In lecture 4, we learned that threads must be scheduled by the operating system so that each gets to run on one of the CPU cores. This is the job of the *scheduler*. A simple thread scheduling algorithm is called *round-robin*. Given a pool of threads ready to run, each thread gets a slice of time to run on a processor core before the scheduler saves the thread state and replaces it by the next thread in the pool. At the end of the pool, the schedule begins again at the beginning.

This assignment demonstrates simple round-robin thread scheduling. After creating the specified number of threads, the program manages the threads round-robin fashion (0, 1, 2, .. n-1, 0, 1, 2, ..). That selected thread's function prints a running message at 1-second intervals during its time slice, while other threads silently loop and wait their turn. A **SIGINT** handler sets a *running* flag to *false*, causing all the threads to terminate and thread management to complete. The program then waits for the threads to terminate before exiting.

The 2019S1CS5007/assignment-4 GitHub repository has a file, "manage_threads.c" with a working *main* function, and functions that you will implement to create and manage the threads and wait for them to terminate. You will also implement a thread function that loops and prints during its interval, and a **SIGINT** function that handles the signal. You are welcome to use code snippets from in-class examples for creating threads and waiting for them to terminate, and for setting up the signal handler.

The thread funcion should print the process ID along with the thread number for the selected thread so that you can use the *kill*program from a *bash* shell to send the program a **SIGINT** signal. Here is some sample output from the program with three threads. The SIGINT signal was sent at the end of the second round-robin cycle.

```
Creating 3 threads
Managing 3 threads
pid: 5774 thread 0 running
pid: 5774 thread 0 running
pid: 5774 thread 0 running
pid: 5774 thread 0 running
pid: 5774 thread 1 running
pid: 5774 thread 1 running
pid: 5774 thread 2 running
pid: 5774 thread 2 running
pid: 5774 thread 2 running
pid: 5774 thread 0 running
pid: 5774 thread 0 running
pid: 5774 thread 1 running
pid: 5774 thread 1 running
pid: 5774 thread 1 running
pid: 5774 thread 2 running
pid: 5774 thread 2 running
Received SIGINT: notifying threads to stop.
Waiting for 3 threads to terminate
pid 5774: thread 1 terminating
pid 5774: thread 0 terminating
pid 5774: thread 2 terminating
thread 0 exited with status 0
thread 1 exited with status 0
thread 2 exited with status 0
3 threads have terminated -- exiting
```

One note on thread creation. The *runner* thread function takes a parameter that is an integer thread number corresponding to the thread index in the thread array for printing purposes. Since the *runner* thread function requires a pointer, the *createThreads*function "pointer-encodes" the unsigned thread number using

the expression *(**NULL**+thread_no)*, and *runner* extracts the thread number from its parameter using the expression *(param-**NULL**)*. This is a well-known trick in *pthread* programming for passing an unsigned integer parameter to the thread function.