/*** Operating systems assignment 1 ***/

/*** Written by Akilesh B, CS13B1042 ***/

Part 1:

The first part of the assignment involves installing Minix OS and customizing the kernel.

1) Installed VM Ware Work station in Ubuntu 14.04 and created a new VM which runs Minix 3.2.1 through the ISO file which were downloaded. 1GB of RAM and 20 GB of disk space was allocated to this.

Customizing the kernel:

I had to modify the kernel so as to print the name of every command being executed on shell.

2) In order to do this, I had to modify the exec.c file present in /usr/src/servers/pm directory.

In this file (/usr/src/servers/pm/exec.c) there is a call to sys_exec in the function exec_restart, rmp->mp_name is the name of the command being executed.

So I added the line: printf("executing %s\n", rmp->mp_name);

After changes the vicinity of code is as follows:

```
if (rmp->mp_tracer != NO_TRACER && !(rmp->mp_trace_flags & TO_NOEXEC))
{

        sn = (rmp->mp_trace_flags & TO_ALTEXEC) ? SIGSTOP : SIGTRAP;

        check_sig(rmp->mp_pid, sn, FALSE /* ksig */);

  }
```

/*** The below line is the ADDED LINE, its the portion which calls kernel with pointers set by VFS ***/

```
/* Call kernel to exec with SP and PC set by VFS. */
   printf("executing %s\n",rmp->mp_name);
r = sys_exec(rmp->mp_endpoint, sp, (vir_bytes)rmp->mp_name, pc, ps_str);
if (r != OK) panic("sys_exec failed: %d", r);
```

I have included 5 lines here : 2 lines before my change, my change which is a single line and 2 lines after the change.

For cd command, go to /usr/src/commands/ash/cd.c

Inside function docd add the below line

```
printf("executing cd\n");      /*** added line ***/
```

For pwd command, go to /usr/src/commands/ash/cd.c

Inside function pwdcmd add the below line

```
printf("executing pwd\n");      /*** added line ***/
```

3) After making these changes, go to /usr/src for rebuilding the OS

make build command rebuilds and installs the OS. Make sure you are a super user (sudo su) before executing command make build.

In order to rebuild only the kernel, go to /usr/src/releasetools and execute command make clean install.

4) Reboot the system. Now, any command which is executed is printed. ie if I give ls, I get executing ls in the console (changes made in the kernel exec.c file).

Goals accomplished: Minix OS was successfully installed. I was able to customize my kernel and print from the kernel so I have rudimentary debugging capabilities. I was able to do it for both Minix 3.2.1 and Minix 3.3.0 (as a VM on windows host machine)

Part 2:

1) After installing Minix 3.2.1 and customizing the kernel, I have to install X window system in Minix 3.2.1

Minix 3.2.1 cannot connect to the internet, so the packages cannot be updated or installed. How did I overcome? I downloaded all the required packages for X window system separately, created a shared folder and mounted it to the VM

After mounting it, add this line in /usr/pkg/etc/pkgin/repositories.conf file

/*** add this line to the end and comment out the URL in the line before*///

file:///home/pkgs/

I have the packages in : /home/pkgs directory.

Usually, when I give pkg_up command for updating the packages, it tries to get the packages from the URL in line 6 of repositories.conf, but I have all the required packages in /home/pkgs/ directory. Hence the changes above.

Now execute the following:

pkg_up   /*** for updating the packages ***/

pkgin install x11 /*** for installing x window system ***/

Now, reboot the system.

2) xdm   /*** open the x window terminal ***/

It asks for username and password, after which x terminal opens up.

3) Write a simple helloworld.c program which prints hello world.

Now, execute the following commands

```
cc helloworld.c   /*** compile ***/

./a.out           /*** running the compiled program ***/
```

Helloworld is succesfully printed to the console.

Goals accomplished: X System was successfully installed and simple hello world program was run.

Difficulties faced in part 1 and part 2 of assignment:

1) Minix 3.3.0 is the latest minix. It has no source code inbuilt and no support for X window system. Their site says

: On the current mainline (soon-to-be MINIX 3.3.0) X11 has not yet been updated, and thus is not yet available.

2) Minix 3.2.1 could not connect to the internet, so packages had to be mounted through a shared folder for installing X Window system.

3) Minix 3.3.0 with changes, was not building successfully when host machine is Ubuntu. (ie Ubuntu machine with Minix 3.3.0 as a VM).

Part 3

Installing version of Linux and customizing the kernel in the same way as in Minix

1) I installed VM Ware player on windows 7 and created a new VM and installed Ubuntu 14.04 through the ISO file which was downloaded. 2GB of RAM and 20 GB of disk space was allocated to this.

2) The kernel code for current version of linux can be obtained by executing this command:

```
sudo apt-get source linux-image-$(uname -r)   /*** to get kernel code *///
```

Pre-requisites before rebuilding kernel:

1) gcc should be installed (sudo apt-get install gcc)

2) ncurses development package should be installed (sudo apt-get install libncurses5-dev)

update your system by : sudo apt-get update

3) Configuring the build environment

sudo apt-get build-dep linux-image-$(uname -r)

Customizing the kernel:

Kernel version: linux-lts-vivd-3.19.0

I had to make changes in exec.c file located in /usr/src/linux-lts-vivid-3.19.0/fs directory

The function do_execveat_common executes a new program, add this line

printk("executing %s\n", filename->name); /** add this line ***/

The vicinity after the changes:

```
retval = exec_binprm(bprm);
        if (retval < 0)
                goto out;
printk("executing %s\n", filename->name);                  /*** added line ***/
        /* execve succeeded */
        current->fs->in_exec = 0;
        current->in_execve = 0;
        acct_update_integrals(current);
```

I have included 5 lines, two lines before the change, the change and two lines after change.

4) Now execute the following commands:

```
sudo make   /*** for compiling these changes ***/
```

```
sudo make modules_install install   /*** this will install linux-kernel into your system, creates some files under /boot directory ***/
```

```
sudo update-grub  /*** for updating grub ***/
```

5) Now reboot the system and observe the changes:

all these print messages are contained in dmesg log

```
executing /bin/ls

executing /bin/mkdir

executing /bin/dmesg
```

Goals accomplished: Ubuntu 14.04 was successfully installed and I was able to customize the kernel and print from the kernel so I have rudimentary debugging capabilities.

/***   Screen shots of my changes are present in the same folder ***/

/*** END ***/