# CS3502 Project 01-A Fall 2022

# Setting up the OS Development Environment

# Learning Objectives:

## The purpose of this assignment is to continue the development examples of setting up an Operating System kernel.

## Submissions will be placed here and they will contribute to the total assignment grade of 100 points (+100 points for Bonus section).

## This is the kernel build process and constitutes the Part I of the Kernel Build assignment. This assignment in D2L contains four parts that will complete the Project Assignment:

## Part I: Set up Environment – install NASM,QEMU, GCC, GDB in WSL, VM or native installation.

## Part II: Get to a working shell prompt.

**PART I**

See the Power Point file in D2L Assignment1.ppt and complete the instructions. Be sure to contact me in class, office hours (or even outside of office hours) if you have any difficulties – and… you ***will*** have difficulties. Contact me early and often.

NOTE: 4Gb is probably the minimum size for Ubuntu16 with all the sudo apt-get installs done but you may want more room (i.e. 20Gb if you have sufficient space). ALSO be sure to install a FIXED size, this will save performance issues later.

Future assignment kernel development sections will depend and build on completing this one.

The files are in D2L and for convenience are (hopefully) embedded here.

## 

**10 points**

**PART II – The debugger and world’s simplest shell prompt**

**See the very bottom of assignment for discussion questions:**

You will need NASM, GCC, QEMU, GDB. The latter three are already installed but from the bash (Windows Subsystem for Linux) or x-term (Ubuntu 16 Virtual Machine) run the following:

sudo apt-get update

sudo apt-get upgrade

sudo apt-get dist-upgrade

sudo apt-get install gdb

## As before retrieve the file kernel.zip.gz from D2L and extract to your home directory or another place. The file is included here but if it does not show up properly retrieve from D2L as well.

## In WSL you may have to copy from Downloads by executing the command:

## cd ~

## cp /mnt/c/Users/<your ID>/Downloads/kernel.zip.gz .

## In the VM though it may be easier to just access D2L from Firefox88 (ignore warnings about compatibility it still runs today) and down load the files directly in the VM in D2L. Otherwise you may have to use sftp to connect to the host machine. See me for any issues and we can go through this part together.

## 

## E.g. run:

## cd ~

## tar -xvf kernel.tar.gz

## This will create eight directories under the directory a4:

## For section 01\_gdb, it will be longer than the others, the rest are just make… screenshot --- for the most part. **RED** is the section assignment, **GREEN** asks specifically for submission information.

## **Part II – Connecting the gnu debugger, Video and Interrupts**

## **01\_gdb - Simple kernel continuation for "vga" graphic manipulation and connecting to debugger**

## Change to the directory **01\_gdb** and run the command:

## make clean

## make

## make run (if you are using qemu -vga std)

## make curses (if you have trouble with qemu -vga std and run qemu -curses)

## You should have a window that looks similar to this.

## 

## Navigate to the kernel directory under 01\_gdb and edit the file kernel.c to insert the following after the line ” \*video\_memory = ‘X’;”. Don’t forget to make sure the brace } at the end doesn’t get overwritten when you add these lines:

## If you are running Windows Subsystem for Linux you can run notepad.exe from a bash prompt e.g. if your files are extracted under your home directory:

## cd ~ --- will take you to home directory

## cd a4

## cd 01\_gdb

## cd kernel

## **If in WSL run:**

## /mnt/c/Windows/notepad.exe kernel.c

## If you are familiar with using “vi” or other editors you can use them instead.

## **If in ubuntu16 Virtual Machine the native text editor is gedit:**

## gedit kernel.c

## **unsigned char a;**

## **unsigned char b;**

## **a = 0;**

## **b = 0;**

## **for( int i = 1; i < 24; ++i) {**

## **for( int j = 0; j < 80; ++j) {**

## **\*(video\_memory + i\*160 + j\*2) = '.';**

## **}**

## **}**

## **\*video\_memory = 'Z';**

## **for( int i = 10; i < 20; ++i) {**

## **for( int j = 50; j < 60; ++j) {**

## **\*(video\_memory + i\*160 + j\*2) = 'X';**

## **}**

## **}**

## **while( 1) {**

## **\*(video\_memory+0) = b;**

## **\*(video\_memory+1) = ++a;**

## **b = b + a/255;**

## **}**

## Once done editing, save the file, in the shell prompt rebuild the edited:

## cd .. ###This should take you to up one level back to **a4/01\_gdb** directory

## make clean

## make

## make curses --OR-- make run

## **Take a screenshot of the result and submit it. Exit this instance, make sure you are in the directory 01\_gdb and run the command:**

## make clean

## make debug

## Then, open a **second x-term/bash shell**, navigate to the directory **01\_gdb** and connect the debugger to the running program in the other shell:

## make gdb

## You will see two windows like these (debugger and connected program):

## 

## In the gdb window at the (gdb) prompt type:

## list 0

## jump \_start

## break \_start

## info break

## jump \_start

## step 1

## step 2

## step 10

## 

## You may have to hit Ctrl-C if the prompt disappears.

## NOTE: On my office computer gdb ran fine, but on my home computer gdb connected but did not control the program output as it did in the office. Depending on your setup, the second window may respond differently.

## **Show a screenshot of the gdb window of the results from the above commands.**

## ***NOTE: If you only allocated 1 CPU to the Ubuntu 16 Virtual Machine this program will freeze your VM. If you only have one CPU allocated go ahead and run it but you’ll have to reset it from the VM Window to stop it.***

## **02\_driver - Simple video driver implementation**

## Change directory to **02\_driver** and run:

## make clean

## Make run (or curses)

## Modify the code so that the “X” that is printed after the text is now 2 rows down and 10 characters (columns) further over instead so that:

## Loading kernel into memory.

## X

## Becomes:

## Loading kernel into memory.

## X

## Instead.

## **Print and submit a screenshot of the result.**

## **03\_strings - Writing more than one character at a function call**

## Change directory to **03\_string** and run:

## make clean

## Make run (or curses)

## **Print and submit a screenshot of the result.**

## **04\_scroll - Writing by lines and scrolling out of view**

## Change directory to **04\_scroll** and run:

## make clean

## Make run (or curses)

## Modify the kernel.c code by changing i<24 value to i<30 and slowing down the scroll by adding the following for loop after **kprint\_at(str, 0, i):**

## **for( int j = 0; j < 100000000; ++j) {**

## **;**

## **}**

## **Print and submit a screenshot of the result.**

## **05\_basic\_int - Building a simple Interrrupt table**

## Change directory to **05\_basic\_int** and run:

## make clean

## Make run (or curses)

## Modify the message “Breakpoint” to print instead “Breakpoint, Halted”

## **Print and submit a screenshot of the result.**

## **06\_irq - Filling in the interrupt request handlers**

## Change directory to **06\_irq** and run:

## make clean

## Make run (or curses)

## Modify the result to print “Z” before the “S”.

## **Print and submit a screenshot of the result.**

## **07\_timers - Implementing a simple timer system**

## Change directory to **07\_timers** and run:

## make clean

## Make run (or curses)

## Change the value of 50 in init\_timer(50) to 1000.

## Change the value of 50 in init\_timer(50) to 10.

## **Print and submit a screenshot of the result.**

## **08\_shell - Implementing the world's simplest shell**

## Change directory to **08\_shell** and run:

## make clean

## Make run (or curses)

## Enter the following shell commands:

## echo

## END

## dir

## end

## **Print and submit a screenshot of the result.**

**15 points**

**You are now ready to begin kernel design of the principle modules of Kernel design: The CPU Scheduler, Memory Manager and File System Implementation**

## **QUESTIONS**

## 01\_gdb – How does the while(1) change the output when \*video\_memory and \*video\_memory +1 are updated with the **a** and **b** values?

## 01\_gdb – What information do you receive from running the debugger step commands while watching the kernel run in the other terminal?

## 02\_driver – In kernel.c why is offset\_from\_vga assigned position multiplied by \*2?

## 03\_strings – Where is the function kprint\_at stored in the 03\_strings build?

## 03\_strings – kprint\_at( “X”, 1, 6) what is each argument representing here?

## 04\_scroll – After modifying to slow scroll down and increasing number of i loops; was the result what you expected, what do you think is happening?

## 05\_basic\_int What is the directive in kernel.c: \_\_asm\_\_ \_\_volatile\_\_("int $2"); doing?

## 06\_irq - Where is the routine that prints the S when an interrupt is received in 06\_irq?

## 07\_timers – What is the value 10/50/1000 doing to the timer result?

## 08\_shell – Why does typing “END” produce the output “?E?N?D” ?

## **For the interested, see the guide website I used as a template for this: https://github.com/cfenollosa/os-tutorial**

**Submission Guidelines:**

* No handwritten submission is accepted, always submit within this document file.
* You may include your freehand drawing/image and handwritten scans in the submission. However, the writing and images must be clearly legible.  If I notice them as illegible to me prior to the due date I will let you know to provide clearer version.  This will not change the due date.  It is best to present non-handwritten submissions, generally, as is done in the professional setting.
* Show all work/calculations.
* **Please complete your entire work in a single Word Document and Save the file as: yournetid\_CS3502\_Project01.docx; otherwise, 25% points will be deducted. For instance, I would save my file as: ogarcia5\_CS3502\_Assignment01.docx. Then upload your file in D2L.**
* Please observe the submission due date and time. After the due date there is a 50% penalty for the next 24 hours. Any submission after 24 hours of the due date will be graded at 0%.
* If you include a reference or an image taken from other sources, please cite them appropriately. APA is preferred, but cite them so they can be found.
* If you resubmit, please make sure to attach the file again. Your latest submission before the due date will be graded.
* **There is NO extended deadline for this assignment.**