**CS356** **Operating System Projects** **Spring 2021**

# Project 2: Android scheduler

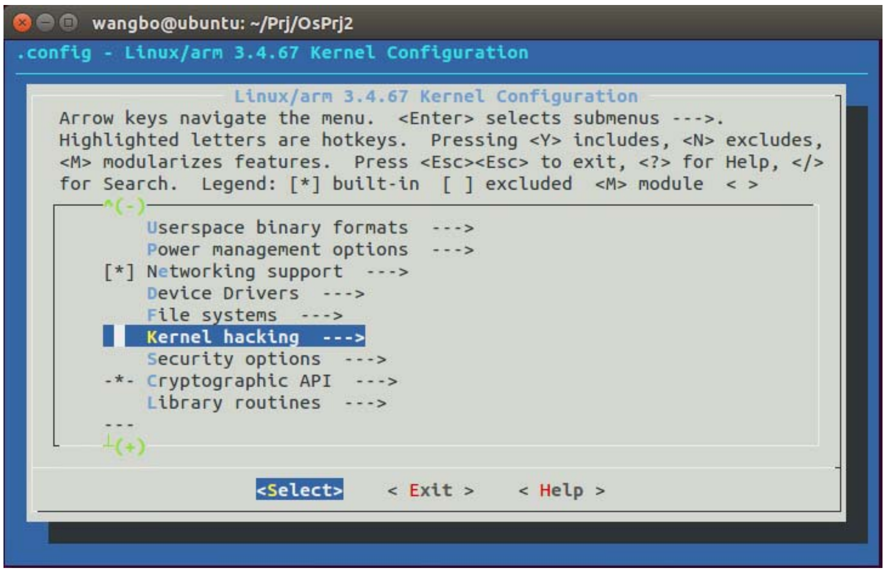
**Objectives:**

- Compile the Android kernel.
- Familiarize Android scheduler
- Implement a weighted round robin scheduler.
- Get experience with software engineering techniques.

**Make sure your system is 64-bits system.**

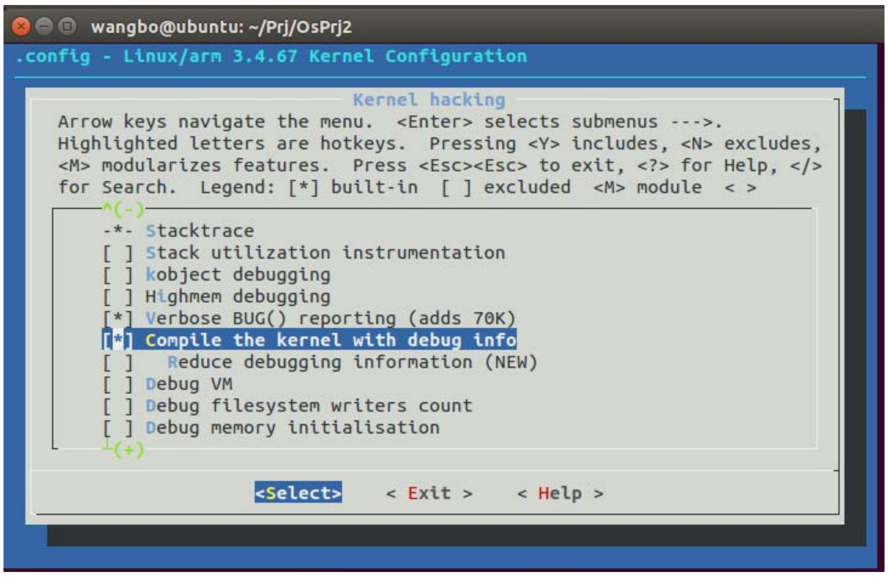
**Problem Statement:**

1. **Compile the Linux kernel.**
   a. Make sure that you have added the following path into your environment variable.
   ANDROID_NDK_HOME/toolchains/arm-linux-androideabi-4.6/prebuilt/linux-x86_64/bin
   b. Open Makefile in KERNEL_SOURCE/goldfish/, and find these:
   ```
   ARCH            ?=  $(SUBARCH)
   CROSS_COMPILE   ?=
   ```
   Change it to:
   ```
   ARCH            ?=  arm
   CROSS_COMPILE   ?=  arm-linux-androideabi-
   ```
   Save it and exit.
   c. Execute the following command in terminal to set compiling config:
   ```
   make goldfish_armv7_defconfig
   ```
   d. Modify compiling config:
   ```
   sudo apt-get install ncurses-dev
   make menuconfig
   ```
   Then you can see a GUI config.

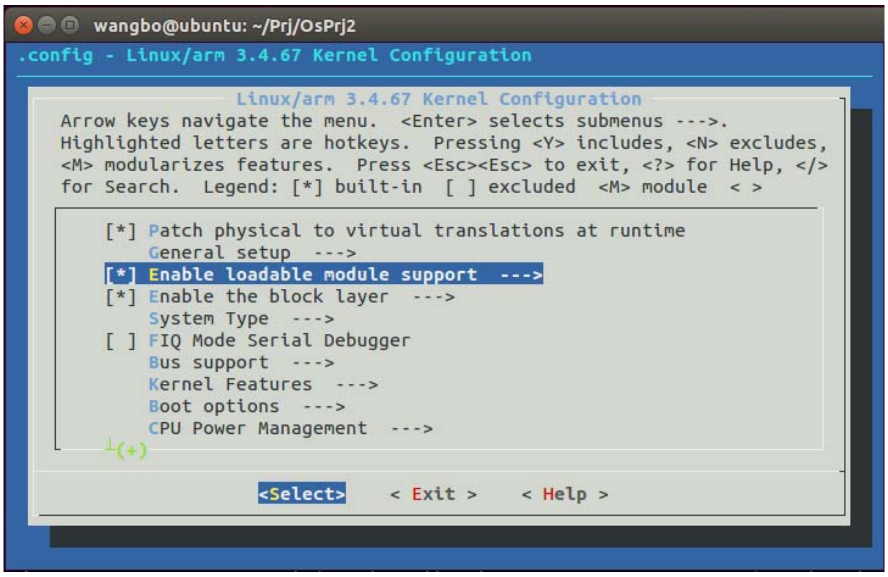


```
wangbo@ubuntu: ~/Prj/OsPrj2
.config - Linux/arm 3.4.67 Kernel Configuration
                 Linux/arm 3.4.67 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
      ^(-)
          Userspace binary formats  --->
          Power management options  --->
      [*] Networking support  --->
          Device Drivers  --->
          File systems  --->
          Kernel hacking  --->
          Security options  --->
      -*- Cryptographic API  --->
          Library routines  --->
      ---
      └(+)

              <Select>    < Exit >    < Help >
```

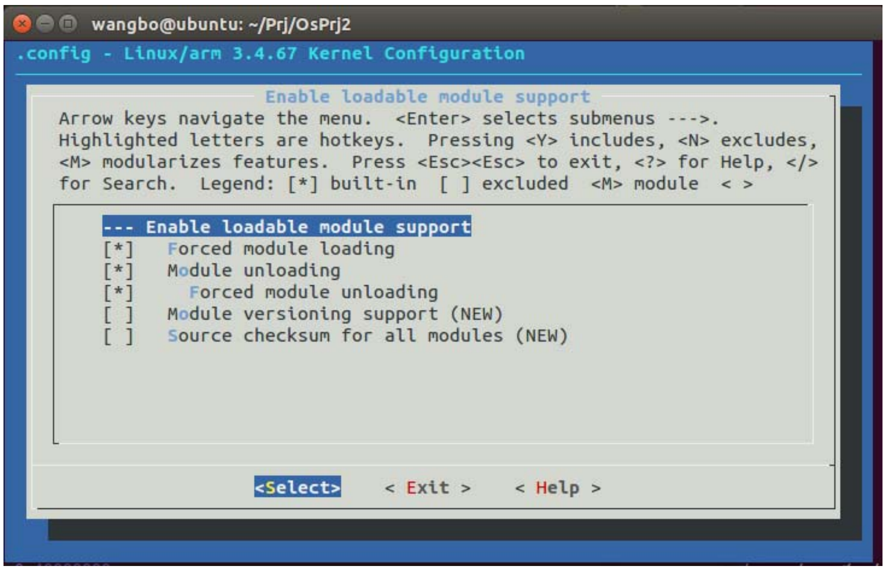

```
wangbo@ubuntu: ~/Prj/OsPrj2
.config - Linux/arm 3.4.67 Kernel Configuration
                         Kernel hacking
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
      ^(-)
      -*- Stacktrace
      [ ] Stack utilization instrumentation
      [ ] kobject debugging
      [ ] Highmem debugging
      [*] Verbose BUG() reporting (adds 70K)
      [*] Compile the kernel with debug info
      [ ]   Reduce debugging information (NEW)
      [ ] Debug VM
      [ ] Debug filesystem writers count
      [ ] Debug memory initialisation
      └(+)

              <Select>    < Exit >    < Help >
```


```
wangbo@ubuntu: ~/Prj/OsPrj2
.config - Linux/arm 3.4.67 Kernel Configuration
                Linux/arm 3.4.67 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      [*] Patch physical to virtual translations at runtime
          General setup  --->
      [*] Enable loadable module support  --->
      [*] Enable the block layer  --->
          System Type  --->
      [ ] FIQ Mode Serial Debugger
          Bus support  --->
          Kernel Features  --->
          Boot options  --->
          CPU Power Management  --->
      └(+)

              <Select>    < Exit >    < Help >
```


```
wangbo@ubuntu: ~/Prj/OsPrj2
.config - Linux/arm 3.4.67 Kernel Configuration
                    Enable loadable module support
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      --- Enable loadable module support
      [*]   Forced module loading
      [*]   Module unloading
      [*]     Forced module unloading
      [ ]   Module versioning support (NEW)
      [ ]   Source checksum for all modules (NEW)

              <Select>    < Exit >    < Help >
```

Open the *Compile the kernel with debug info* in *Kernel hacking* and *Enable loadable module support* with *Forced module loading, Module unloading and Forced module unloading* in it.

Save it and exit.

e. Compile

```
make -j4
```

The number of -j* depends on the number of cores of your system.

## 2. Implement weighted round robin scheduler.

The Linux scheduler repeatedly switches between all the running tasks on the system, attempting to give a fair amount of CPU time to each task. Fair-share group scheduling is a strategy that shares the CPU among sets of tasks according to the groups that own those tasks. In this assignment we refer to a schedulable execution unit (like a process or a thread) as "task". This was chosen in order to remain consistent with the Linux scheduler terminology.

You will add a new scheduling policy, Round-robin scheduling, to support fair-share group scheduling. **Round-robin scheduling** treats all tasks equally, but there are times when it is desirable to give some tasks preference over others. In Android, tasks are classified into different groups so that the various task groups can be handled appropriately (The Linux default scheduler also has the concept of groups but it means something different there). The group of the task is going to determine the weight that the process will be assigned. Typically, Android tasks are assigned to the foreground or background group.

- ✧ **Foreground process**: The app you're using is considered the foreground process. Other processes can also be considered foreground processes — for example, if they're interacting with the process that's currently in the foreground. There are only a few foreground processes at any given time.
- ✧ **Background process**: Background processes are not currently visible to the user. They have no impact on the experience of using the phone. At any given time, many background processes are currently running. You can think of these background processes as "paused" apps. They're kept in memory so you can quickly resume using them when you go back to them, but they aren't using valuable CPU time or other non-memory resources.

Earlier versions of Android had a system group which was meant for kernel threads. Now, you can assign kernel threads and system processes to the foreground group by default.

From user level, you can run *ps -P* on the device or emulator to check the assigned groups for each task. To get information for a specific pid, you can run *cat /proc/{pid}/cgroup*. Observe the change of a task's group when you launch an app and go back to the launcher by pressing the home button. The *add_tid_to_cgroup* function in [1] will help you to understand how Android set tasks to each group.

At the kernel level, a task's group information can be found using a task pointer. Refer to the line 96 in *kernel/sched/debug.c* and use that function appropriately. The return value will be "/" for a foreground (and system group in earlier versions) group, "/bg_non_interactive" for a background group. Whenever Android changes the group of a task, *cpu_cgroup_attach* function in *kernel/sched/core.c* will be called. By default, start with a weight of 10 for foreground processes and 1 for background processes i.e. foreground processes will have a time slice of 100ms whereas background ones will get 10ms. You should try tweaking the weights to see how they impact how responsive your device feels.

Your scheduler should operate alongside the existing Linux scheduler. Therefore, you should add a new scheduling policy, SCHED_WRR. The value of SCHED_WRR should be **6**.

Tasks using the SCHED_WRR (your scheduler) policy should take priority over tasks using the SCHED_NORMAL (or SCHED_BATCH) policy, but not over tasks using the SCHED_RR or SCHED_FIFO policies (the real-time tasks).

You are required to print some kernel information to show your process is using WRR scheduler properly.

**Implementation Details:**

1. Most of the code of interest for this assignment is in *kernel/sched.c* and *include/linux/sched.h*. These are probably not the only files you will need to look at, but they're a good start. While there is a fair amount of code in these files, a key goal of this assignment is for you to understand how to abstract the scheduler code so that you learn in detail the parts of the scheduler that are crucial for this assignment and ignore the parts that are not.
2. Finish your code in */include/linux/wrr.c*. Remember that in wrr.c, when you want to allocate time slice to a task with WRR as its policy, you should judge whether it is a foreground task or background task and allocate corresponding time slice to it.
3. Add some printk(“”) in wrr.c or some other places which proves there is a task using WRR as a policy.
4. We will provide an apk *processtest.apk* for test. You need rite a test file, which can change the scheduler in user space.
   - ✧ Change the apk’s scheduler to WRR when the apk is in foreground groups, and give out some information (pid, name, times lice, and some others you like).
   - ✧ Change the apk’s scheduler to WRR when the apk is in background groups, and give out some information (pid, name, time slice, and some others you like).
5. Any extended ideas can be considered into the bonus! Here are some of the ideas we provide, I hope you won’t be limited to these:
   - ✧ Can you come up with a method to compare the performance of RR, FIFIO, NORMAL and WRR?
   - ✧ Can you build WRR in a multi-cpu architecture and implement load balance?
6. The blog [2] relevant to linux kernel scheduler could be helpful to you.

**Material to be submitted:**

1. Compress the source code of the programs into **Prj2+StudentID.tar** file. It contains all *.c, *.h files you have changed in Linux kernel. Use meaningful names for the file so that

the contents of the file are obvious. Enclose a README file that lists the files you have submitted along with a one sentence explanation. Call it **Prj2README**.

2. Only internal documentation is needed. Please state clearly the purpose of each program at the start of the program. Add comments to explain your program. (-5 points, if insufficient.)
3. Test runs: Screen captures of the scheduler test to show that your scheduler works.
4. A project report of the project2. In this report, you should give an explanation how your WRR scheduler works. It would be better to present the analysis on the performance of different schedulers.
5. Send your **Prj2+StudentID.tar** file to TA.
6. Due date: **June 4, 2021**, submit on-line **before midnight**.
7. Demo slots: **June 5-6, 2021**. Demo slots will be posted in the WeChat group. Please sign your name in one of the available slots.
8. You are encouraged to present your design of the project optionally. The presentation date will be in the **afternoon of June 6, 2021**. Please pay attention to the WeChat group.

## References

[1]. http://androidxref.com/5.1.1_r6/xref/system/core/libcutils/sched_policy.c
[2]. https://helix979.github.io/jkoo/post/os-scheduler/