Lab 1

CIS-657 PRINCIPLES OF OPERATING SYSTEMS

PROF. MOHAMMED ABDALLAH

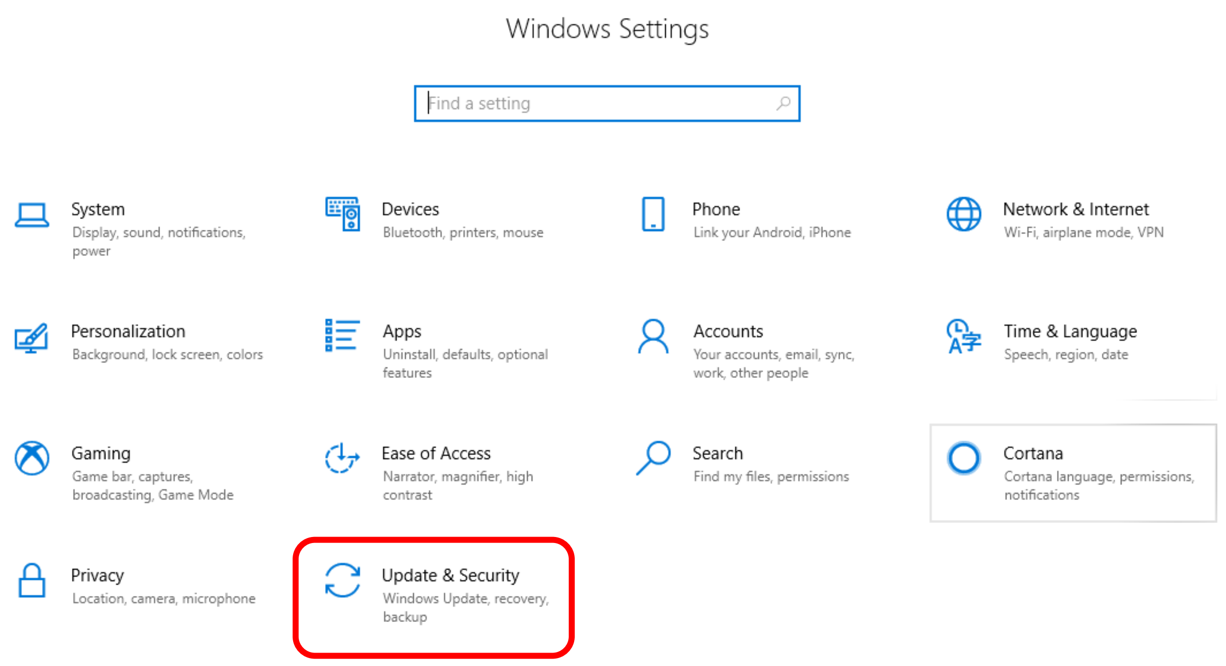
Anthony Redamonti
SYRACUSE UNIVERSITY

Section A: Installation and Setup

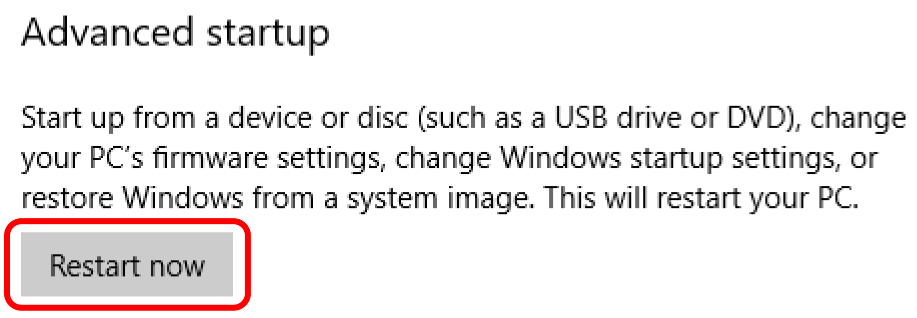
In the lab instructions, there is a list of four different options of configurations between VirtualBox and XINU versions. Option 1 was chosen, as it was the most tested and reliable option. The following setup was performed on a Windows 10 Enterprise Host OS.

Option 1: VirtualBox 5.2.28 and XINU 2019 appliances

First, the Virtualization must be enabled on the host PC. Click on the home button (bottom left corner) and select "Settings". Next, select "Update & Security" shown below.

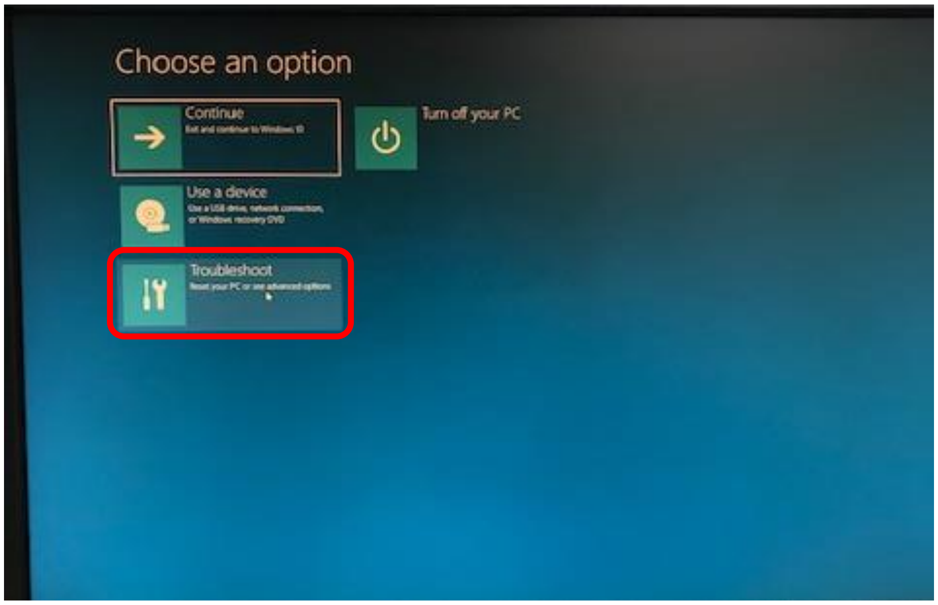

Windows Settings
Find a setting
System
Display, sound, notifications, power
Devices
Bluetooth, printers, mouse
Phone
Link your Android, iPhone
Network & Internet
Wi-Fi, airplane mode, VPN
Personalization
Background, lock screen, colors
Apps
Uninstall, defaults, optional features
Accounts
Your accounts, email, sync, work, other people
Time & Language
Speech, region, date
Gaming
Game bar, captures, broadcasting, Game Mode
Ease of Access
Narrator, magnifier, high contrast
Search
Find my files, permissions
Cortana
Cortana language, permissions, notifications
Privacy
Location, camera, microphone
Update & Security
Windows Update, recovery, backup

Click on "Recovery" then under Advanced Setup, select "Restart Now" shown below.

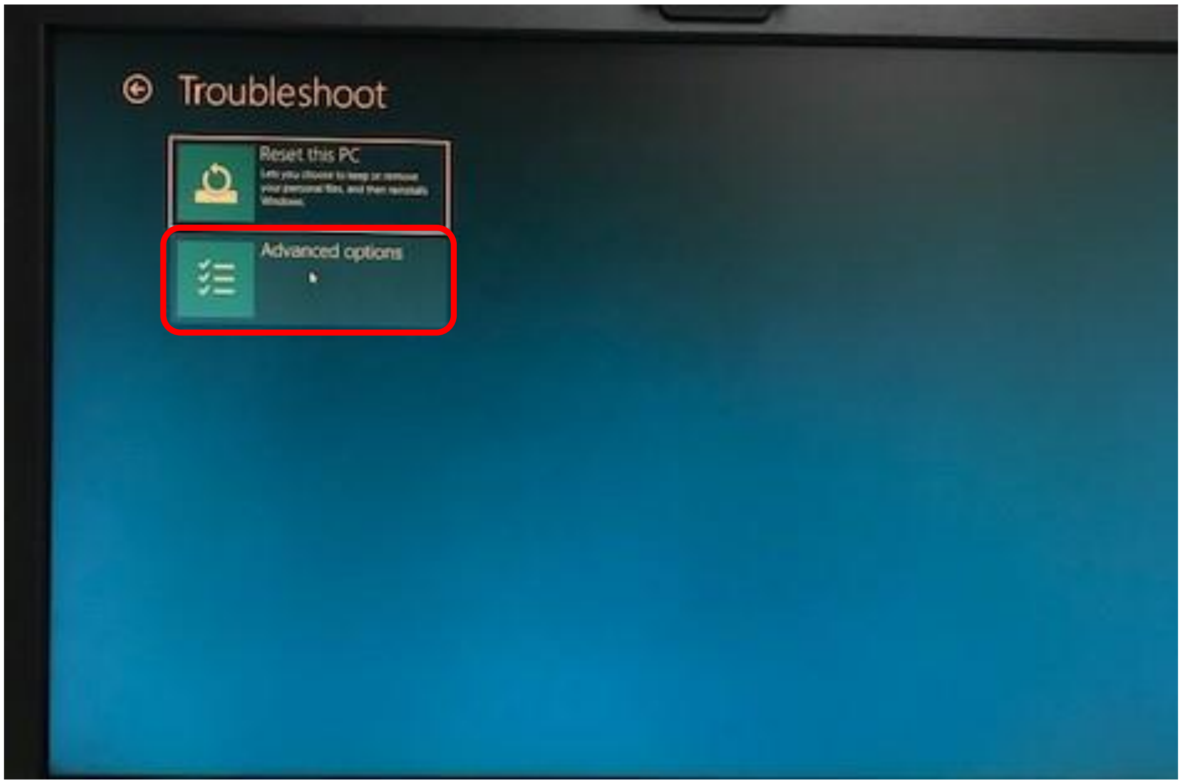

Advanced startup

Start up from a device or disc (such as a USB drive or DVD), change your PC's firmware settings, change Windows startup settings, or restore Windows from a system image. This will restart your PC.

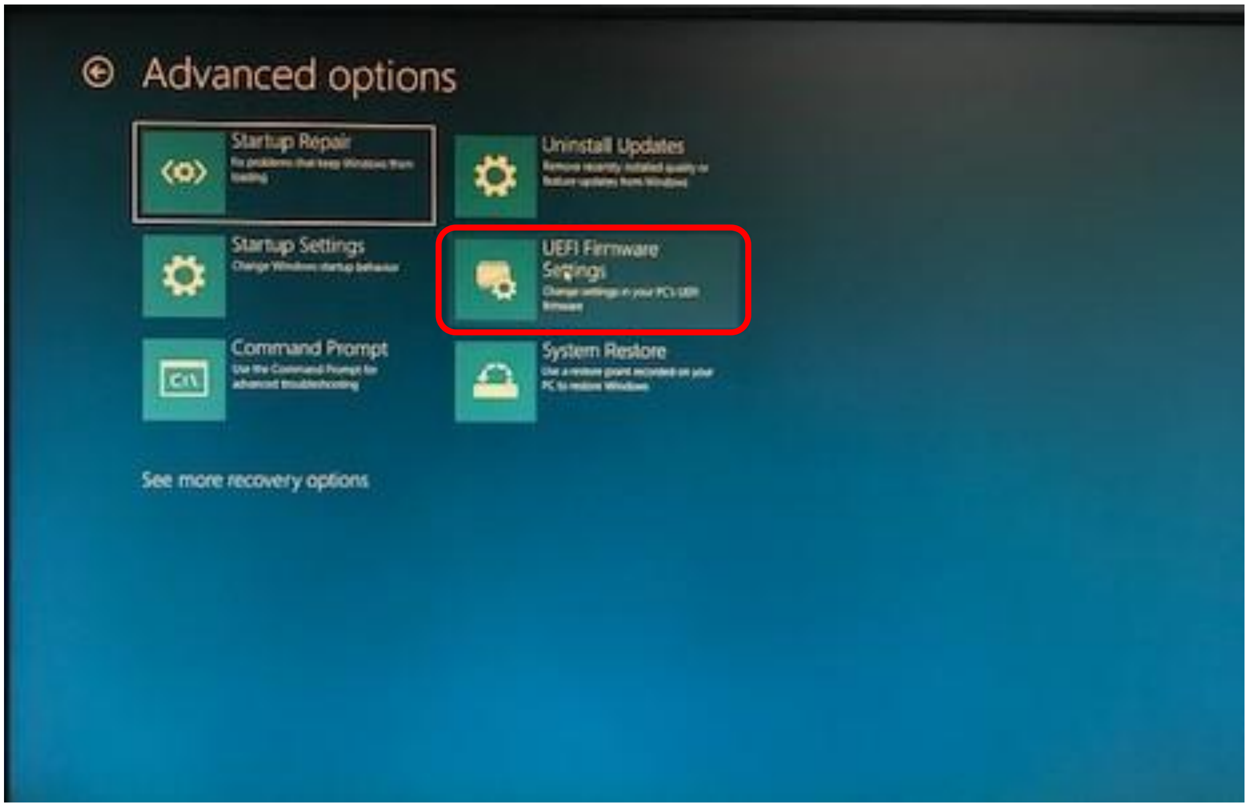
Select the “Troubleshoot” button shown below.


Choose an option
Continue
Exit and continue to Windows 10
Turn off your PC
Use a device
Use a USB drive, network connection, or Windows recovery DVD
Troubleshoot
Reset your PC or see advanced options

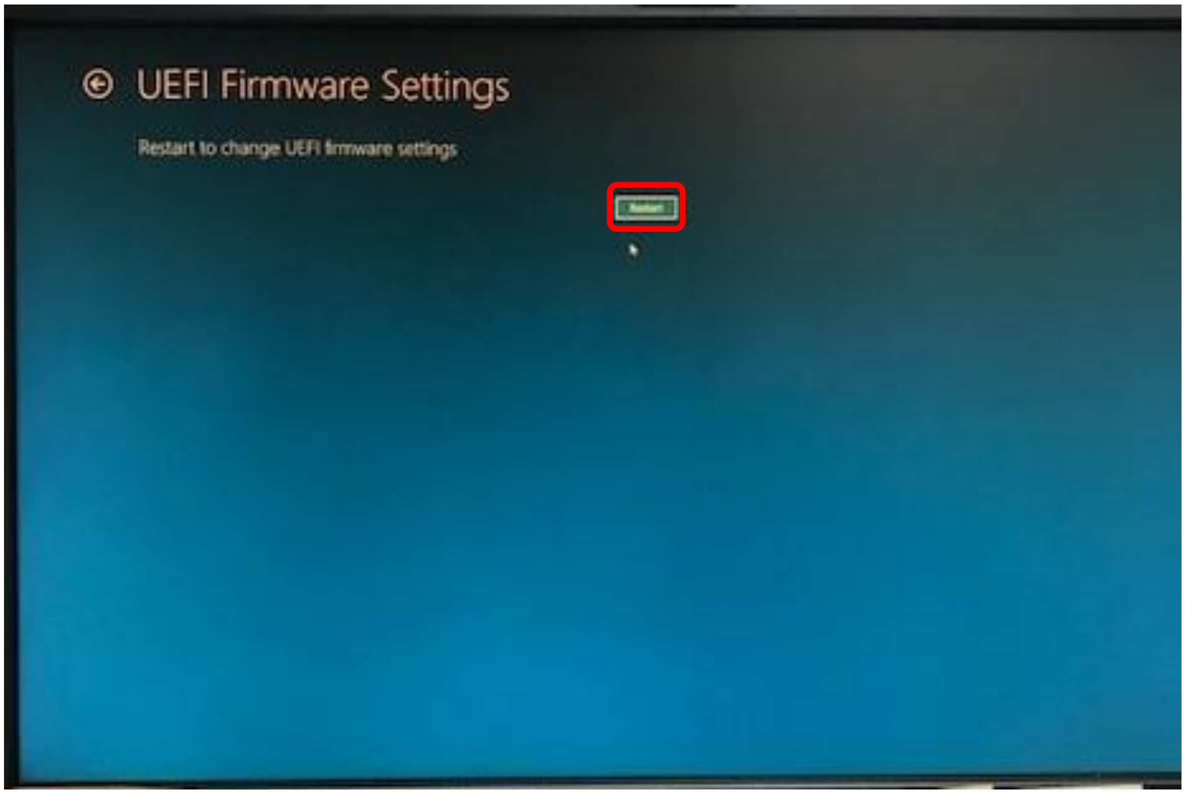
Select “Advanced options” shown below.


Troubleshoot
Reset this PC
Lets you choose to keep or remove your personal files, and then reinstalls Windows.
Advanced options

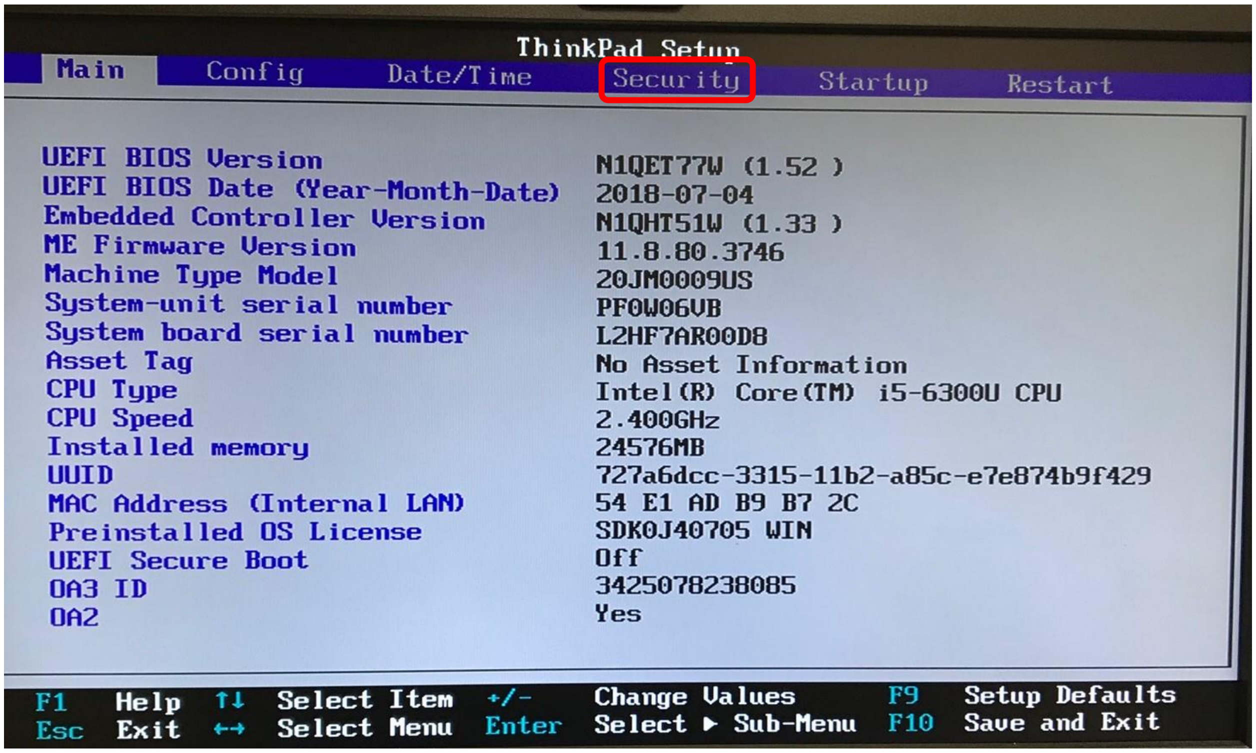
Next, select “UEFI Firmware Settings” shown below.



Click “restart” shown below.



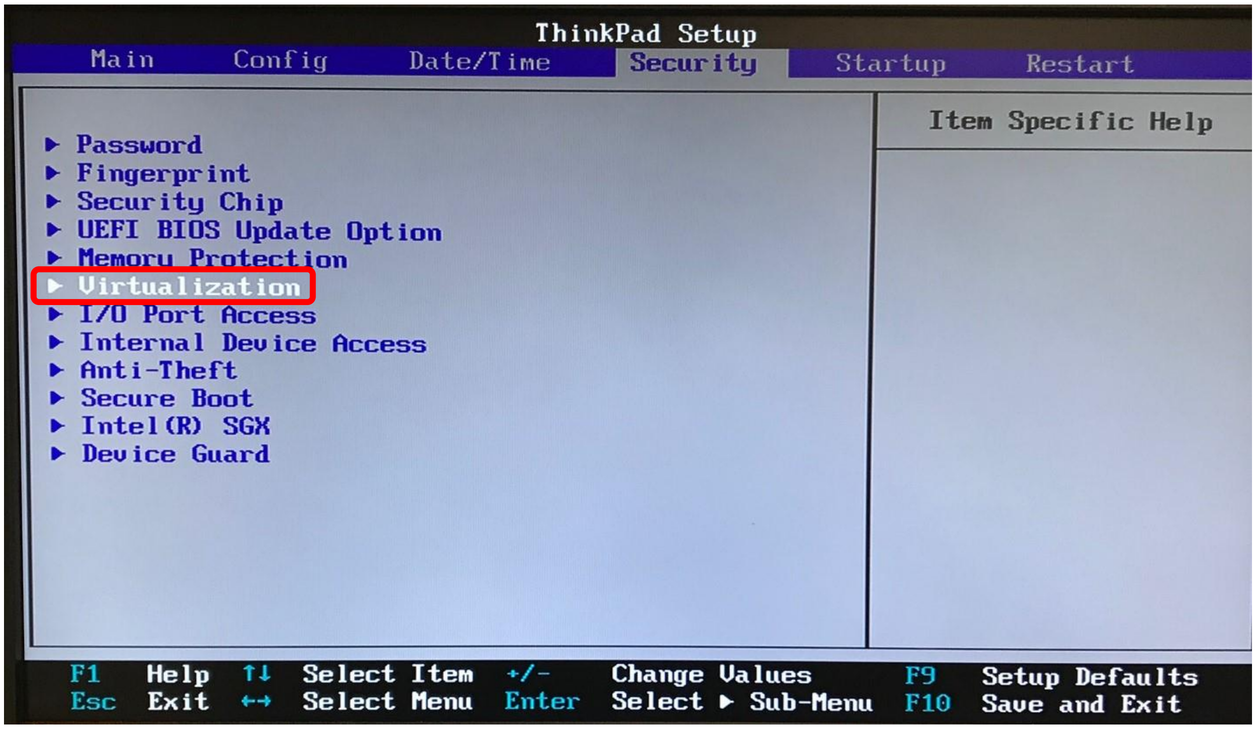
Navigate to the “Security” tab.



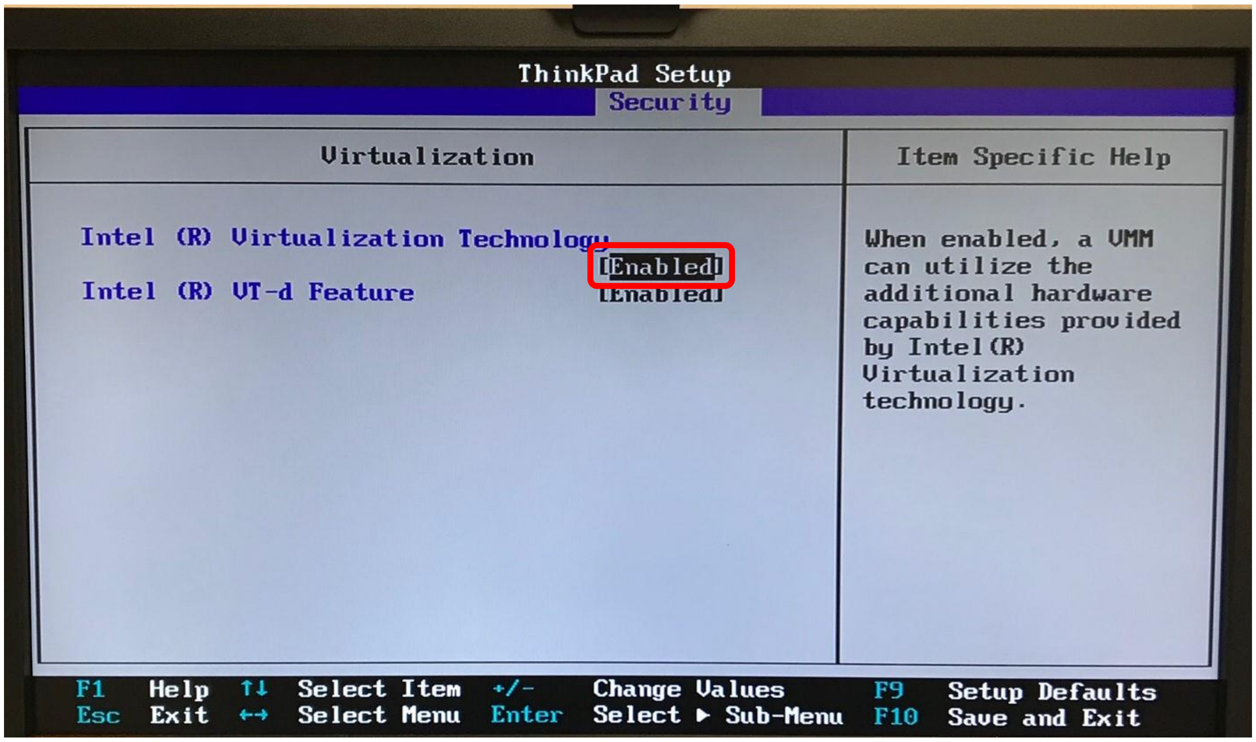
```
UEFI BIOS Version                       N1QET77W (1.52 )
UEFI BIOS Date (Year-Month-Date)        2018-07-04
Embedded Controller Version             N1QHT51W (1.33 )
ME Firmware Version                     11.8.80.3746
Machine Type Model                      20JM0009US
System-unit serial number               PF0W06VB
System board serial number              L2HF7AR00D8
Asset Tag                               No Asset Information
CPU Type                                Intel(R) Core(TM) i5-6300U CPU
CPU Speed                               2.400GHz
Installed memory                        24576MB
UUID                                    727a6dcc-3315-11b2-a85c-e7e874b9f429
MAC Address (Internal LAN)              54 E1 AD B9 B7 2C
Preinstalled OS License                 SDK0J40705 WIN
UEFI Secure Boot                        Off
OA3 ID                                  3425078238085
OA2                                     Yes
```

F1 Help ↑↓ Select Item +/- Change Values F9 Setup Defaults
Esc Exit ↔ Select Menu Enter Select ▶ Sub-Menu F10 Save and Exit

Navigate to “Virtualization”.

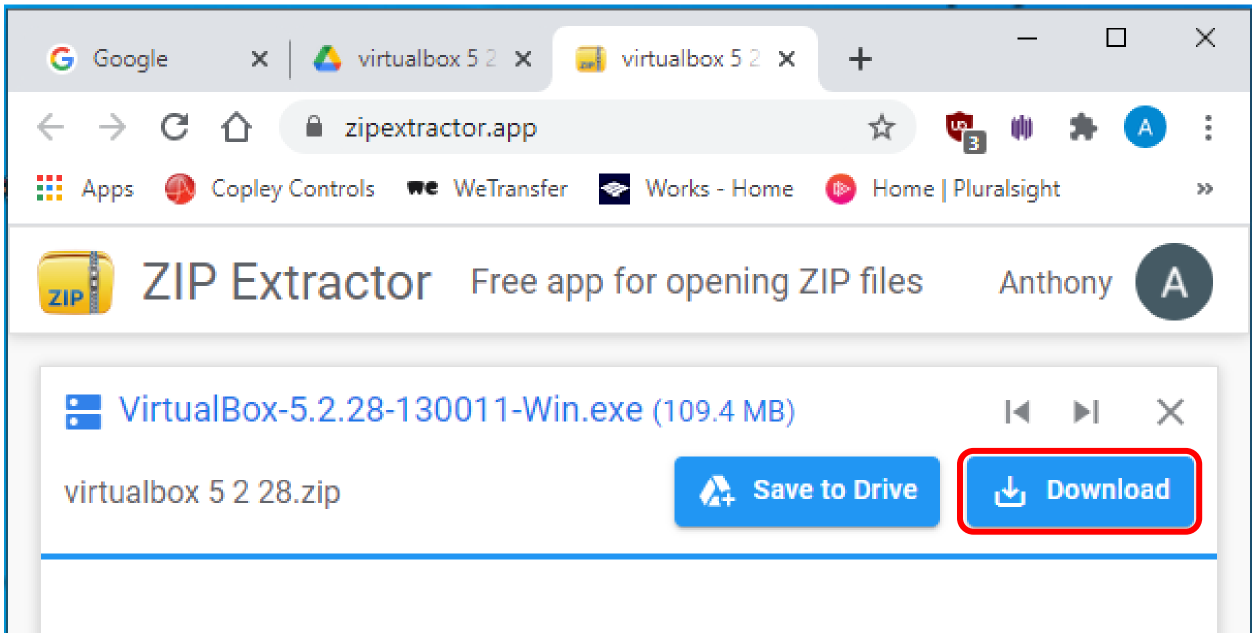


Change the virtualization setting to "Enabled".



```
ThinkPad Setup
Security

Virtualization                                   Item Specific Help

Intel (R) Virtualization Technology  [Enabled]   When enabled, a VMM
Intel (R) VT-d Feature               [Enabled]   can utilize the
                                                 additional hardware
                                                 capabilities provided
                                                 by Intel(R)
                                                 Virtualization
                                                 technology.

F1  Help  ↑↓ Select Item  +/-   Change Values        F9  Setup Defaults
Esc Exit  ↔  Select Menu  Enter Select ▶ Sub-Menu    F10 Save and Exit
```

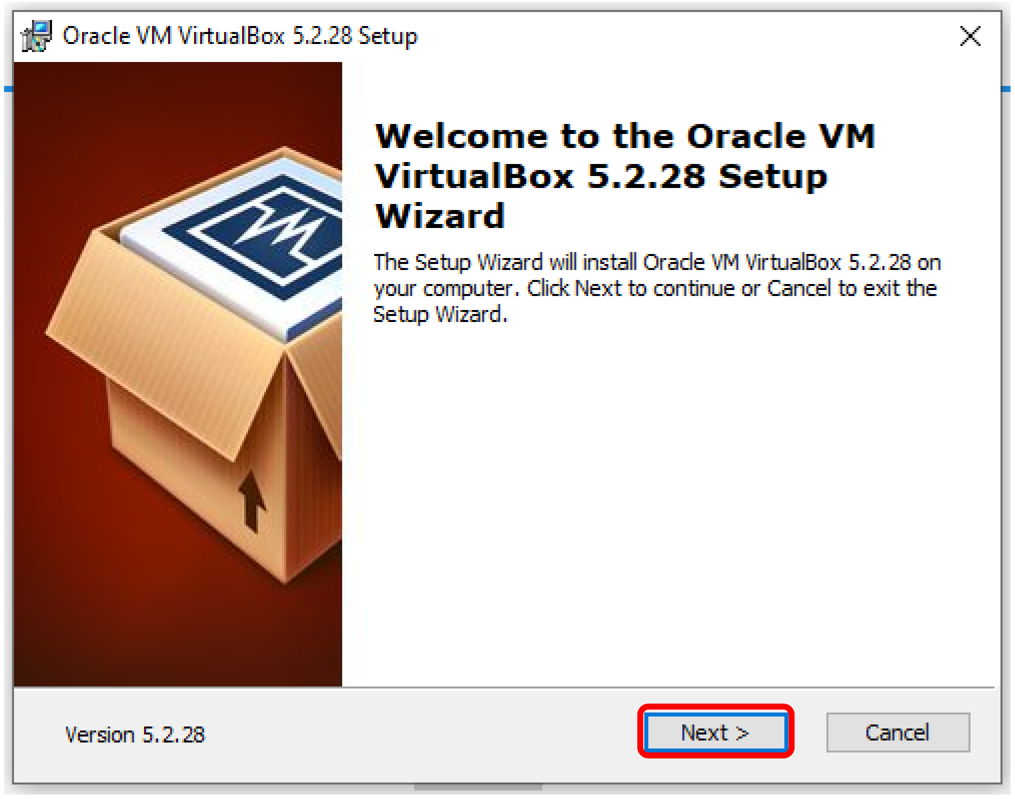
Now that the virtualization feature has been enabled, download the VirtualBox Version 5.2.28 executable from the link provided in the lab instructions.



VirtualBox-5.2.28-130011-Win.exe (109.4 MB)

virtualbox 5 2 28.zip

Click "Next" shown below.



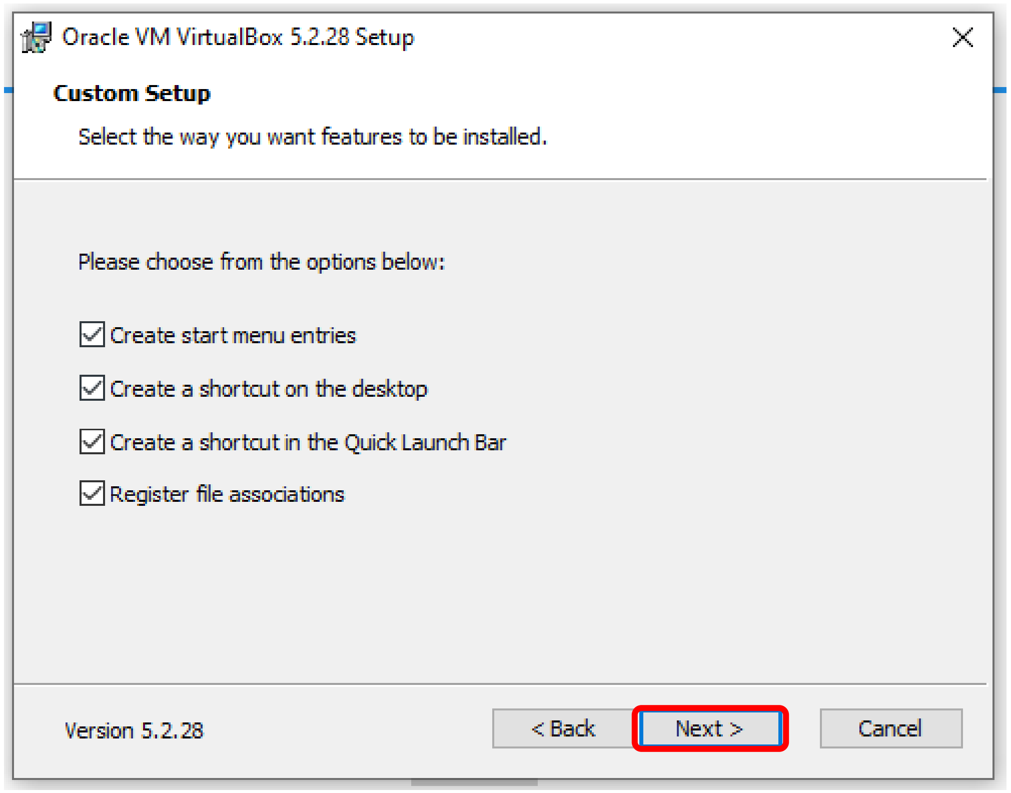
Oracle VM VirtualBox 5.2.28 Setup

Welcome to the Oracle VM VirtualBox 5.2.28 Setup Wizard

The Setup Wizard will install Oracle VM VirtualBox 5.2.28 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Version 5.2.28

Click the "Next" button provided that all the boxes in the options menu are checked.



Oracle VM VirtualBox 5.2.28 Setup

Custom Setup

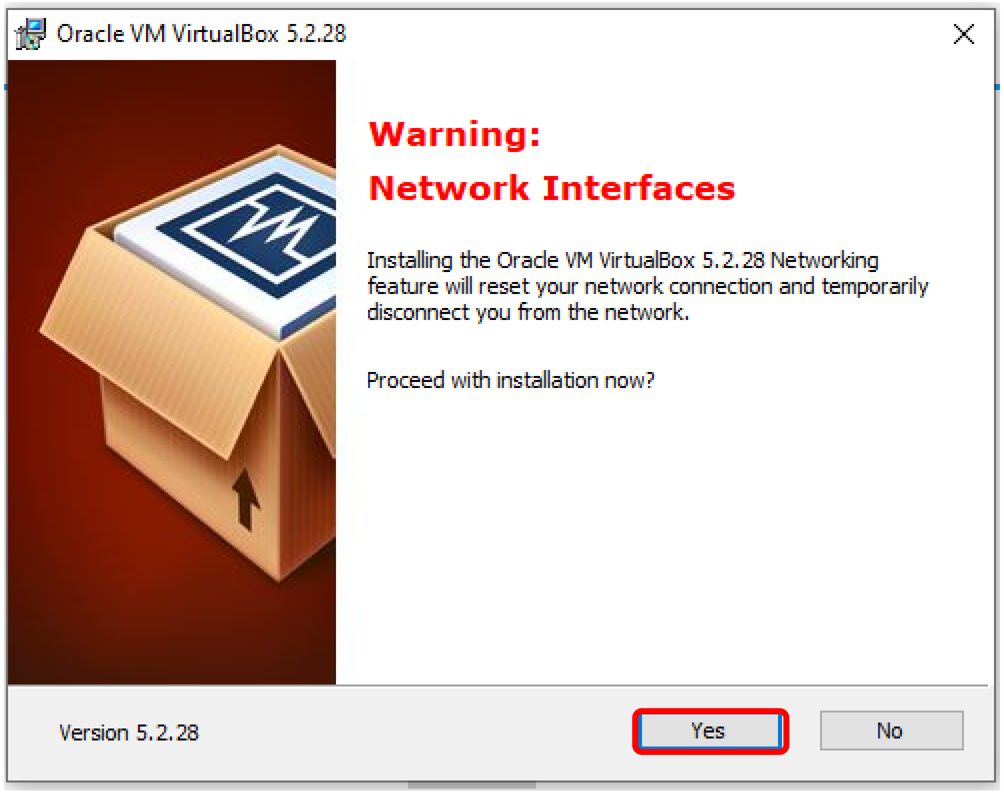
Select the way you want features to be installed.

Please choose from the options below:

- ☑ Create start menu entries
- ☑ Create a shortcut on the desktop
- ☑ Create a shortcut in the Quick Launch Bar
- ☑ Register file associations

Version 5.2.28

Click the “Yes” button shown below.


Oracle VM VirtualBox 5.2.28

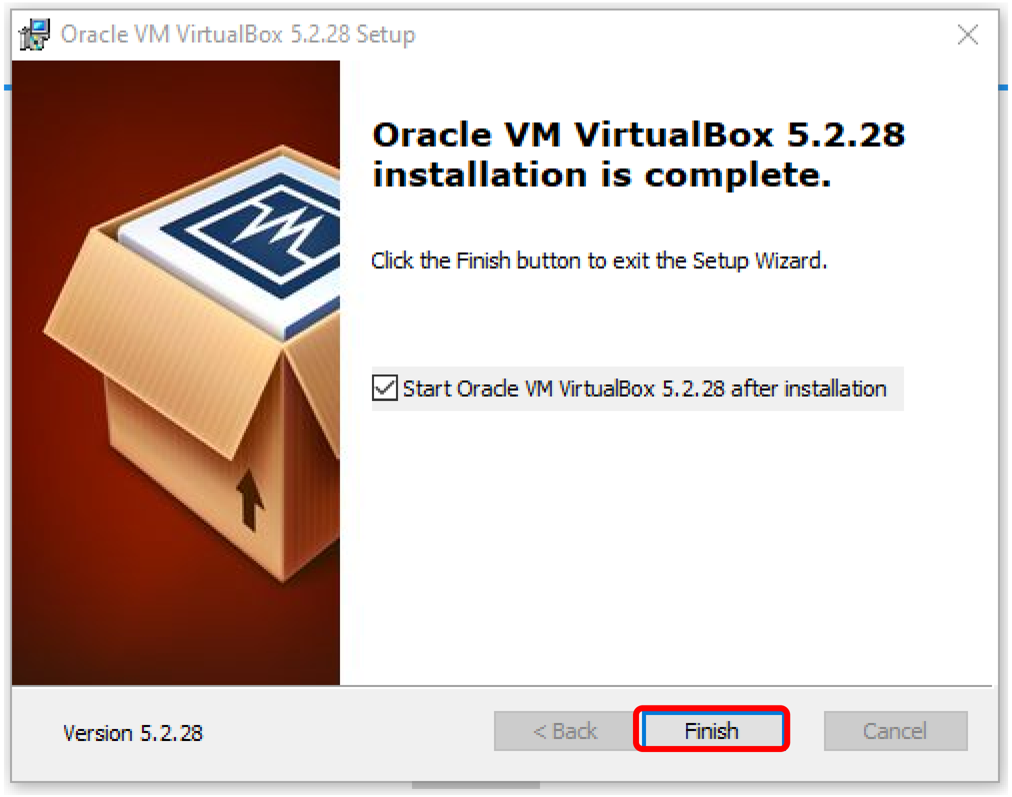
Warning:
Network Interfaces

Installing the Oracle VM VirtualBox 5.2.28 Networking feature will reset your network connection and temporarily disconnect you from the network.

Proceed with installation now?

Version 5.2.28

Click the “Finish” button shown below.


Oracle VM VirtualBox 5.2.28 Setup

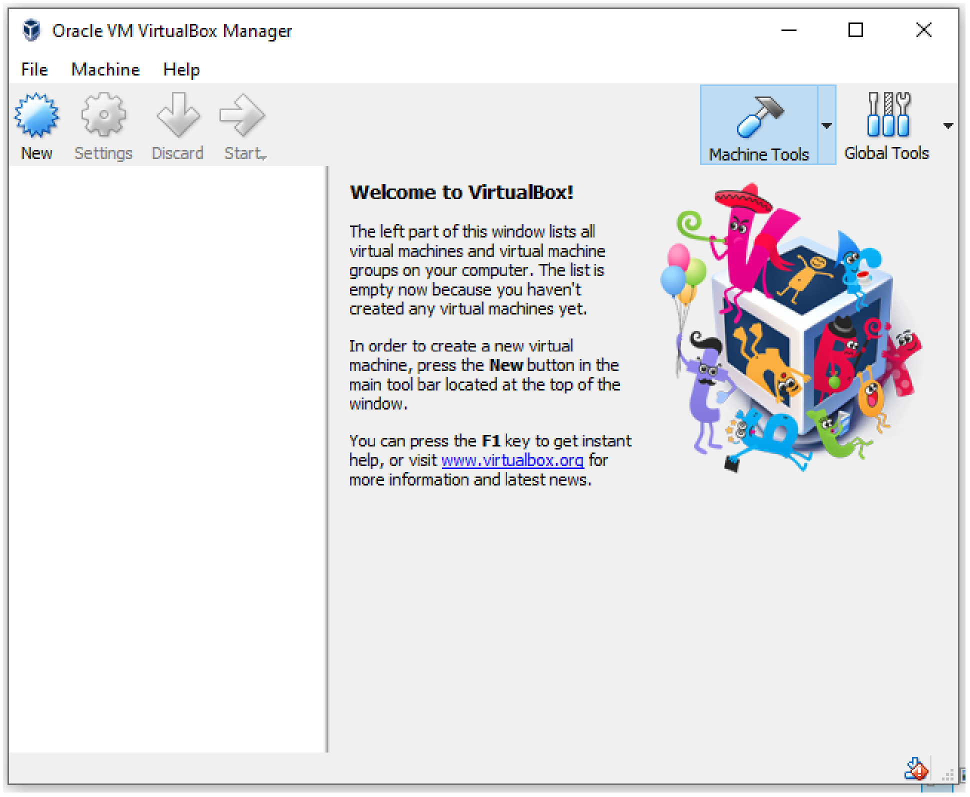
Oracle VM VirtualBox 5.2.28 installation is complete.

Click the Finish button to exit the Setup Wizard.

☑ Start Oracle VM VirtualBox 5.2.28 after installation

Version 5.2.28

The screen below should appear.



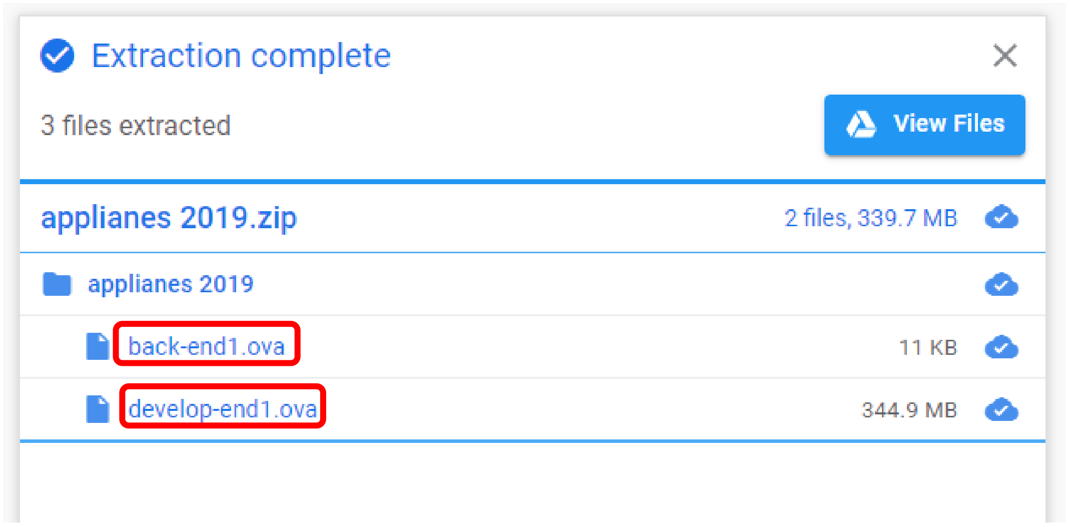
Welcome to VirtualBox!

The left part of this window lists all virtual machines and virtual machine groups on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

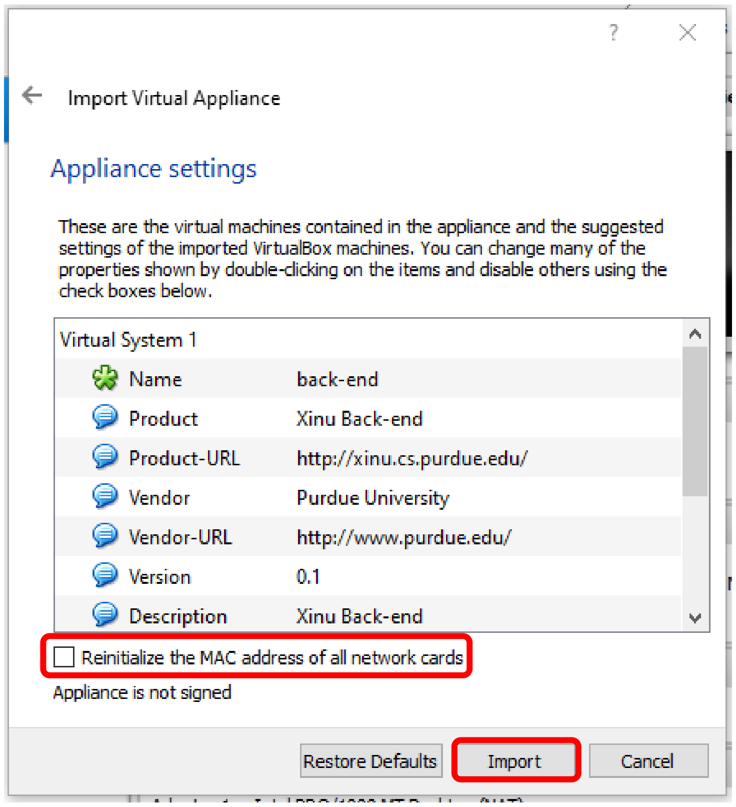
Close the VirtualBox application and download the two main files from the Xinu 2019 appliances link provided in the lab instructions.



3 files extracted

applianes 2019.zip	2 files, 339.7 MB
applianes 2019	
back-end1.ova	11 KB
develop-end1.ova	344.9 MB

Open the VirtualBox application again and select File > Import Appliance. Click on the folder icon and select the file “back-end1.ova”. Click “Next”. Be sure to uncheck the “Reinitialize the MAC address of all network cards” checkbox. Click the “Import” button shown below.



Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

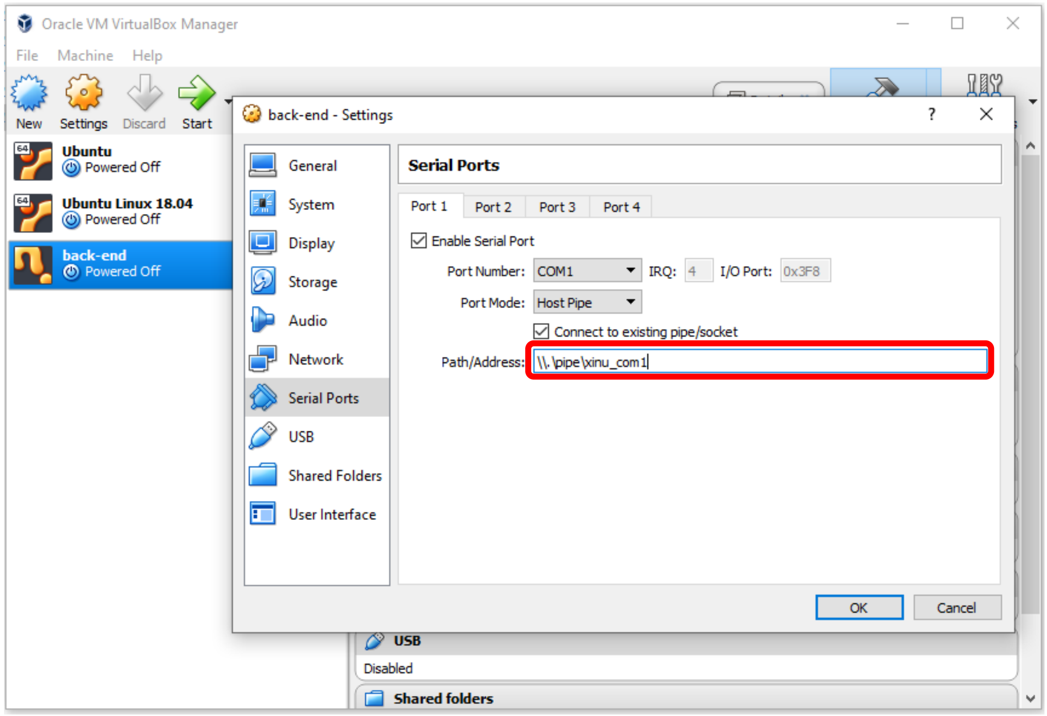
Virtual System 1

Name	back-end
Product	Xinu Back-end
Product-URL	http://xinu.cs.purdue.edu/
Vendor	Purdue University
Vendor-URL	http://www.purdue.edu/
Version	0.1
Description	Xinu Back-end

☐ Reinitialize the MAC address of all network cards

Appliance is not signed

Select “back-end” from the main menu and click the “Settings” button. In the settings menu, select “Serial Ports”. Check the “Enable Serial Port” checkbox. The Port Number drop-down menu should have “COM1” selected, and the Port Mode drop-down menu should have “Host Pipe” selected. Check the “Connect to existing pipe/socket” checkbox. Edit the Path/Address to “\\.\pipe\xinu_com1”. Click “OK”.



Serial Ports

☑ Enable Serial Port

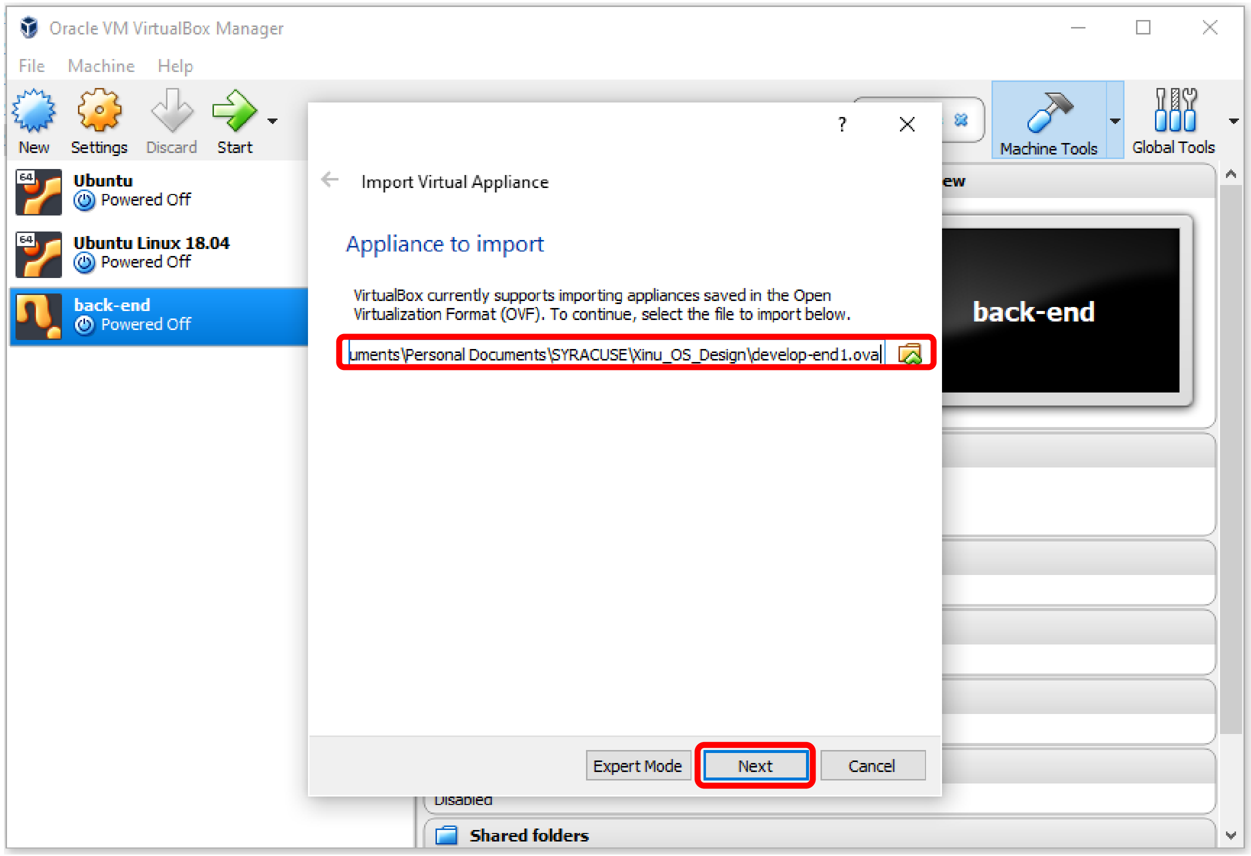
Port Number: COM1 IRQ: 4 I/O Port: 0x3F8

Port Mode: Host Pipe

☑ Connect to existing pipe/socket

Path/Address: \\.\pipe\xinu_com1

Next, import the develop-end1.ova file. Click File > Import Appliance. Click on the folder icon and select the file “develop-end1.ova”. Click “Next”.

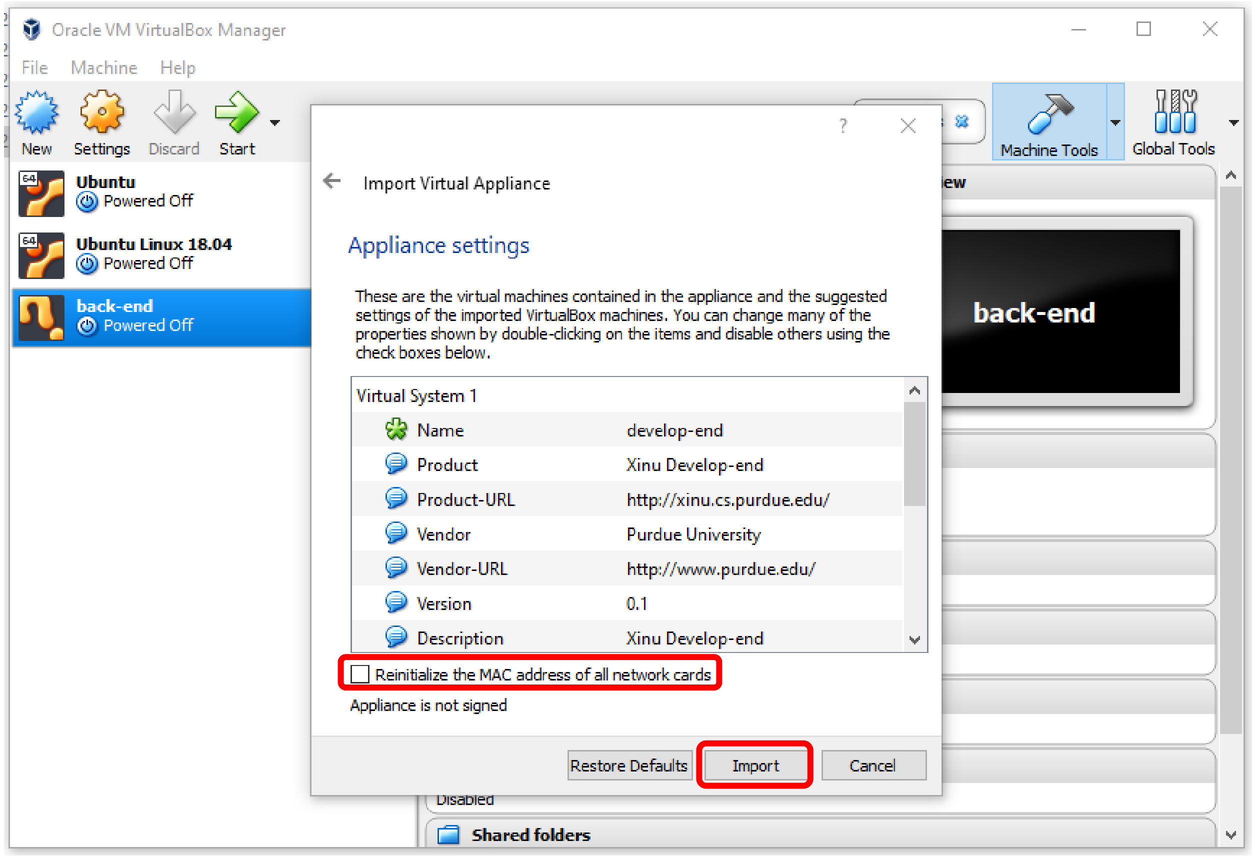


Appliance to import

VirtualBox currently supports importing appliances saved in the Open Virtualization Format (OVF). To continue, select the file to import below.

uments\Personal Documents\SYRACUSE\Xinu_OS_Design\develop-end1.ova

Be sure to uncheck the “Reinitialize the MAC address of all network cards” checkbox. Click the “Import” button shown below.


Oracle VM VirtualBox Manager
File Machine Help
New Settings Discard Start
Machine Tools Global Tools
Ubuntu Powered Off
Ubuntu Linux 18.04 Powered Off
back-end Powered Off
back-end
Disabled
Shared folders
Import Virtual Appliance

Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

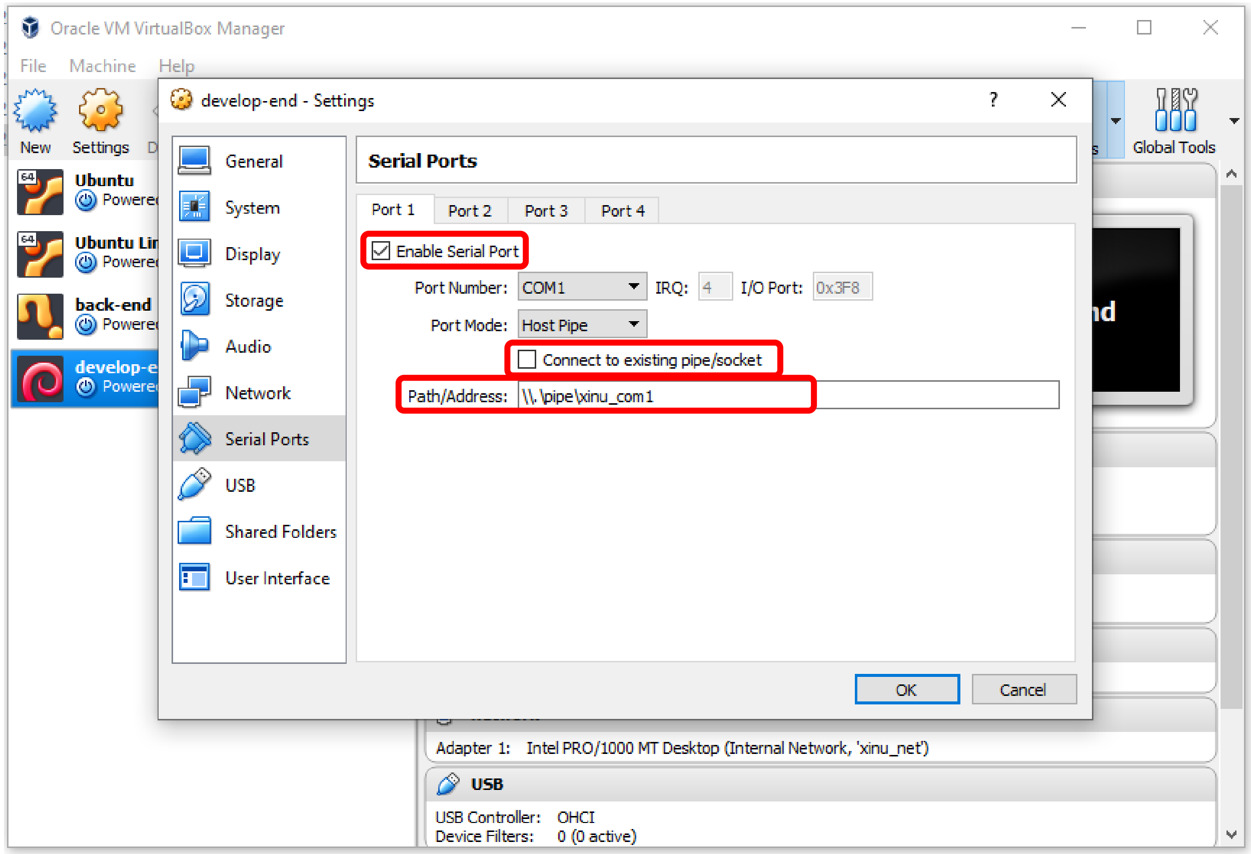
Virtual System 1

Name	develop-end
Product	Xinu Develop-end
Product-URL	http://xinu.cs.purdue.edu/
Vendor	Purdue University
Vendor-URL	http://www.purdue.edu/
Version	0.1
Description	Xinu Develop-end

☐ Reinitialize the MAC address of all network cards

Appliance is not signed

Select “develop-end” from the main menu, and click the “Settings” button. In the settings menu, select “Serial Ports”. Check the “Enable Serial Port” checkbox. The Port Number drop-down menu should have “COM1” selected, and the Port Mode drop-down menu should have “Host Pipe” selected. <u>Do not check</u> the “Connect to existing pipe/socket” checkbox. Edit the Path/Address to “\\.\pipe\xinu_com1”. Click “OK”.



Serial Ports

☑ Enable Serial Port

Port Number: COM1 IRQ: 4 I/O Port: 0x3F8

Port Mode: Host Pipe

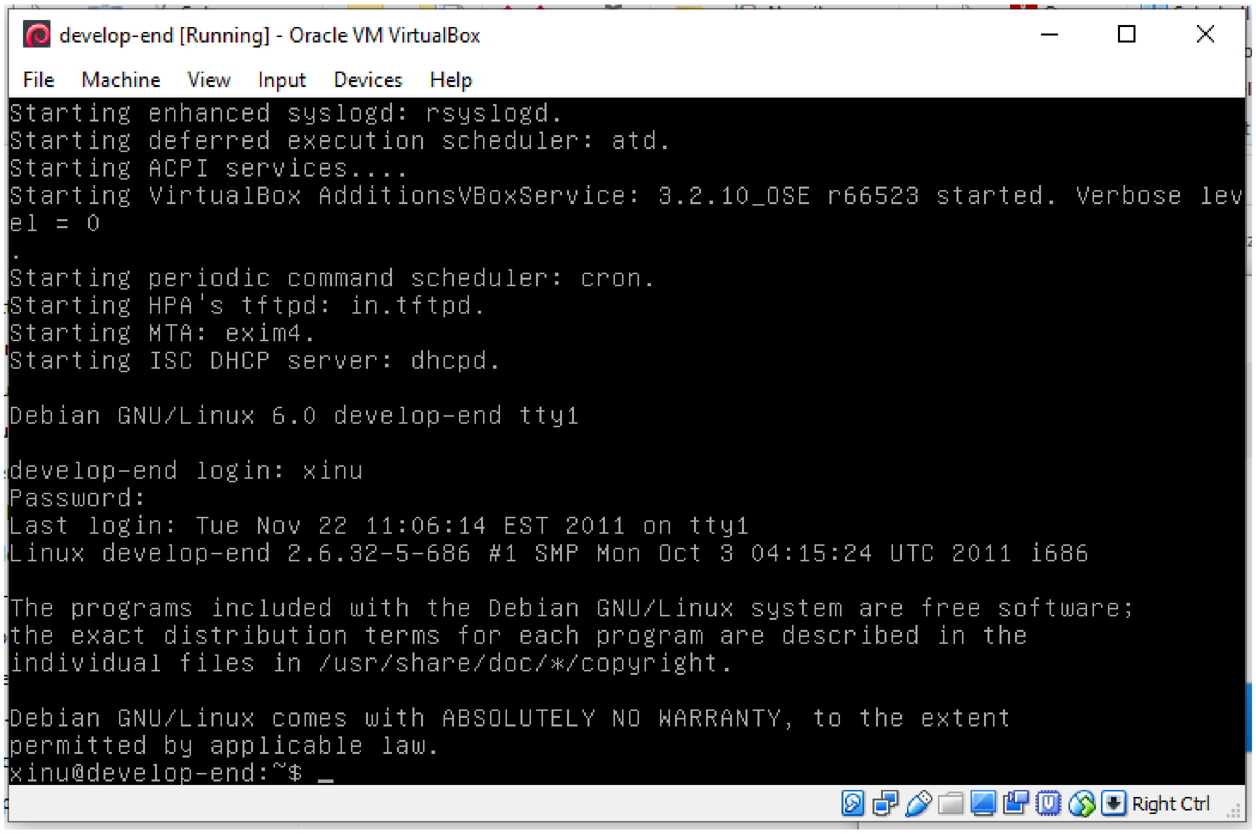
☐ Connect to existing pipe/socket

Path/Address: \\.\pipe\xinu_com1

Adapter 1: Intel PRO/1000 MT Desktop (Internal Network, 'xinu_net')

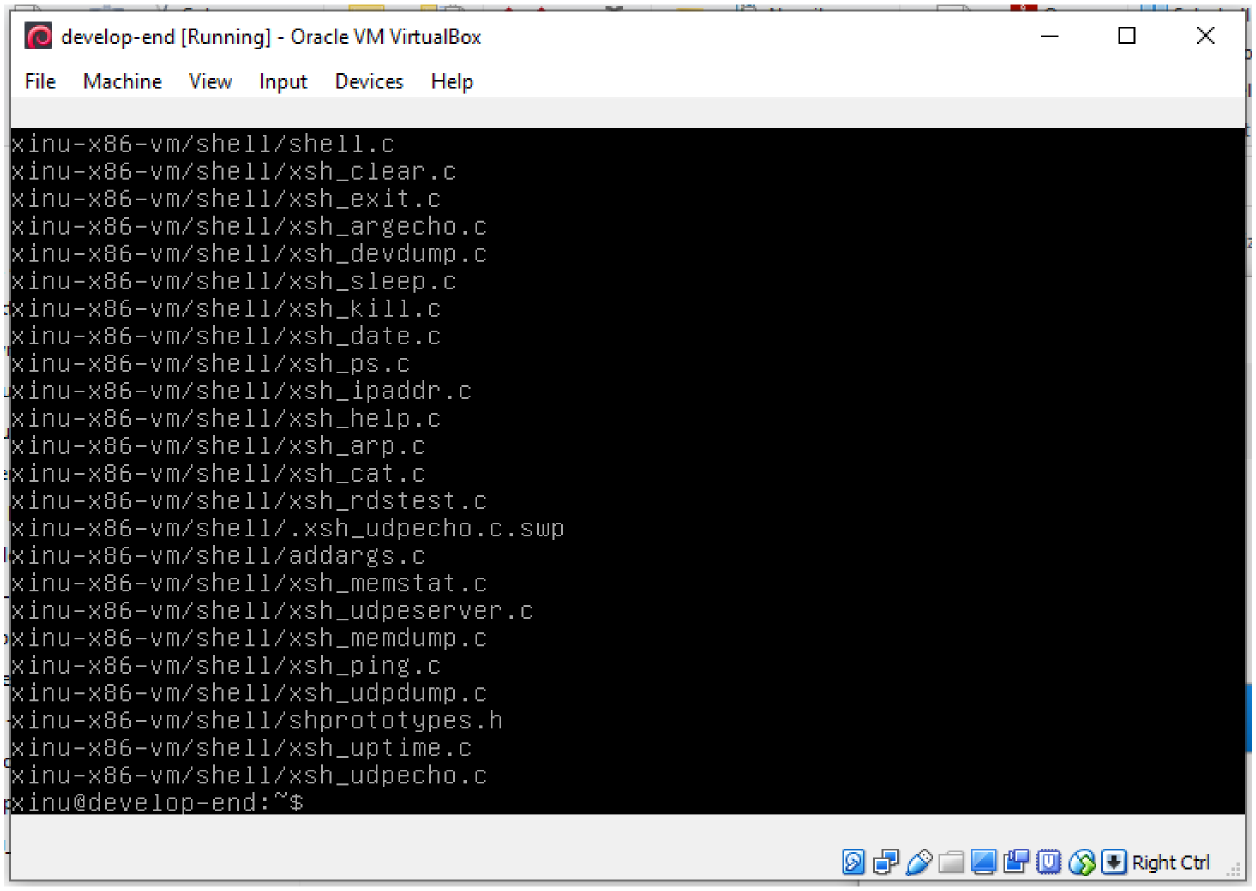
USB Controller: OHCI
Device Filters: 0 (0 active)

Start the develop-end and use the username “xinu” and the password “xinurocks”. The console should look like the screen below.


develop-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

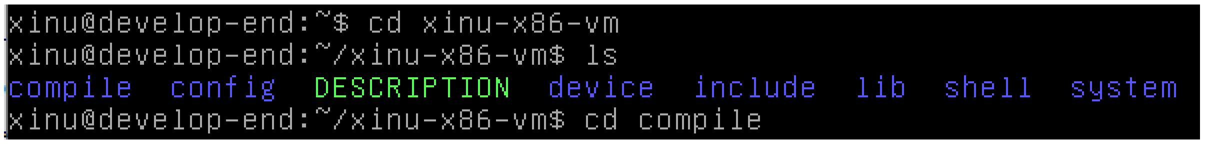
```
Starting enhanced syslogd: rsyslogd.
Starting deferred execution scheduler: atd.
Starting ACPI services....
Starting VirtualBox AdditionsVBoxService: 3.2.10_OSE r66523 started. Verbose lev
el = 0
.
Starting periodic command scheduler: cron.
Starting HPA's tftpd: in.tftpd.
Starting MTA: exim4.
Starting ISC DHCP server: dhcpd.

Debian GNU/Linux 6.0 develop-end tty1

develop-end login: xinu
Password:
Last login: Tue Nov 22 11:06:14 EST 2011 on tty1
Linux develop-end 2.6.32-5-686 #1 SMP Mon Oct 3 04:15:24 UTC 2011 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
xinu@develop-end:~$ _
```

Next, navigate to the home directory and type “ls”. There should be two folders present, one of which is “xinu-x86-vm.tar.gz”. To untar this file, type “tar -zxvf xinu-x86-vm.tar.gz”.


develop-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

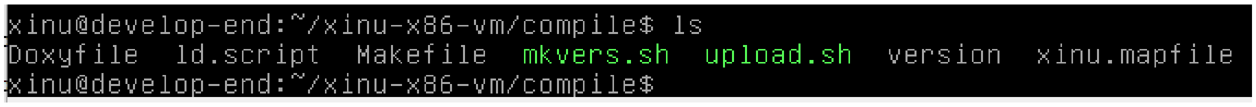
```
xinu-x86-vm/shell/shell.c
xinu-x86-vm/shell/xsh_clear.c
xinu-x86-vm/shell/xsh_exit.c
xinu-x86-vm/shell/xsh_argecho.c
xinu-x86-vm/shell/xsh_devdump.c
xinu-x86-vm/shell/xsh_sleep.c
xinu-x86-vm/shell/xsh_kill.c
xinu-x86-vm/shell/xsh_date.c
xinu-x86-vm/shell/xsh_ps.c
xinu-x86-vm/shell/xsh_ipaddr.c
xinu-x86-vm/shell/xsh_help.c
xinu-x86-vm/shell/xsh_arp.c
xinu-x86-vm/shell/xsh_cat.c
xinu-x86-vm/shell/xsh_rdstest.c
xinu-x86-vm/shell/.xsh_udpecho.c.swp
xinu-x86-vm/shell/addargs.c
xinu-x86-vm/shell/xsh_memstat.c
xinu-x86-vm/shell/xsh_udpeserver.c
xinu-x86-vm/shell/xsh_memdump.c
xinu-x86-vm/shell/xsh_ping.c
xinu-x86-vm/shell/xsh_udpdump.c
xinu-x86-vm/shell/shprototypes.h
xinu-x86-vm/shell/xsh_uptime.c
xinu-x86-vm/shell/xsh_udpecho.c
xinu@develop-end:~$
```

Next, type “cd xinu-x86-vm”. Then “ls”. One of the folders listed should be “compile”. Type “cd compile”. The series of commands is shown below.



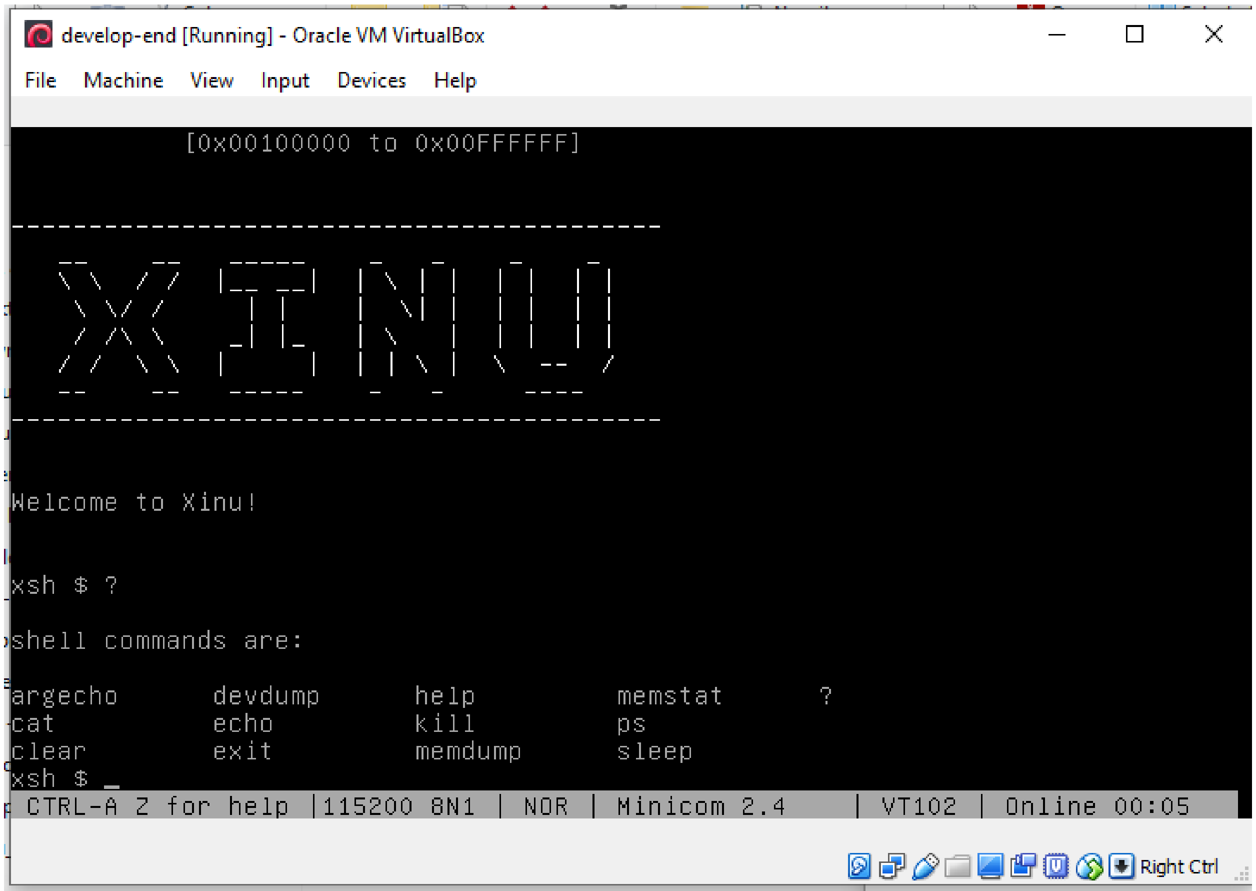
```
xinu@develop-end:~$ cd xinu-x86-vm
xinu@develop-end:~/xinu-x86-vm$ ls
compile  config  DESCRIPTION  device  include  lib  shell  system
xinu@develop-end:~/xinu-x86-vm$ cd compile
```

Next, type “ls” to view the files available in the compile folder.



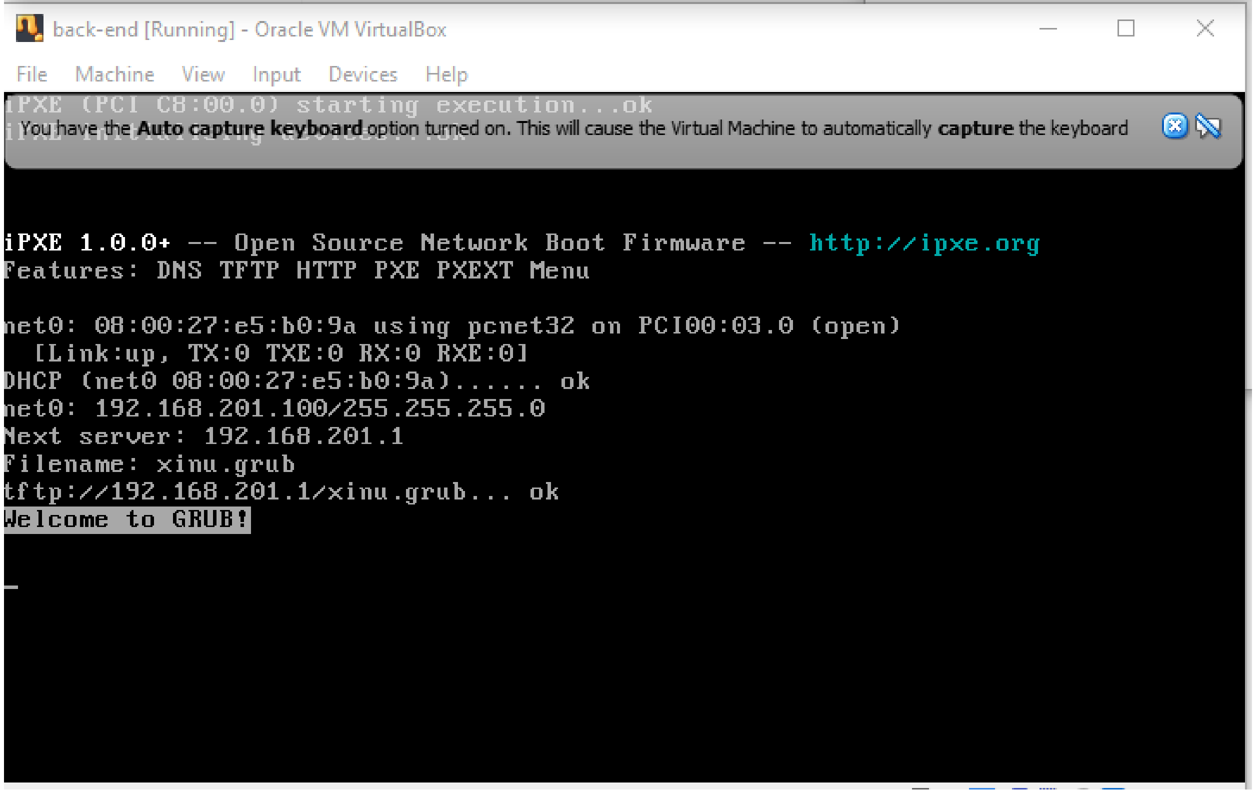
```
xinu@develop-end:~/xinu-x86-vm/compile$ ls
Doxyfile  ld.script  Makefile  mkvers.sh  upload.sh  version  xinu.mapfile
xinu@develop-end:~/xinu-x86-vm/compile$
```

Type “make clean;make;./upload.sh” followed by “sudo minicom”. If prompted for a password, use “xinurocks”. The screen should then display “Welcome to minicom 2.4”. Open up the back-end. The develop-end should display the image “XINU” followed by the greeting, “Welcome to Xinu!” Type the “?” character to see a list of available commands. The develop-end and back-end at this specific step are displayed on the next page.



develop-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
          [0x00100000 to 0x00FFFFFF]

------------------------------------------

Welcome to Xinu!

xsh $ ?

shell commands are:

argecho      devdump      help         memstat      ?
cat          echo         kill         ps
clear        exit         memdump      sleep
xsh $ _
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4     | VT102 | Online 00:05
```



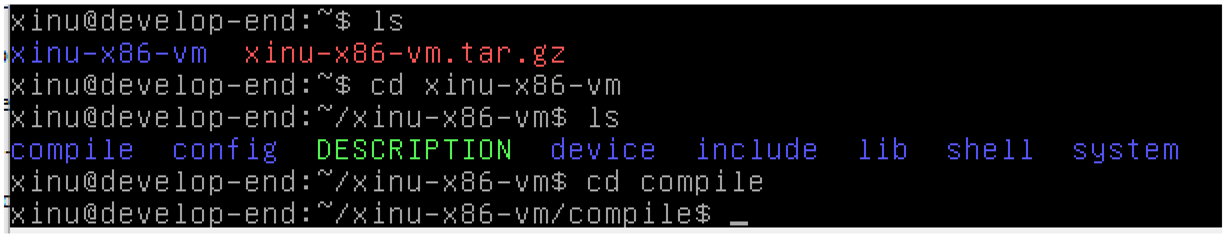
back-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard

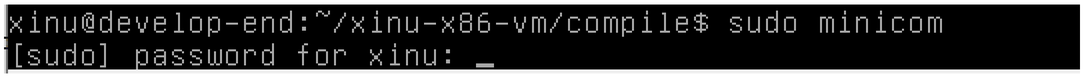
```
iPXE (PCI C8:00.0) starting execution...ok

iPXE 1.0.0+ -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS TFTP HTTP PXE PXEXT Menu

net0: 08:00:27:e5:b0:9a using pcnet32 on PCI00:03.0 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
DHCP (net0 08:00:27:e5:b0:9a)...... ok
net0: 192.168.201.100/255.255.255.0
Next server: 192.168.201.1
Filename: xinu.grub
tftp://192.168.201.1/xinu.grub... ok
Welcome to GRUB!

_
```

Section B: Edit and Type Your First Program

Close the develop-end and back-end. To run the compiled code again, simply navigate to the develop-end "/xinu-x86-vm/compile" folder as before and run "sudo minicom". There is no need to recompile the project. See the commands below.

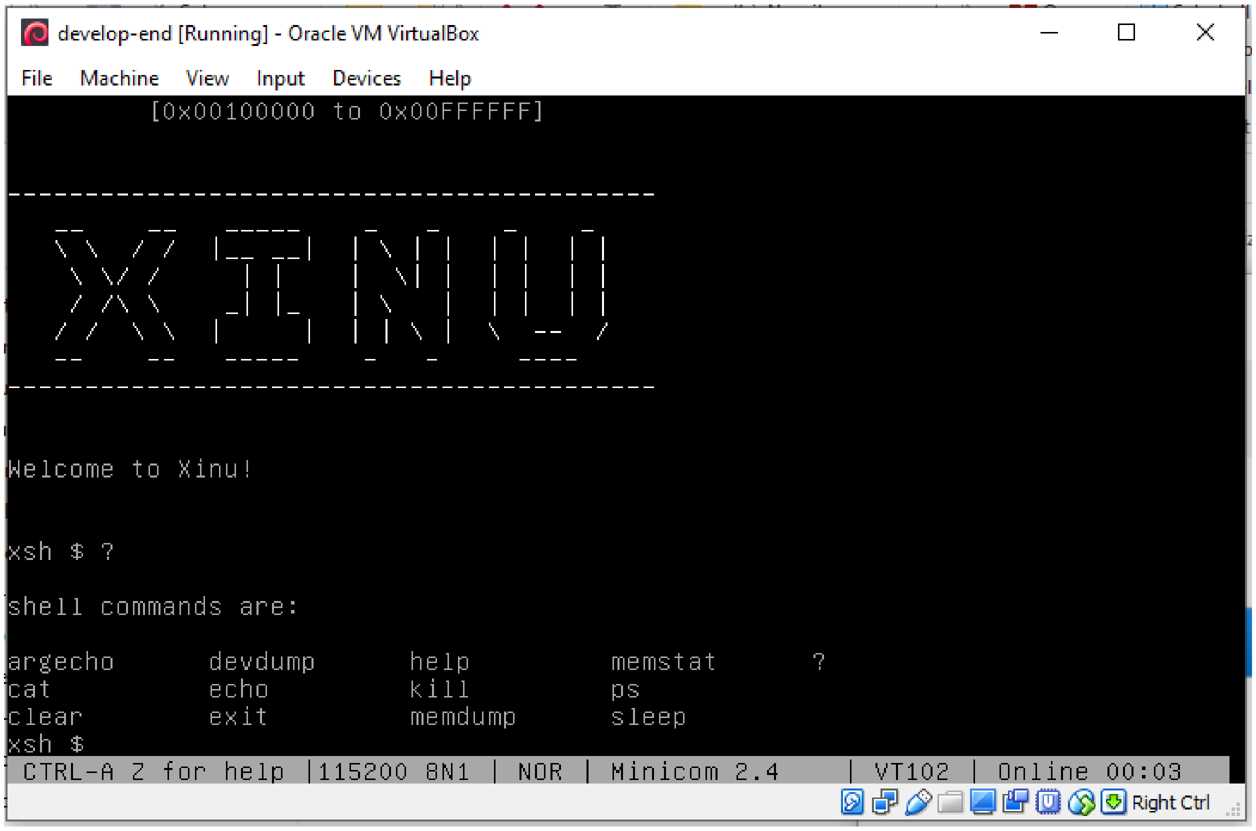


```
xinu@develop-end:~$ ls
xinu-x86-vm  xinu-x86-vm.tar.gz
xinu@develop-end:~$ cd xinu-x86-vm
xinu@develop-end:~/xinu-x86-vm$ ls
compile  config  DESCRIPTION  device  include  lib  shell  system
xinu@develop-end:~/xinu-x86-vm$ cd compile
xinu@develop-end:~/xinu-x86-vm/compile$ _
```



```
xinu@develop-end:~/xinu-x86-vm/compile$ sudo minicom
[sudo] password for xinu: _
```

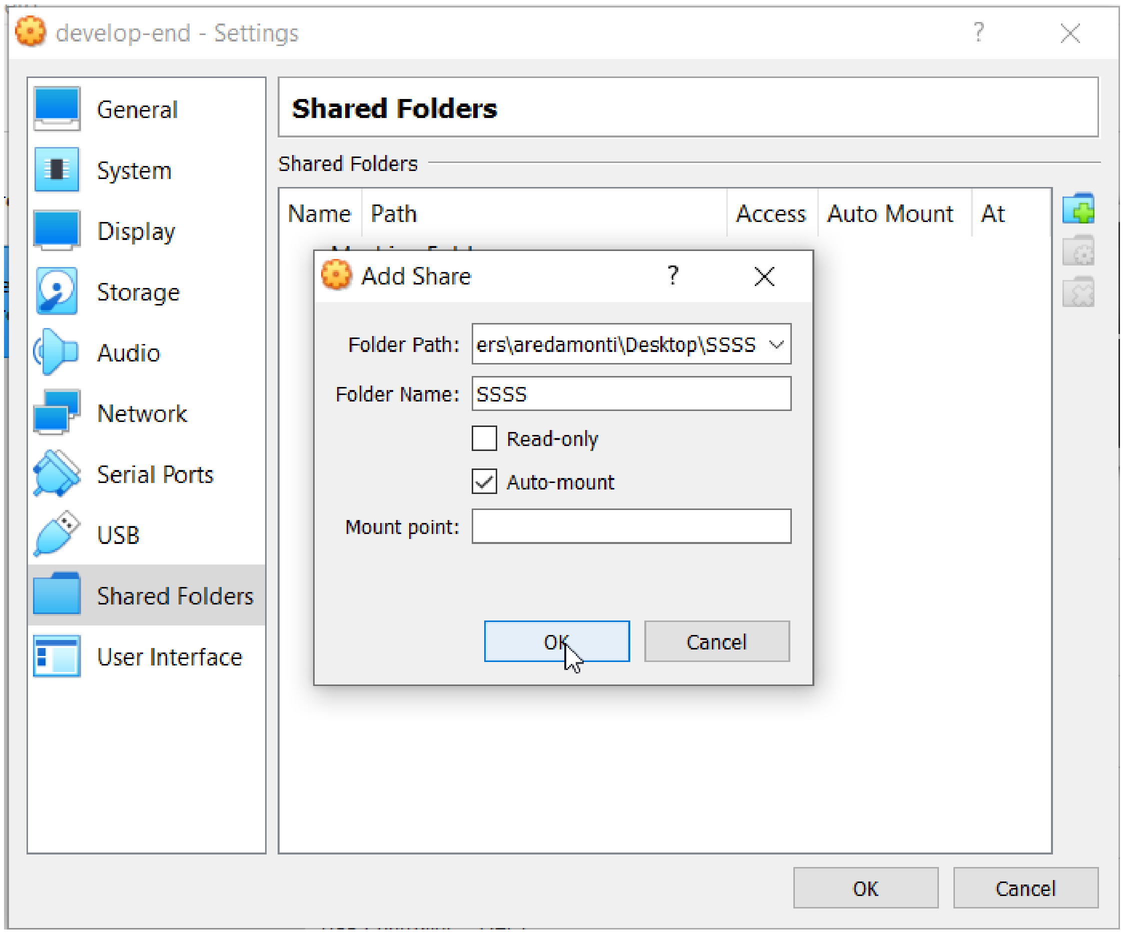
Then open the back-end. Type the "?" character for a list of available commands.



```
       [0x00100000 to 0x00FFFFFF]


------------------------------------------

------------------------------------------


Welcome to Xinu!


xsh $ ?

shell commands are:

argecho      devdump      help         memstat      ?
cat          echo         kill         ps
clear        exit         memdump      sleep
xsh $
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4     | VT102 | Online 00:03
```

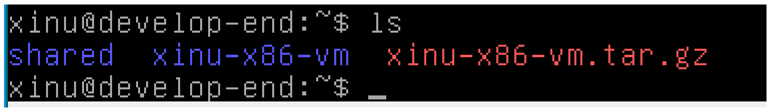
Close the develop-end and back-end. Create an empty folder on the desktop of the host OS (Windows). The name "SSSS" was arbitrarily chosen for the folder in this example. Insert a file into the folder. In VirtualBox, click on the develop-end and select Settings > Shared Folders > Add. Find and select the "SSSS" folder on the host desktop. Check the "Auto-mount" option checkbox.

The screen below displays the correct settings for the shared folder as they were just described. Click "OK".



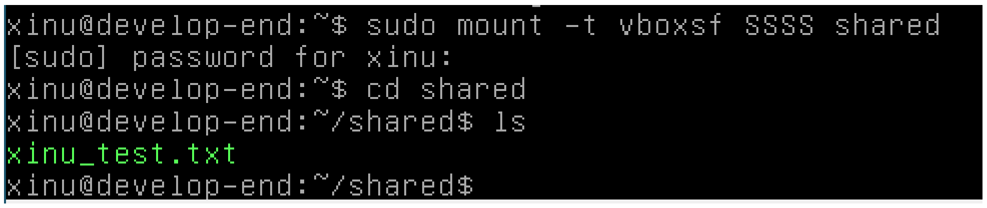
Folder Path: ers\aredamonti\Desktop\SSSS
Folder Name: SSSS
[ ] Read-only
[✓] Auto-mount
Mount point:

Open the develop-end and type "mkdir shared" in the home directory. An "ls" command will show the newly created "shared" directory.



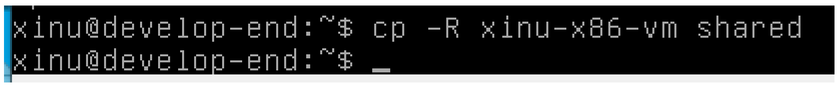
```
xinu@develop-end:~$ ls
shared  xinu-x86-vm  xinu-x86-vm.tar.gz
xinu@develop-end:~$ _
```

Reboot the develop-end. Before using the shared folder, be sure to mount it using the following command: "sudo mount -t vboxsf SSSS shared". The "shared" folder will contain everything in the "SSSS" folder on the host desktop.

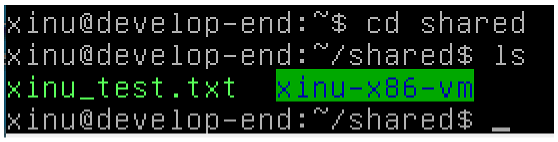


```
xinu@develop-end:~$ sudo mount -t vboxsf SSSS shared
[sudo] password for xinu:
xinu@develop-end:~$ cd shared
xinu@develop-end:~/shared$ ls
xinu_test.txt
xinu@develop-end:~/shared$
```

Next, copy the home directory into the shared folder. Navigate to the home directory and type “cp -R xinu-x86-vm shared”.



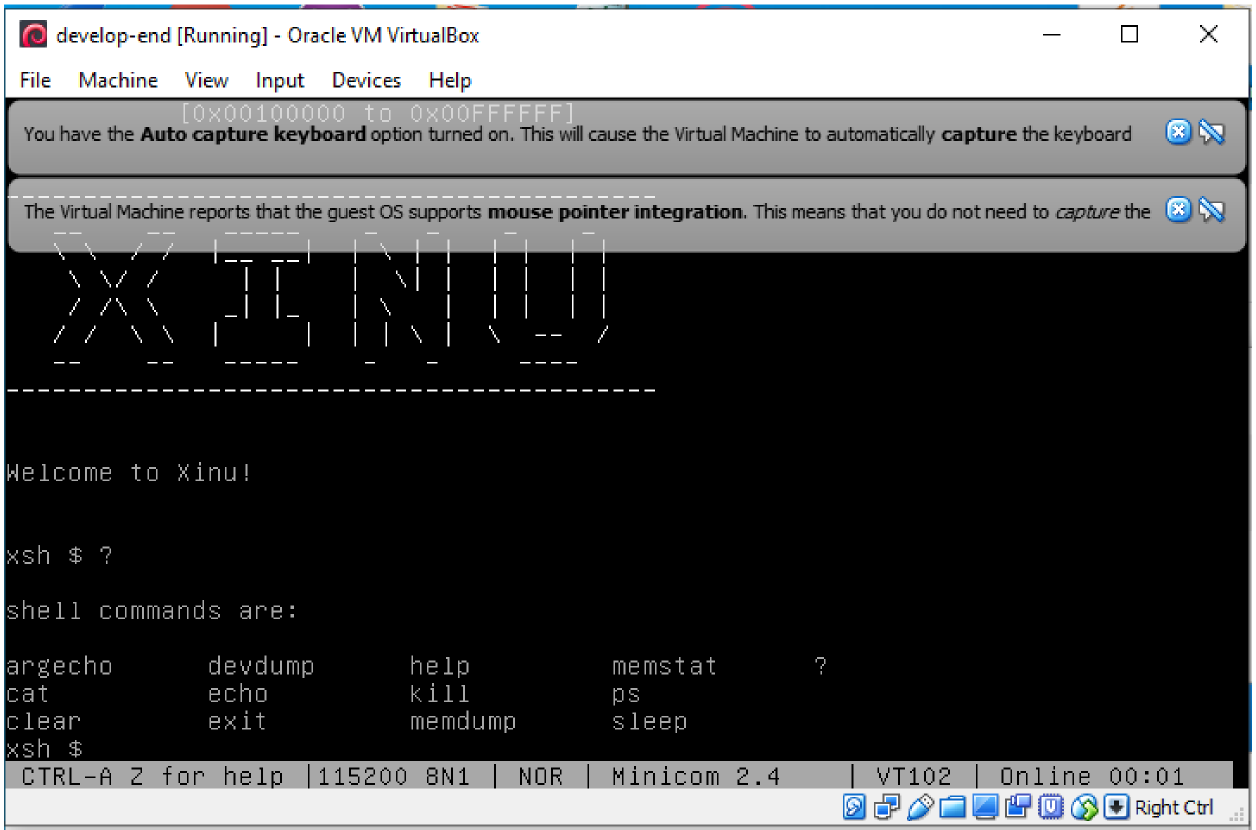
```
xinu@develop-end:~$ cp -R xinu-x86-vm shared
xinu@develop-end:~$ _
```



```
xinu@develop-end:~$ cd shared
xinu@develop-end:~/shared$ ls
xinu_test.txt  xinu-x86-vm
xinu@develop-end:~/shared$ _
```

To compile in the shared folder, follow the same steps previously performed.

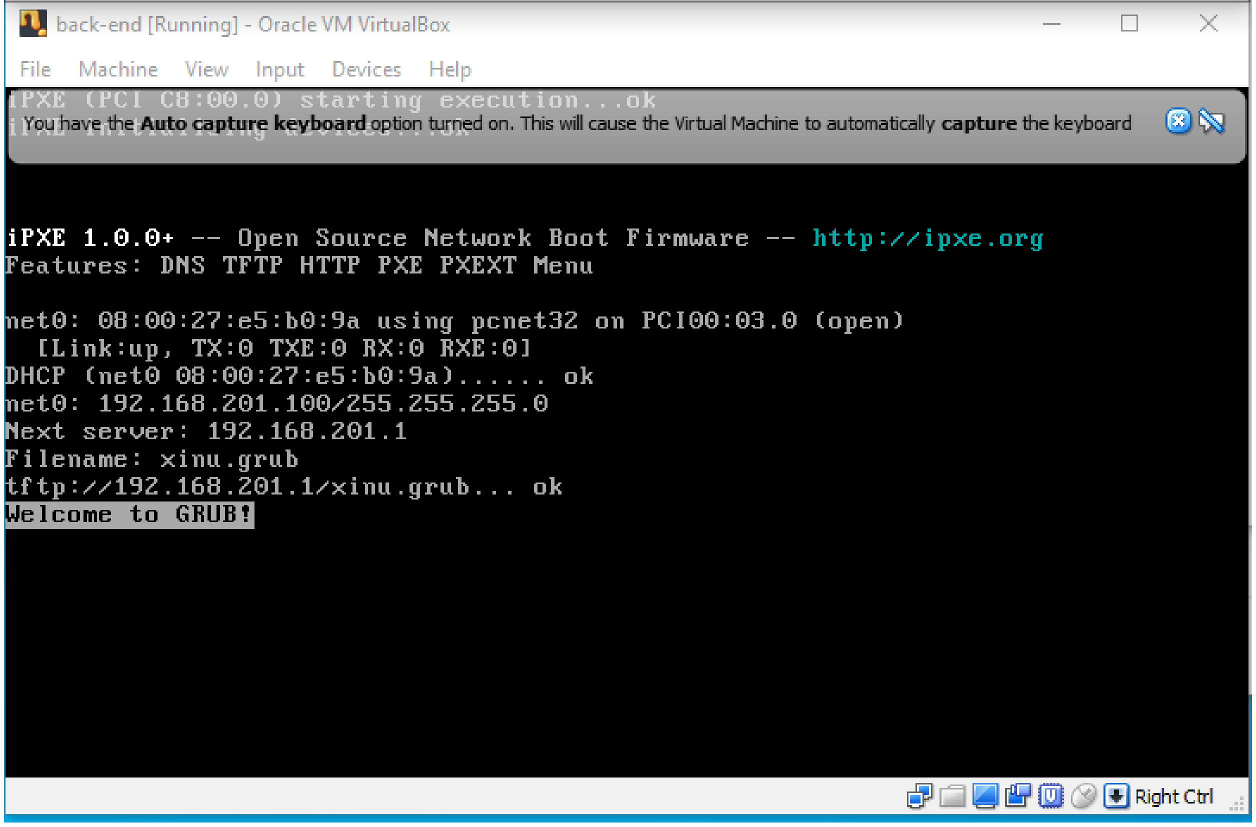
Starting from the beginning, first mount the “shared” folder using the command “sudo mount -t vboxsf SSSS shared”. Next, navigate to the “/shared/xinu-x86-vm/compile” folder. To compile, type “make clean;make;./upload.sh”. Then type “sudo minicom” and open the back-end. Type the “?” character in the develop-end to see a list of available commands. The develop-end and back-end screens should look like the image on the next page.


develop-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard

The Virtual Machine reports that the guest OS supports **mouse pointer integration**. This means that you do not need to *capture* the

```
[0x00100000 to 0x00FFFFFF]
------------------------------------------

------------------------------------------


Welcome to Xinu!


xsh $ ?

shell commands are:

argecho       devdump       help          memstat       ?
cat           echo          kill          ps
clear         exit          memdump       sleep
xsh $
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4      | VT102 | Online 00:01
```

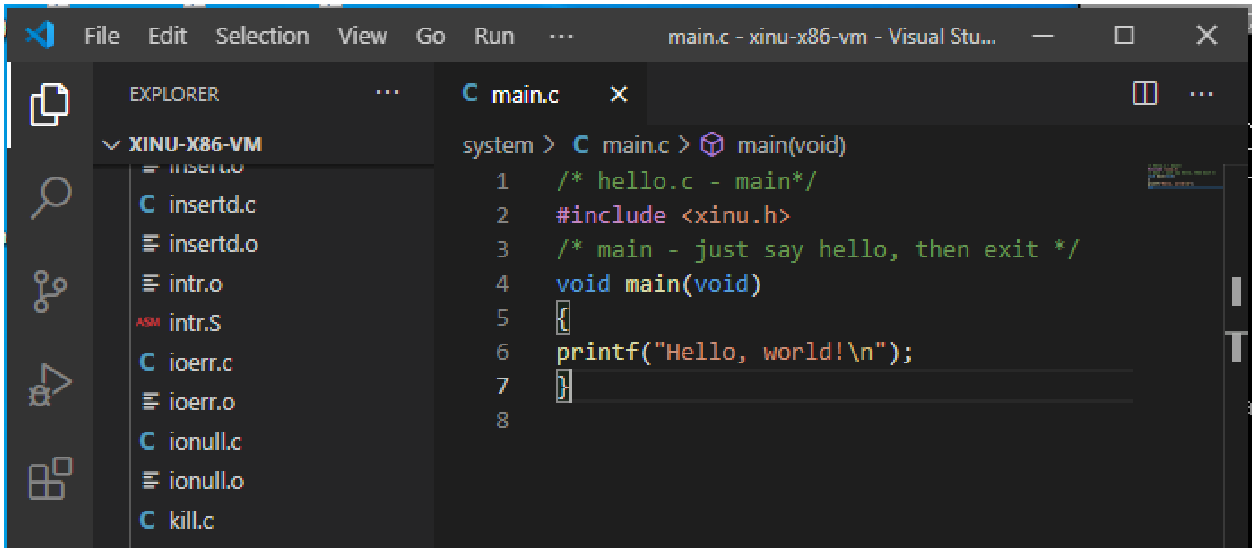

back-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard

```
iPXE (PCI C8:00.0) starting execution...ok
iPXE initialising devices...ok


iPXE 1.0.0+ -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS TFTP HTTP PXE PXEXT Menu

net0: 08:00:27:e5:b0:9a using pcnet32 on PCI00:03.0 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
DHCP (net0 08:00:27:e5:b0:9a)...... ok
net0: 192.168.201.100/255.255.255.0
Next server: 192.168.201.1
Filename: xinu.grub
tftp://192.168.201.1/xinu.grub... ok
Welcome to GRUB!
```

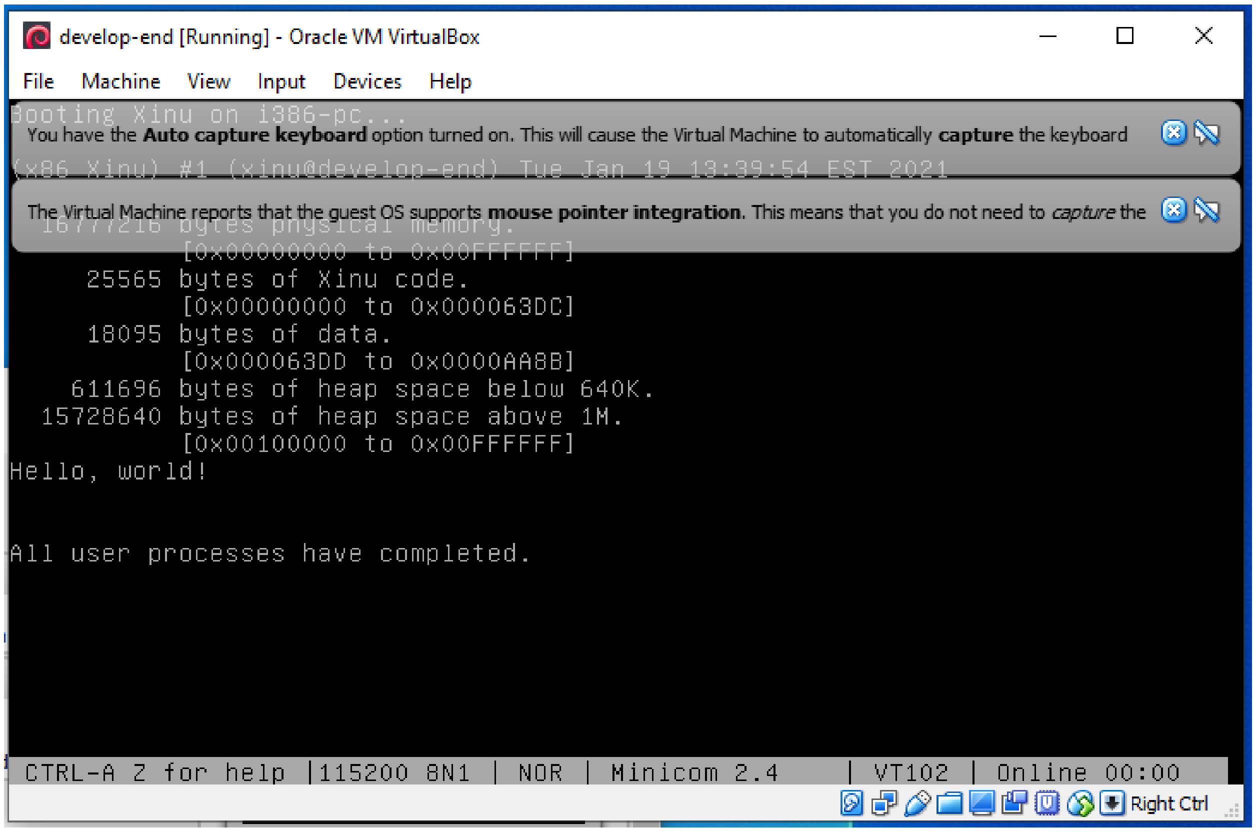
Section C: Run Your First Program

The main.c file was edited to simply print “Hello, world!” on the console. A screenshot of the program and output are below. A backup of the original “xinu-x86-vm” folder was created before any edits were made.



File Edit Selection View Go Run ···
main.c - xinu-x86-vm - Visual Stu...
EXPLORER
XINU-X86-VM
insertd.c
insertd.o
intr.o
intr.S
ioerr.c
ioerr.o
ionull.c
ionull.o
kill.c
main.c
system > main.c > main(void)

```
/* hello.c - main*/
#include <xinu.h>
/* main - just say hello, then exit */
void main(void)
{
printf("Hello, world!\n");
}
```



develop-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard

The Virtual Machine reports that the guest OS supports **mouse pointer integration**. This means that you do not need to *capture* the

```
Booting Xinu on i386-pc...

(x86 Xinu) #1 (xinu@develop-end) Tue Jan 19 13:39:54 EST 2021

  16777216 bytes physical memory.
           [0x00000000 to 0x00FFFFFF]
     25565 bytes of Xinu code.
           [0x00000000 to 0x000063DC]
     18095 bytes of data.
           [0x000063DD to 0x0000AA8B]
    611696 bytes of heap space below 640K.
  15728640 bytes of heap space above 1M.
           [0x00100000 to 0x00FFFFFF]
Hello, world!


All user processes have completed.

 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4     | VT102 | Online 00:00
```

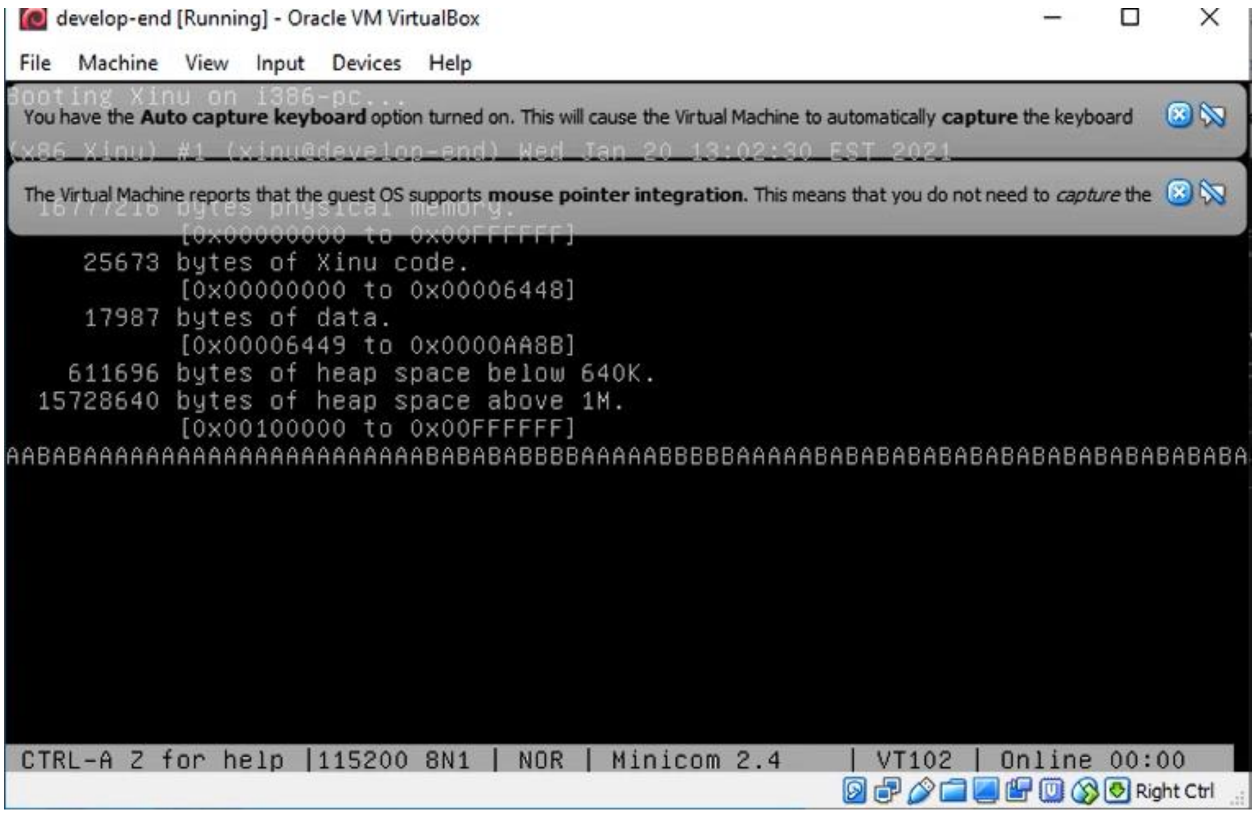
Section D: Type and Run A Second Program To Create Processes

Below is the program written to create two processes “sndA” and “sndB”. SndA willl continually print the character ‘A’, and sndB will continually print the character ‘B’. SndA and sndB have the same priority of 20.

```c
/* main.c - main*/
#include <xinu.h>

void sndA(void), sndB(void);

/* main - Example of creating processes in Xinu */
void main(void)
{
resume(create(sndA, 1024, 20, "p 1", 0));
resume(create(sndB, 1024, 20, "p 2", 0));
}

/* sndA - Repeatedly emit 'A' on console */

void sndA(void)
{
while(1){
putc(CONSOLE, 'A');
}
}
/* sndB - Repeatedly emit 'B' on console */

void sndB(void)
{
while(1){
```

```
putc(CONSOLE, 'B');

}

}
```

A snapshot of the output of the code is below. Note the random switching between the two processes. The pattern is unpredictable. The code is an example of two concurrent processes running the same code. There is context switching occurring between the two processes, and there are no guarantees about the order or rate of execution.


develop-end [Running] - Oracle VM VirtualBox

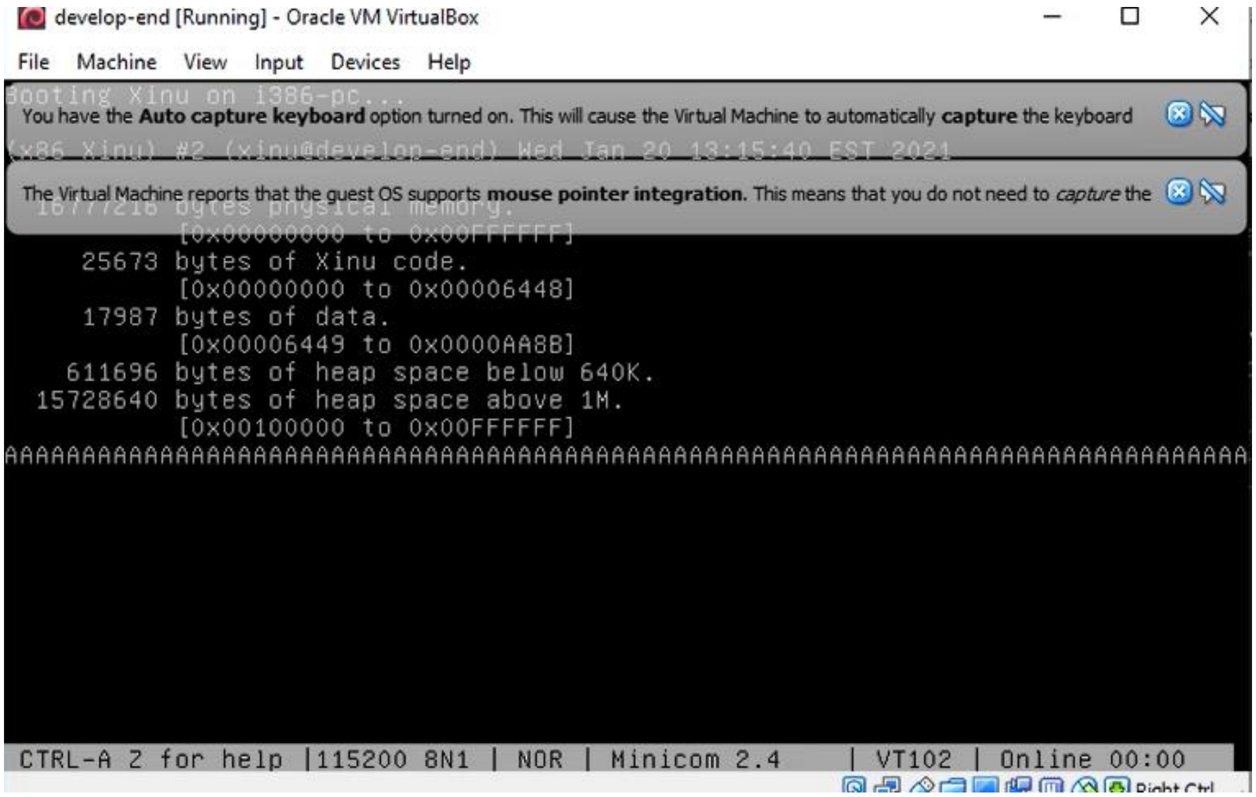
You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard

The Virtual Machine reports that the guest OS supports **mouse pointer integration.** This means that you do not need to *capture* the

```
Booting Xinu on i386-pc...

(x86 Xinu) #1 (xinu@develop-end) Wed Jan 20 13:02:30 EST 2021

  16777216 bytes physical memory.
           [0x00000000 to 0x00FFFFFF]
     25673 bytes of Xinu code.
           [0x00000000 to 0x00006448]
     17987 bytes of data.
           [0x00006449 to 0x0000AA8B]
    611696 bytes of heap space below 640K.
  15728640 bytes of heap space above 1M.
           [0x00100000 to 0x00FFFFFF]
AABABAAAAAAAAAAAAAAAAAAAAAABABABABBBBAAAAABBBBBAAAAABABABABABABABABABABABABABABA
```

```
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4      | VT102 | Online 00:00
```

The priority of sndA was changed from 20 to 40. The output should be mostly ‘A’ characters since the priority of sndA is double that of sndB. A screenshot of the output can be found at the beginning of the next page. Indeed, the output is all ‘A’ characters. While it is still possible to have sndB be the running process, it is far less likely now that its priority is half of sndA.


develop-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

```
Booting Xinu on i386-pc...
You have the Auto capture keyboard option turned on. This will cause the Virtual Machine to automatically capture the keyboard
(x86 Xinu) #2 (xinu@develop-end) Wed Jan 20 13:15:40 EST 2021
The Virtual Machine reports that the guest OS supports mouse pointer integration. This means that you do not need to capture the
  16777216 bytes physical memory.
           [0x00000000 to 0x00FFFFFF]
     25673 bytes of Xinu code.
           [0x00000000 to 0x00006448]
     17987 bytes of data.
           [0x00006449 to 0x0000AA8B]
    611696 bytes of heap space below 640K.
  15728640 bytes of heap space above 1M.
           [0x00100000 to 0x00FFFFFF]
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4     | VT102 | Online 00:00
```

The code is an example of CPU scheduling, which refers to how the OS decides how to choose which process to run from a list of ready processes located in the ready queue. There are various scheduling algorithms, but Xinu will run the highest priority process unless there are multiple highest priority processes. Xinu would use a round-robin approach for this case where each high priority process gets a Quantum time slice. To achieve round-robin scheduling, context switching is used. Context switching is when a process is stopped in the middle of execution, machine state is saved (registers, program status word, program counter, etc.) and the program counter jumps to another process, loads the new process (general purpose registers, program status word, etc.) and begins execution. Context switching gives the appearance to the user that multiple processes are executing simultaneously. On a single threaded uni-core processor, that is not possible, but because of the fast rate of context switching compared to human perception, it appears to execute multiple processes simultaneously.

<u>Analysis of queue.c and main.c</u>

The file “queue.c” begins by declaring the queuetab, which is an array of the qentry (queue entry) data type. Thus, the size of the array is equal to the number of queue entries (NQENT).

The definition of NQENT from queue.h is as follows: NQENT = (NPROC + 4 + NSEM + NSEM). NPROC represents the number of processes. Each process is allocated an entry slot in the queue table array. The number “4” represents two slots (head and tail) for the ready list and two slots (head and tail) for the sleep list. “NSEM + NSEM” (NSEM*2) represents slots for the head and tail of each semaphore implemented in the OS.

Enqueue() takes two arguments: “pid” and “q”. The “pid” is the process ID of the process to insert into the queue, which is identified by the other argument “q” (the queue ID). The local variables “tail” and “prev” are declared and represent the tail and previous node indexes in the linked list (queue). The validity of the arguments is checked using the functions “isbadqid(q)” and “isbadpid(pid)”. If either argument is invalid, the enqueue function will return SYSERR. If the arguments are valid, the local variable “tail” is assigned as the queuetail(q). The queue ID is really the head of the queue, and the tail of the queue is the very next entry in the queuetab. Therefore, the queuetail is defined as q+1. The local variable “prev” is assigned to queuetab[tail].prev, which is the last queue entry before the tail. The linked list is then rearranged so that the new process “queuetab[pid]” will be inserted at the tail-end of the queue just before the tail. The function ends by returning the pid of the process that was successfully inserted into the queue.

Dequeue() takes one argument “q” which is the queue ID of the queue to perform dequeue(). If the queue ID is invalid, SYSERR is returned. Likewise, if the queue is empty, EMPTY is returned. If neither of those conditions is met then the function removes the first process from the queue at the beginning (head-end) of the queue, sets its qprev and qnext pointers to EMPTY, and returns the pid of the process.

The file “main.c” takes in command line arguments. Integer “argc” represents the number of arguments and character type “**argv” is a double pointer to those arguments. Unsigned 32-bit integer “retval” (return value) is declared. The shell process is created and put onto the ready queue (resumed). The recvclr() function is called to clear out any message in the current process. A while-loop is entered where the process will wait for a message by calling “receive()”. The process will be blocked waiting on the message. The process state will be “PR_RECV” and then it will call “resched()” to perform a context switch to another process. When the message is sent, the process will be placed back on the ready queue and eventually run. The kprintf function is called to print: “\n\n\rMain process recreating shell\n\n\r”. The “shell” process is then recreated using the “resume(create(shell, 4096, 1, ”shell”, 1, CONSOLE))” system call. Because “receive()” and “resume(create)” are in an infinite while-loop, the shell process will be continually created, blocking for a message, and calling kprintf.

<u>Analysis of 2019 and 2020 Xinu Source Code</u>

After viewing the two versions of the Xinu source code, the following differences were prevalent. After navigating to each version’s respective “system” folder, the 2020 version does not contain object files (.o extension), while the 2019 version does. Object files are typically created by the compiler using the source files (.c extension). The 2020 version also contains more source files (.c) such as clkhandler.c, exit.c, gettime.c, getticks.c, getutime.c, and a binary file clkdisp.S. The 2020 version also contains a folder that is not in the 2019 version called “net”. The net folder contains various source files dealing with communications over the Ethernet port such as udp.c (send/receive UDP packets) and ip.c (send/receive IP packets). Net.c manages the memory for these using the buffer-pool “netbufpool” (network buffer pool).

Reviewing the DESCRIPTION file located in each version of Xinu, there are differences between the two versions. First, the tty driver in the 2020 version uses port-mapped I/O in place of the memory-mapped I/O used on the Galileo hardware platform. Also, the 2020 version’s startup sequence (start.S) differs from the Galileo startup sequence, and the 2020 version removed the platform initialization used by the Galileo (platinit.c).