

**CS 695 / SWE 660 Software Engineering for Real-Time Embedded Systems**

# **QEMU LAB ASSIGNMENT**

**Group 6 (TUE/THUR)**



## **Group 6 (Tue/Thu)**

### **Group Members:**

- **Poorvi Lakkadi (G01389351)**
- **Prabath Reddy Sagili Venkata (G01393364)**
- **Pranitha Kakumanu (G01379534)**
- **Sai Hruthik Karumanchi (G01352466)**
- **Sai Sujith Reddy Ravula (G01409395)**
- **Sri Harish Jayaram (G01393332)**

# INTRODUCTION

- In this lab assignment, we understand the powerful capabilities of QEMU, a versatile emulator that allows us to run various Linux environments on different hardware platforms.
- As part of the assignment, we work with three distinct platforms: x86\_64, Raspberry Pi 3B, and BeagleBone Black, and understand how to set up, configure, and run emulated systems.
- Along the way, we'll demonstrate a "Hello World" program that retrieves system information, giving us a view of what's possible in these emulated environments.

# ENVIRONMENT SETUP

Given that the host computer is running on Windows, we have established Debian-based Linux systems (specifically, Ubuntu or Linux Mint) using VirtualBox for our lab assignment.

These following two commands install QEMU and its dependencies, enabling the user to run virtualized ARM64 (aarch64) and other system architectures on their Debian-based Linux system.

**`sudo apt-get install -y qemu-system-aarch64`**


This command installs the QEMU emulator for the ARM64 architecture on a Debian-based Linux system, automatically confirming any prompts with the "-y" flag.

**`sudo apt install qemu qemu-kvm`**

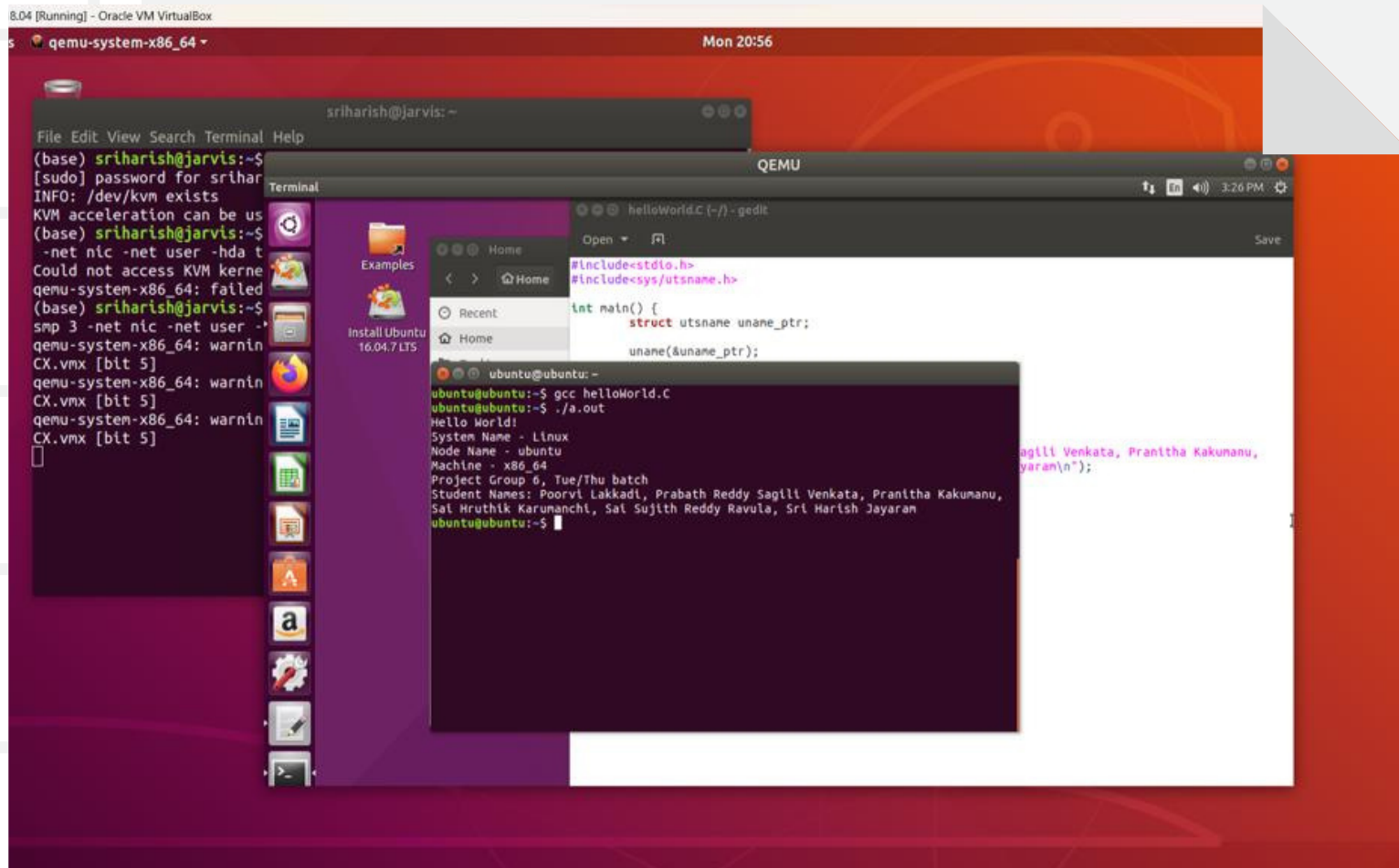
This command installs the QEMU emulator and the QEMU-KVM package on a Linux system using the APT package manager.



# EMULATING X86\_64 LINUX WITH QEMU

- Use case: Emulating Linux on an x86\_64 platform.
  - Command: `qemu-system-x86_64`
  - Specify memory allocation with `-m` option.
  - Create a virtual disk with `qemu-img create`.
  - Boot using a lightweight Linux live image.
  - Write a C program and execute the file.
- 

# STEPS




- Generate a virtual disk image, named "test-image.img," in the qcow2 format with a size of 10GB using the following command:
- Download a Linux ISO image from the official repository and place it in your working directory.
- Boot the Linux ISO image, connecting it to the virtual disk image, using the following command:

***qemu-system-x86\_64 -m 1024 -boot d -enable-kvm -smp 3 -net nic -net user -hda test-image.img -cdrom ubuntu-16.04.iso***

- Open a text editor (e.g., nano) to create and edit a C program named "hello\_world.c" using the following command:
- Inside the text editor, write the complete C code for your program. Save the file and exit the text editor. Then, compile the program using the gcc compiler:
- Execute the complied program with the following command:

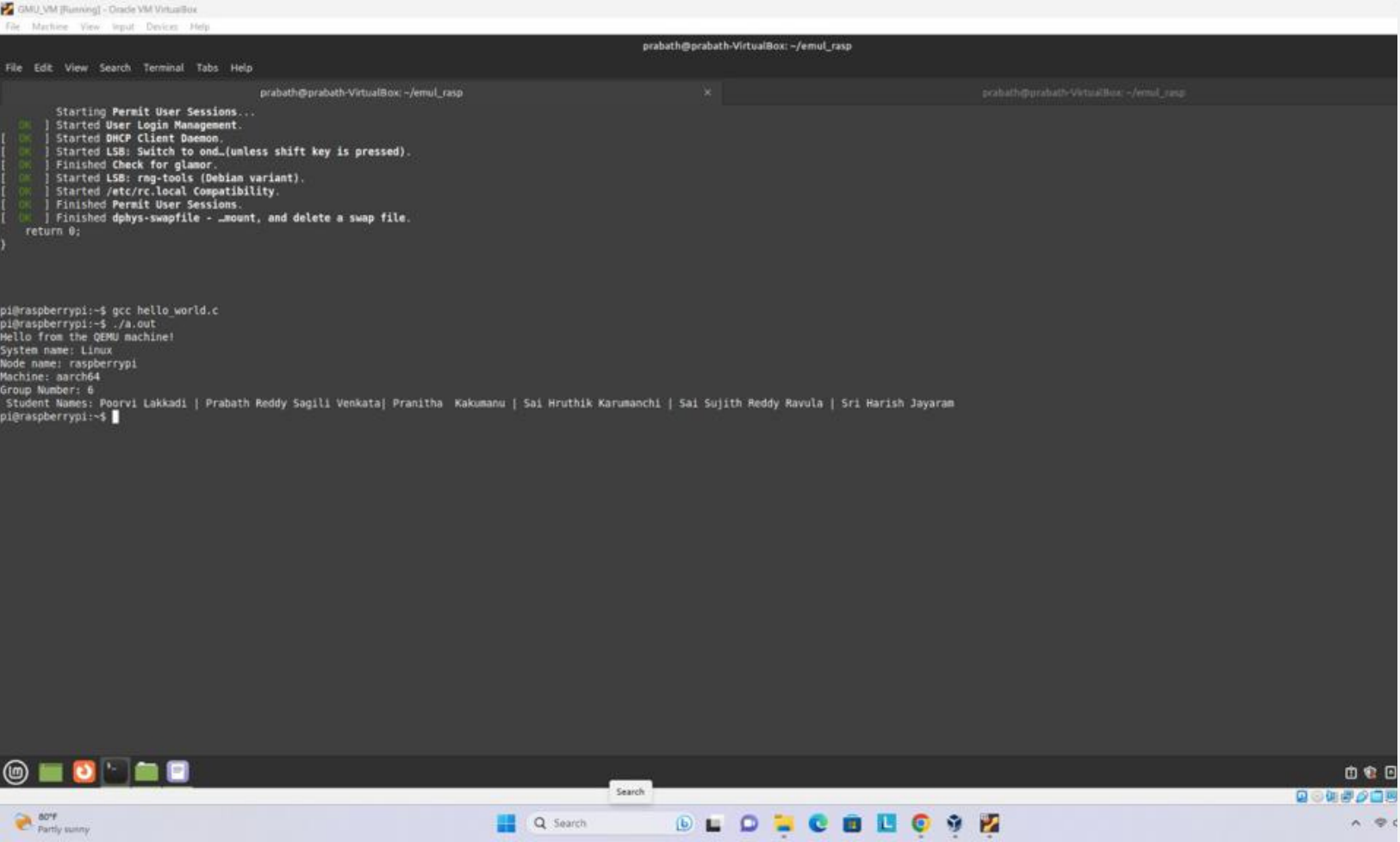


# EMULATING RASPBERRY PI 3B WITH QEMU

- Use case: Creating a Raspberry Pi 3B emulator.
  - Command: `qemu-system-aarch64` with specific machine.
  - Find the appropriate machine name with `qemu-system-aarch64 --machine help`.
  - Implement the "Hello World" program on the emulated Raspberry Pi.
  - Testing ARM architecture emulation.
- 



# STEPS



- Mount the first partition of the image by calculating the offset (Boot Start \* Sector size, in this case, 8192 \* 512 = 4194304 bytes) and using the following commands:

***sudo mkdir /mnt/image***

***sudo mount -o loop,offset=4194304 ./2023-05-03-raspbios-bullseye-arm64.img /mnt/image/***

- List the contents of the mounted directory using the ls command:

***ls -ls /mnt/image/***

- Copy the Kernel and Device Tree files from the mounted image to your home directory:

***cp /mnt/image/bcm2710-rpi-3-b-plus.dtb ~***

***cp /mnt/image/kernel8.img ~***

- Generate an encrypted password hash for SSH using the openssl passwd -6 command and make note of the returned hash. Then, enable SSH by creating an empty "userconf" file and a "ssh" file:

***openssl passwd -6***

***echo 'pi:<your\_hash>' | sudo tee /mnt/image/userconf***

***sudo touch /mnt/image/ssh***

- Resize the image to 8GB so that it aligns with a power of 2:

***qemu-img resize ./2023-05-03-raspbios-bullseye-arm64.img 8G***

- Run QEMU with the specified configuration, including the Raspberry Pi 3B machine, CPU settings, memory allocation, kernel, and image paths:

***qemu-system-aarch64 -machine raspi3b -cpu cortex-a72 -nographic -dtb bcm2710-rpi-3-b-plus.dtb -m 1G -smp 4 -kernel***

***kernel8.img -sd 2023-05-03-raspbios-bullseye-arm64.img -append "rw earlyprintk loglevel=8 console=ttyAMA0,115200***

***dwc\_otg.lpm\_enable=0 root=/dev/mmcblk0p2 rootdelay=1" -device usb-net,netdev=net0 -netdev user,id=net0,hostfwd=tcp::2222-:22***

- Once the Raspberry Pi emulation opens, enter the username and password to log in. Then, execute your hello\_world program as follows:

***sudo nano hello\_world.c***

- Type the entire C code, save the file, compile it, and run it:


***gcc hello\_world.c***

***./a.out***



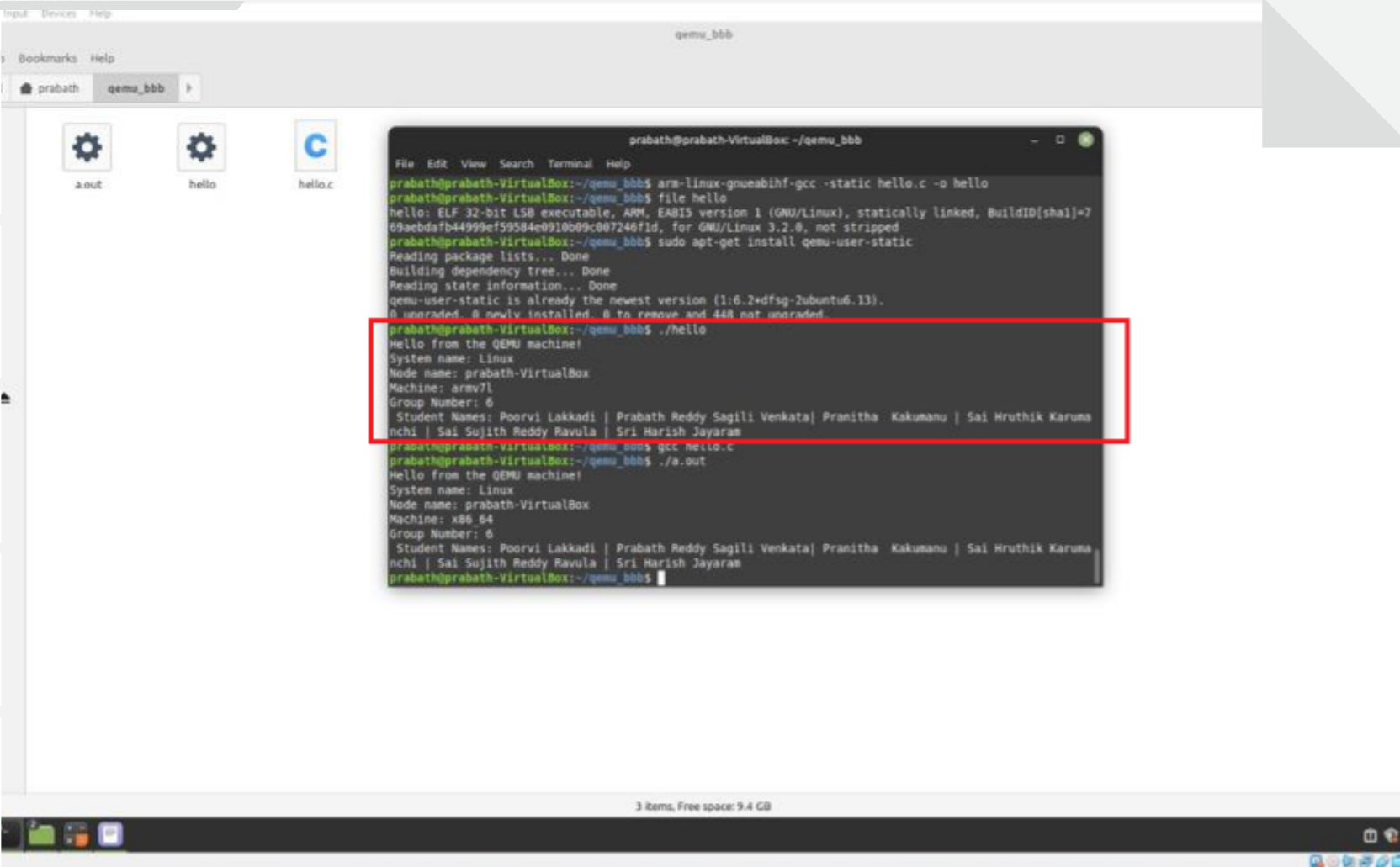


# **EMULATING BEAGLEBONE BLACK WITH QEMU**

- Use case: Creating an emulator for BeagleBone Black.
  - Machine choice: realview-pb-a8 or armhf.
  - Implement the Hello World program.
- 

# STEPS

- Ensure that the APT cache recognizes the cross-build essential tools for ARM by running the following command:  
***apt-cache search cross-build-essential***
- Install the cross-compilation tools specifically designed for armhf (32-bit hard float) that closely emulates the Cortex-A8 architecture using this command:  
***sudo apt install crossbuild-essential-armhf***
- Check the cross-compiler installation to ensure its correctly configured by running the following command:  
***arm-linux-gnueabi-hf-gcc -v***
- Install QEMU user mode support for ARM command line emulation:  
***sudo apt install qemu-user-static***
- Add the armhf architecture package to your system by executing the following command:  
***sudo dpkg --add-architecture armhf***
- Double-check that the armhf architecture is added as a foreign architecture by running:  
***dpkg --print-foreign-architectures***
- Always make sure your package information is up to date by running:  
***sudo apt update***
- Compile your C program for ARM architecture with the -static flag, producing an ARM binary:  
***arm-linux-gnueabi-hf-gcc -static hello.c -o hello***
- Execute the cross-compiled ARM binary using QEMU user mode emulation. Verify the architecture using `utsname.machine`, which should display "arm7l" when executing the binary:  
***./hello***





# “HELLO\_WORLD” C PROGRAM


```
#include <stdio.h>
#include <sys/utsname.h>

int main() {
    struct utsname sysInfo;

    // Use uname to retrieve system information
    if (uname(&sysInfo) == -1) {
        perror("uname");
        return 1; // Exit with an error code
    }

    // Print system information and project group
    printf("Hello from the QEMU machine!\n");
    printf("System name: %s\n", sysInfo.sysname);
    printf("Node name: %s\n", sysInfo.nodename);
    printf("Machine: %s\n", sysInfo.machine);
    printf("Project Group Number: 6\n ");
    printf("Student Names: Poorvi Lakkadi | Prabath Reddy Sagili Venkata | Pranitha Kakumanu | Sai Hruthik Karumanchi | Sai Sujith Reddy Ravula | Sri Harish Jayaram\n");

    return 0;
}
```



**THANK YOU**

