Operating Systems Assignment 1
Fahad Syed, Travis Thiel, Austin Tsai

Design Decisions:
The scheduler was written with the following implementation and design decisions:
1) Our multi-level priority queue has 3 priorities: short allocates 8 milliseconds to a thread as a time slice. medium allocates 16 milliseconds as a time slice. FIFO is our lowest priority and runs each task to completion. This model was based off of an example in Dr. Francisco's lecture slides.
2) Yield is our scheduler. Most (but not all) scheduling occurs in yield. Yield calls a helper function "run_from_queue" which will schedule from one of our three threads probabilistically. 50% of the time, we schedule from the short queue. 30% of the time, we schedule from the medium queue. 20% of the time, we schedule from the FIFO queue. Random numbers are generated using the random_0_100 helper functions.
3) To decrease scheduling overhead, some scheduling state is safely modified in create(), exit(), and join(). Although it is possible to call our scheduler and read previous state information from global variables, it was faster to do some scheduling tasks within their respective functions than to pass the necessary references and values to a separate struct where yield would read them and do the state modification. To ensure state is safely modified, a variable named schedule_lock is used. Yield will return if schedule_lock is 1. This prevents yield from performing its scheduling routines while queues are being modified which could lead to undefined behavior.
4) my_pthread_create() adds a created thread to the run queue. my_pthread_exit() adds the current thread to the terminated queue and calls yield(). my_pthread_join() puts the current thread on the waiting queue if the target thread is still running.
5) Maintenance does not occur in cycles, but instead triggers when a thread has been waiting longer than 1 second to finish. In this case, the thread is dequeued from its current queue and enqueued onto the short queue.
6) Intentional yield calls will not demote a thread to a lower queue so long as their is sufficient time remaining within the slice. This is to prevent a thread from taking advantage of yield to prevent being put on a lower queue.

Debug Testing:
To test our scheduler, we used a file called pthread_test.c. This file was compiled and altered as new functionality was added to the scheduling library. We used it to test swapping, thread promotion, thread demotion, different time slices, and benchmarking. The scheduler was tested with 4 threads (including main) serially in FIFO to test joining and return values. The scheduler was also tested with 4 threads using a multi-level priority queue to determine how well the library could handle 4 threads running simultaneously. test_func2 and main purposefully take to complete in order to test thread demotion to longer queues and to test exiting before joining.

Debug Output:
The functions within the scheduling library contain lines of debug output for testing and benchmarking that are currently
commented out. We found these output lines useful to debugging as gdb didn't play nicely with multi-threaded environments.

Benchmarks:
Here are our benchmarks from running parallelCal 6:

```
running time: 2092 micro-seconds
sum is: 83842816
verified sum is: 83842816
```