

Joe's Notes for Operating Systems Project A

*****Note: These instructions are derived from Dr. Black's Project A document. They are not a substitute for the official project document but a distillation of his instructions. You should still read through Dr. Black's original Project A document in order to get a complete understanding of what is going on in the project.*****

1) Create a new folder on your Ubuntu Desktop in your Virtual Machine. Name it "ProjectA." (Do not use spaces in your folder name or file names.)

- You can do this in two ways.

a) Open the terminal and type the following commands:

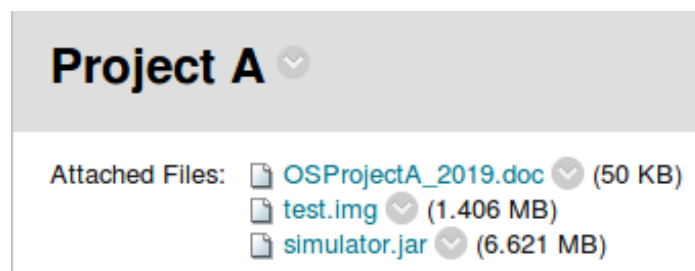
```
~$ cd Desktop  
~/Desktop$ mkdir ProjectA
```

(The '\$' represents your command prompt. Anything following a '\$' is a command entered at the command prompt.)

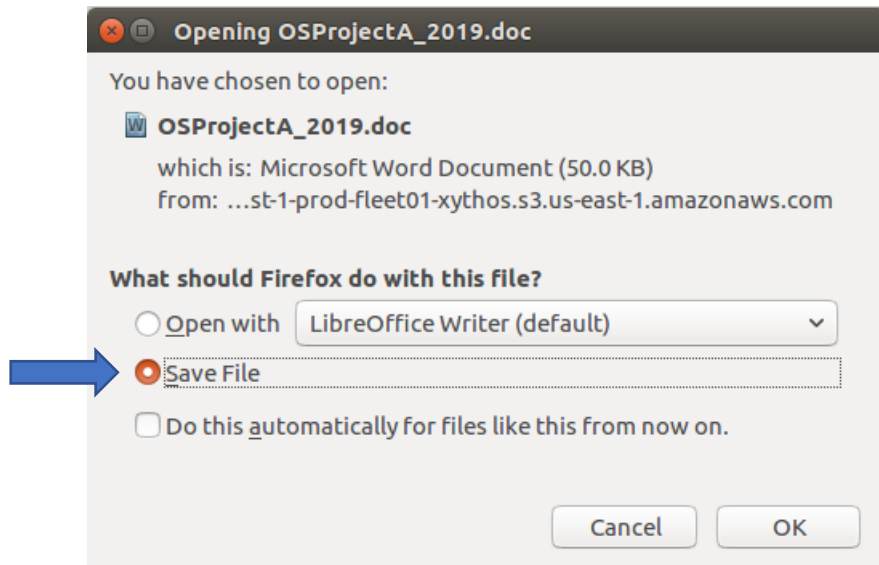
OR...

b) You can right-click on your Ubuntu Desktop and create a folder as you would in Windows or MacOS.

2) In Ubuntu, open a web browser (either Firefox or the Ubuntu Browser). Go to Blackboard and open the Project A assignment. Download all three files found there and place them into your newly created 'ProjectA' folder.



When you click on each file, make sure you 'Save' the file and NOT 'Open' it. (see next page)



These files will be placed in the 'Downloads' directory automatically. Move all three files (*OSProjectA_2019.doc*, *test.img*, *simulator.jar*) from 'Downloads' to 'ProjectA'. (Be sure that the ProjectA folder is located on your Desktop.)

3) Test the simulator.

To run 'simulator.jar' we need to use the Java Runtime Environment (JRE). In the terminal, navigate to 'ProjectA':

```
:~$ cd Desktop/ProjectA
```

...and type the following command:

```
:~/Desktop/ProjectA$ java -jar simulator.jar
```

This should open the simulator. You will probably get a message saying that you don't have Java and you should see a list of install suggestions. One of the suggestions is 'default-jre.' We will install that one. Enter the following command:

```
:~/Desktop/ProjectA$ sudo apt install default-jre
```

You will need to enter your user password here. Note that nothing will show up as you're typing your password. After the installation, run the 'java -jar simulator.jar' command again. The simulator should start.

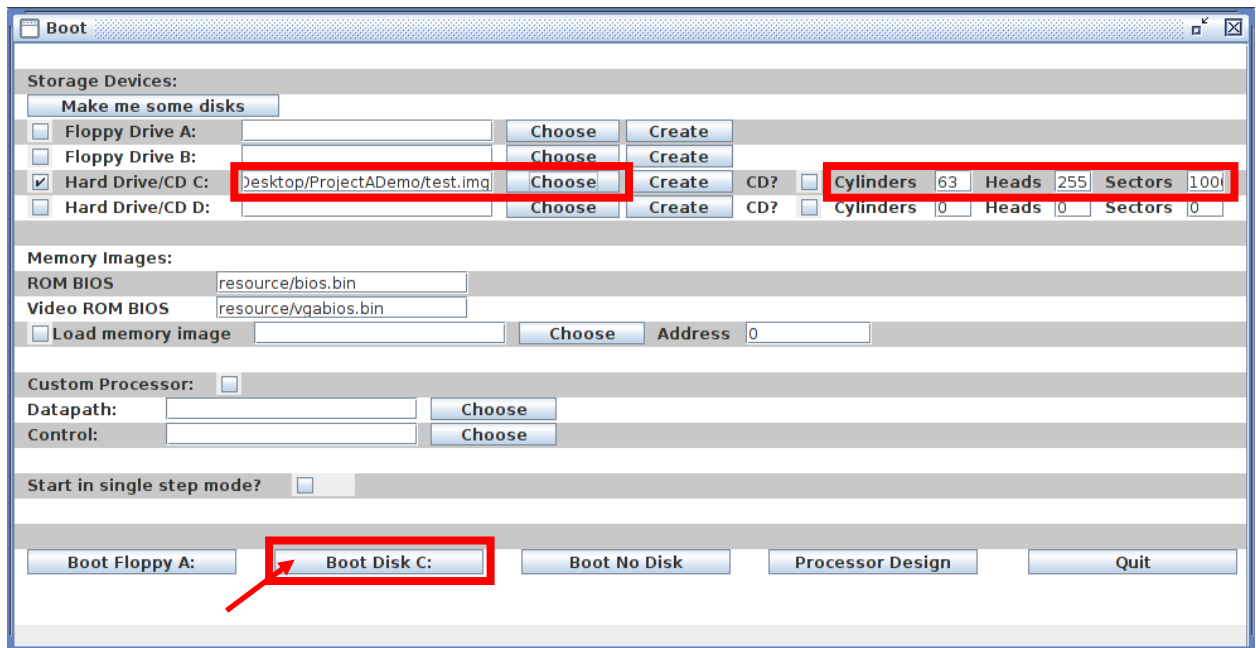
When the simulator starts, you should see a 'Boot' window. Click the 'Choose' button next to 'Hard Drive/CD C:', and select the 'test.img' file in your ProjectA folder. To the left of the 'Choose' button, you will see 'Cylinders', 'Heads', and 'Sectors'. Set them as follows:

Cylinders: 63

Heads: 255

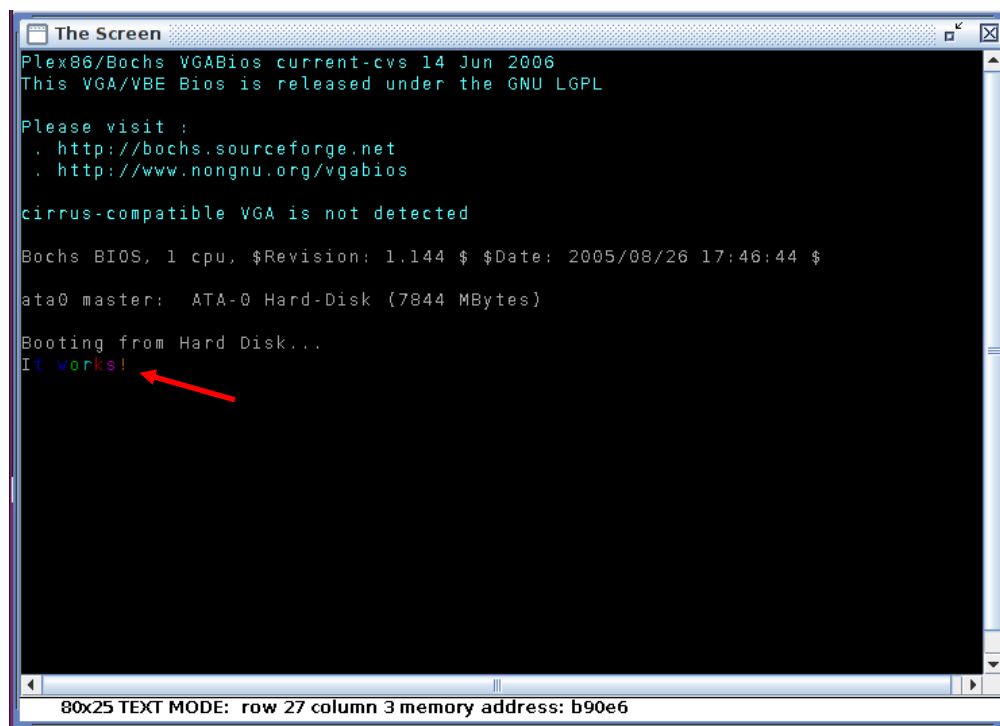
Sectors: 1000

After you properly set up the Boot screen (shown below), click 'Boot Disk C:'.



**Note: In my example above, the path to 'test.img' is 'Desktop/ProjectADemo/test.img'. Yours will be 'Desktop/ProjectA/test.img' if you called your folder 'ProjectA'.*

After clicking the 'Boot Disk C:' button, the screen should load, and you should see a colorful message: 'It works!' (shown below).



Your simulator is working properly!

4) Make 'bootload.asm' and assemble it.

- Create a blank text file named 'bootload.asm':

```
:~/Desktop/ProjectA$ nano bootload.asm
```

- Copy and paste the bootload.asm assembly program found at the end of the 'OSProjectA_2019.doc' into the newly created text file in nano.

- Assemble bootload.asm with the following command:

```
:~/Desktop/ProjectA$ nasm bootload.asm
```

This will create a machine code file named 'bootload'.

5) Make a diskc.img file containing all zeros using the command shown below:

```
:~/Desktop/ProjectA$ dd if=/dev/zero of=diskc.img bs=512 count=1000
```

You should now see a file named 'diskc.img' in your ProjectA folder

6) Now, copy the 'bootload' machine code to the first sector of 'diskc.img':

```
:~/Desktop/ProjectA$ dd if=bootload of=diskc.img bs=512 count=1 conv=notrunc
```

7) Make 'kernel.asm' and assemble it.

- Create a blank text file named 'kernel.asm':

```
:~/Desktop/ProjectA$ nano kernel.asm
```

- Copy and paste the kernel.asm assembly program found at the end of the 'OSProjectA_2019.doc' into the newly created text file in nano.

- Assemble kernel.asm with the following command:

```
:~/Desktop/ProjectA$ as86 kernel.asm -o kernel_asm.o
```

You should now see a file called 'kernel_asm.o' in your ProjectA folder.

8) Now, we will write our kernel in C. Create a new text file:

```
:~/Desktop/ProjectA$ nano kernel.c
```

The C code below is my solution. Make sure you understand what is going on in this code.

```
GNU nano 2.5.3 File: kernel.c

// Joe's Operating System Kernel
// kernel.c

void main() {

    int startVidMem = 0xb800;
    int vidMemOffset = 0x0;
    int white = 0x7;
    char* letters = "Hello World\0";

    while(*letters != 0x0) {
        putInMemory(startVidMem, vidMemOffset, *letters);
        vidMemOffset++;
        putInMemory(startVidMem, vidMemOffset, white);
        vidMemOffset++;
        // advance letters pointer
        letters++;
    }

    while(1);
}
```

Type this code into your newly created 'kernel.c' text file and save it. If you have written your own solution, you may of course use that instead.

9) Compile 'kernel.c' using the 'bcc' compiler. (*Note: you can't use 'gcc' to compile your C code because gcc produces 32-bit machine code; we want 16-bit code, so we need to use bcc.)

```
:~/Desktop/ProjectA$ bcc -ansi -c -o kernel_c.o kernel.c
```

10) Link 'kernel_c.o' and 'kernel_asm.o':

```
:~/Desktop/ProjectA$ ld86 -o kernel -d kernel_c.o kernel_asm.o
```

You need to do this step because you make a call to 'putInMemory' in your 'kernel.c'. Your C code won't know what to do with 'putInMemory' unless it is linked with the 'kernel.asm' where 'putInMemory' is defined and where the actual body of the function lives.

The output file, 'kernel', is the fully executable machine code for your kernel.

11) Copy the executable machine code, 'kernel', onto 'diskc.img' at sector 3.

```
:~/Desktop/ProjectA$ dd if=kernel of=diskc.img bs=512 conv=notrunc seek=3
```

12) Boot 'diskc.img' in the simulator.

Start the simulator:

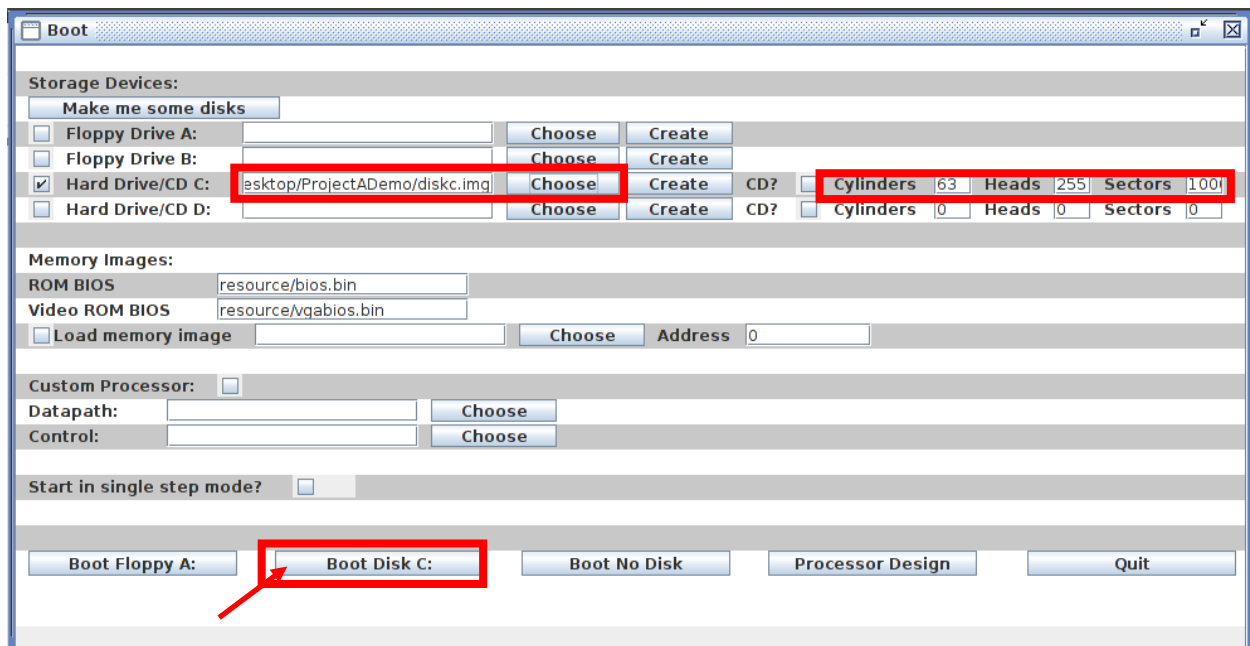
```
~/Desktop/ProjectA$ java -jar simulator.jar
```

Click the 'Choose' button next to 'Hard Drive/CD C:' and select the 'diskc.img' file in your ProjectA folder. To the left of the 'Choose' button, you will see 'Cylinders', Heads', and 'Sectors'. Set them the same as before (they may already be set):

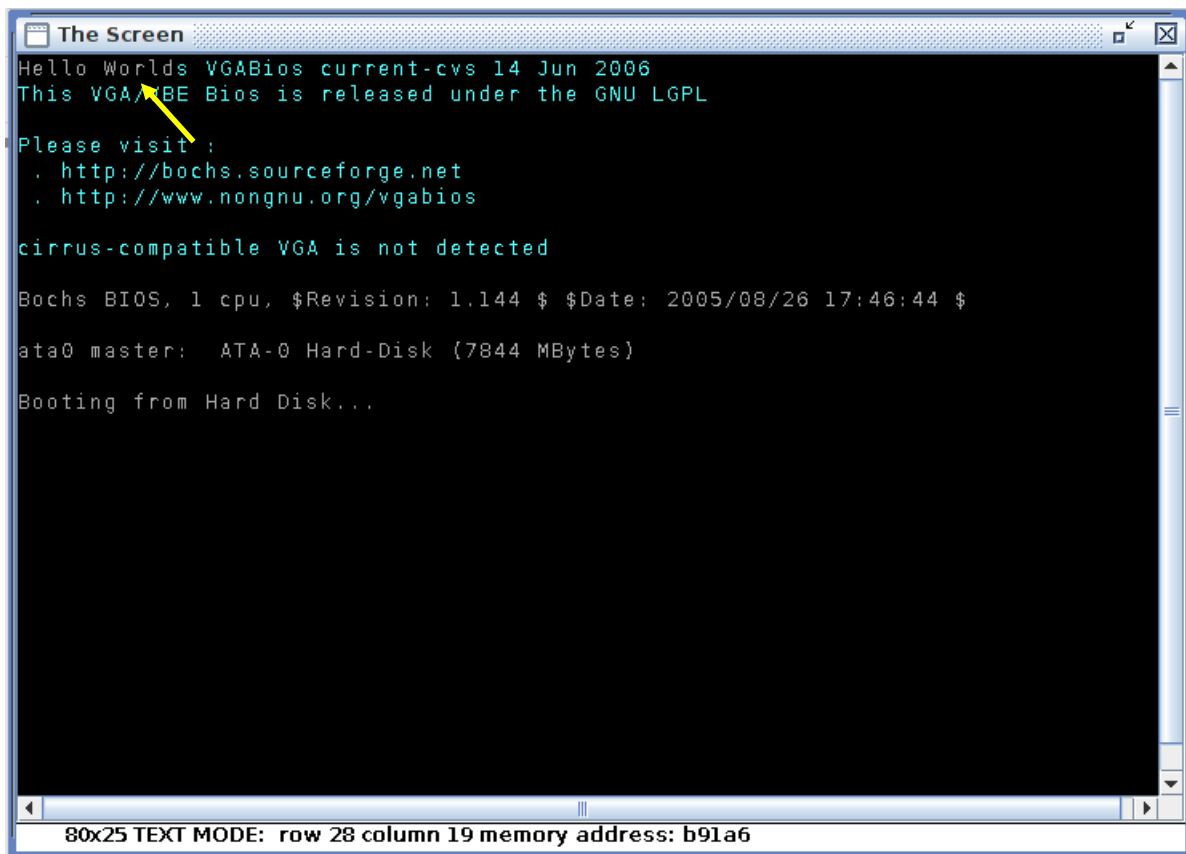
Cylinders: 63

Heads: 255

Sectors: 1000



If everything works as it should, you will see "Hello World" displayed in the upper left corner of your screen (shown below – yellow arrow).



```
The Screen
Hello Worlds VGABios current-cvs 14 Jun 2006
This VGA/MBE Bios is released under the GNU LGPL
Please visit :
. http://bochs.sourceforge.net
. http://www.nongnu.org/vgabios

cirrus-compatible VGA is not detected

Bochs BIOS, 1 cpu, $Revision: 1.144 $ $Date: 2005/08/26 17:46:44 $
ata0 master: ATA-0 Hard-Disk (7844 MBytes)
Booting from Hard Disk...

80x25 TEXT MODE: row 28 column 19 memory address: b91a6
```

You have successfully compiled and run your first operating system kernel!

- 13) Make a shell script file in order to run all your compile commands with a single command line input.

This step is really for your convenience. ***If you make any changes to your 'kernel.c' or your 'kernel.asm', you will need to recompile and relink them.*** Typing in all those compile and link commands is time consuming. Instead we can make a script to run them all on demand.

Make a new text file:

```
~/Desktop/ProjectA$ nano compileOS.sh
```

Simply put all the compile, linking, and copying commands in this file, each on separate lines (shown below).



```
joe@joe-VirtualBox: ~/Desktop/ProjectA
GNU nano 2.5.3 File: compileOS.sh
bcc -ansi -c -o kernel_c.o kernel.c
as86 kernel.asm -o kernel_asm.o
ld86 -o kernel -d kernel_c.o kernel_asm.o
dd if=kernel of=diskc.img bs=512 conv=notrunc seek=3

[ Read 5 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text ^T To Linter ^_ Go To Line
```

Remember what these commands do:

`bcc -ansi -c -o kernel_c.o kernel.c` // compile an object file from your kernel C code

`as86 kernel.asm -o kernel_asm.o` // compile an object file from your kernel assembly code. Remember, this is where your 'putInMemory' function definition is.

`ld86 -o kernel -d kernel_c.o kernel_asm.o` // links your kernel assembly object file with your kernel C object file and creates a kernel machine code executable

`dd if=kernel of=diskc.img bs=512 conv=notrunc seek=3` // puts your kernel executable machine code in sector 3 of your 'diskc.img' file.

After all these commands, your 'diskc.img' has your bootloader at sector 0 (we don't have to do this again, the 'bootload' machine code is already there) and your 'kernel' machine code at sector 3. At this point, that's all we need on our disk. You can now run all the commands in 'compileOS.sh'. But first you must tell the OS that you want 'compileOS.sh' to be executable by entering the following command:

```
~/Desktop/ProjectA$ chmod +x compileOS.sh
```

Now, to run the script:

```
~/Desktop/ProjectA$ ./compileOS.sh
```


14) In order to submit all your work, you need to gather all your files into an archive. This is like a '.zip' file but it's a different kind of archive format; it's a '.tar' file. To do this make sure you are in your 'ProjectA' folder in the command line:

```
~/Desktop/ProjectA$
```

Type the following command using your last name (YourLastName_ProjectA.tar)

```
~/Desktop/ProjectA$ tar -cvf Matta_ProjectA.tar *
```

Replace your last name where 'Matta' (my last name) is.

This command takes all the files in your 'ProjectA' folder and places it in your archive 'YourLastName_ProjectA.tar' file. Pretty good that you can fit all those files into one file!

If you want to confirm that you got all your files you want to submit in your archive file, run the following command (replacing your last name for 'Matta', of course):

```
~/Desktop/ProjectA$ tar -tvf Matta_ProjectA.tar
```

You should see a list of all your files for this project that are contained in the '.tar' file.

Submit your .tar file on Blackboard.

You've completed Project A and have written your first operating system kernel!