmain is defined in the c file

start:
    cli                  ;block interrupts
    mov esp, stack_space ;set stack pointer
    call kmain
    hlt                  ;halt the CPU

section .bss
resb 8192                ;8KB for stack
stack_space:
```

The **dd** defines a double word of size 4 bytes.

Building the kernel

We will now create object files from `kernel.asm` and `kernel.c` and then link it using our linker script.

```
nasm -f elf32 kernel.asm -o kasm.o
```

will run the assembler to create the object file `kasm.o` in ELF-32 bit format.

```
gcc -m32 -c kernel.c -o kc.o
```

The `-c` option makes sure that after compiling, linking doesn't implicitly happen.

```
ld -m elf_i386 -T link.ld -o kernel kasm.o kc.o
```

will run the linker with our linker script and generate the executable named **kernel**.

Configure your grub and run your kernel

GRUB requires your kernel to be of the name pattern `kernel-<version>`. So, rename the kernel. I renamed my kernel executable to `kernel-701`.

Now place it in the **/boot** directory. You will require superuser privileges to do so.

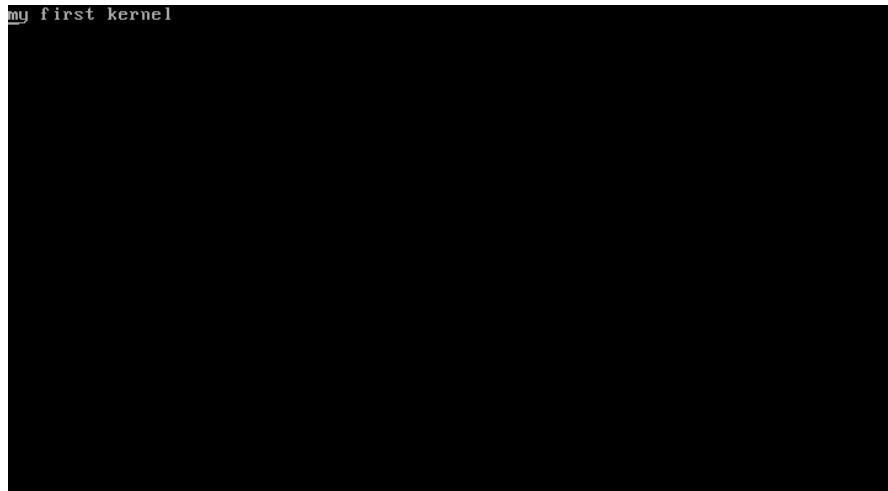
In your GRUB configuration file `grub.cfg` you should add an entry, something like:

```
title myKernel
    root (hd0,0)
    kernel /boot/kernel-701 ro
```

Don't forget to remove the directive `hiddenmenu` if it exists.

Reboot your computer, and you'll get a list selection with the name of your kernel listed.

Select it and you should see:



That's your kernel!!

PS:
* It's always advisable to get yourself a virtual machine for all kinds of kernel hacking.

* To run this on **grub2** which is the default bootloader for newer distros, your config should look like this (Thanks to [Rubén Laguna \(//rubenlaguna.com/\)](http://rubenlaguna.com/) from comments for the config):

```
menuentry 'kernel 7001' {
    set root='hd0,msdos1'
    multiboot /boot/kernel-7001 ro
}
```

* Also, if you want to run the kernel on the `qemu` emulator instead of booting with GRUB, you can do so by:

```
qemu-system-i386 -kernel kernel
```

Also, see the next article in the Kernel series:
[Kernel 201 - Let's write a Kernel with keyboard and screen support \(http://arjunsreedharan.org/post/99370248137/kernel-201-lets-write-a-kernel-with-keyboard-and\)](http://arjunsreedharan.org/post/99370248137/kernel-201-lets-write-a-kernel-with-keyboard-and)


263 points ([//reddit.com/r/programming/comments/2310xe/kernel_101_lets_write_a_kernel/](http://reddit.com/r/programming/comments/2310xe/kernel_101_lets_write_a_kernel/))  ([//reddit.com/r/programming/comments/2310xe/kernel_101_lets_write_a_kernel/](http://reddit.com/r/programming/comments/2310xe/kernel_101_lets_write_a_kernel/))

90 CommentsArjun Sreedharan

Recommend 9ShareSort by Best




Join the discussion...

- 


Ron • a year ago

great post. Didn't know it was this easy ;)

24 ^ | v • Reply • Share >
- 


Livid • a year ago

Great. Really appreciate it. Could you please write more about how to handle keyboard input, as well as a really basic file system? I'd love to learn more.

17 ^ | v • Reply • Share >
- 


Arjun Sreedharan Mod → Livid • a year ago

Hey, i am actually planning to write another post with addition of a keyboard driver among others. :)

35 ^ | v • Reply • Share >
- 

boris → Arjun Sreedharan • a year ago

I would be extremely enthusiastic about such a post! :]

8 ^ | v • Reply • Share >
- 

guybrush → Arjun Sreedharan • a year ago

Please do write about keyboard input, or share a tip on where to look next.

^ | v • Reply • Share >



Arjun Sreedharan Mod guybrush • 7 months ago

here's it - at last:

<http://arjunsreedharan.org/pos...>

5 | • Reply • Share >



capcase • a year ago

I am interested in a follow-up post about how to write your own bootloader. thanks!

11 | • Reply • Share >



Terrence Andrew Davis • a year ago

I wrote a kernel and compiler.

<http://www.templeos.org/Temple...>

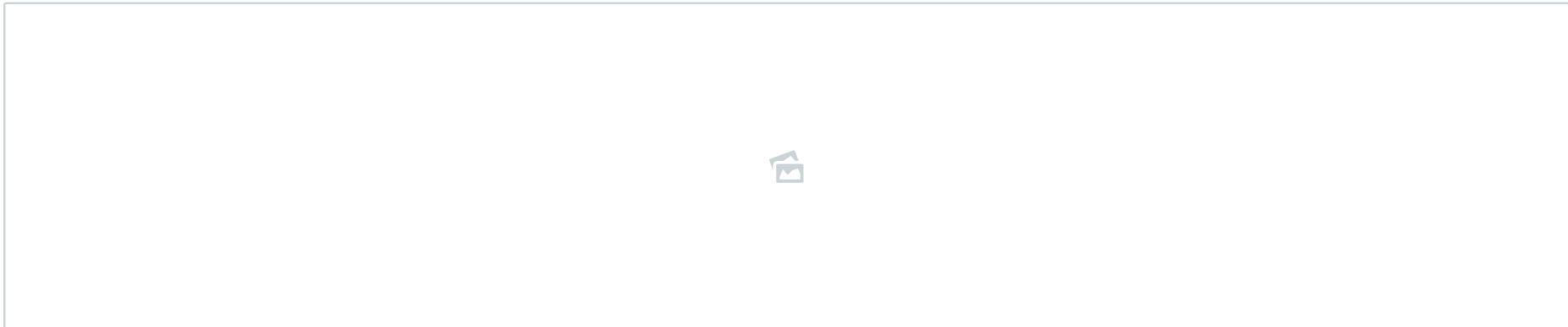
And other tools, and bootloaders, an editor, and a graphic's library.

15 | • Reply • Share >



Anon Terrence Andrew Davis • a year ago

Please leave troll.



7 | • Reply • Share >



Vidar Hokstad Anon • a year ago

He is known to have mental problems, but why do you feel the need to make a comment like that when he for once makes an on-topic, relevant comment without anything offensive?

16 | • Reply • Share >



AnonReply Anon • a year ago

He is not a troll but a religious fanatic with really good programing skills. I know him from other sites.

8 | • Reply • Share >



Anon Anon • a year ago

you are a little cock sucker, you fucking trolling nigger

1 | • Reply • Share >



Anon Terrence Andrew Davis • a year ago

Good work

2 | • Reply • Share >



Arjun Menon • a year ago

How does the CPU begin execution at physical address 0xFFFFFFF0 if you have less than 4 GiB of main memory? Suppose you had a system with 256 MiB of main memory – what physical address range will these 2^{28} bytes of memory correspond to? (Are physical addresses mapped in reverse order?) In addition, what subsystem is responsible for copying BIOS and placing the jump instruction at 0xFFFFFFF0 before BIOS starts executing? By the way, thanks for writing this piece!

3 | • Reply • Share >



James Cooper → Arjun Menon • a year ago

Every address is just a number that is broadcast on the address bus. What is listening to this address depends on what is connected. In a 32-bit memory space, the upper memory addresses are not mapped to RAM, but rather to memory mapped hardware. The BIOS ROM will also be mapped in here. Likewise for older 16-bit processors, where the video RAM and BIOS were mapped into parts of the memory between 640KB and 1MB.

For more info:

<http://en.wikipedia.org/wiki/B...>

<http://en.wikipedia.org/wiki/R...>

http://wiki.osdev.org/Memory_M...

3 ^ | v • Reply • Share >



James Cooper → James Cooper • a year ago

I forgot to elaborate that this is only for physical linear memory addresses. The x86 processor has many different addressing modes and those that are used by your typical 32-bit or 64-bit operating system (e.g. protected mode) will allow for things like virtual addresses, where the logical memory addresses referred to in a program are mapped through descriptor tables in the processor to some other physical address. But during initial boot, the processor will start in real mode. There are many layers of compatibility in place so that code from the original 8086 can still function on a modern x86 processor. It makes writing a bootloader and OS kernel quite an experience!

1 ^ | v • Reply • Share >



Arjun Sreedharan Mod → Arjun Menon • a year ago

Since RAM is faster than ROM, BIOS copies itself to the RAM. This is called `shadowing`. So on translation, the address [0xFFFFFFF0] could map to a BIOS memory block.

1 ^ | v • Reply • Share >



Rubén Laguna → Arjun Menon • a year ago

I found an excellent article that explains it. <http://duartes.org/gustavo/blo...>

Basically the north bridge chip (part of the chipset of the motherboard) routes memory requests, most of the time it routes to RAM modules but certain address are used for communication with PCI card, video memory, bios flash, etc. It's all on the link above.

^ | v • Reply • Share >



Budi Mulyawan → Arjun Menon • a year ago

I believe this is virtual address space <http://en.wikipedia.org/wiki/V...> which get mapped into physical ram

^ | v • Reply • Share >



James Cooper → Budi Mulyawan • a year ago

No, virtual memory only exists in protected mode. The processor starts in real mode for backwards compatibility.

1 ^ | v • Reply • Share >



yadli → James Cooper • a year ago

I think that will depend on the processor and the motherboard. The memory controller is linked to the memory banks, but there might be a multiplexer to wire the memory bus somewhere else, so that all accessible things appear on a same address line. (Now perhaps all these are integrated on the chip)

The name of this is not "virtual memory" but "unified addressing" imo.

Of course X86 has independent addresses for I/O ports, unlike some ARM chips.

^ | v • Reply • Share >



vikrantc14 • 5 months ago

Awesome! First step to my life's aim... You are my Guru from now on.

2 ^ | v • Reply • Share >



anees • a year ago

super duper post! hardly few people explain such stuff these days. Thanks a ton. Anxiously waiting for the next post.


2 ^ | v • Reply • Share >




joe • 9 months ago

u are the best man :)


1 ^ | v • Reply • Share >

 **Manoj** • a year ago
Great post! I had already tried this but I am eager to know how to write keyboard driver.


1 ^ | v • Reply • Share >

 **Shi Wei** • a year ago
interesting,like to learn more

1 ^ | v • Reply • Share >

 **Kieron** • a year ago
In the screen-clearing section, I'm having trouble understanding how the full screen is cleared, and not just half. If each character is represented by 2 bytes, shouldn't the 'while' loop be 'while (j < 80 * 25 * 2)'?

1 ^ | v • Reply • Share >

 **Arjun Sreedharan** Mod → Kieron • a year ago
You are right. Will Fix.Thanks


^ | v • Reply • Share >

 **Kieron** → Arjun Sreedharan • a year ago
Right, so it breaks out of the loop after 1000 iterations, or half of a 80 * 25 screen, no?


^ | v • Reply • Share >

 **Arjun Sreedharan** Mod → Kieron • a year ago
Right Kieron. Good catch. thanks :)


^ | v • Reply • Share >

 **Kieron** → Arjun Sreedharan • a year ago
I thought I was losing my mind :)

^ | v • Reply • Share >

 **Arjun Sreedharan** Mod → Kieron • a year ago
Now Fixed ;)


1 ^ | v • Reply • Share >

 **raghu** • 12 days ago
Sir_^_Great Stuff.....Some people make complex things look easy.....Brilliant...


^ | v • Reply • Share >

 **rock** • a month ago
hi~I follow your step. It works when I used 'qemu-system-i386 -kernel kernel', but it CANT work when I reboot the system. It shows 'invalid magic number' when loading kernel-701.


^ | v • Reply • Share >

 **rock** → rock • a month ago
It work when I change:
linux => multiboot, in grub.cfg

^ | v • Reply • Share >

 **Arjun Sreedharan** Mod → rock • a month ago
yes you need to use the `multiboot` command, since this is a multiboot spec compliant kernel. `linux` command is to load linux kernels, which aren't technically multiboot compliant.

^ | v • Reply • Share >

 **BS** • 2 months ago
ld -m elf_i386 -T link.ld -o kernel kasm.o kc.o
error at this command...pls help

^ | v • Reply • Share >

 **BS** → BS • 2 months ago
ld:linkld:5: svntax error

^ | v • Reply • Share >

 **Arjun Sreedharan** Mod → BS • 2 months ago

Are you using the same code as github.com/arjun024/mkernel ?

^ | v • Reply • Share >

 **BS** → Arjun Sreedharan • 2 months ago

```
ld -melf_i386 -e kmain kasm.o kc.o -o kernel-2197
```

i used this code to link them

^ | v • Reply • Share >

 **Arjun Sreedharan** Mod → BS • 2 months ago

First, your entry is NOT `kmain`, it's `start`. You have to use the linker file to link using the `-T` flag. Also you don't need `-e` since ENTRY is given in the linker file.

You should run:

```
$ ld -m elf_i386 -T linker.ld -o kernel kasm.o kc.o
```

If you still get issues, could you try cloning the repo afresh and then build it.

^ | v • Reply • Share >

 **Hida Fluffminer** • 2 months ago

How do I make A Bootable virtual disk of this (*.iso) for VirtualBox? This would really help to emulate!

^ | v • Reply • Share >

 **Arjun Sreedharan** Mod → Hida Fluffminer • 2 months ago

Hida Fluffminer, have a look here:

[https://github.com/arjun024/mk...](https://github.com/arjun024/mkernel)

^ | v • Reply • Share >

 **Monin Jose** • 5 months ago

i have tried on emulator, its working, can u help me on creating a bootable disk with this files. please mail me superusersmiliey@gmail.com

^ | v • Reply • Share >

 **Arjun Sreedharan** Mod → Monin Jose • 5 months ago

you can put `grub-mkrescue` to use.

see [https://github.com/arjun024/mk...](https://github.com/arjun024/mkernel)

hope this helps.

^ | v • Reply • Share >

 **Sky Venom** • 5 months ago

Bro, Please .. I know this is hard buh am finding it very difficult to get the system loading and displaying the 'my first kernel' ... Can you be kind to make a video of it please

^ | v • Reply • Share >

 **Arjun Sreedharan** Mod → Sky Venom • 5 months ago

TBH, i am little busy to make a video; but i will certainly help you out - please tell me where you are stuck. really sorry about the video

^ | v • Reply • Share >

 **liuxinglanyue** • 6 months ago

ding

^ | v • Reply • Share >

 **w** • 7 months ago

cool, but maybe you should initialize esp?

^ | v • Reply • Share >

 **Arjun Sreedharan** Mod → w • 7 months ago



Well, I am under the assumption GRUB will set up `esp` and anyway I wanted to keep the article as simple as possible. Now i think it's a good idea to set up a stack of our own.

To do that - set aside some memory in the .bss section

```
section .bss
resb 8192 ;8kb
stack_space:
```

and then before `call kmain`:

```
mov esp, stack_space
```

^ | v • Reply • Share ›

Load more comments

AROUND THE WEB

WHAT'S THIS?

The People's Chemist

Nurse Confesses: "I Would Have NEVER Vaccinated My Own Children!"

NPR

With So Much Oil Flowing, U.S. May Be Reaching Storage Limits

Lifestyle Journal

Unique Method Regrows Lost Hair (Do This Daily)

TheFix.com

Alcohol and Energy Drinks A Dangerous Combo, Study Says


ALSO ON ARJUN SREEDHARAN


[Simple and free file storage for your website using dropbox ...](#) 1 comment

[Hide data inside pointers](#) 13 comments

[Character literal is not a character in C !!](#) 5 comments


[Some jQuery bugs](#) 3 comments


 (<http://instantcoffeee.tumblr.com/>)instantcoffeee
(<http://instantcoffeee.tumblr.com/>) likes this

 (<http://the-friend-with-little-benefits.tumblr.com/>)the-friend-with-little-benefits (<http://the-friend-with-little-benefits.tumblr.com/>) likes this

 (<http://katrona.tumblr.com/>)katrona (<http://katrona.tumblr.com/>)
likes this


 (<http://saken-mx.tumblr.com/>)saken-mx (<http://saken-mx.tumblr.com/>) likes this


 (<http://abducode.tumblr.com/>)abducode
(<http://abducode.tumblr.com/>) likes this


 (<http://blackmesasecurity.tumblr.com/>)blackmesasecurity
(<http://blackmesasecurity.tumblr.com/>) likes this


 (<http://optiklab.tumblr.com/>)optiklab (<http://optiklab.tumblr.com/>)
likes this


 (<http://rconnor.tumblr.com/>)rconnor (<http://rconnor.tumblr.com/>)
likes this


 (<http://asaphprado.tumblr.com/>)asaphprado
(<http://asaphprado.tumblr.com/>) likes this


 (<http://bahsana.tumblr.com/>)bahsana (<http://bahsana.tumblr.com/>)
likes this


 (<http://yoanribeiro.tumblr.com/>)yoanribeiro
(<http://yoanribeiro.tumblr.com/>) likes this


 (<http://booksandfandoms.tumblr.com/>)booksandfandoms
(<http://booksandfandoms.tumblr.com/>) likes this


 (<http://green-hanuta.tumblr.com/>)green-hanuta (<http://green-hanuta.tumblr.com/>) likes this


 (<http://gamer-fox.tumblr.com/>)gamer-fox (<http://gamer-fox.tumblr.com/>) reblogged this from computerpile
(<http://computerpile.tumblr.com/>)


 (<http://memiux.com/>)memiux (<http://memiux.com/>) reblogged this
from arjunsreedharan (<http://arjunsreedharan.org/>)


 (<http://splenty.tumblr.com/>)splenty (<http://splenty.tumblr.com/>) likes
this


 (<http://rainyclouds1359.tumblr.com/>)rainyclouds1359
(<http://rainyclouds1359.tumblr.com/>) likes this

 (<http://computerpile.tumblr.com/>)computerpile
(<http://computerpile.tumblr.com/>) likes this

 (<http://silverlinethings.tumblr.com/>)silverlinethings
(<http://silverlinethings.tumblr.com/>) likes this

 (<http://d5z6.tumblr.com/>)d5z6 (<http://d5z6.tumblr.com/>) likes this


 (<http://sgtoctopus.tumblr.com/>)sgtoctopus
(<http://sgtoctopus.tumblr.com/>) likes this


 (<http://goshpenny.tumblr.com/>)goshpenny
(<http://goshpenny.tumblr.com/>) likes this


 (<http://sankara.me/>)sankara-me (<http://sankara.me/>) likes this


#!
(<http://inputmismatchexception.tumblr.com/>)inputmismatchexception
(<http://inputmismatchexception.tumblr.com/>) likes this


 (<http://compiler-errors.tumblr.com/>)compiler-errors (<http://compiler-errors.tumblr.com/>) reblogged this from see-plus-plus (<http://see-plus-plus.tumblr.com/>)

 (<http://rossn.tumblr.com/>)rossn (<http://rossn.tumblr.com/>) likes this


 (<http://sinyeol.tumblr.com/>)sinyeol (<http://sinyeol.tumblr.com/>) likes this


 (<http://ilikebadmusic.tumblr.com/>)ilikebadmusic (<http://ilikebadmusic.tumblr.com/>) likes this

 (<http://hauleth.niemier.pl/>)hauleth (<http://hauleth.niemier.pl/>) likes this


 (<http://m0x3.tumblr.com/>)m0x3 (<http://m0x3.tumblr.com/>) likes this


 (<http://benknis.tumblr.com/>)benknis (<http://benknis.tumblr.com/>) likes this


 (<http://armish.tumblr.com/>)armish (<http://armish.tumblr.com/>) likes this


 (<http://letgoat.tumblr.com/>)letgoat (<http://letgoat.tumblr.com/>) likes this


 (<http://see-plus-plus.tumblr.com/>)see-plus-plus (<http://see-plus-plus.tumblr.com/>) likes this


 (<http://see-plus-plus.tumblr.com/>)see-plus-plus (<http://see-plus-plus.tumblr.com/>) reblogged this from arjunsreedharan (<http://arjunsreedharan.org/>)


 (<http://thefox21.tumblr.com/>)thefox21 (<http://thefox21.tumblr.com/>) likes this


 (<http://gaussianmixture.tumblr.com/>)gaussianmixture (<http://gaussianmixture.tumblr.com/>) likes this

 (<http://fitoria.tumblr.com/>)fitoria (<http://fitoria.tumblr.com/>) likes this

 (<http://lche.tumblr.com/>)lche (<http://lche.tumblr.com/>) likes this

 (<http://akhlog.tumblr.com/>)akhlog (<http://akhlog.tumblr.com/>) reblogged this from arjunsreedharan (<http://arjunsreedharan.org/>)


 (<http://adrizzlingrain.tumblr.com/>)adrizzlingrain (<http://adrizzlingrain.tumblr.com/>) likes this


 (<http://theshadowzero.tumblr.com/>)theshadowzero (<http://theshadowzero.tumblr.com/>) reblogged this from shecodesandstyle (<http://shecodesandstyle.tumblr.com/>)


 (<http://ladylike-maniac.tumblr.com/>)ladylike-maniac (<http://ladylike-maniac.tumblr.com/>) likes this


 (<http://saybrr.com/>)yawali (<http://saybrr.com/>) likes this

 (<http://robotsthatkill.tumblr.com/>)robotsthatkill (<http://robotsthatkill.tumblr.com/>) reblogged this from arjunsreedharan (<http://arjunsreedharan.org/>)

 (<http://deathwarmedover.tumblr.com/>)deathwarmedover (<http://deathwarmedover.tumblr.com/>) likes this

 (<http://alliyve.tumblr.com/>)alliyve (<http://alliyve.tumblr.com/>) likes this

 (<http://shecodesandstyle.tumblr.com/>)shecodesandstyle
(<http://shecodesandstyle.tumblr.com/>) reblogged this from
arjunsreedharan (<http://arjunsreedharan.org/>)

 (<http://shecodesandstyle.tumblr.com/>)shecodesandstyle
(<http://shecodesandstyle.tumblr.com/>) likes this

 (<http://giako87.tumblr.com/>)giako87 (<http://giako87.tumblr.com/>)
likes this

[Show more notes](#)