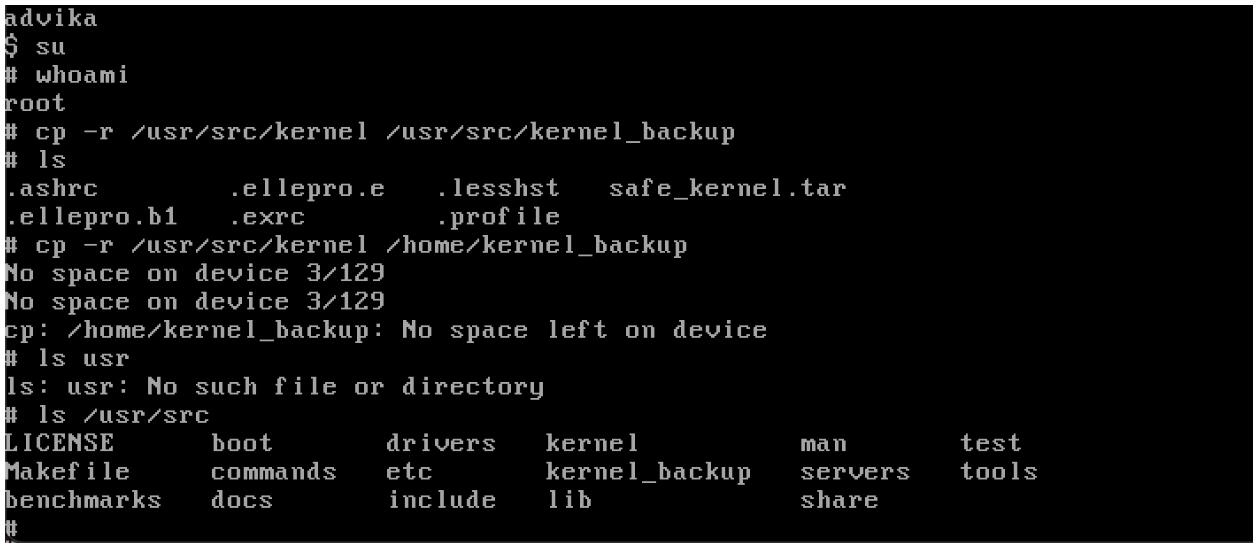
CPE 453: Operating Systems
Lab 4: Minix Scavenger Hunt II
Advika Deodhar (amdeodha) and Daniel Gutierrez (jguti151)

Executive Summary: In this lab we modified the MINIX kernel to display "@" at the cursor anytime the system was idle.

Task 1: Building a Modified Kernel
Approach:

- First we booted up the Minix OS system by doing the boot up command for Qemu and changing it a little so that the display was full screen
  `qemu-system-i386 -display gtk,zoom-to-fit=on -nic user,model=virtio-net-pci -rtc base=utc -net user -net nic -m 256 -fda testfloppy -hda minix_3_1_8.vmdk`
- That command did not work for Advika, on Mac, so she just used the command below
- `qemu-system-i386 -nic user,model=virtio-net-pci -rtc base=utc -net user -net nic -m 256 -fda testfloppy -hda minix_3_1_8.vmdk`
- We then took notes on the boot and startup process of Minix from reading the manpages described in the lab spec
- usage() - describes the installation process of Minix
  - `make hdboot` makes the image file then copies it into the directory /minix
  - The newest image is taken to be experimental, and the oldest image is the one that works for sure.
  - The oldest image gets taken out and the newest image gets put in.
  - Make sure to check what /minix is holding before running hdboot because if you want your new image to be the one that works for sure you have to remove the old image first
  - Shutting down is a special process - you can't just turn Minix off it has to flush stuff from the cache first
- monitor() - describes the Boot Monitor
  - When the monitor is loaded it executes the function main
  - You can enter monitor mode by hitting the escape key
  - Inside monitor mode, there are different commands like ls, echo, and boot which are the important ones for us
  - Without any arguments, boot will load and execute the Minix 3 image
  - exit exists the monitor mode
- boot() - describes going from power on to the log in prompt
  - The monitor loads the kernel from `/boot/image`
  - This starts the different tasks that Minix is made up of, and then control is transferred to the init process
  - Init starts the daemons /etc/rc then the /usr file system is mounted

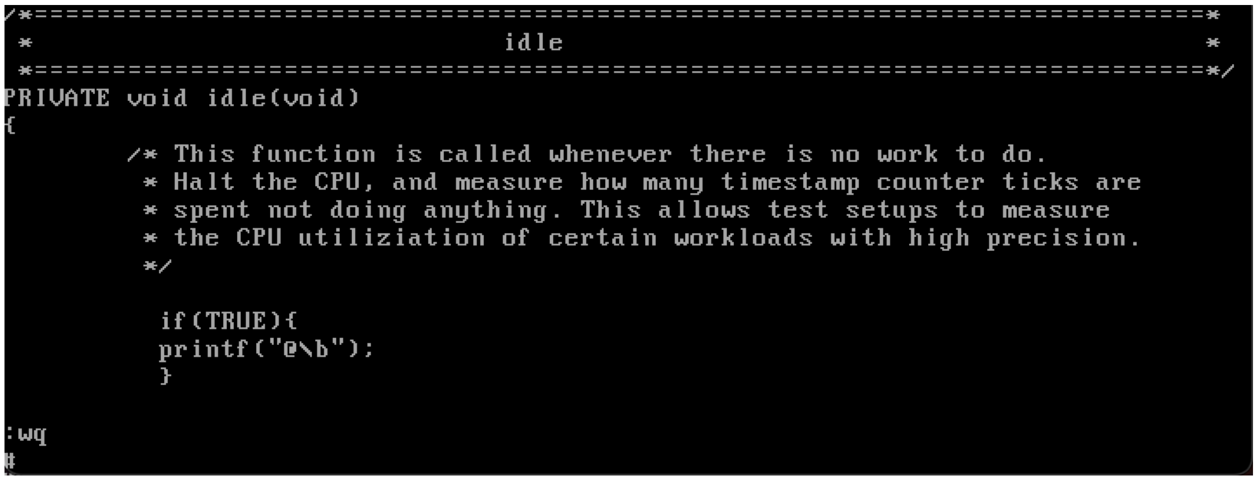
- We then made a copy of the original kernel before we edited it just to make sure we have a backup incase something goes horribly wrong
    - We used the command "`cp -r /usr/src/kernel /usr/src/kernel_backup`" and that worked. We double checked that the copy was made by ls-ing in /usr/src, which is shown below.



```
advika
$ su
# whoami
root
# cp -r /usr/src/kernel /usr/src/kernel_backup
# ls
.ashrc         .ellepro.e    .lesshst    safe_kernel.tar
.ellepro.b1    .exrc         .profile
# cp -r /usr/src/kernel /home/kernel_backup
No space on device 3/129
No space on device 3/129
cp: /home/kernel_backup: No space left on device
# ls usr
ls: usr: No such file or directory
# ls /usr/src
LICENSE      boot        drivers    kernel          man        test
Makefile     commands    etc        kernel_backup   servers    tools
benchmarks   docs        include    lib             share
#
```

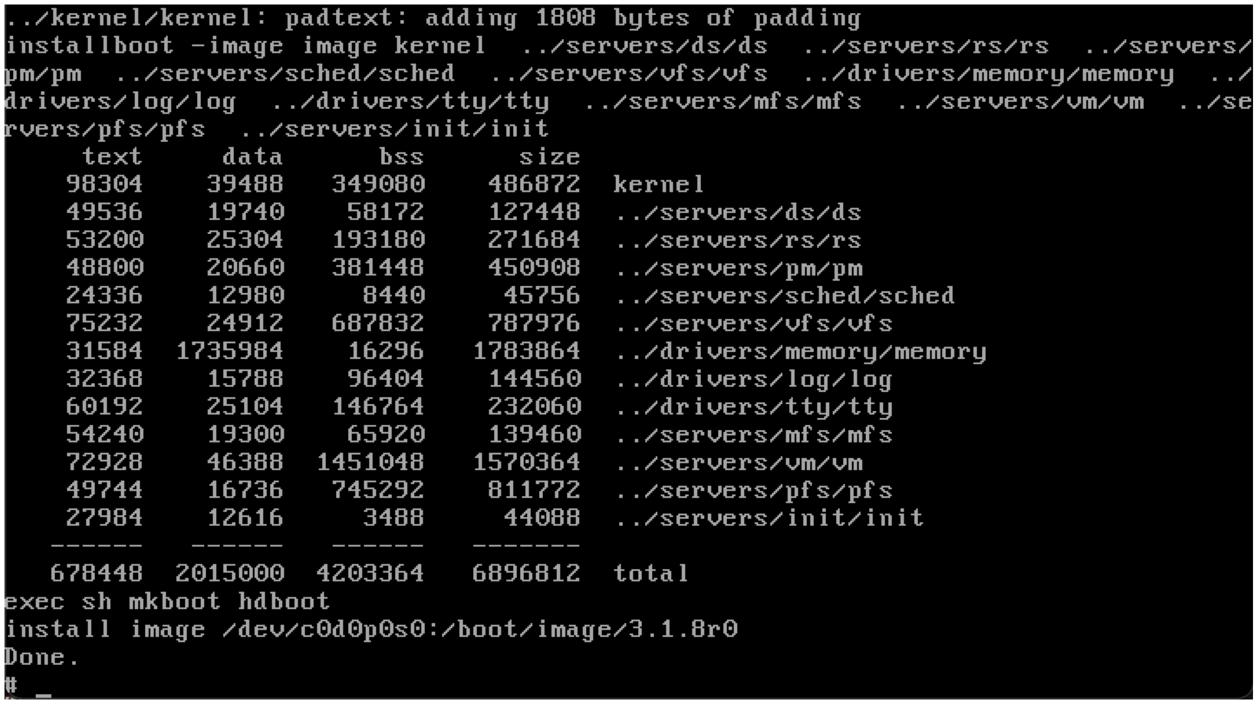
-
- Next, we then navigated to the kernel source files by going into `/usr/src/kernel`. There were many files to browse through but eventually we found that the idle function in proc.c
- The files we looked through were `main.c, system.c,` then `proc.c`
- Once inside `proc.c`, we then added the following code into the idle() as shown below. This made it so that whenever the kernel is idle and not running something, the @ symbol would be printed out.

```
if (TRUE) {
      printf("@\b");
}
```



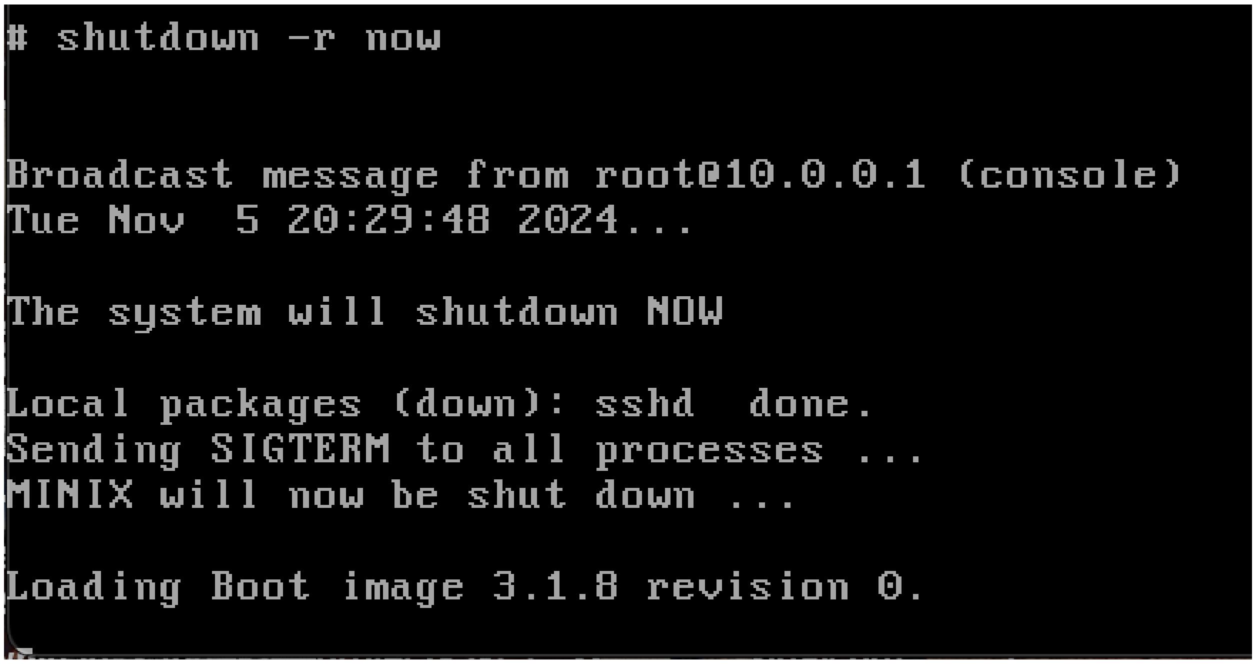
```
/*===========================================================================*
 *                              idle                                         *
 *===========================================================================*/
PRIVATE void idle(void)
{
        /* This function is called whenever there is no work to do.
         * Halt the CPU, and measure how many timestamp counter ticks are
         * spent not doing anything. This allows test setups to measure
         * the CPU utiliziation of certain workloads with high precision.
         */

          if(TRUE){
          printf("@\b");
          }

:wq
#
```

- Then to rebuild the kernel after editing it, we had to go into "`cd /usr/src/tools`" as that is where the various builds and scripts of the Minix system are stored.
- We then ran "`make hdboot`", which compiles all the kernel files and then combines them into a single kernel file. We learned from the usage manpage that his also saves the currently running kernel as the "old image" and then puts in the freshly built kernel as the "new image." The successful `make hdboot` proof is below.



```
../kernel/kernel: padtext: adding 1808 bytes of padding
installboot -image image kernel  ../servers/ds/ds  ../servers/rs/rs  ../servers/
pm/pm  ../servers/sched/sched  ../servers/vfs/vfs  ../drivers/memory/memory  ../
drivers/log/log  ../drivers/tty/tty  ../servers/mfs/mfs  ../servers/vm/vm  ../se
rvers/pfs/pfs  ../servers/init/init
     text     data      bss      size
    98304    39488   349080    486872  kernel
    49536    19740    58172    127448  ../servers/ds/ds
    53200    25304   193180    271684  ../servers/rs/rs
    48800    20660   381448    450908  ../servers/pm/pm
    24336    12980     8440     45756  ../servers/sched/sched
    75232    24912   687832    787976  ../servers/vfs/vfs
    31584  1735984    16296   1783864  ../drivers/memory/memory
    32368    15788    96404    144560  ../drivers/log/log
    60192    25104   146764    232060  ../drivers/tty/tty
    54240    19300    65920    139460  ../servers/mfs/mfs
    72928    46388  1451048   1570364  ../servers/vm/vm
    49744    16736   745292    811772  ../servers/pfs/pfs
    27984    12616     3488     44088  ../servers/init/init
   ------   ------   ------   -------
   678448  2015000  4203364   6896812  total
exec sh mkboot hdboot
install image /dev/c0d0p0s0:/boot/image/3.1.8r0
Done.
# _
```

- Once we added the code to idle, we had to wq to save the file and then do the shutdown command since after reading the manpages we learned that shutdown will boot using the new image.
- We used the command "`shutdown -r now`" which would shutdown the OS, the -r would reboot it, and the "now" would reboot it right away instead of its default time interval
- After this command it took a long time for it to shut down, so we decided to just close Qemu and just restart it again from our terminal. The screen it got stuck on got stuck on the screen below.



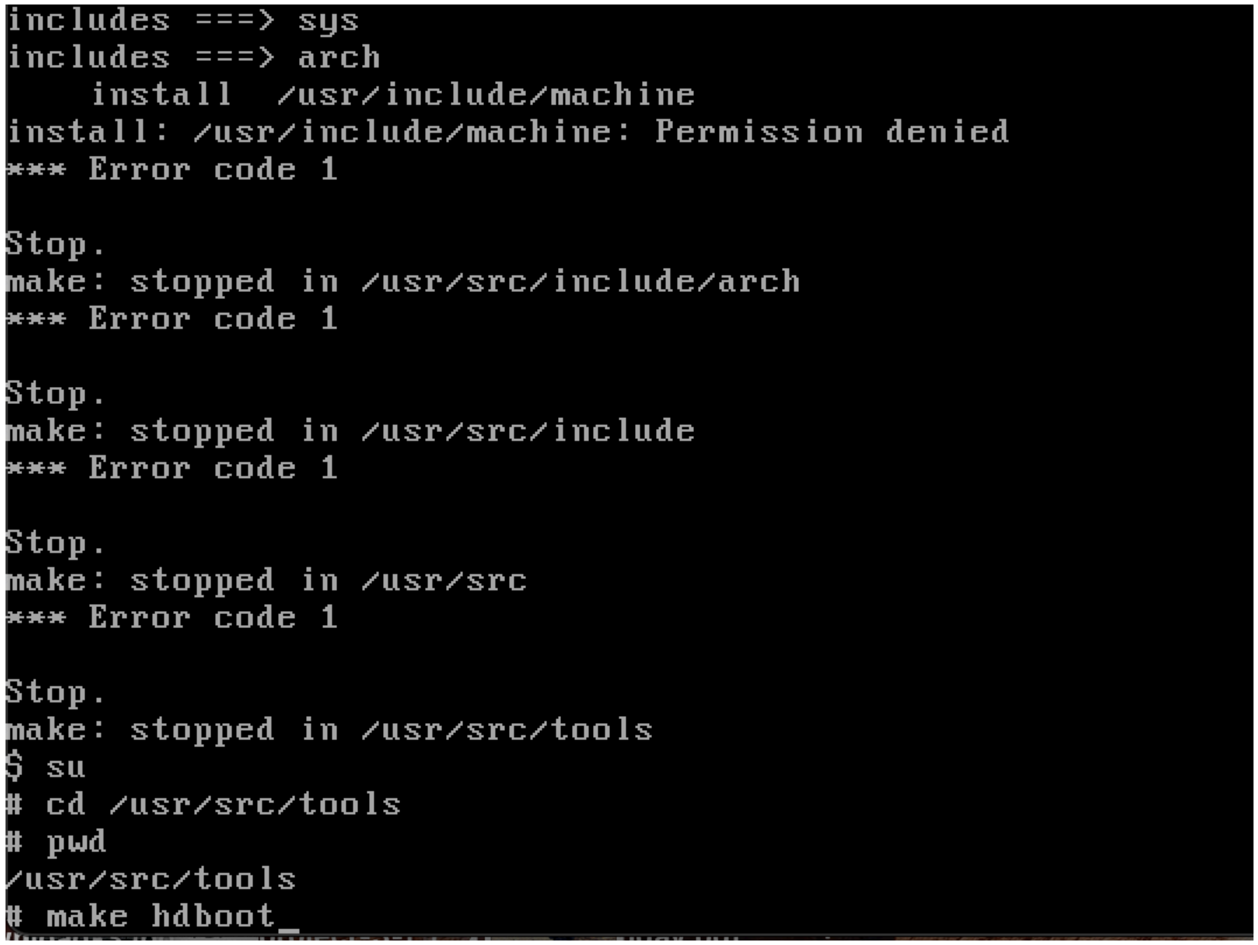
```
# shutdown -r now


Broadcast message from root@10.0.0.1 (console)
Tue Nov  5 20:29:48 2024...

The system will shutdown NOW

Local packages (down): sshd  done.
Sending SIGTERM to all processes ...
MINIX will now be shut down ...

Loading Boot image 3.1.8 revision 0.
```

- We then moved onto the testing.

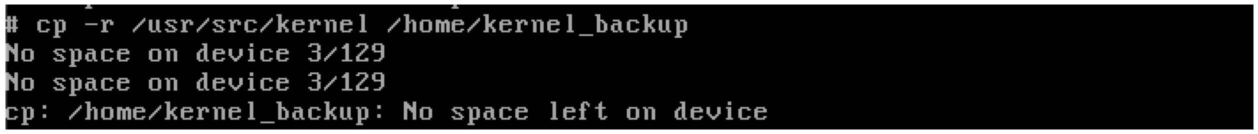
Note: We probably could've added to switch_to_user(void) where it calls idle in `proc.c`. Maybe without even the if statement, but we did not test this.

Problems Encountered:

- At first we tried to copy the kernel over so that we would have a backup of our kernel incase something went wrong
- We tried to do the cp -r command to copy the files while in usr but they kept saying permission denied so we had to switch to root in order to copy over the kernel files

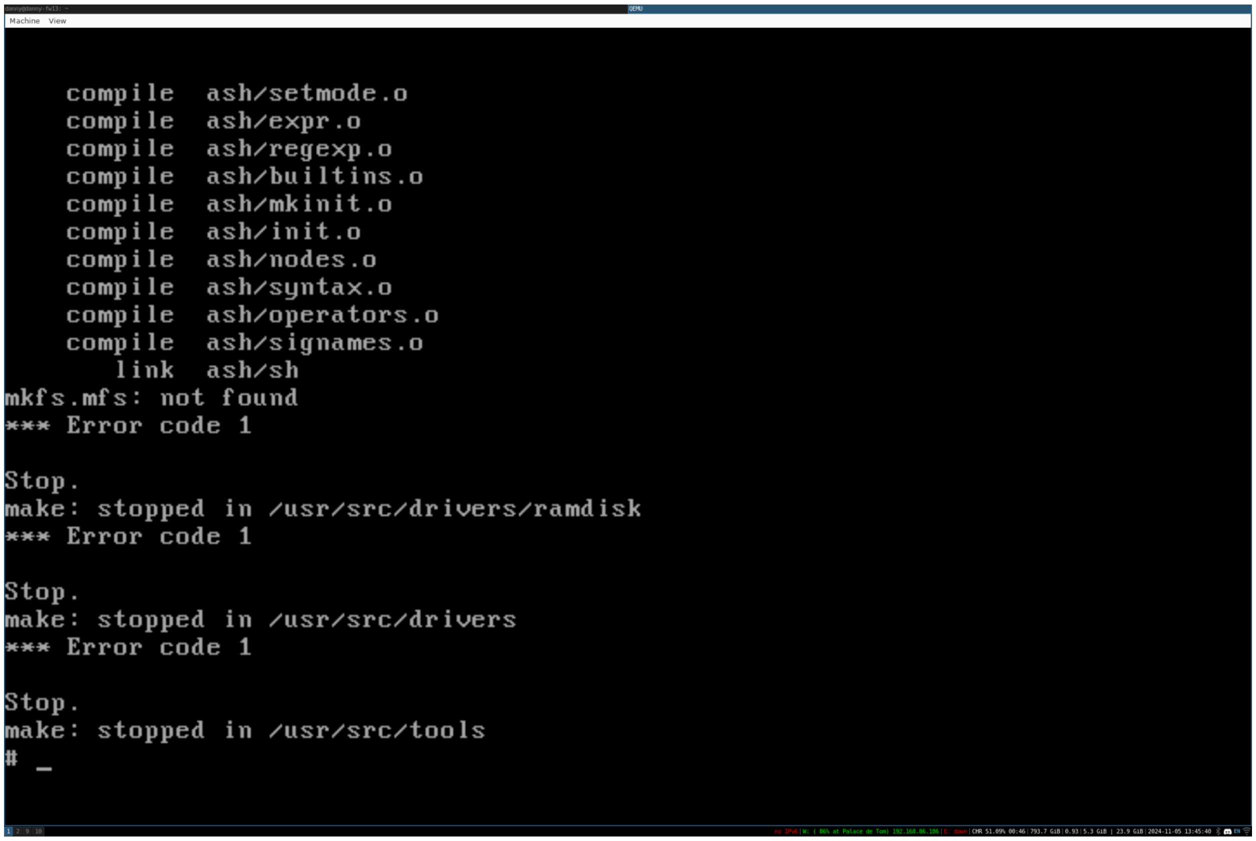


```
includes ===> sys
includes ===> arch
    install  /usr/include/machine
install: /usr/include/machine: Permission denied
*** Error code 1

Stop.
make: stopped in /usr/src/include/arch
*** Error code 1

Stop.
make: stopped in /usr/src/include
*** Error code 1

Stop.
make: stopped in /usr/src
*** Error code 1

Stop.
make: stopped in /usr/src/tools
$ su
# cd /usr/src/tools
# pwd
/usr/src/tools
# make hdboot_
```

- We tried doing the command `cp -r /usr/src/kernel /home/safe_kernel` but were getting the error that there was "`No space on the device 3/129`."
- 

```
# cp -r /usr/src/kernel /home/kernel_backup
No space on device 3/129
No space on device 3/129
cp: /home/kernel_backup: No space left on device
```

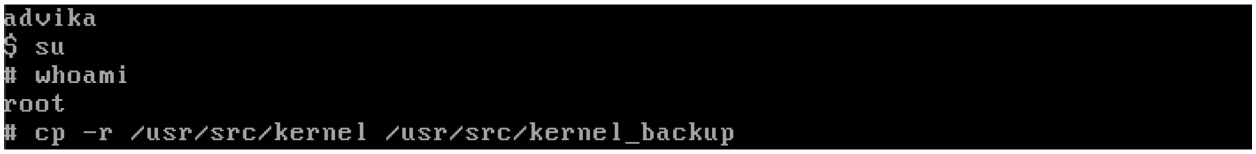
- We then thought it would be smaller if we compressed it into a tar file so we tried doing the command "`tar cvf safe_kernel.tar /usr/src/kernel`" but were also getting the same error "`No space on the device 3/129`."
- We ran into some struggles when using VI to edit the idle function. It was very hard to know when we were in insert mode, and to backspace if we messed up. Also the arrow keys would type out "l" and "h" when Advika tried to use them to move around.
- Danny ran into an error that said "`mkfs.mfs: not found`" when he was running `make hdboot`. The error is shown below.



```
    compile  ash/setmode.o
    compile  ash/expr.o
    compile  ash/regexp.o
    compile  ash/builtins.o
    compile  ash/mkinit.o
    compile  ash/init.o
    compile  ash/nodes.o
    compile  ash/syntax.o
    compile  ash/operators.o
    compile  ash/signames.o
       link  ash/sh
mkfs.mfs: not found
*** Error code 1

Stop.
make: stopped in /usr/src/drivers/ramdisk
*** Error code 1

Stop.
make: stopped in /usr/src/drivers
*** Error code 1

Stop.
make: stopped in /usr/src/tools
# _
```

Solutions:

- With the permission denied error for copying over the files, we realized that we had to be in root in order to do so. Once we switched from usr to root, copying the files was possible.



```
advika
$ su
# whoami
root
# cp -r /usr/src/kernel /usr/src/kernel_backup
```

- With the "`No space on the device 3/129`" issue we realized it was because Minix has 3 partitions, ROOT, HOME, AND USER. We realized that all of the files and code that makes up the kernel takes up a lot of space, and so the USER partition may have more room to store that since the kernel happens to be under usr/src anyways. We then tried copying it to "`/usr/src/kernel_backup`" instead of "`home/safe_kernel`" and it worked just fine.
- Danny was able to solve the `mkfs.mfs: not found` error by just logging in as root, not doing su(), and doing `make hdboot` again.

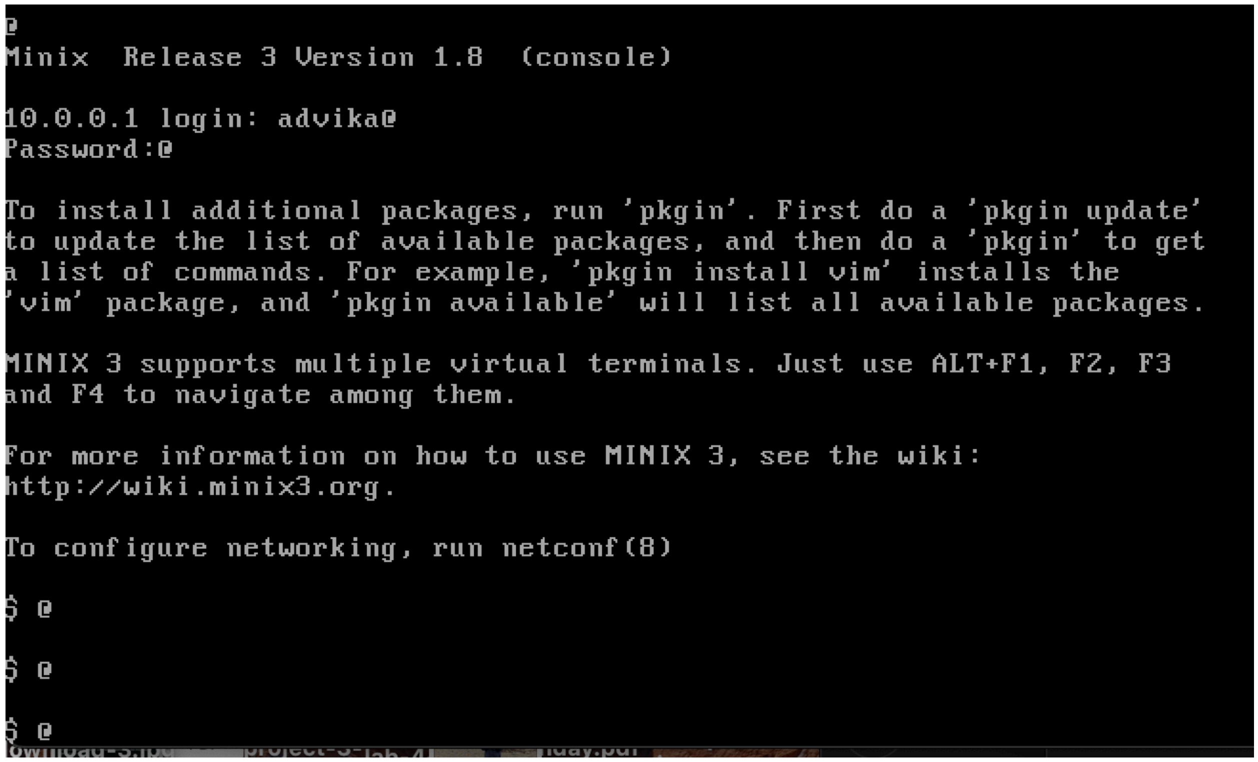
- We ran into some struggles when using VI to edit the idle function. We solved this by just closing out of the file if we messed up and not saving it, then opening it again and typing very very carefully.

Lessons Learned:

- We confirmed that as we knew, root has a lot more permissions to carry out tasks that usr does not, one of them being copying some of the kernel source files to make a backup.
- The Minix system is made up of 3 partitions on disk, so when the OS boots up, the one partition given is split into 3 different partitions, one with the root filesystem, one with home, and one with user. All of the file system programs are under`/usr/src` and that is where all of the files for the kernel are. The user partition has way more space than the home partition. This is because the home partition is just used for small personal projects whereas usr has to hold all of the system and application files used to run the programs.
- We also learned that VI is incredibly frustrating as there are no backspaces, you must overwrite the parts you mistyped, and that it does not tell you when you are in insert mode. We learned to type very carefully.
- We are not sure why the `mfks.mfs: not found` error was solved by going into root not with `su`, but instead `login root`.
- VI sucks and we completely understand why VIM was made.

Task 2: Test It

Approach:

- After rebooting with the new image, we saw that while the kernel was idle, and not running anything, it would print out the @ symbol. This is shown below.
  

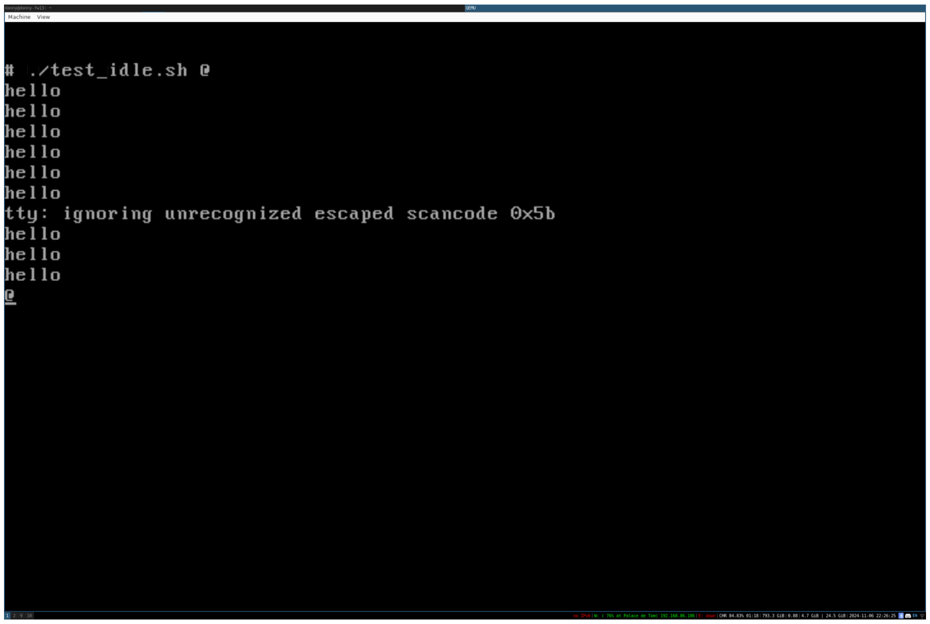
```
@
Minix  Release 3 Version 1.8  (console)

10.0.0.1 login: advika@
Password:@

To install additional packages, run 'pkgin'. First do a 'pkgin update'
to update the list of available packages, and then do a 'pkgin' to get
a list of commands. For example, 'pkgin install vim' installs the
'vim' package, and 'pkgin available' will list all available packages.

MINIX 3 supports multiple virtual terminals. Just use ALT+F1, F2, F3
and F4 to navigate among them.

For more information on how to use MINIX 3, see the wiki:
http://wiki.minix3.org.

To configure networking, run netconf(8)

$ @

$ @

$ @
```

-
- We then wanted to make sure that the @ symbol would not be visible while the kernel is doing something, so we wrote the following programs to test that.
- We wrote 2 shell scripts.

  test_busy.sh
  ```
  #!/bin/sh
  while true
  do
       echo "hello world"
  done
  ```

  test_idle.sh
  ```
  #!/bin/sh
  while true
  do
       echo "hello world"
       sleep 1
  done
  ```

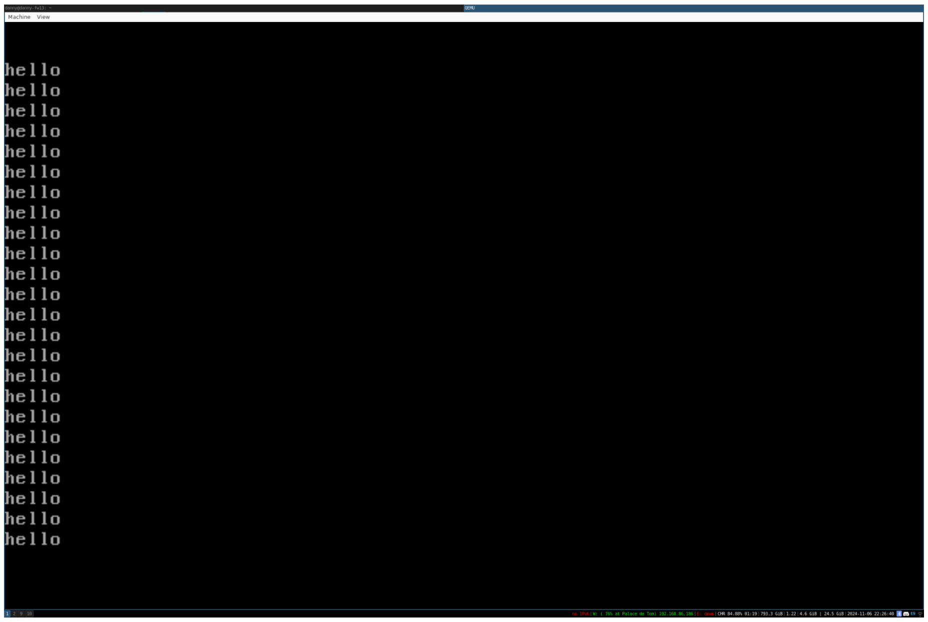
- In this second program, we wanted to see that it would print the @ symbol once after the “hello world” as it would be idle as it sleeps.
- We then chmod +x <name>.sh to be able to run the script.
- test_idle.sh working. Verified by seeing the @ at the cursor when its sleeping



```
# ./test_idle.sh @
hello
hello
hello
hello
hello
hello
tty: ignoring unrecognized escaped scancode 0x5b
hello
hello
hello
@
```

Note: Ignore the tty message, I tried to Alt-Tab to take a screenshot.

- test_busy.sh working. Verified by no seeing an @.



```
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

Problems Encountered:

- We tried finding Bash but it isn't part of Minix, it only has sh, or the Bourne Shell.

Solutions:

- There is still sh to run our programs, so the "`#!/bin/sh`" on the top of the programs tells it to run using sh.

Lessons Learned:

- We learned that editing the kernel is kind of a process and isn't super simple, and that was just a few lines of code in one file. Project 4 should be fun :).