CS 460 – Programming Assignment 2
Implement A Round-Robin Scheduler
(Due midnight Friday 2/20/2015)

## 1. Overview

In this programming assignment, you will implement a round-robin scheduler for threads and use context-switch functions that you implemented in the first assignment. *When a thread calls yield(), it will put itself at the end of a queue of waiting threads and pull the next thread off that queue*.

The assignment must be done on lab machines (lx.encs.vancouver.wsu.edu) using your encs login account. This assignment can be done individually or with a team of two.

Example binaries are also included so you know what the output programs should be.

## 2. Instructions

Login to lx.encs.vancouver.wsu.edu and copy `/encs_share/cs/class/cs460/proj2` to your home directory. The directory includes

- Makefile
- queue.h & queue.c : ADT for thread library
- scheduler.h: header file specifying scheduler functions and thread states
- scheduler.c: ***You are to complete this file***
- main*.c: Test files with main functions

Copy your thread_start.s and thread_switch.s to this folder.

We extend the thread table entry data structure to include a value of an enumeration that the scheduler can use to make decisions about what to do with a thread. The thread states are defined as an enumeration in scheduler.h

```
typedef enum {
  RUNNING,
  READY,
  BLOCKED,
  DONE
} state_t;
```

**You are to complete the scheduler.c file to implement functions defined in scheduler.h.**

**2.1. thread_start and thread_finish**: we'll need to make some slight modifications to our thread_start assembly routine.

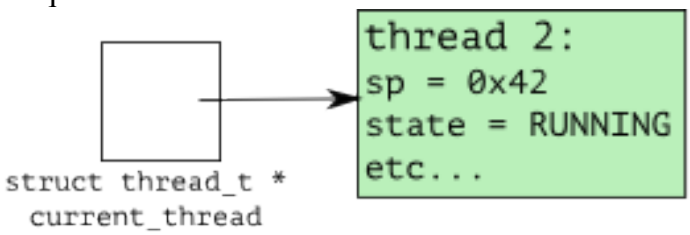
***Question 1*: How can you solve the problem of allowing a thread to terminate gracefully (no segmentation fault) in assignment 1?**

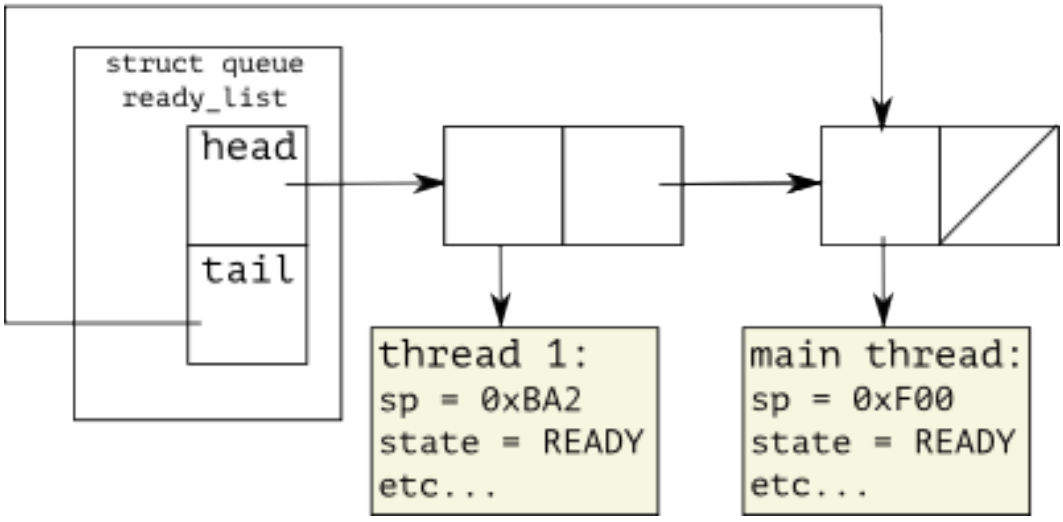
Implement the thread_finish function accordingly, then modify your thread_start routine to push the address of thread_finish on the bottom of the new thread's stack. This can be accomplished with the instruction:

pushq $thread_finish.

We'll also want to pass our initial argument to the target function. We'd also like to be able to pass parameters to the initial function of a thread. Add a new field, void * initial_arg, and change the type signature of the initial function to void(*initial_func)(void*). The polymorphic void* argument type will allow our functions to take arbitrary parameters. Once you're done with %rdi, you can overwrite it with the value of the initial argument.

**2.2.** **scheduler_begin**: it should initialize and/or allocate any data structures the scheduler will need. Here's an idea of how you might lay out your scheduler, using the provided queue ADT:




**2.3.** **thread_fork**: This function encapsulates everything necessary to allocate a new thread and then jump to it. thread_fork should:

- Allocate a new thread table entry, and allocate its control stack.
- Set the new thread's initial argument and initial function.

- Set the current thread's state to READY and enqueue it on the ready list.
- Set the new thread's state to RUNNING.
- Save a pointer to the current thread in a temporary variable, then set the current thread to the new thread.
- Call thread_start with the old current thread as old and the new current thread as new.

**2.4. yield**: Yield is very similar to thread_fork, with the main difference being that it is pulling the next thread to run off of the ready list instead of creating it. yield should:

- If the current thread is not DONE, set its state to READY and enqueue it on the ready list.
- Dequeue the next thread from the ready list and set its state to RUNNING.
- Save a pointer to the current thread in a temporary variable, then set the current thread to the next thread.
- Call thread_switch with the old current thread as old and the new current thread as new.

**2.5. scheduler_end**: recall that we need a way to prevent the main thread from terminating prematurely if there are other threads still running.

***Question 2*: How to solve the premature termination problem in assignment 1**?

Implement the solution you see fit in scheduler_end.

Test your code with the given main files!

## 1. Submission guidelines and grading criteria

### a. Submission

Please provide sufficient comments for your code. Clearly indicate if a step is incomplete.

Please zip all files

- A report file that includes test results
- The code, and
- A README file that describes how to run your code.

Submit the zip file on prog2 dropbox on Angel (Those having difficulties accessing Angel can send the file to the TA's email address: solongo.munkhjargal@email.wsu.edu)

The code is expected to run on the lab machines (lx.encs.vancouver.wsu.edu).

### b. Grading criteria

| Criteria | Points (/100) |
|---|---|
| Step 1 | 20 |

| | |
|---|---|
| Step 2 | 20 |
| Step 3 | 20 |
| Step 4 | 20 |
| Step 2 | 20 |