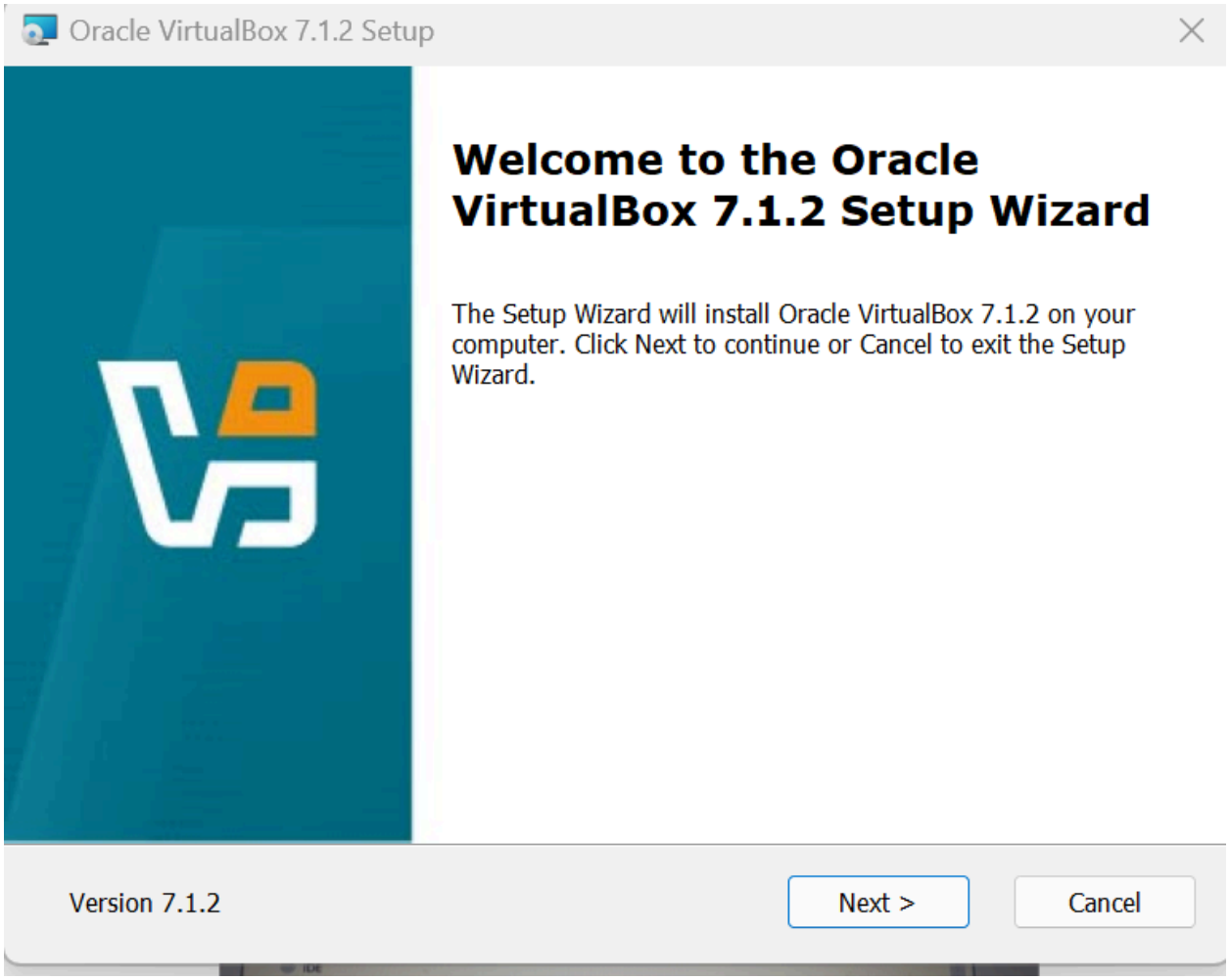
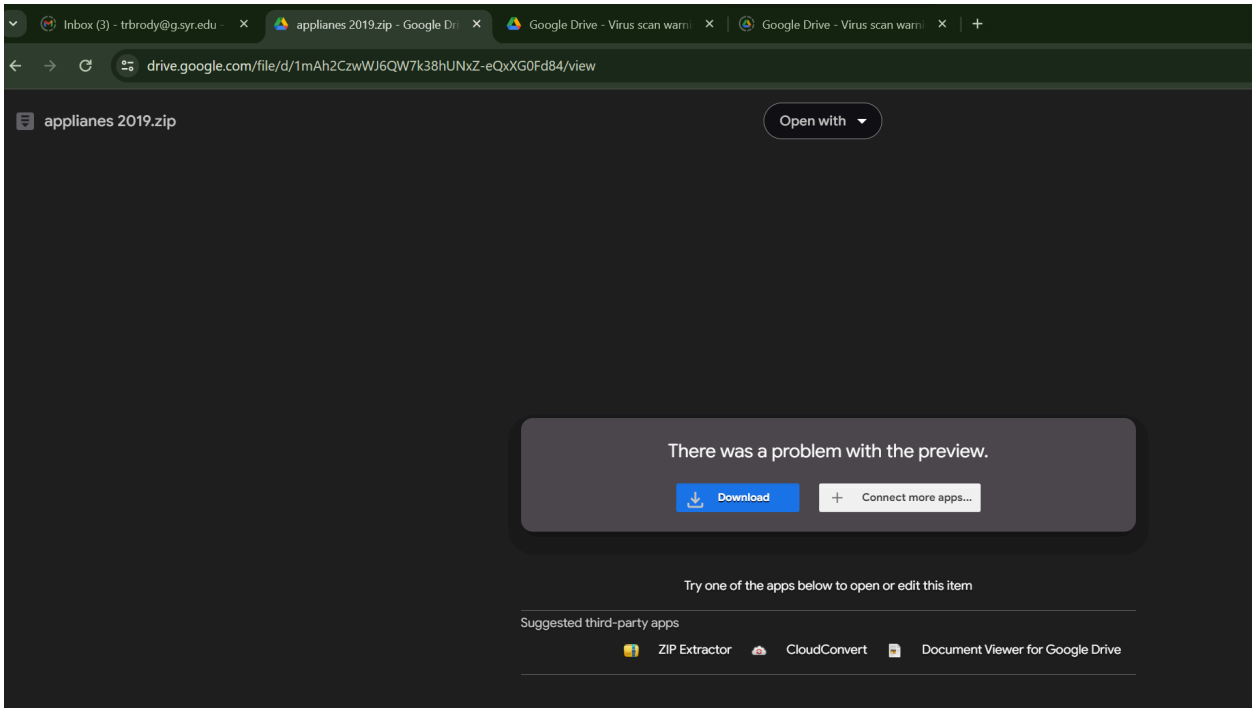
Tristan Brody
2024-1002:14195 CIS-657
October 13 2024
Lab 1 Findings
Syracuse University

First, I installed VirtualBox 7.1.2 for Windows (most recent version available online), selecting configurations to match figures 1-3 in the lab instructions

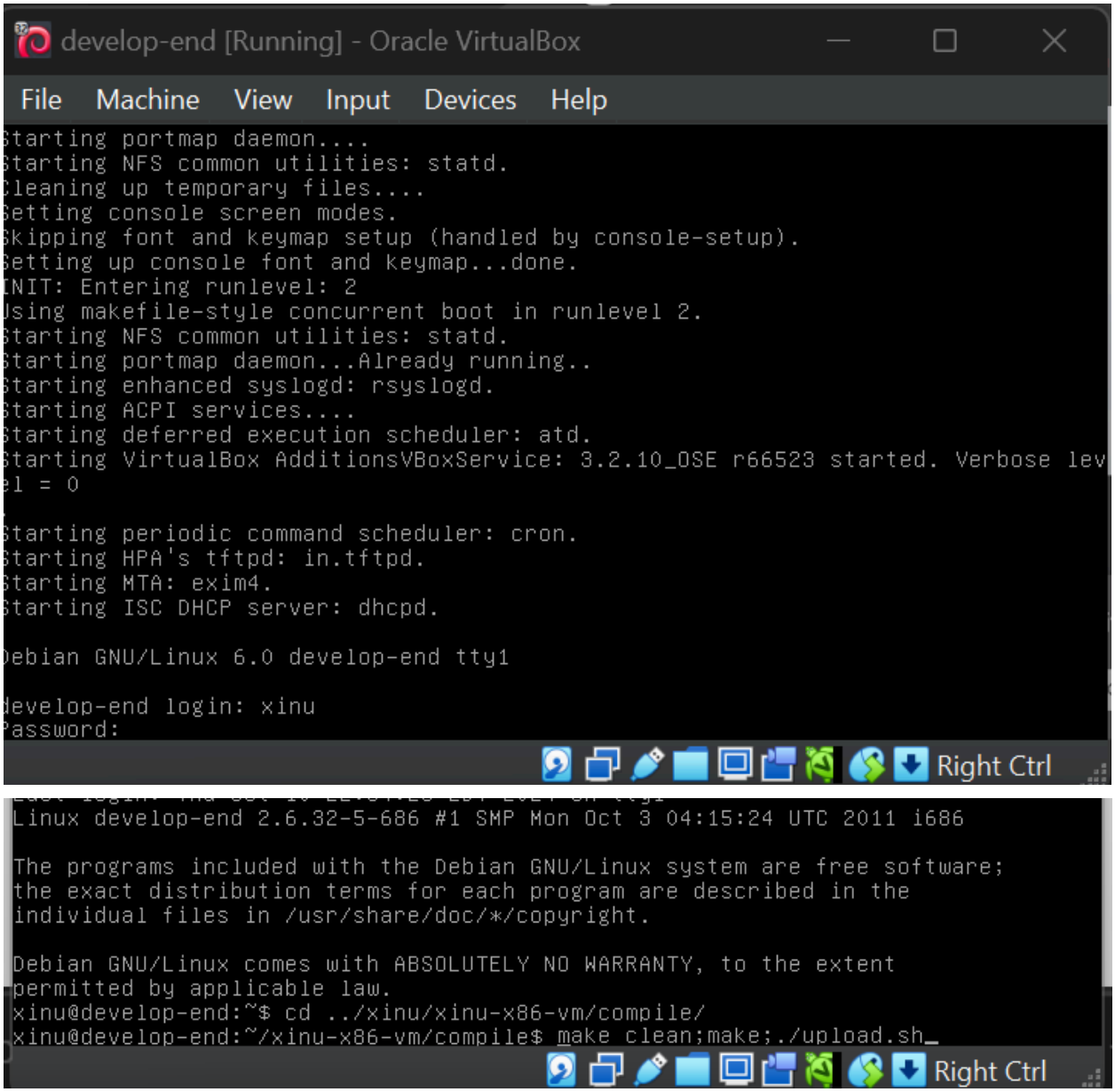


Next, I downloaded the Xinu 2019 appliances

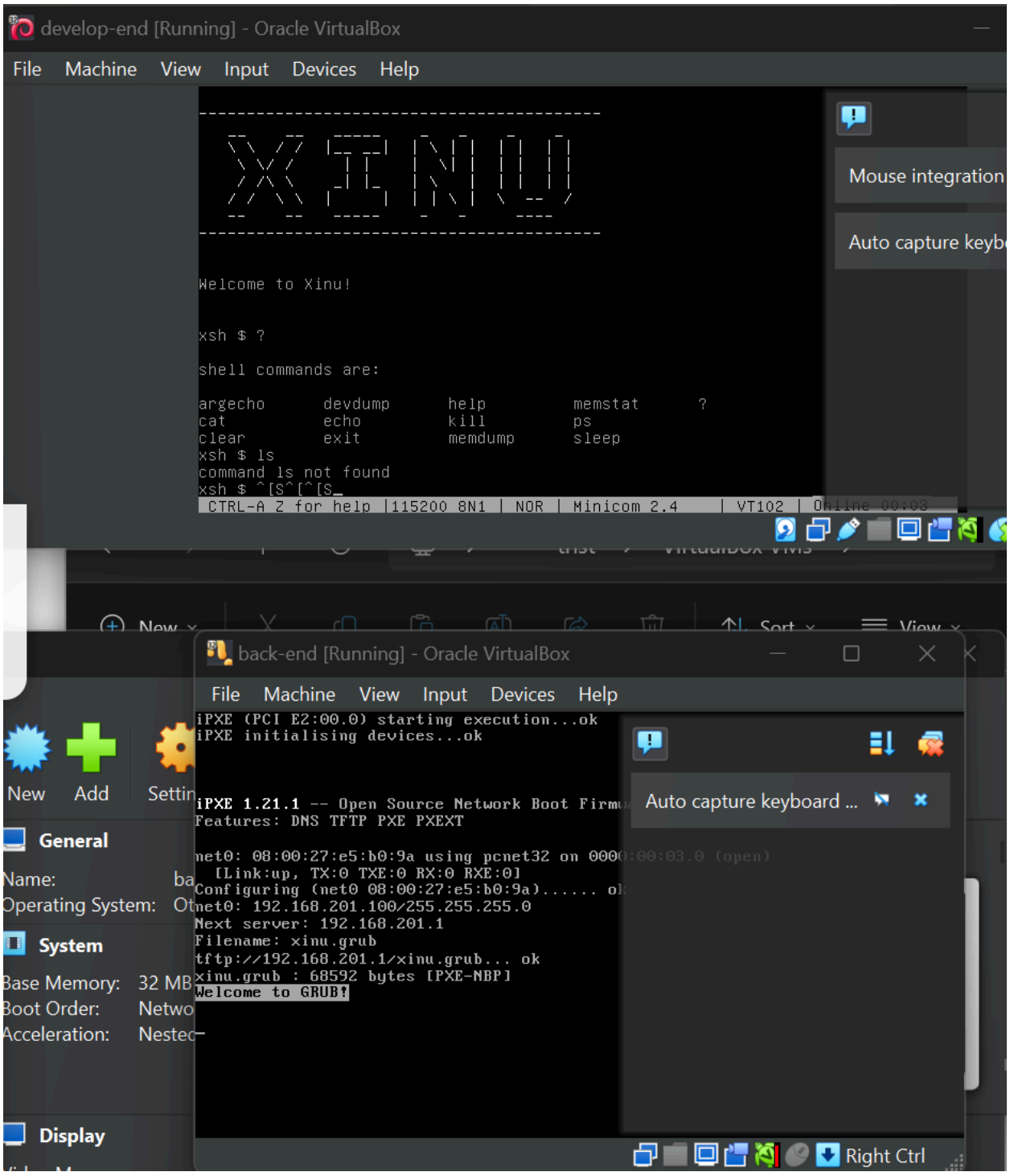


I imported the develop-end and back-end to VirtualBox, paying close attention to the special configurations called out in figures 9 and 10 in the Lab 1 instructions.

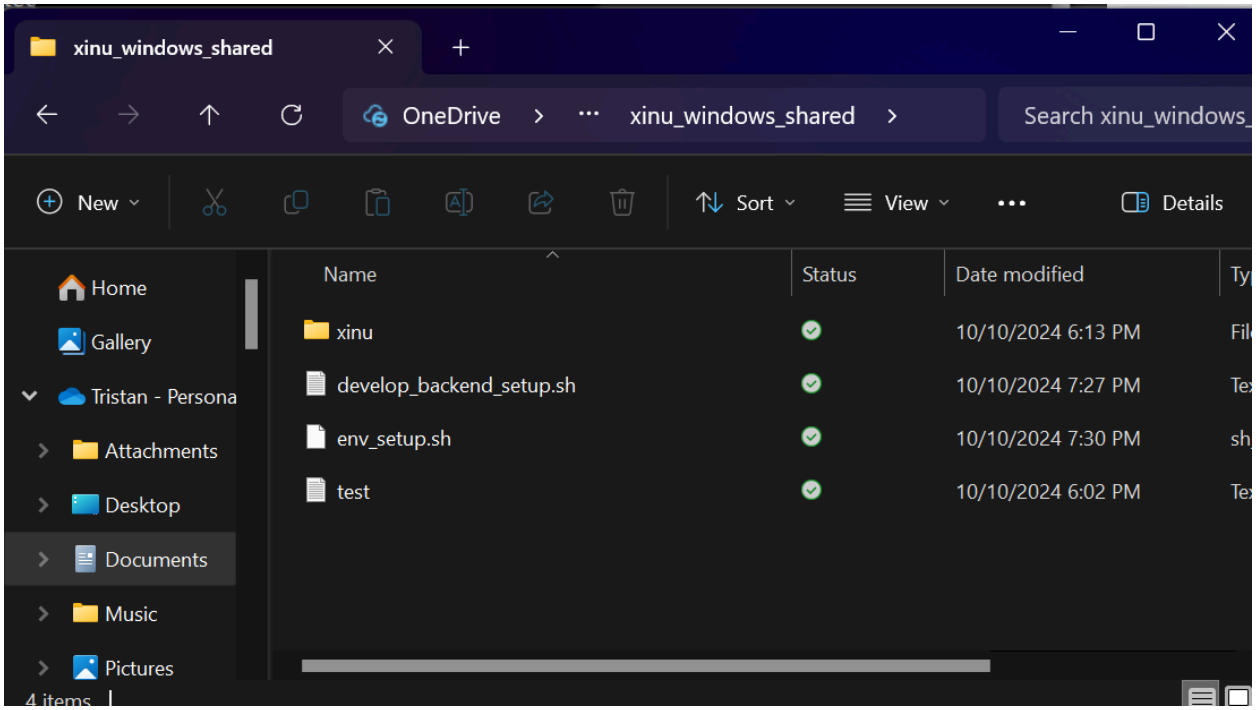
To set up Xinu 2019 within VirtualBox, I followed the Lab 1 instructions to compile and upload the xinu-x86-vm directory



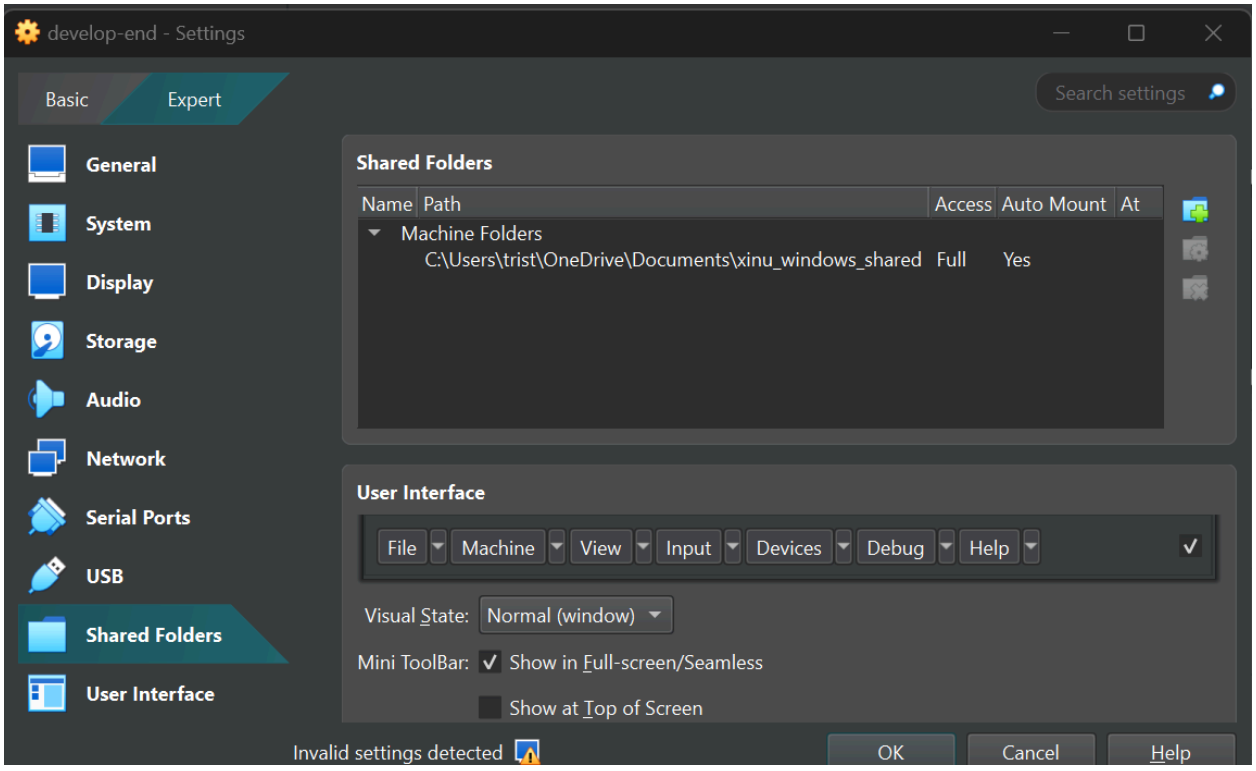
After running sudo-minicom and opening the back-end machine, I was able to see the Xinu prompt and enter ? as shown below:



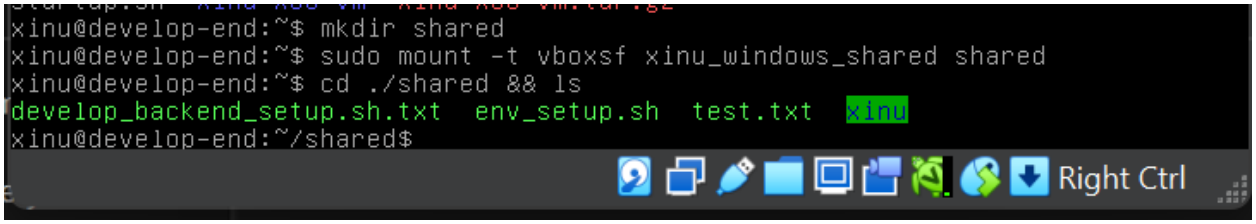
In order to easily access and edit files in the xinu folder, I followed Lab 1's instructions on creating and mounting a shared folder in Windows 11 for access within develop-end:



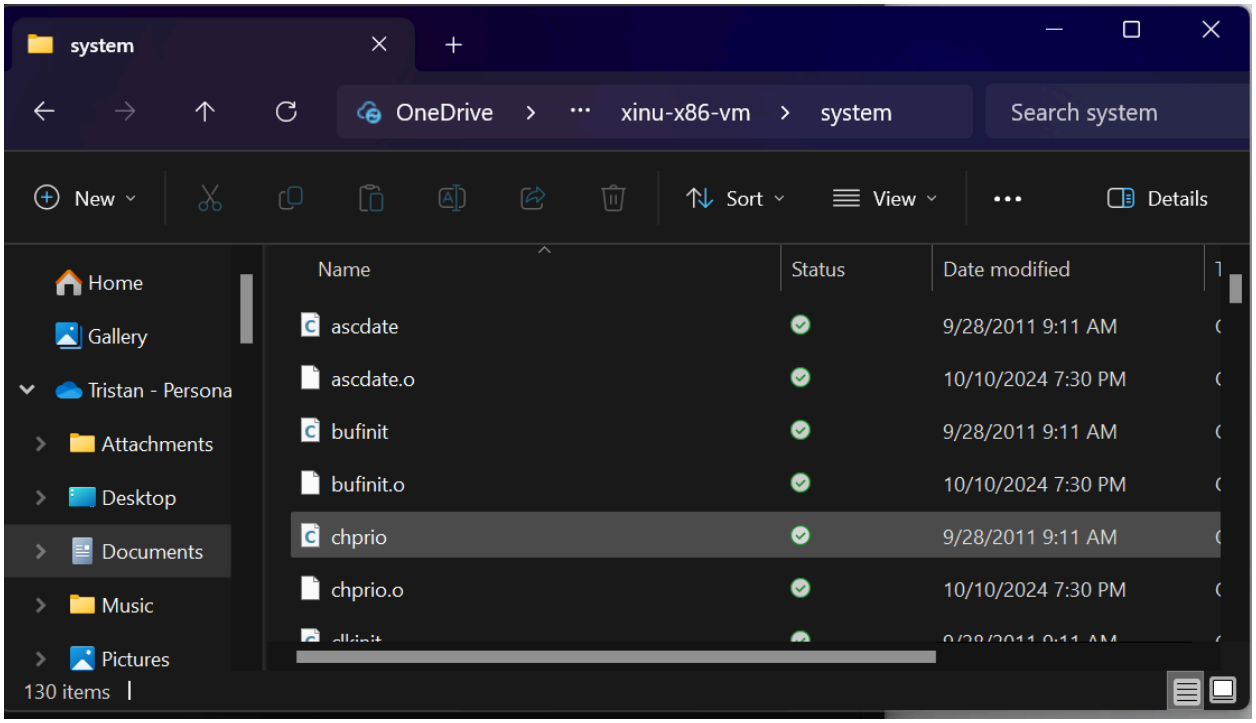
Settings for shared folder in develop-end:



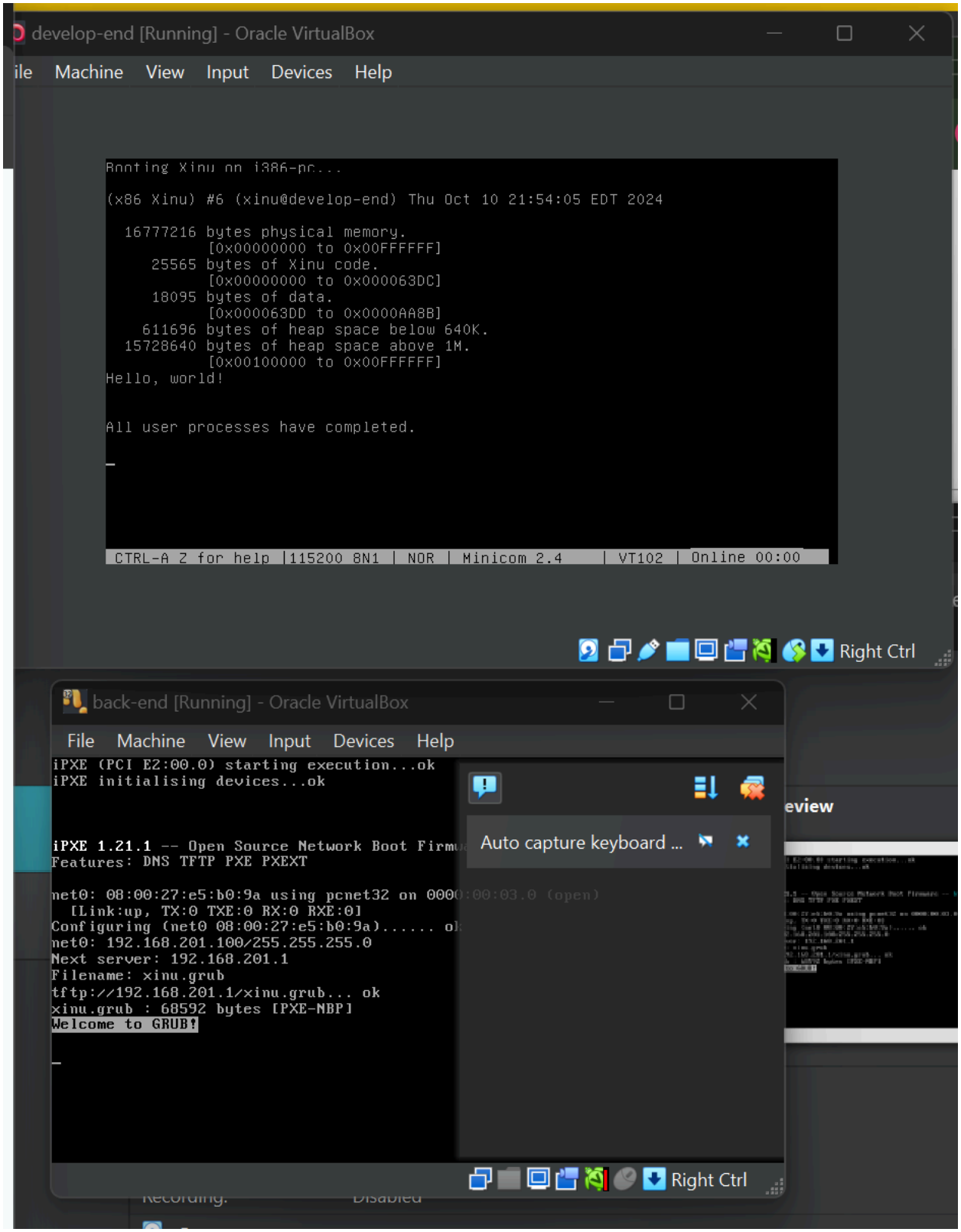
Mounting the shared Windows 11 folder + adding xinu -



Ability to access shared files from xinu folder within shared Windows folder:



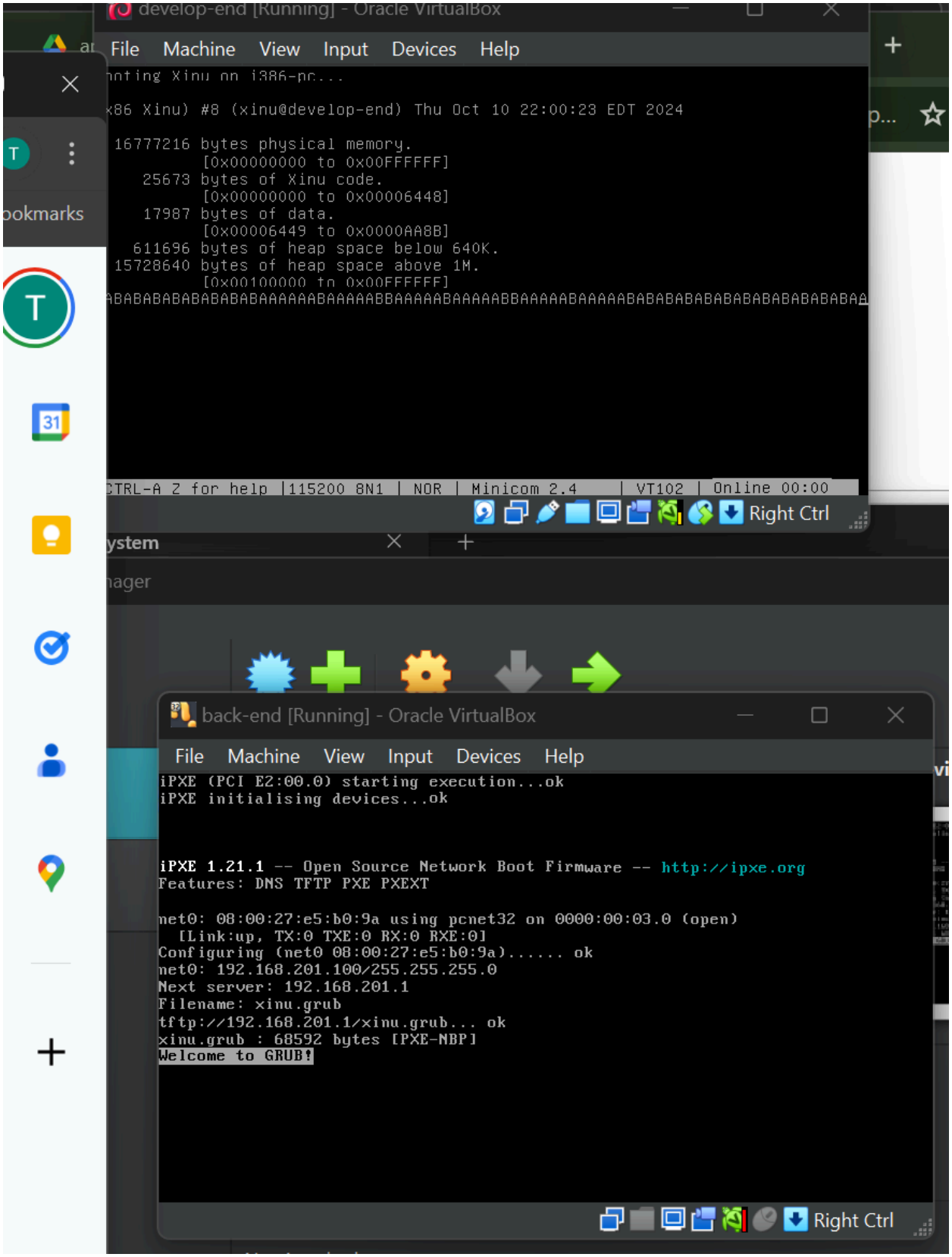
Next, I edited the main.c to match the code in Section C. Below was the output after booting xinu:



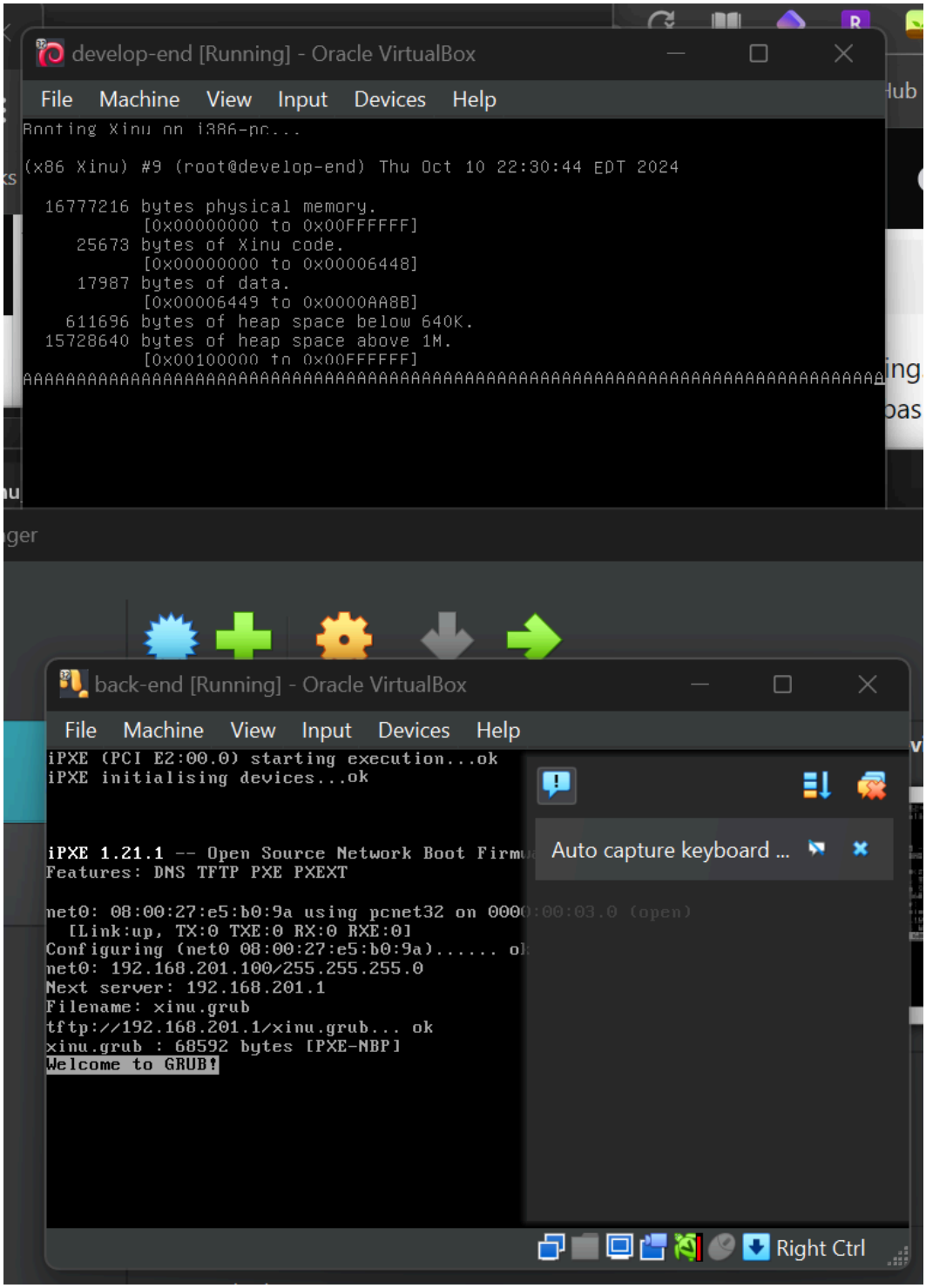
Section D - running main.c as
#include <xinu.h>

```
void sndA(void), sndB(void);
/*------------------------------------------------------------------------
* main - Example of creating processes in Xinu
*------------------------------------------------------------------------*/
void main(void)
{
resume(create( sndA, 1024, 20, "process 1", 0) );
resume( create(sndB, 1024, 20, "process 2", 0) );
}
/*------------------------------------------------------------------------
sndA - Repeatedly emit ’A’ on the console without terminating */
void sndA(void)
{
while( 1 )
putc(CONSOLE, 'A');
}
/*------------------------------------------------------------------------
sndB - Repeatedly emit ’B’ on the console without terminating*/
void sndB(void)
{
while( 1 )
putc(CONSOLE, 'BX');
}
```

Output in xinu in screenshot below -


develop-end [Running] - Oracle VirtualBox
File Machine View Input Devices Help
nting Xinu on i386-pc...
x86 Xinu) #8 (xinu@develop-end) Thu Oct 10 22:00:23 EDT 2024
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]
ABABABABABABABABAAAAAABAAAAABBAAAAABAAAAABBAAAAABAAAAABABABABABABABABABABABAA
CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4 | VT102 | Online 00:00
Right Ctrl
back-end [Running] - Oracle VirtualBox
File Machine View Input Devices Help
iPXE (PCI E2:00.0) starting execution...ok
iPXE initialising devices...ok
iPXE 1.21.1 -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS TFTP PXE PXEXT
net0: 08:00:27:e5:b0:9a using pcnet32 on 0000:00:03.0 (open)
[Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (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
xinu.grub : 68592 bytes [PXE-NBP]
Welcome to GRUB!
Right Ctrl

Upon updating the priority of processes in main.c, I would expect to only see As since we're changing the priority of sndA to be greater than sndB, and sndA runs forever in an infinite loop. Screenshot of output in xinu below:



Summary of main.c and queue.h files in the Xinu system files:

The original main.c file in the Xinu system folder creates a process for the Xinu shell which calls the shell method. Next, it calls the recvclr function, which handles clearing and returning any

messages from the current running process. Then, it enters an infinite loop that waits to receive a message confirming the shell has exited, in which case it recreates it.

The original queue.h file in the Xinu system folder defines the interface for implementing a queue for use with the Xinu operating system. It defines constants for key aspects of the queue data structure, including the head of the queue, the tail, whether or not the queue is empty, and so on. It also includes the signatures for functions you would need to implement in a queue.c file, including enqueue and dequeue for FIFO queues and insert for priority-based queues.