CST 8703 Lab 0 - Hello Real-Time World

Kyle Chisholm

2022-05-01

Real-Time Systems and Embedded Programming

Submission Deadline: May 01, 2022

Online version of this document: https://chishok.github.io/CST8703/CST8703-Lab0

This lab will provide a tutorial on how to install and configure a Raspberry Pi for remote development. POSIX with realtime extensions will be introduced and a Linux application written in C will showcase basic clock and pthread functions.

> ***NOTE:*** There is currently a global shortage of Raspberry Pi boards. If you do not have a Raspberry Pi, simply use WSL (follow instructions below), native Ubuntu Linux, or Virtualbox to configure your development environment, build and run software in a POSIX-compliant environment.

Background

The Portable Operating System Interface (POSIX) provides an interface for operating systems, applications, and real-time embedded development on platforms like Linux, VxWorks, QNX, and many more. FreeRTOS, a popular Real-Time Operating System (RTOS), also provides a wrapper for the POSIX threading API (commonly known as pthreads).

A robust development environment and tools are necessary for building and debugging applications. Proprietary Real-Time Operating Systems (RTOS) typically provide a host IDE with extensive documentation to get started. For Free and Open-Source Software (FOSS) such as the Linux operating system, development environments are as numerous are there are developers who use them. This lab provides a setup guide for the Raspberry Pi and a C development environment. Python, Bash, and CMake will also be used for scripting and managing the build and execution process. VS Code will be used as the primary IDE.

> ***NOTE:*** This course is focused on developing real-time embedded applications. However, there are many other aspects of creating embedded Linux systems, including cross-compiling with toolchains, bootloaders, kernel configuration, filesystems, build systems, drivers. etc. Mastering Embedded Linux Programming is a good reference to go deeper into these areas of embedded Linux systems, if interested.

Useful Links And References For Help With POSIX

The Linux `man` pages are an indispensable tool for any systems developer, and especially for POSIX real-time systems. You can access man pages online at https://man7.org/linux/man-pages or simply use the command line tool `man` on any modern Linux distribution. Type `man man` in the terminal for more information. In addition to the course textbook, the following resources are helpful when writing code for embedded Linux:

Reference	Description
https://man7.org/tlpi [1]	The ultimate reference for Linux and UNIX system programming.
Modern C	Great resource for C language coding.
Mastering Embedded Linux Programming	Build full embedded solutions and cross-compile with Yocto or buildroot.

Coding Best Practices, Standards, and Formatting

Adhering to coding standards and formatting can go a long way in maintaining readable "clean" code. Several configuration files, tools, and VS Code extensions are helpful when writing and debugging code. A brief description of the tools will be provided during the installation procedures in the Methods section, but you are encouraged to learn more about them and personalize your development environment.

Materials

1. Raspberry Pi 4 or Raspberry Pi Zero 2 W.
2. Desktop computer with Linux, Windows, or Mac operating system.
3. Wired or wireless local network.

Methods

The following procedures will guide you through the installation and configuration of all necessary software for developing code with this lab series in CST 8703.

Install Prerequisites (Windows 10)

1. Install OpenSSH.

   Follow instructions online to install OpenSSH.

   Follow instructions online to start and configure OpenSSH.

2. Follow instructions online to install Chocolatey. Chocolatey has extensive documentation here: https://docs.chocolatey.org/.

3. Install Git with Chocolatey. Open a powershell prompt as an administrator and run:

   ```
   choco install git.install
   ```

4. Install Python with Chocolatey. Open a powershell prompt as an administrator and run:

   ```
   choco install python3
   ```

5. Follow instructions online to install Windows Subsystem for Linux (WSL).

   Reboot the system and open Ubuntu for the first time. The installation will continue and you will be prompted to enter a username and password. Upgrade Ubuntu with the commands:

   ```
   sudo apt update
   sudo apt upgrade -y
   ```

   You can also install different distributions, such as Debian:

   ```
   Invoke-WebRequest -Uri https://aka.ms/wsl-debian-gnulinux -OutFile ~/debian.appx -UseBasicParsing
   Add-AppxPackage -Path ~/debian.appx
   ```

6. Follow instructions online to install Windows Terminal. If you want to customize a pretty terminal prompt, OhMyPosh is recommended.

7. Follow instructions online to install VS Code.

Install Prerequisites (Ubuntu 20.04)

In a bash terminal, run the following:

```
sudo apt install -y \
    git \
    openssh-client \
    python3
sudo snap install code --classic
```

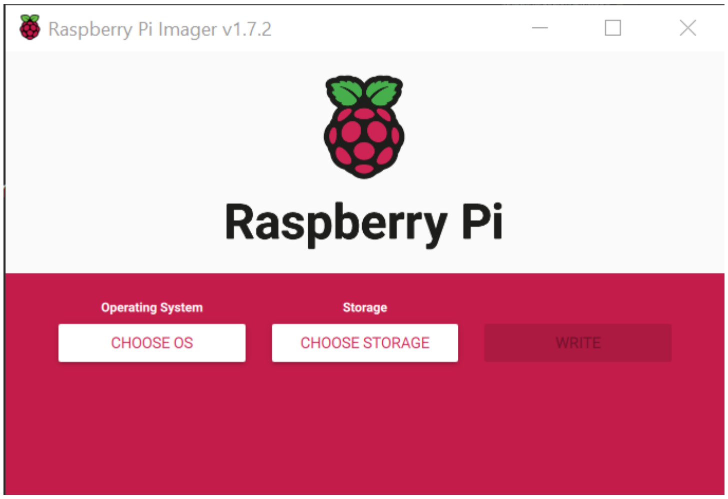
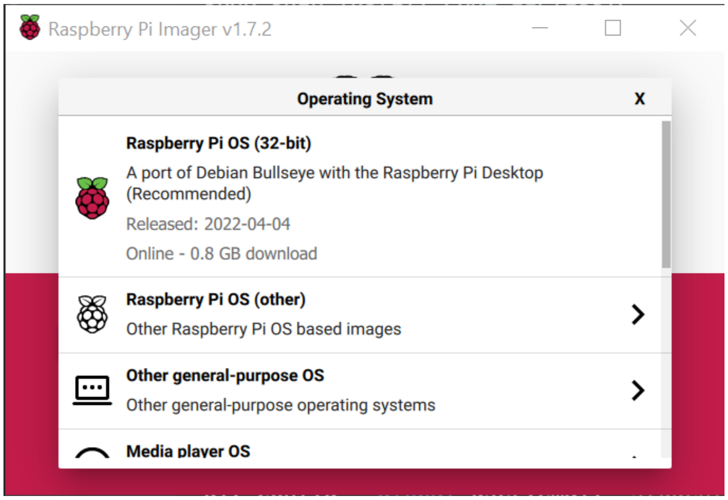
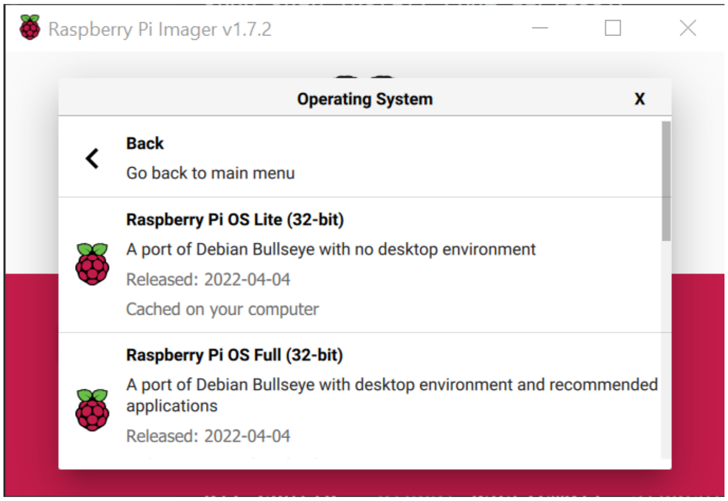
Install Raspberry Pi OS

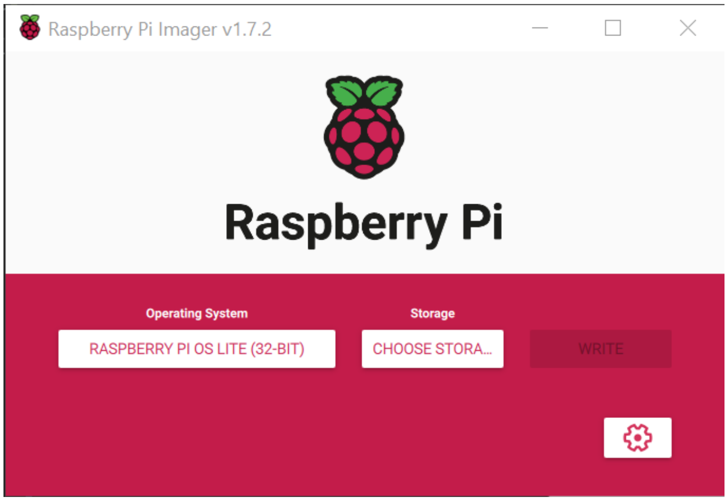
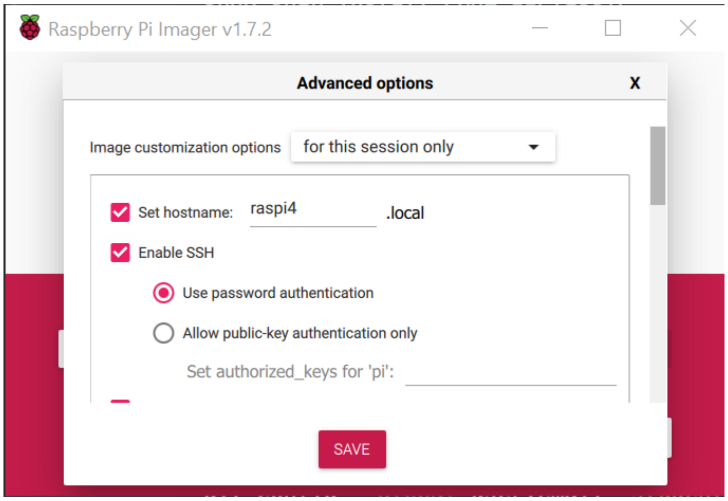
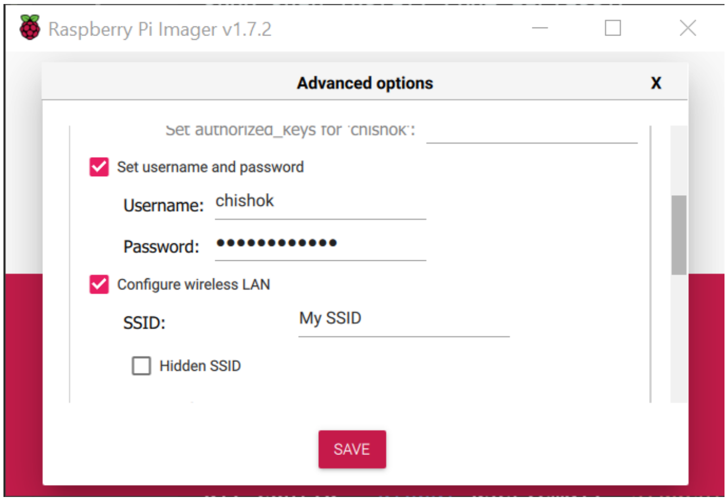
1. Download and install the Raspberry Pi OS Imager application.

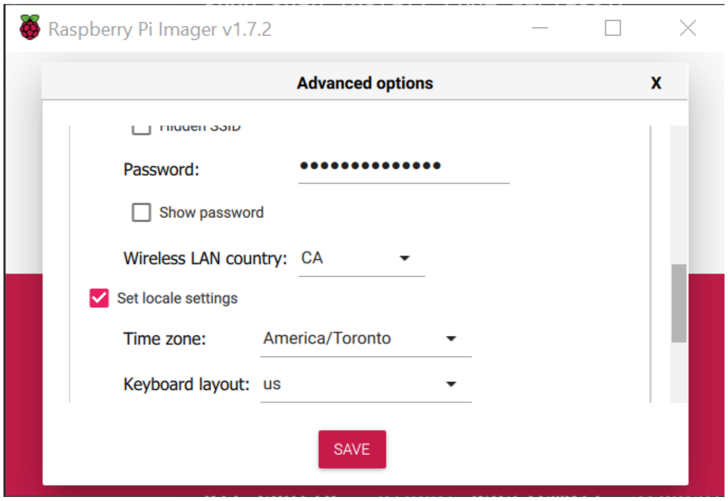
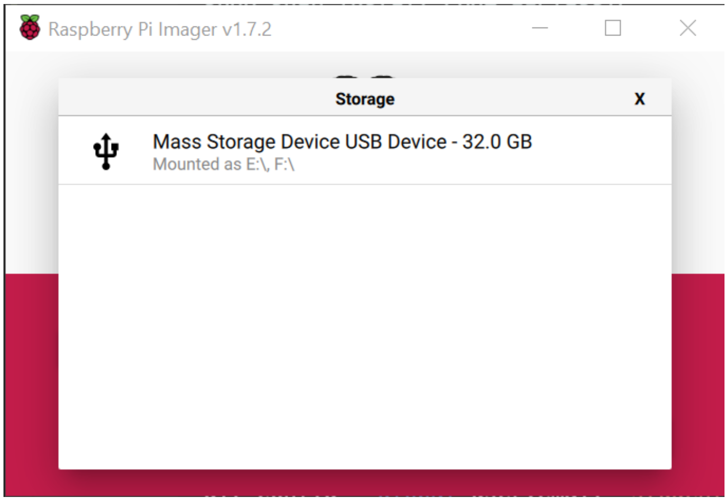
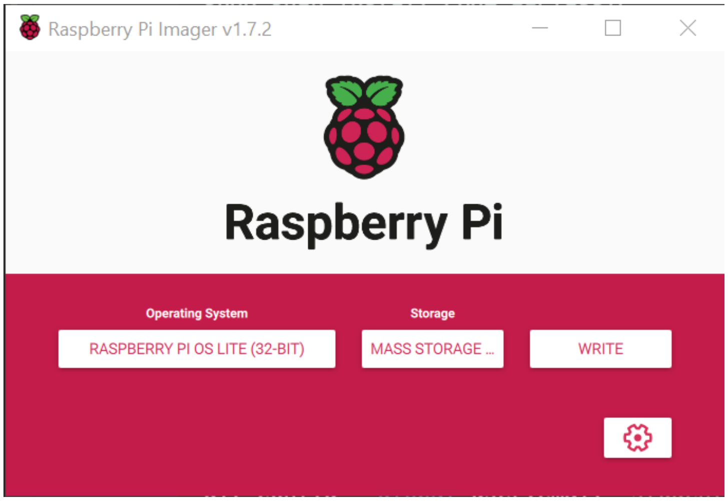
2. Insert a blank MicroSD card into a reader attached to the host computer.

3. Run the program “Raspberry Pi Imager”

4. Follow these steps to configure the Raspberry Pi OS:

Step	Screenshot
1. Select ‘CHOOSE OS’	 Raspberry Pi Imager v1.7.2 Raspberry Pi Operating System CHOOSE OS Storage CHOOSE STORAGE WRITE
2. Select ‘Raspberry Pi OS (other)’	 **Raspberry Pi OS (32-bit)** A port of Debian Bullseye with the Raspberry Pi Desktop (Recommended) Released: 2022-04-04 Online - 0.8 GB download **Raspberry Pi OS (other)** Other Raspberry Pi OS based images **Other general-purpose OS** Other general-purpose operating systems **Media player OS**
3. Select ‘Raspberry Pi OS Lite (32-bit)’	 **Back** Go back to main menu **Raspberry Pi OS Lite (32-bit)** A port of Debian Bullseye with no desktop environment Released: 2022-04-04 Cached on your computer **Raspberry Pi OS Full (32-bit)** A port of Debian Bullseye with desktop environment and recommended applications Released: 2022-04-04

Step	Screenshot
4. Click on the button with the gear symbol to set advanced options	 Raspberry Pi Imager v1.7.2 Raspberry Pi Operating System: RASPBERRY PI OS LITE (32-BIT) Storage: CHOOSE STORA... WRITE
5. Set custom hostname and enable SSH	 Raspberry Pi Imager v1.7.2 Advanced options Image customization options: for this session only ☑ Set hostname: raspi4 .local ☑ Enable SSH ● Use password authentication ○ Allow public-key authentication only Set authorized_keys for 'pi': SAVE
6. Set username and password and configure wireless LAN with your router SSID and password.	 Raspberry Pi Imager v1.7.2 Advanced options Set authorized_keys for 'chishok': ☑ Set username and password Username: chishok Password: •••••••••••• ☑ Configure wireless LAN SSID: My SSID □ Hidden SSID SAVE

Step	Screenshot
7. Set your locale and click 'SAVE'	 □ Hidden SSID Password: •••••••••••••• □ Show password Wireless LAN country: CA ☑ Set locale settings Time zone: America/Toronto Keyboard layout: us
8. Select 'CHOOSE STORAGE' and choose the MicroSD card.	 Mass Storage Device USB Device - 32.0 GB Mounted as E:\, F:\
9. Select 'WRITE' and wait for process to complete.	 Operating System: RASPBERRY PI OS LITE (32-BIT) Storage: MASS STORAGE ...

5. Insert the microSD into your Raspberry Pi and connect it to a power source.

Set Up Raspberry Pi With Development Tools

When your Raspberry Pi is connected to power, it will turn on and connect to the local network with SSID and password provided in the previous section. You can also connect to the local network with an Ethernet cable instead of relying on WiFi. The first step in setting up for remote development is to connect to the Raspberry Pi with ssh by entering the following command:

```
ssh <username>@<hostname>
```

Enter `<username>` and `<hostname>` as configured when creating the OS image in the previous section. For example:

```
ssh chishok@raspi4
```

If provided a warning "The authenticity of host '...' can't be established.", simply respond "yes".

If the hostname cannot be resolved, look at your router's DHCP settings to find the assigned IP address corresponding to the Raspberry Pi and use that instead. You may also configure a reserved IP address just for your Pi by changing the appropriate settings in your router. For example if the IP address is `192.168.1.120`, login with the command:

```
ssh chishok@192.168.1.120
```

Once connected to your Pi, the following steps will guide you through the initial setup for remote C/C++ and Python development:

1. Install C/C++ development tools with the following commands:

   ```
   sudo apt update
   sudo apt upgrade -y
   sudo apt install -y \
       git \
       build-essential \
       cmake \
       clang-format
   ```

2. Install Python with the following command:

   ```
   sudo apt install -y \
       python3 \
       python3-pip \
       python3-venv \
       python3-setuptools \
       python3-flake8 \
       python3-flake8-docstrings \
       python3-autopep8
   ```

3. Configure Git with your name and email:

   ```
   git config --global user.name "First Last"
   git config --global user.email "student@algonquincollege.com"
   ```

4. Create ssh key pair with the command:

```
ssh-keygen -t rsa
```

Do not enter a passphrase and accept all default options.

5. Create local directory `~/.local` and add it to `PATH` environment variable with the following commands:

   ```
   mkdir -p "${HOME}/.local/bin"
   echo "export PATH=\${PATH}:${HOME}/.local/bin" >> ${HOME}/.bash_profile
   ```

6. For lab submissions, environment variables `AC_USERNAME` and `AC_EMAIL` must be set with your student id name and email. These variables can be set when logging in in by adding them to the bash profile script as follows:

   ```
   echo "export AC_USERNAME=chishok" >> "${HOME}/.bash_profile"
   echo "export AC_EMAIL=chishok@algonquincollege.com" >> "${HOME}/.bash_profile"
   ```

   Replace `chishok` and `chishok@algonquincollege.com` with your own student name and email.

7. Source profile script from `~/.bashrc`:

   ```
   echo "source ${HOME}/.bash_profile" >> ${HOME}/.bashrc
   ```

Hint: Use SSH key pair to Connect To Raspberry Pi

Exit the ssh connection with the command `exit` to return to your desktop computer command line. Follow these steps to conveniently connect to the raspberry pi without having to enter a username and password:

1. Generate key pair

   ```
   ssh-keygen -t rsa
   ```

   Do not enter a passphrase and accept all default options.

2. Copy your public key to the Raspberry Pi's authorized keys file:

   From Windows:

   ```
   ssh-keygen -t rsa
   cat $env:USERPROFILE\.ssh\id_rsa.pub | ssh chishok@raspi4 "cat >> .ssh/authorized_keys"
   ```

   From Linux:

   ```
   ssh-copy-id chishok@raspi4
   ```

You should now be able to `ssh` into your Raspberry Pi from your computer without having to enter a password.

Set Up VS Code for SSH Remote Development With The Raspberry Pi

1. Open VS Code and install the Remote Development Extension Pack.

2. Follow the instructions online to connect to a remote host by SSH.

3. You can now use VS Code as though it was running on the Raspberry Pi. Using the integrated terminal is also very convenient.
4. While connected remotely, install the extensions in the table below. Refer to online help for more information about managing extensions on remote hosts.

Extension	Description
C/C++ Extension Pack	C/C++ build and debug, CMake integration, code-completion suggestions, etc.
Clang-Format	C/C++ code formatting.
Python	Python integration, debugging, linting, formatting, etc.

Optional extensions for themes and syntax highlighting (install on desktop computer):

Extension	Link
Better C++ Syntax	Improves syntax highlighting.
C/C++ Themes	Color syntax themes for C/C++ code.
Material Icon Theme	Improved file icons.

Retrieve Lab Source

On the Raspberry Pi (or local Linux system, if desired), checkout the code sample for this lab:

```
git clone https://github.com/chishok/CST8703-Lab0
```

Run VS Code on your desktop computer and connect remotely with SSH (or WSL if on Windows without a Raspberry Pi). Open the folder `CST8703-Lab0` checked out with Git. The following screenshot shows how the folder should eventually appear:

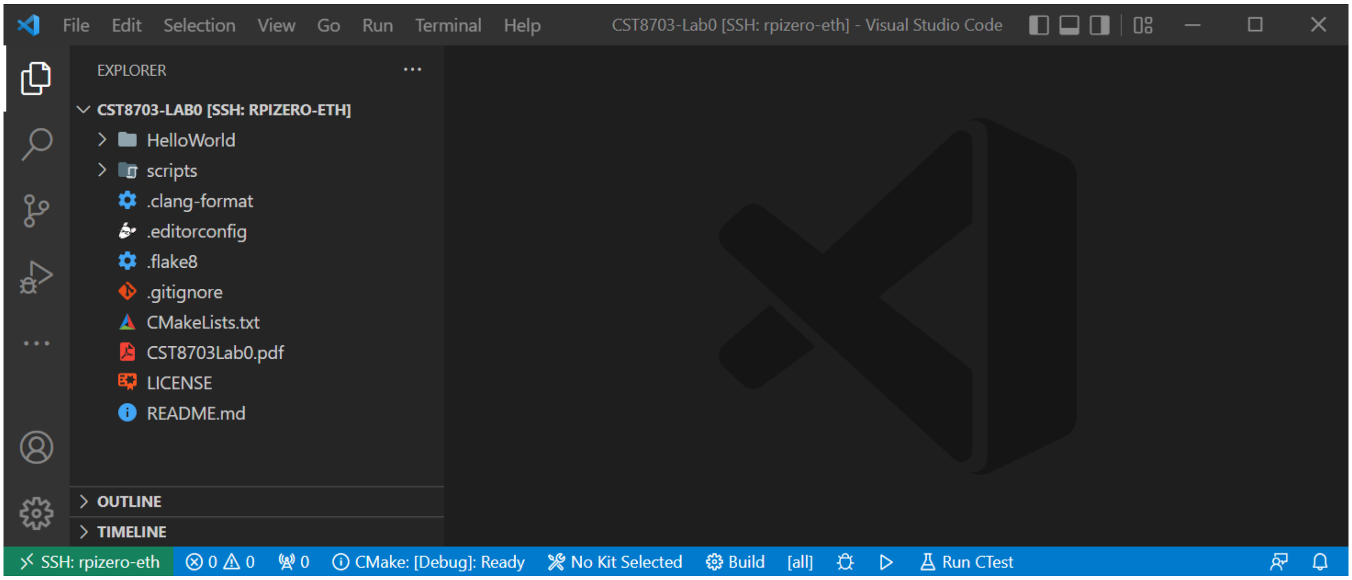
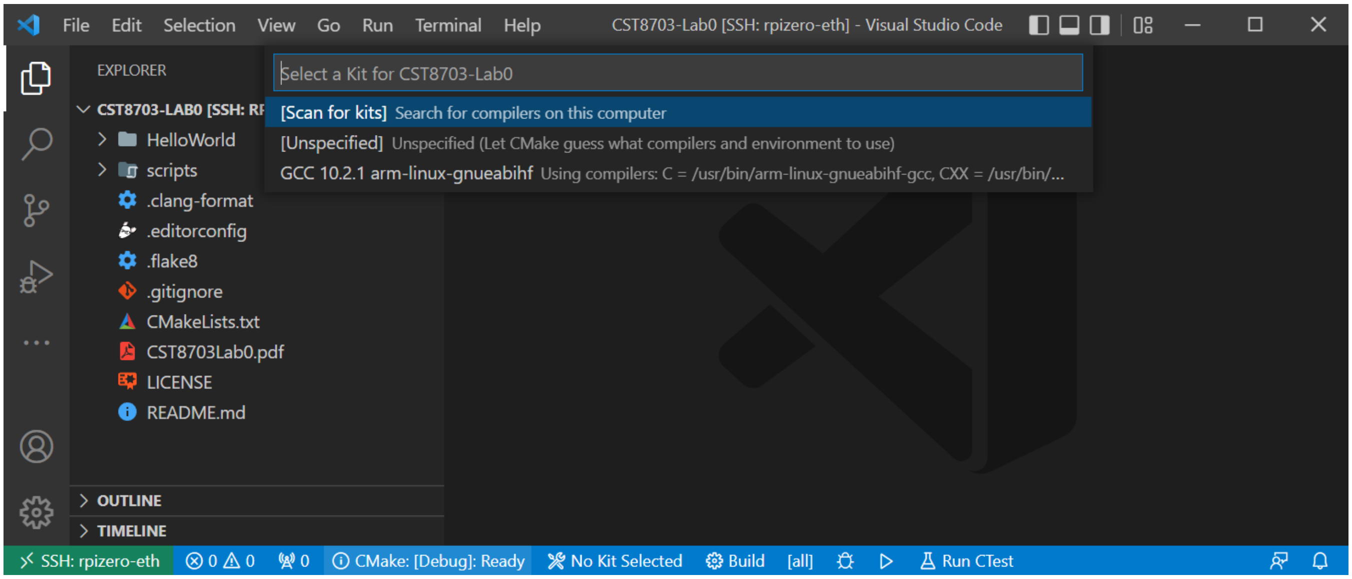


Figure 1: Screenshot of VS Code

Build and Debug

NOTE: For more information on how to use CMake Tools on VS Code, refer to the help available online.

1. Click on kits button at the bottom of VS Code window taskbar. Select the GCC compiler kit.



Select a Kit for CST8703-Lab0

[Scan for kits] Search for compilers on this computer

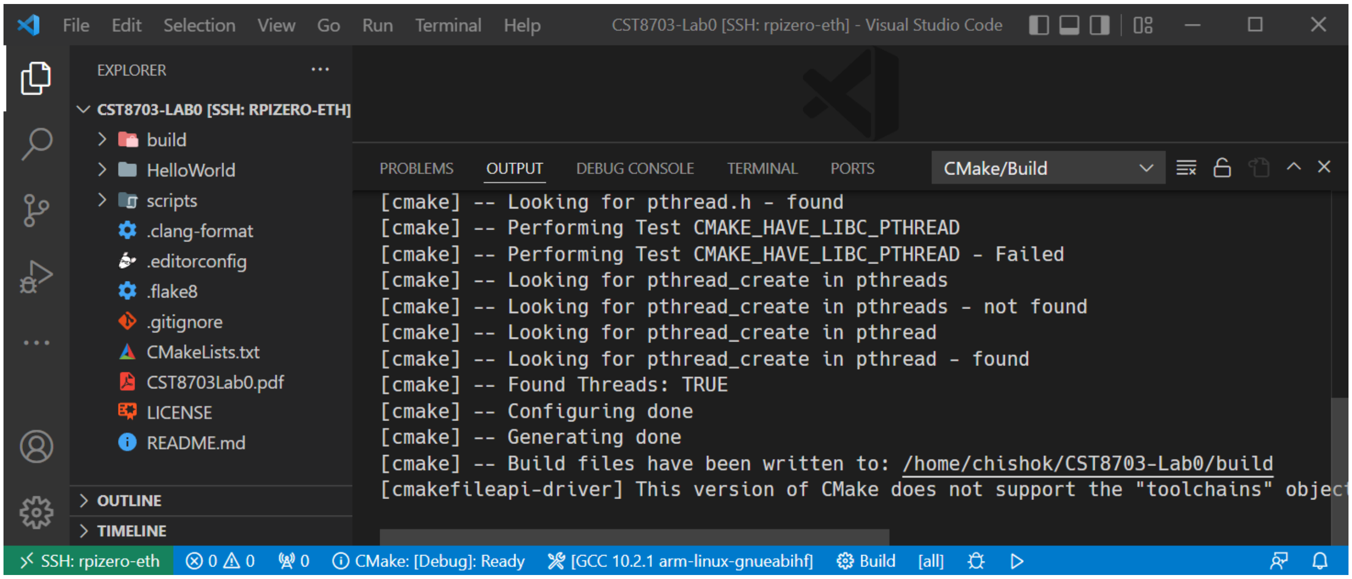
[Unspecified] Unspecified (Let CMake guess what compilers and environment to use)

GCC 10.2.1 arm-linux-gnueabihf Using compilers: C = /usr/bin/arm-linux-gnueabihf-gcc, CXX = /usr/bin/...

Figure 2: Screenshot of VS Code with CMake kits selection

2. Select the CMake button and choose the Build type "Debug."

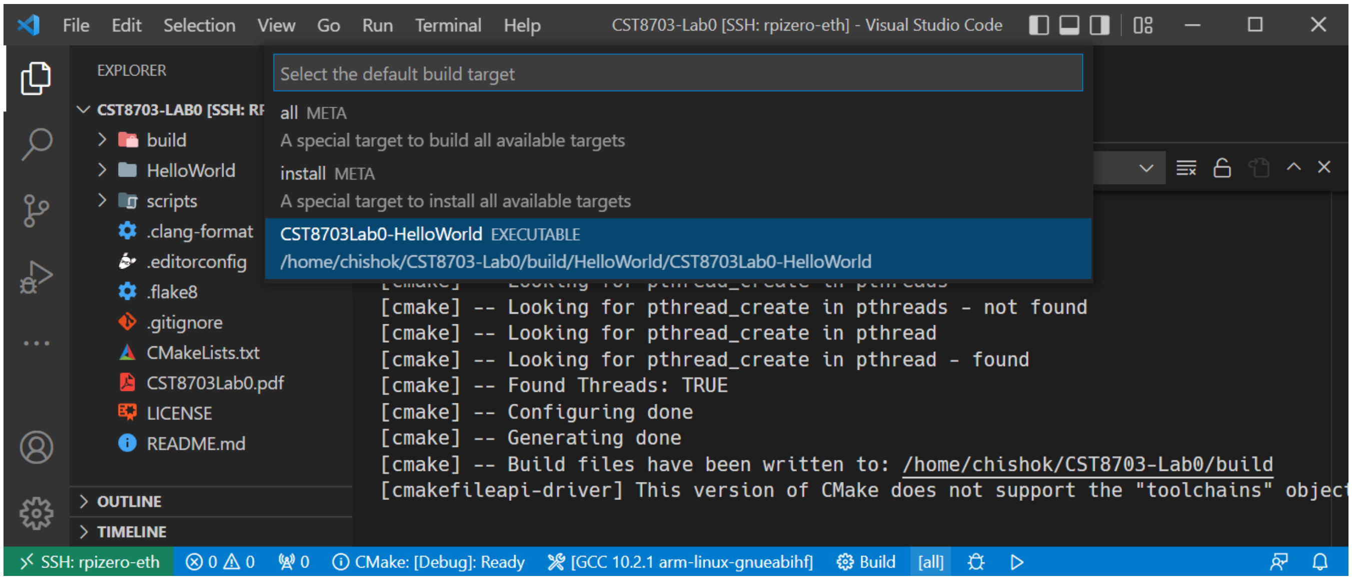
   CMake will configure and a `build` folder is created where target programs are created. The following screenshot shows the output on successful build:



```
[cmake] -- Looking for pthread.h - found
[cmake] -- Performing Test CMAKE_HAVE_LIBC_PTHREAD
[cmake] -- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Failed
[cmake] -- Looking for pthread_create in pthreads
[cmake] -- Looking for pthread_create in pthreads - not found
[cmake] -- Looking for pthread_create in pthread
[cmake] -- Looking for pthread_create in pthread - found
[cmake] -- Found Threads: TRUE
[cmake] -- Configuring done
[cmake] -- Generating done
[cmake] -- Build files have been written to: /home/chishok/CST8703-Lab0/build
[cmakefileapi-driver] This version of CMake does not support the "toolchains" objec
```

Figure 3: Screenshot of VS Code with CMake configuration output

3. Select a target program by clicking on `[all]` button on bottom taskbar and choosing `CST8703Lab0-HelloWorld` as shown here:



Select the default build target

all META
A special target to build all available targets

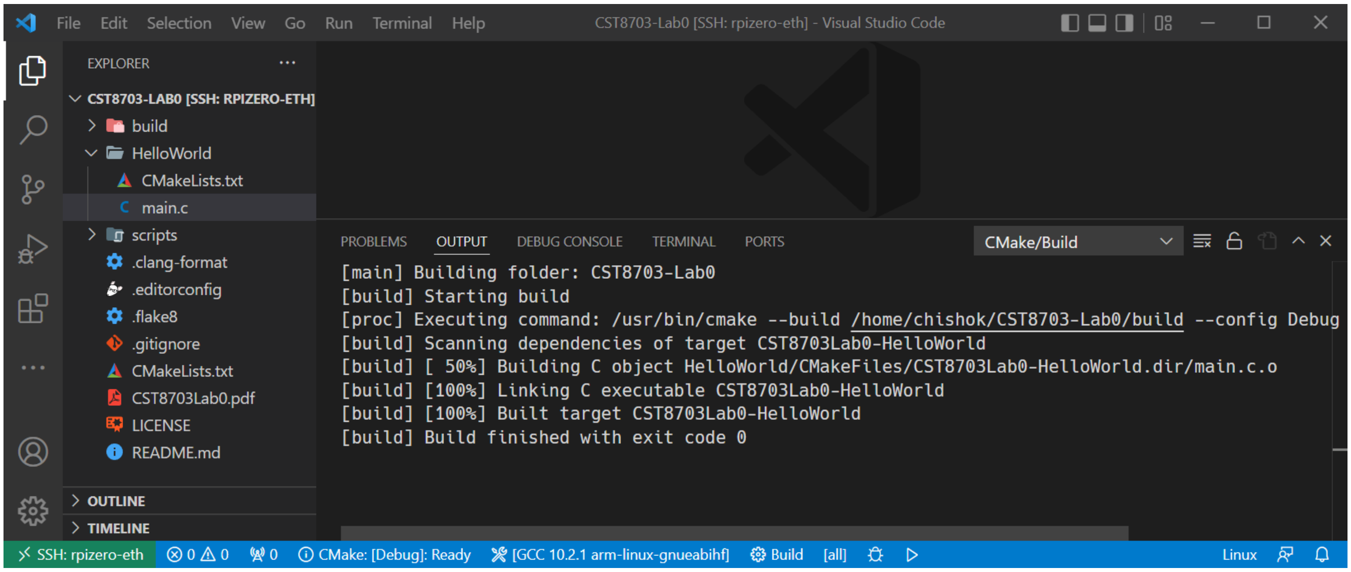
install META
A special target to install all available targets

CST8703Lab0-HelloWorld EXECUTABLE
/home/chishok/CST8703-Lab0/build/HelloWorld/CST8703Lab0-HelloWorld

```
[cmake] -- Looking for pthread_create in pthreads - not found
[cmake] -- Looking for pthread_create in pthread
[cmake] -- Looking for pthread_create in pthread - found
[cmake] -- Found Threads: TRUE
[cmake] -- Configuring done
[cmake] -- Generating done
[cmake] -- Build files have been written to: /home/chishok/CST8703-Lab0/build
[cmakefileapi-driver] This version of CMake does not support the "toolchains" objec
```

Figure 4: Screenshot of VS Code to select target

4. Click the “Build” button on the bottom taskbar to build the target. The build process output is shown here:



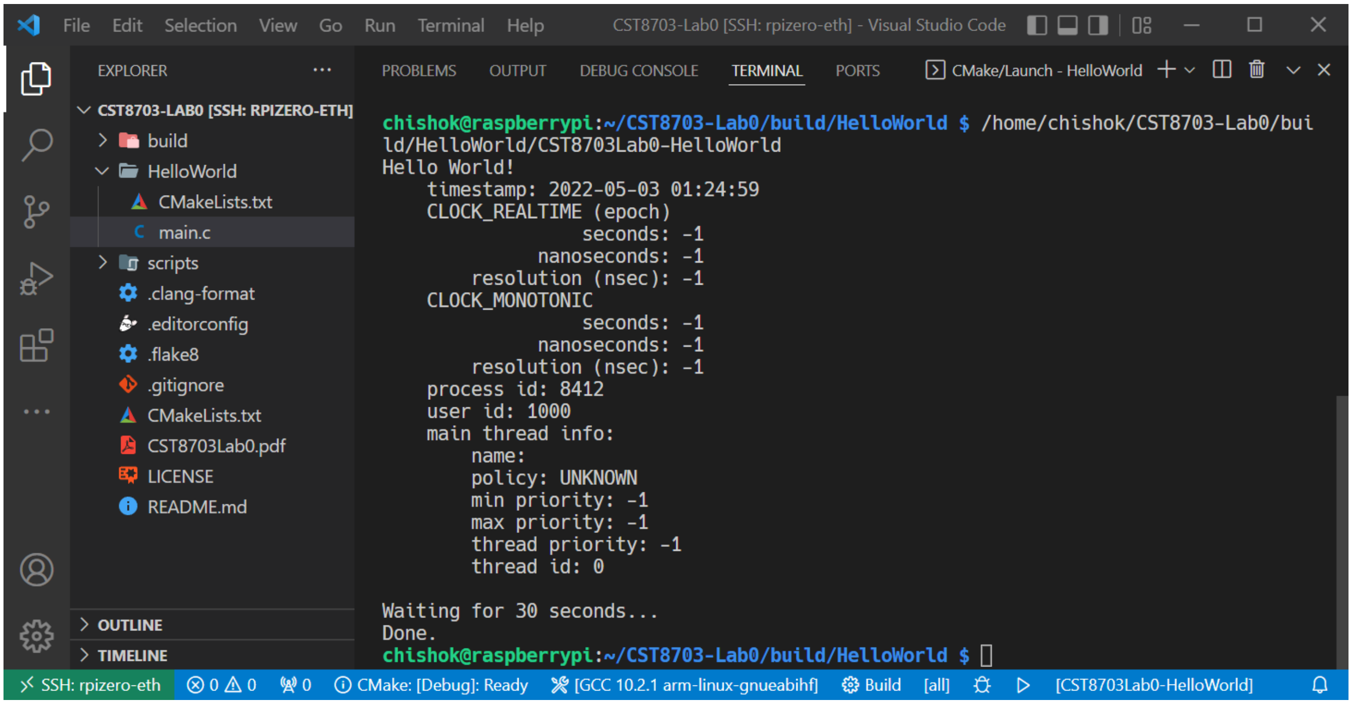
```
[main] Building folder: CST8703-Lab0
[build] Starting build
[proc] Executing command: /usr/bin/cmake --build /home/chishok/CST8703-Lab0/build --config Debug
[build] Scanning dependencies of target CST8703Lab0-HelloWorld
[build] [ 50%] Building C object HelloWorld/CMakeFiles/CST8703Lab0-HelloWorld.dir/main.c.o
[build] [100%] Linking C executable CST8703Lab0-HelloWorld
[build] [100%] Built target CST8703Lab0-HelloWorld
[build] Build finished with exit code 0
```

Figure 5: Screenshot of VS Code with build output

5. Click the “Play” button beside the target name to run the program. The terminal appears with output from running the program as shown here:

6. Open `HelloWorld/main.c` and create a breakpoint on line 112.Debug the program by clicking on the “Bug” button on the bottom taskbar. VS Code enters debug mode as shown here:

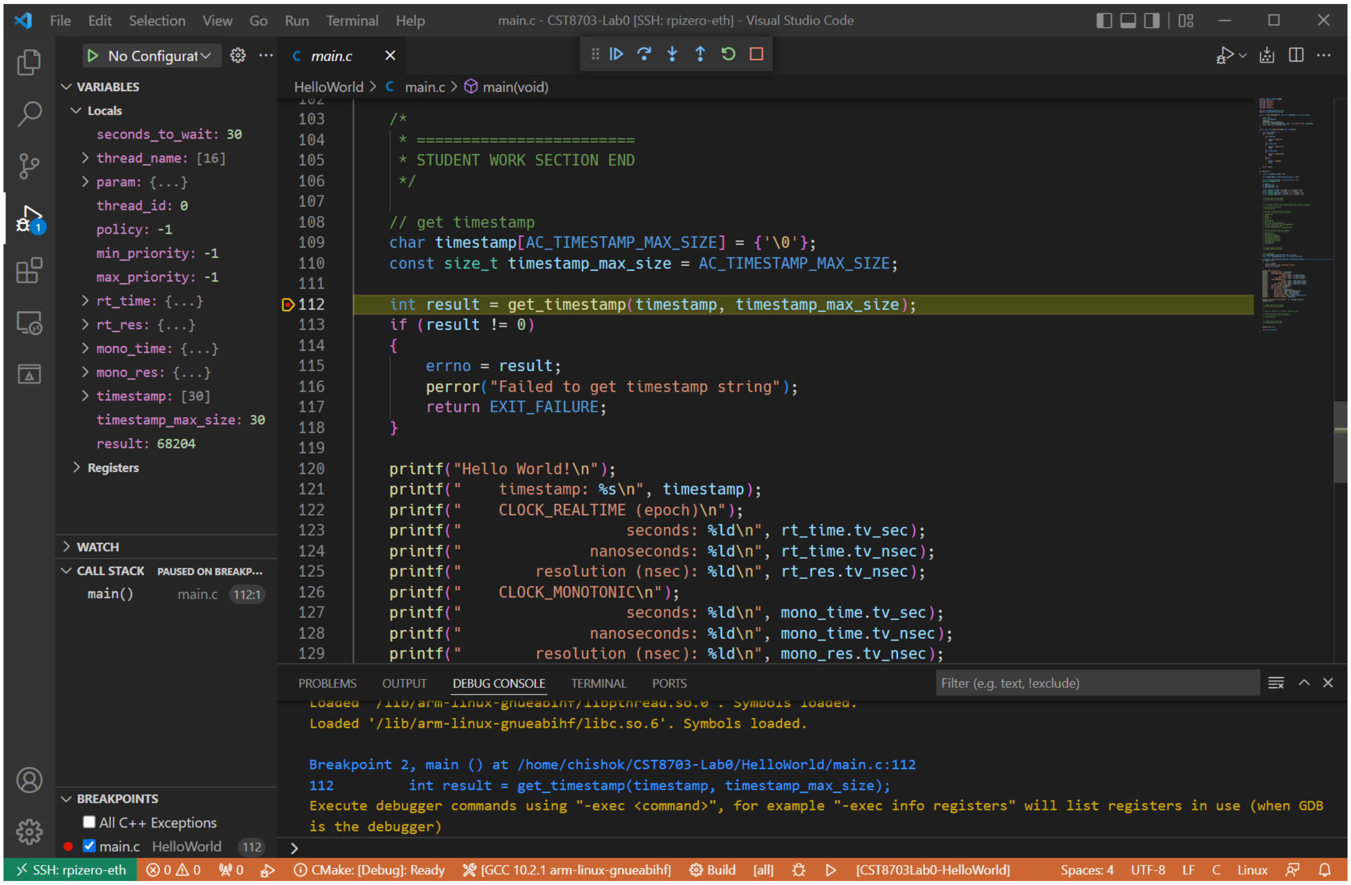
   More information on debugging in VS Code can be found online.



```
chishok@raspberrypi:~/CST8703-Lab0/build/HelloWorld $ /home/chishok/CST8703-Lab0/bui
ld/HelloWorld/CST8703Lab0-HelloWorld
Hello World!
    timestamp: 2022-05-03 01:24:59
    CLOCK_REALTIME (epoch)
                  seconds: -1
              nanoseconds: -1
        resolution (nsec): -1
    CLOCK_MONOTONIC
                  seconds: -1
              nanoseconds: -1
        resolution (nsec): -1
    process id: 8412
    user id: 1000
    main thread info:
        name:
        policy: UNKNOWN
        min priority: -1
        max priority: -1
        thread priority: -1
        thread id: 0

Waiting for 30 seconds...
Done.
chishok@raspberrypi:~/CST8703-Lab0/build/HelloWorld $ 
```

Figure 6: Screenshot of VS Code with run output



```
seconds_to_wait: 30
> thread_name: [16]
> param: {...}
thread_id: 0
policy: -1
min_priority: -1
max_priority: -1
> rt_time: {...}
> rt_res: {...}
> mono_time: {...}
> mono_res: {...}
> timestamp: [30]
timestamp_max_size: 30
result: 68204
```

main() main.c 112:1

□ All C++ Exceptions

☑ main.c HelloWorld 112

```
    /*
     * ========================
     * STUDENT WORK SECTION END
     */

    // get timestamp
    char timestamp[AC_TIMESTAMP_MAX_SIZE] = {'\0'};
    const size_t timestamp_max_size = AC_TIMESTAMP_MAX_SIZE;

    int result = get_timestamp(timestamp, timestamp_max_size);
    if (result != 0)
    {
        errno = result;
        perror("Failed to get timestamp string");
        return EXIT_FAILURE;
    }

    printf("Hello World!\n");
    printf("    timestamp: %s\n", timestamp);
    printf("    CLOCK_REALTIME (epoch)\n");
    printf("                  seconds: %ld\n", rt_time.tv_sec);
    printf("              nanoseconds: %ld\n", rt_time.tv_nsec);
    printf("        resolution (nsec): %ld\n", rt_res.tv_nsec);
    printf("    CLOCK_MONOTONIC\n");
    printf("                  seconds: %ld\n", mono_time.tv_sec);
    printf("              nanoseconds: %ld\n", mono_time.tv_nsec);
    printf("        resolution (nsec): %ld\n", mono_res.tv_nsec);
```

```
Loaded '/lib/arm-linux-gnueabihf/libpthread.so.0'. Symbols loaded.
Loaded '/lib/arm-linux-gnueabihf/libc.so.6'. Symbols loaded.

Breakpoint 2, main () at /home/chishok/CST8703-Lab0/HelloWorld/main.c:112
112         int result = get_timestamp(timestamp, timestamp_max_size);
Execute debugger commands using "-exec <command>", for example "-exec info registers" will list registers in use (when GDB
is the debugger)
```

Figure 7: Screenshot of VS Code while debugging

Provide thread and timing information

Alter the target program to set scheduling parameters of the main thread, retrieve time, and cause the main program to sleep for 30 seconds. Make your modifications between the comments

```
    /*
     * STUDENT WORK SECTION BEGIN
     * ==========================
     */
```

and

```
    /*
     * ========================
     * STUDENT WORK SECTION END
     */
```

Hints and instruction may be provided as comments in the code. Write code to do the following:

1. Set the following for the main thread:
    1. Scheduling priority: 90
    2. Scheduling policy: `SCHED_RR`
2. Retrieve the following information from the main thread:
    1. Scheduling priority
    2. Scheduling policy
    3. Thread name
    4. Thread ID
    5. Minimum priority for current scheduling policy
    6. Maximum priority for current scheduling policy
3. Get the current time form the following real-time clocks:
    1. `CLOCK_REALTIME`
    2. `CLOCK_MONOTONIC`
4. Sleep until 30 seconds after current time was retrieved on `CLOCK_MONOTONIC`. Use the absolute time flag (see `man clock_nanosleep` for more information).

Analysis

While running the program during the 30 second pause in execution, you can access and terminate process using the following Linux tools and commands:

```
HELLO_PID=$(pgrep -f HelloWorld)
ps -F -p ${HELLO_PID}
kill -s TERM ${HELLO_PID}
```

Submission

If you forked the repo on your personal GitHub, commit and push your changes, then download your repo as a zip file. If you do not use GitHub, compress the source code in a zip file or tar file. Have a look at `man tar` or `man zip` for more information. Once the code is submitted, it will be run using the command:

```
./scripts/run_submission.sh
```

The expected contents of output_user.txt should be similar to:

```
pthread_setschedparam: Operation not permitted
Hello World!
    timestamp: 2022-04-12 00:28:02
    realtime (epoch) seconds: 1649737682
                 nanoseconds: 34219300
    monotonic seconds: 16825
          nanoseconds: 810159100
    process id: 6393
    user id: 1000
    Main thread info:
        policy: SCHED_OTHER
        min priority: 0
        max priority: 0
        thread priority: 0
        thread id: 140268182656832
```

The expected contents of output_root.txt:

```
Hello World!
    timestamp: 2022-04-12 00:29:00
    realtime (epoch) seconds: 1649737740
                 nanoseconds: 710771700
    monotonic seconds: 16884
          nanoseconds: 486711600
    process id: 6446
    user id: 0
    Main thread info:
        policy: SCHED_RR
        min priority: 1
        max priority: 99
        thread priority: 60
        thread id: 140070128928576
```

References

[1] M. Kerrisk, *The linux programming interface : A linux and unix system programming handbook*. No Starch Press, 2010.