Project 1: Scheduler

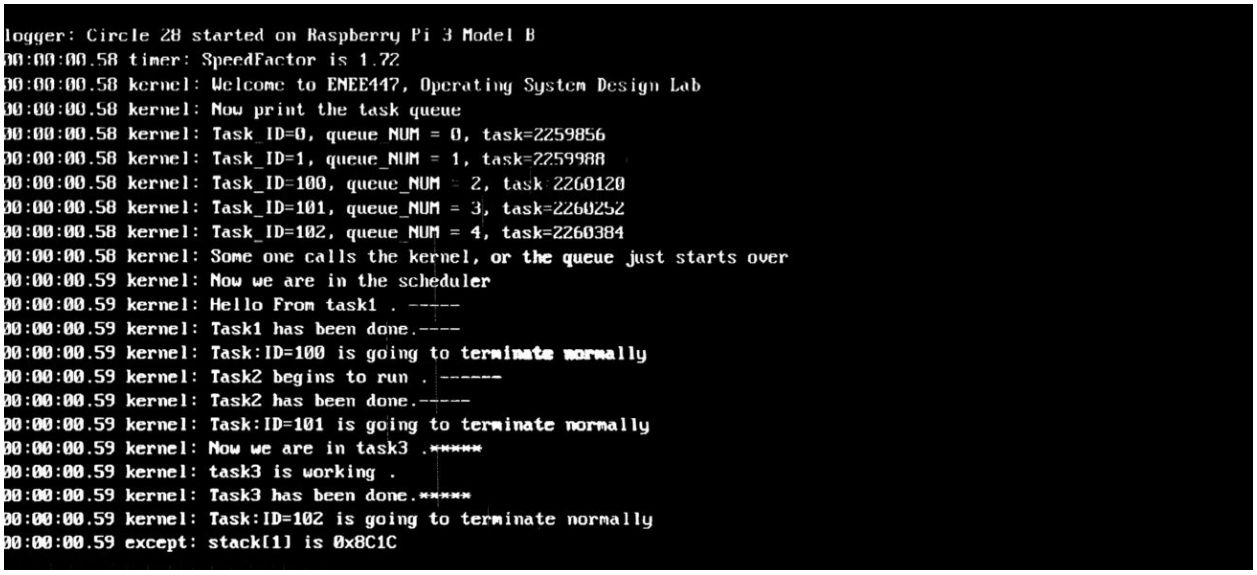
ENEE 447: Operating Systems — Spring 2017
Assigned Date: Sunday, Feb 12.
Due Date: **Sunday, Feb 19. @11:59 p.m.**

1. Purpose

In this project, you will be given some chance to have some hands on experience with scheduler in the operating system. The main goals of this lab are to help you understand what task is in OS, what happen during a context switch, how context switch work, as well as learning two scheduling algorithms.

2. Introduction

Before we get start, make sure you are able to compile and generate the image file to test run application on your RPi3. You will be able to see following on screen (through HDMI cable) if you correctly compile your code from project 1 folder and import the corresponding files and image into your RPi3.



```
logger: Circle 28 started on Raspberry Pi 3 Model B
00:00:00.58 timer: SpeedFactor is 1.72
00:00:00.58 kernel: Welcome to ENEE447, Operating System Design Lab
00:00:00.58 kernel: Now print the task queue
00:00:00.58 kernel: Task_ID=0, queue_NUM = 0, task=2259856
00:00:00.58 kernel: Task_ID=1, queue_NUM = 1, task=2259988
00:00:00.58 kernel: Task_ID=100, queue_NUM = 2, task=2260120
00:00:00.58 kernel: Task_ID=101, queue_NUM = 3, task=2260252
00:00:00.58 kernel: Task_ID=102, queue_NUM = 4, task=2260384
00:00:00.58 kernel: Some one calls the kernel, or the queue just starts over
00:00:00.59 kernel: Now we are in the scheduler
00:00:00.59 kernel: Hello From task1 . -----
00:00:00.59 kernel: Task1 has been done.----
00:00:00.59 kernel: Task:ID=100 is going to terminate normally
00:00:00.59 kernel: Task2 begins to run . ------
00:00:00.59 kernel: Task2 has been done.-----
00:00:00.59 kernel: Task:ID=101 is going to terminate normally
00:00:00.59 kernel: Now we are in task3 .*****
00:00:00.59 kernel: task3 is working .
00:00:00.59 kernel: Task3 has been done.*****
00:00:00.59 kernel: Task:ID=102 is going to terminate normally
00:00:00.59 except: stack[1] is 0x8C1C
```

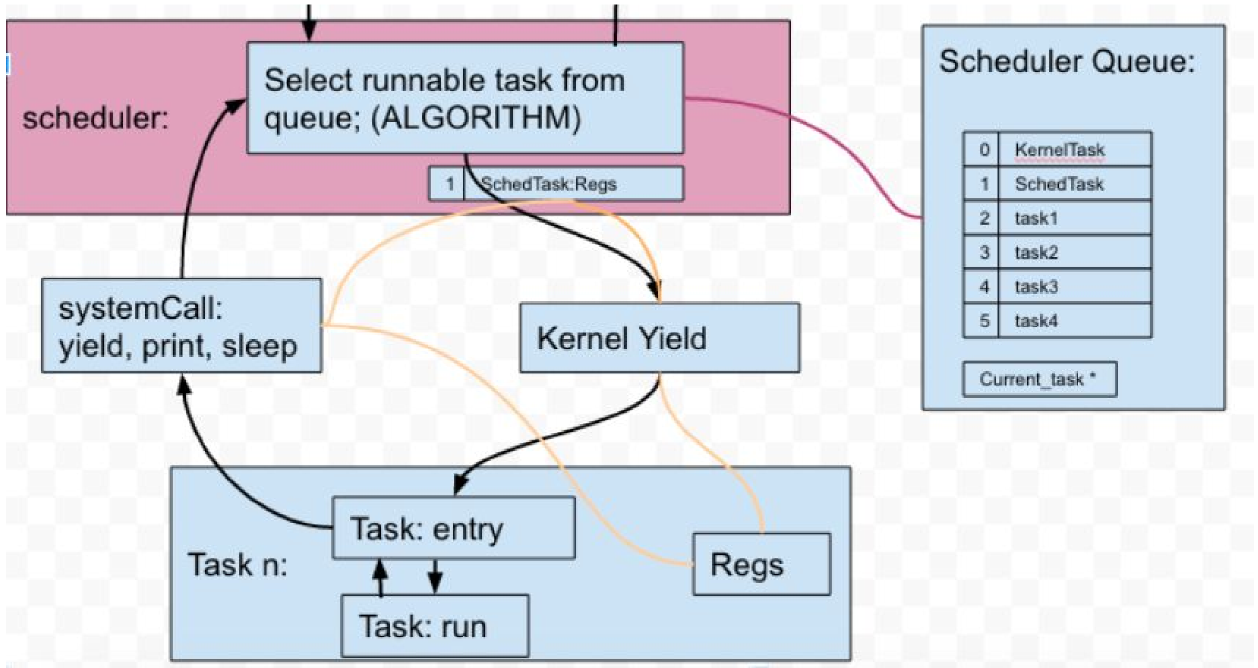
What is a scheduler?
Imaging homeworks and projects are coming to you and assume that it is not productive to multi-tasking on different tasks at the same time. We would like to organize our tasks and accomplish them one at a time. Similar to human beings, Computer (only consider single core right now), need to organize those tasks in a way so it can efficiently

accomplish all tasks. Scheduler's job is to help organize those tasks with specified policy for best efficiency.

Scheduler Implementation

Here is an overview of how everything works. Array *taskQueue[]* is used to stores all tasks inside the scheduler. The first task is always the **kernel_task** (Task_ID=0, stores in *taskQueue[0]*). The second task inside *taskQueue[1]* is the the **scheduler's task** itself (Task_ID=1). All **user tasks (**Task_ID=100+**)** can be stored in any empty space in *taskQueue[].* When the operating system boots, it prepares a kernel task as well as the scheduler task and add them into *taskQueue[]* in kernel mode. After this point, we could start to prepare any user tasks and add them to the *taskQueue[].* The scheduler we provided, will automatically skip the first two tasks (kernel and scheduler) in *taskQueue[]* and only process user tasks if exists according to the scheduling policy.

Context Switches



Context switching is the process of switching between two tasks. During context switch, two operations are performed.

1. store old task's register values, and
2. load new register values into the register files.

For example, the context switch from **kernel task** to **user task** will be done by calling *kernelYieldByNum(taskNum)* function in sched.cpp file. Context switch from **user task** to **kernel task** will be achieved by calling the *yield()* function instead. Please to Pi-OSschedulerDesign.docx for design details.

CPP

Most of the codes are provided in C++. For those of you don’t familiar with C++, no worry, you can write all your code in C, even in .cpp file. However, the ability to read and understand code in C++ is essential for future projects and career.

2.1 Scheduling algorithm

2.1.1 FIFO

The most basic algorithm we can implement is known as **First In, First Out (FIFO)** scheduling. It is very easy to understand FIFO scheduling that processor will process the task that occur first. In this project, the code for FIFO policy has already been given to you. This scheduling algorithm will sect runnable task from the queue in order (except kernel task and scheduler task). When one task is over, return control to scheduler, scheduler would select the next runnable task. When there’s no runnable in the queue, scheduler stops.

2.1.2 Priority Scheduling

This scheduling algorithm will sect runnable task according to priority. User task has a priority range of 20~99. The smaller value, the higher priority. Scheduler will always select the runnable task with the highest priority (lowest value) in the queue to run. When one task is over, return control to scheduler, scheduler would select the next runnable task. When there’s no runnable in the queue, scheduler stops.

3. Problems

Problem 1: Understand the code

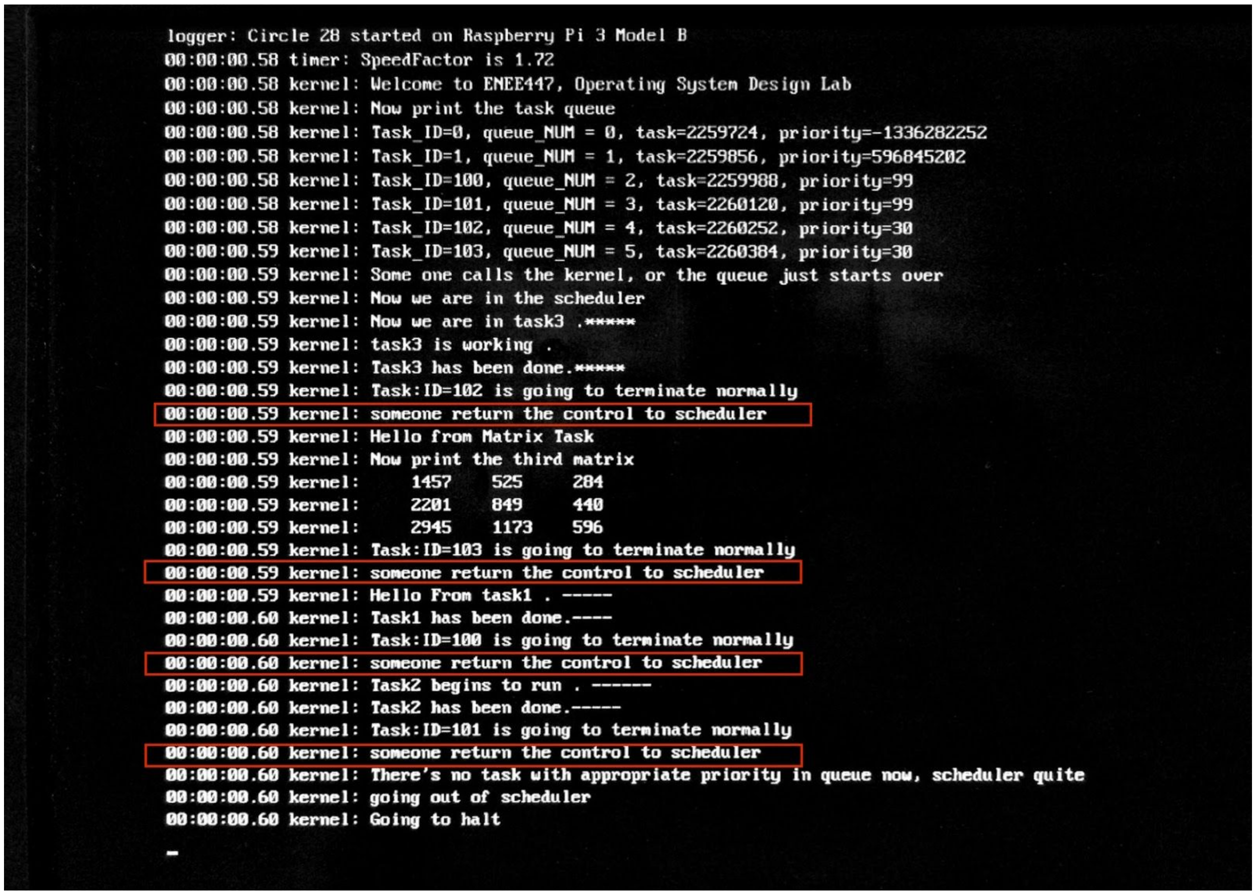
Currently, the default implementation of scheduler is running user tasks consecutively with following behavior:

Scheduler-> Task1 -> Task2 -> Task3 ….

However, this kind of behavior is incorrect. The correct behavior should be something like this:

Scheduler -> Task1 -> **Scheduler** -> Task2 -> **Scheduler-**> ...

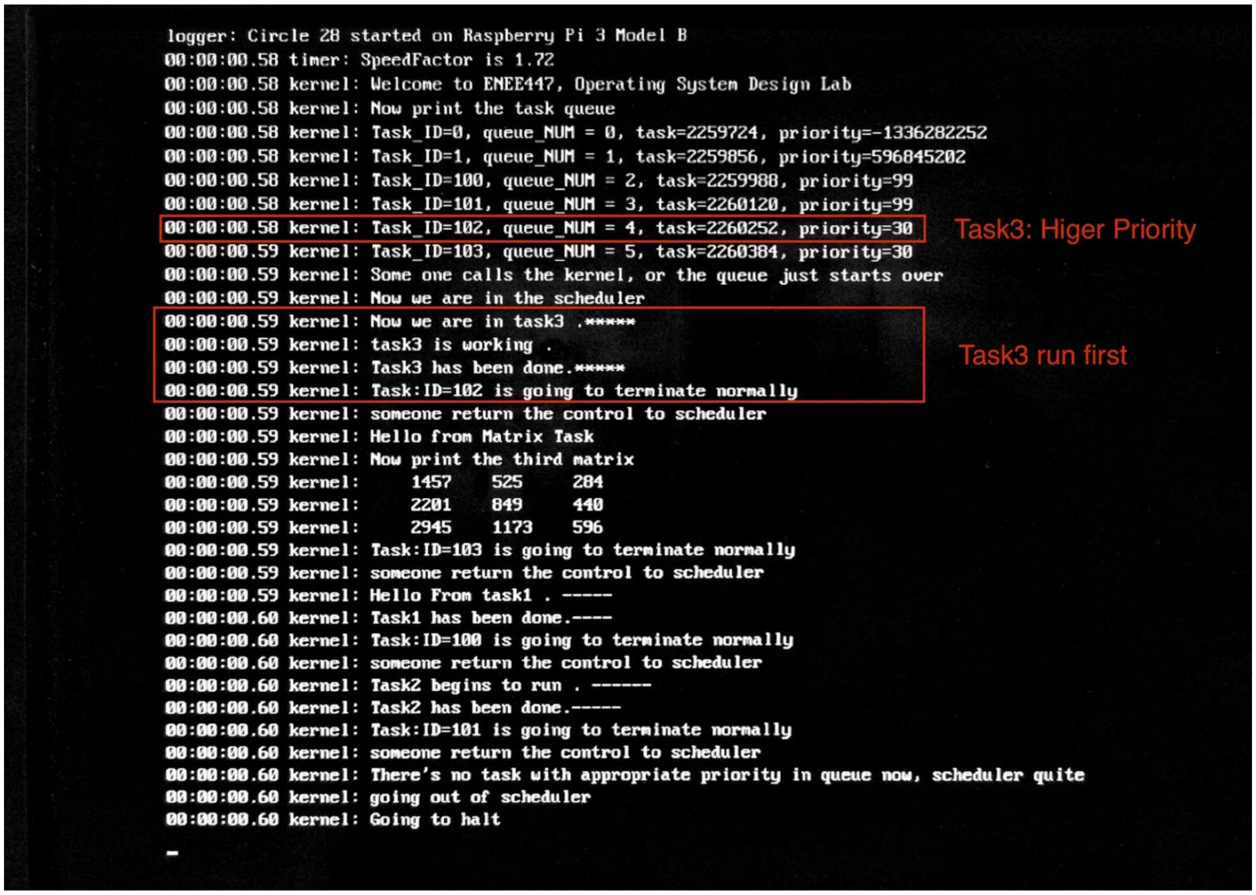
For this task, The code you need to write is very minimal. Your main goal is to understand how a scheduler work by reading through the code we provided. After you have achieve the correct behavior for the scheduler, please use FIFO scheduling policy to display your output to screen.



```
logger: Circle 28 started on Raspberry Pi 3 Model B
00:00:00.58 timer: SpeedFactor is 1.72
00:00:00.58 kernel: Welcome to ENEE447, Operating System Design Lab
00:00:00.58 kernel: Now print the task queue
00:00:00.58 kernel: Task_ID=0, queue_NUM = 0, task=2259724, priority=-1336282252
00:00:00.58 kernel: Task_ID=1, queue_NUM = 1, task=2259856, priority=596845202
00:00:00.58 kernel: Task_ID=100, queue_NUM = 2, task=2259988, priority=99
00:00:00.58 kernel: Task_ID=101, queue_NUM = 3, task=2260120, priority=99
00:00:00.58 kernel: Task_ID=102, queue_NUM = 4, task=2260252, priority=30
00:00:00.59 kernel: Task_ID=103, queue_NUM = 5, task=2260384, priority=30
00:00:00.59 kernel: Some one calls the kernel, or the queue just starts over
00:00:00.59 kernel: Now we are in the scheduler
00:00:00.59 kernel: Now we are in task3 .*****
00:00:00.59 kernel: task3 is working .
00:00:00.59 kernel: Task3 has been done.*****
00:00:00.59 kernel: Task:ID=102 is going to terminate normally
00:00:00.59 kernel: someone return the control to scheduler
00:00:00.59 kernel: Hello from Matrix Task
00:00:00.59 kernel: Now print the third matrix
00:00:00.59 kernel:     1457    525     284
00:00:00.59 kernel:     2201    849     440
00:00:00.59 kernel:     2945    1173    596
00:00:00.59 kernel: Task:ID=103 is going to terminate normally
00:00:00.59 kernel: someone return the control to scheduler
00:00:00.59 kernel: Hello From task1 . -----
00:00:00.60 kernel: Task1 has been done.----
00:00:00.60 kernel: Task:ID=100 is going to terminate normally
00:00:00.60 kernel: someone return the control to scheduler
00:00:00.60 kernel: Task2 begins to run . ------
00:00:00.60 kernel: Task2 has been done.-----
00:00:00.60 kernel: Task:ID=101 is going to terminate normally
00:00:00.60 kernel: someone return the control to scheduler
00:00:00.60 kernel: There's no task with appropriate priority in queue now, scheduler quite
00:00:00.60 kernel: going out of scheduler
00:00:00.60 kernel: Going to halt

_
```

Problem 2: Implement a priority scheduling

As we discussed at section 2.1.2, with priority scheduling algorithm, the scheduler will select the task with higher priority to run first disregard the order we added tasks. A snapshot of result of the priority scheduling is displayed below:



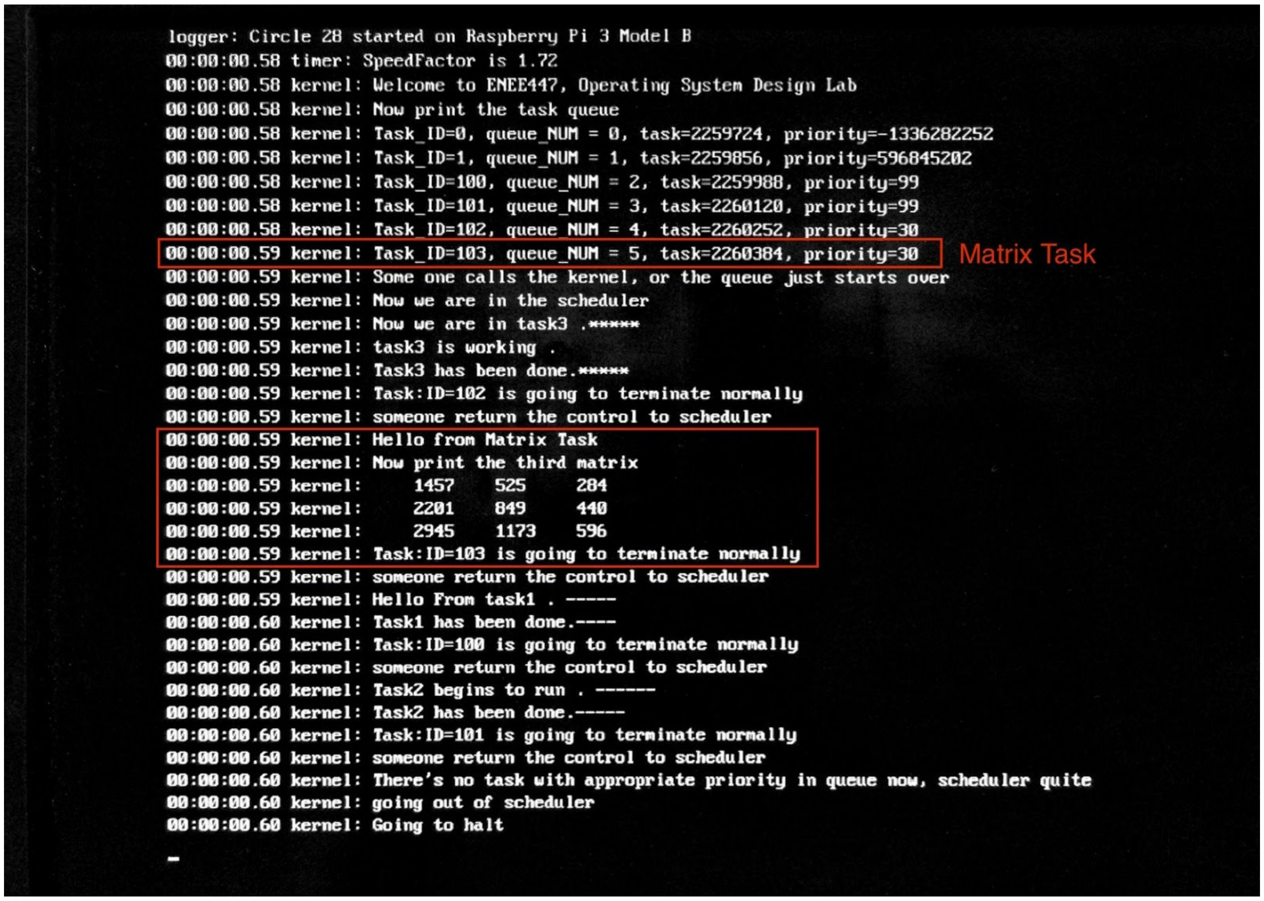
```
logger: Circle 28 started on Raspberry Pi 3 Model B
00:00:00.58 timer: SpeedFactor is 1.72
00:00:00.58 kernel: Welcome to ENEE447, Operating System Design Lab
00:00:00.58 kernel: Now print the task queue
00:00:00.58 kernel: Task_ID=0, queue_NUM = 0, task=2259724, priority=-1336282252
00:00:00.58 kernel: Task_ID=1, queue_NUM = 1, task=2259856, priority=596845202
00:00:00.58 kernel: Task_ID=100, queue_NUM = 2, task=2259988, priority=99
00:00:00.58 kernel: Task_ID=101, queue_NUM = 3, task=2260120, priority=99
00:00:00.58 kernel: Task_ID=102, queue_NUM = 4, task=2260252, priority=30
00:00:00.59 kernel: Task_ID=103, queue_NUM = 5, task=2260384, priority=30
00:00:00.59 kernel: Some one calls the kernel, or the queue just starts over
00:00:00.59 kernel: Now we are in the scheduler
00:00:00.59 kernel: Now we are in task3 .*****
00:00:00.59 kernel: task3 is working .
00:00:00.59 kernel: Task3 has been done.*****
00:00:00.59 kernel: Task:ID=102 is going to terminate normally
00:00:00.59 kernel: someone return the control to scheduler
00:00:00.59 kernel: Hello from Matrix Task
00:00:00.59 kernel: Now print the third matrix
00:00:00.59 kernel:     1457    525     284
00:00:00.59 kernel:     2201    849     440
00:00:00.59 kernel:     2945    1173    596
00:00:00.59 kernel: Task:ID=103 is going to terminate normally
00:00:00.59 kernel: someone return the control to scheduler
00:00:00.59 kernel: Hello From task1 . -----
00:00:00.60 kernel: Task1 has been done.----
00:00:00.60 kernel: Task:ID=100 is going to terminate normally
00:00:00.60 kernel: someone return the control to scheduler
00:00:00.60 kernel: Task2 begins to run . ------
00:00:00.60 kernel: Task2 has been done.-----
00:00:00.60 kernel: Task:ID=101 is going to terminate normally
00:00:00.60 kernel: someone return the control to scheduler
00:00:00.60 kernel: There's no task with appropriate priority in queue now, scheduler quite
00:00:00.60 kernel: going out of scheduler
00:00:00.60 kernel: Going to halt

_
```

Problem 3: Adding your own task to scheduler

For this task, you will be asked to implement a simple user task that is doing matrix multiplication of given two matrices.The matrix multiplication task should fulfilled following requirements:

1. Has higher priority **(priority of your task should have a value less than 99)** comparing to other user tasks
2. Implement an algorithm to perform matrix multiplication of two given matrices.

Make sure you scheduler select and perform the task with higher priority first. A snapshot of the priority scheduling with matrix multiplication task is displayed below:



```
logger: Circle 28 started on Raspberry Pi 3 Model B
00:00:00.58 timer: SpeedFactor is 1.72
00:00:00.58 kernel: Welcome to ENEE447, Operating System Design Lab
00:00:00.58 kernel: Now print the task queue
00:00:00.58 kernel: Task_ID=0, queue_NUM = 0, task=2259724, priority=-1336282252
00:00:00.58 kernel: Task_ID=1, queue_NUM = 1, task=2259856, priority=596845202
00:00:00.58 kernel: Task_ID=100, queue_NUM = 2, task=2259988, priority=99
00:00:00.58 kernel: Task_ID=101, queue_NUM = 3, task=2260120, priority=99
00:00:00.58 kernel: Task_ID=102, queue_NUM = 4, task=2260252, priority=30
00:00:00.59 kernel: Task_ID=103, queue_NUM = 5, task=2260384, priority=30
00:00:00.59 kernel: Some one calls the kernel, or the queue just starts over
00:00:00.59 kernel: Now we are in the scheduler
00:00:00.59 kernel: Now we are in task3 .*****
00:00:00.59 kernel: task3 is working .
00:00:00.59 kernel: Task3 has been done.*****
00:00:00.59 kernel: Task:ID=102 is going to terminate normally
00:00:00.59 kernel: someone return the control to scheduler
00:00:00.59 kernel: Hello from Matrix Task
00:00:00.59 kernel: Now print the third matrix
00:00:00.59 kernel:     1457    525     284
00:00:00.59 kernel:     2201    849     440
00:00:00.59 kernel:     2945    1173    596
00:00:00.59 kernel: Task:ID=103 is going to terminate normally
00:00:00.59 kernel: someone return the control to scheduler
00:00:00.59 kernel: Hello From task1 . -----
00:00:00.60 kernel: Task1 has been done.----
00:00:00.60 kernel: Task:ID=100 is going to terminate normally
00:00:00.60 kernel: someone return the control to scheduler
00:00:00.60 kernel: Task2 begins to run . ------
00:00:00.60 kernel: Task2 has been done.-----
00:00:00.60 kernel: Task:ID=101 is going to terminate normally
00:00:00.60 kernel: someone return the control to scheduler
00:00:00.60 kernel: There's no task with appropriate priority in queue now, scheduler quite
00:00:00.60 kernel: going out of scheduler
00:00:00.60 kernel: Going to halt
_
```

4. Submission

Plan your time accordingly, **no extension**, any late submission (after Feb 19 11:59pm) will result in penalty. If you submit one day late, 10% will be deducted from your project 1 score. If you submit two days late, 20% will be deducted, etc.

Zip all your files:

- Design/Documentation Report in PDF format
- All Project files, make sure compilable.

During the lab section following due date, you will demonstrate the results to your TA.

5. Honor Code

Never ruin others fun and never show your code to others!