In this mini-project, you will first learn how to build your own custom kernel on Raspberry Pi3. This kernel will be patched with RT-Gang: A Real-Time Gang Scheduling Framework which is a custom scheduling feature under Linux kernel. You will then learn to compile and use an out-of-source-tree kernel module (BWLOCK). You should return mini-proj1-1.txt, mini-proj1-2.txt, mini-proj1-3.txt, mini-proj1-4.txt.

EECS 753 Mini Project #1
Part 0. Install Raspbian
1
Part 1. Build your own kernel with RT-Gang
2
Part 2. RT-Gang experiment
3

Part 0. Install Raspbian

First thing first. Install the Raspbian OS on the Pi 3. Use the 32GB micro-SD card you received. Follow the detailed instruction in the following. Using NOOB may be easiest, but directly installing Raspbian is also not that hard. https://www.raspberrypi.org/downloads/.

Setup gpu memory and cpu clock as follows to maximize memory space and system stability.

```
$ sudo nano /boot/config.txt
```

```
gpu_mem=64
force_turbo=1
arm_freq=1200
```

Pi3A+ has small DRAM (512MB). So, it’s a good idea to setup additional swap space.

```
$ sudo nano /etc/dphys-swapfile
```

```
CONF_SWAPSIZE=1500
```

Next, setup wifi as follows. Note that this assumes the campus wide 'JAYHAWK' wifi network:
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

network={
  ssid="JAYHAWK"
  priority=1
  proto=RSN
  key_mgmt=WPA-EAP
  pairwise=CCMP
  auth_alg=OPEN
  eap=PEAP
  identity="kuonlineID"
  password="kupassword"
  phase1="peaplabel=0"
  phase2="auth=MSCHAPV2"
};

After modifying the network setting, reboot the system. Check if wifi is connected using `$ ifconfig wlan0` command. In your home, you need to add a setting for your home network; please check [https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md](https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md).

Part 1. Build your own kernel with RT-Gang

Once you boot to the Pi 3, it’s time to install your own kernel. All the necessary information can be found in the following document.

Below is essentially a copy of the document above, with some additional commentary:

```
$ sudo apt-get install git bc bison flex libssl-dev
$ git clone --depth=1 https://github.com/raspberrypi/linux -b rpi-4.19.y
$ cd linux
$ patch -p1 < rtgang-v4.19.patch
$ KERNEL=kernel7
$ make bcm2709_defconfig

Before you begin, let’s backup the current kernel just in case:
$ sudo cp /boot/kernel7.img /boot/kernel7.img.backup

Then, do the following to build the kernel:

$ make -j4 zImage modules dtbs

Now, this will take about 2 hours. So, take a break, sleep, or watch a movie while the pi is busy compiling the kernel. After a long time, if the compilation was successful, proceed the following to install the compiled kernel (and other stuff).```
```
$ sudo make modules_install
$ sudo cp arch/arm/boot/dts/*/*.dtb /boot/
$ sudo cp arch/arm/boot/dts/overlays/*/*.dtb* /boot/overlays/
$ sudo cp arch/arm/boot/dts/overlays/README /boot/overlays/
$ sudo cp arch/arm/boot/zImage /boot/$KERNEL.img
```

If everything went smoothly, then reboot the system and check if the kernel is the one you just compiled.
```
$ uname -a
Linux raspberrypi 4.19.29-v7+ #31 SMP Wed Mar 20 07:55:27 GMT 2019 armv7l GNU/Linux
```

Make sure that `RT_GANG_LOCK` is available as a scheduling feature:
```
$ cat /sys/kernel/debug/sched_features
GENTLE_FAIR_SLEEPERS NO_RT_GANG_LOCK START_DEBIT NO_NEXT_BUDDY LAST_BUDDY
CACHE_HOT_BUDDY WAKEUP_PREEMPTION NO_HRTICK NO_DOUBLE_TICK LB_BIAS NONTASK_CAPACITY
TTMU_QUEUE NO_SIS_AVG_CPU SIS_PROP NO_WARN_DOUBLE_CLOCK RT_PUSH_IPI RT_RUNTIME_SHARE
NO_LB_MIN ATTACH_AGE_LOAD WA_IDLE WA_WEIGHT WA_BIAS UTIL_EST
```

Copy the output of `uname -a` command on your Pi 3 and save it as `mini-proj1-1.txt` file. You should return the file as a proof.

Part 2. Build Throttling Module

Switch to a tty (e.g., tty1) terminal. This can be achieved using the following key combination on most keyboards: CTRL + SHIFT + F1

Create a folder under `home/pi/` which will serve as the root directory for this project:
```
$ cd /home/pi
$ mkdir project_1
$ cd project_1
$ root_dir=`pwd`
```

Clone the RT-Gang repository under root directory:
```
$ git clone https://github.com/CSL-KU/RT-Gang.git
```

Go to the throttling module inside the directory and build it:
```
$ cd RT-Gang/throttling/kernel_module
$ make
```
Building Perf-Kernel Module

make[1]: Entering directory '/home/pi/work/gits/linux'
CC [M]  /home/pi/project_1/RT-Gang/throttling/kernel_module/src/bwlockmod.o
CC [M]  /home/pi/project_1/RT-Gang/throttling/kernel_module/src/perf.o
CC [M]  /home/pi/project_1/RT-Gang/throttling/kernel_module/src/interrupt.o
CC [M]  /home/pi/project_1/RT-Gang/throttling/kernel_module/src/overflow.o
LD [M]  /home/pi/project_1/RT-Gang/throttling/kernel_module/bwlockmod.o
Building modules, stage 2.
MODPOST 1 modules
CC     /home/pi/project_1/RT-Gang/throttling/kernel_module/bwlockmod.mod.o
LD [M]  /home/pi/project_1/RT-Gang/throttling/kernel_module/bwlockmod.ko
make[1]: Leaving directory '/home/pi/work/gits/linux'

> Build Complete. Outputs can be seen in exe directory

This should create 'exe' directory; containing the kernel module for throttling:

$ ls exe
bwlockmod.ko

Part 3. RT-Gang experiment

1. Clone IsolBench suite under the root directory:

$ cd ${root_dir}
$ git clone https://github.com/CSL-KU/IsolBench.git

Build the bandwidth-rt benchmark from IsolBench:

$ cd IsolBench/bench
$ make bandwidth-rt

gcc -O2 -Wall -g -c -o bandwidth-rt.o bandwidth-rt.c

gcc -O2 -Wall -g -Wl,--no-as-needed -lrt bandwidth-rt.o -o bandwidth-rt -lpthread

2. Create symbolic links to bandwidth-rt executable. These are used in the experiment script:

$ sudo cp -v bandwidth-rt /usr/local/bin
$ sudo ln -s /usr/local/bin/bandwidth-rt /usr/local/bin/tau_1
$ sudo ln -s /usr/local/bin/bandwidth-rt /usr/local/bin/tau_2
$ sudo ln -s /usr/local/bin/bandwidth-rt /usr/local/bin/tau_be_mem
$ sudo ln -s /usr/local/bin/bandwidth-rt /usr/local/bin/tau_be_cpu

3. Install the following packages: trace-cmd, kernelshark

$ sudo apt install trace-cmd kernelshark

4. Go to the directory containing the RT-Gang experiment script:

$ cd ${root_dir}/RT-Gang/experiments/pi/fig5
5. Switch to super-user shell:

```bash
$ sudo bash
```

6. Place the platform in maximum performance mode using the `max_perf.sh` script:

```bash
$ ./max_perf.sh
```

7. Verify that the platform is in maximum performance mode using `perf_state.sh` script:

```bash
$ ./perf_state.sh
```

```bash
WARNING - Must Be Run Sudo
WARNING - Use Only on Pi
Cores active
0-3
Scaling governors (0, 1, 2, 3)
  performance
  performance
  performance
  performance
CPU available frequencies
600000 1200000
CPU cycle frequencies (0, 1, 2, 3)
 1200000
 1200000
 1200000
 1200000
Throttling
-1
End Performance States
```

8. Execute the experiment script:

```bash
$ . fig5.sh
```

```bash
Solo Experiments
Co-Sched Experiment
[3]+ Done           chrt -f 5 tau_2 -o -t 0 -c 2 -n 2 -m 192 -i 210 --jobs
500 --period 30 -v 1 &> /tmp/tau_2.mint
[1]- Terminated    tau_be_mem -t 0 -c 2 -n 1 -m 1024 &>
/tmp/tau_be_mem.mint
[2]+ Terminated    tau_be_cpu -t 0 -c 3 -n 1 -m 8 &>/tmp/tau_be_cpu.mint
RT-Gang Experiment
[3]+ Done           chrt -f 5 tau_2 -o -t 0 -c 2 -n 2 -m 192 -i 210 --jobs
500 --period 30 -v 1 &> /tmp/tau_2.rtg
[1]- Terminated    tau_be_mem -t 0 -c 2 -n 1 -m 1024 &>
/tmp/tau_be_mem.rtg
[2]+ Terminated    tau_be_cpu -t 0 -c 3 -n 1 -m 8 &>/tmp/tau_be_cpu.rtg
```
Once the experiment is complete, following trace files (in addition to a number of log files) should be visible in the current directory: trace.solo, trace.mint, trace.rtg

Start the graphical user interface (GUI):

```
$ service lightdm start
```

9. Once the GUI is running, launch the terminal and go to the experiment directory:

```
$ cd /home/pi/project_1/RT-Gang/experiments/pi/fig5
```

10. Analyze the corun execution trace in kernel-shark:

```
$ kernelshark -i trace.mint
```

![](image)

11. From the ‘Filter’ option in the taskbar, select ‘tasks’. In the resulting pop-up, select the following tasks: tau_1, tau_2, tau_be_mem, tau_be_cpu
12. Zoom into a portion of the trace:

13. Zoom into one hyper period (60-msec):
14. You should be able to see an execution trace similar to the top part of Figure-5 (Corun Execution) in the RT-Gang paper.

15. Analyze the RT-Gang execution trace in kernel-shark:

```bash
$ kernelshark -i trace.rtg
```
16. Repeat the same steps as described earlier. In the ‘Filter -> tasks’ menu, select all ‘kthrottle’ instances in addition to the other tasks.

The hyper-period for this scenario should look like this:

Take screenshots of one hyper-period (60-msec) of execution in each scenario (corun, RT-Gang).