# Project - 1
# CSE 4/521 – Introduction to Operating Systems
# Due: July 15th @11:59pm, 2018

**Objective:**
Implement correct Thread scheduling functionality in PintOS

*Submission Deadline:* There are 6 soft deadlines and one hard deadline for Project 1.

- The hard deadline is **July 15th, 2018**.
- Every team has a buffer time of 5 days to work around the hard deadline for Project 1 and Project 2 combined.
- If a team misses the soft deadline for any of the 3 individual components of Project 1, that team would have to meet me in person.
- A total of 6 soft deadlines and 1 hard deadline exists for this project.
- The design doc soft deadline is for submission in-class.

*Grading*: (Your `make grade` score + 5% of your score for design doc) would be your final grade for Project 1.
Please note the academic integrity policy at: http://academicintegrity.buffalo.edu/policies

*QUICK LINKS*:
PintOS reference website - https://web.stanford.edu/class/cs140/projects/pintos/pintos_1.html
Autograder - https://autograder.cse.buffalo.edu/

## Bird's eye description of thread scheduling in PintOS:

PintOS already implements thread creation and thread completion, a simple scheduler to switch between threads, and synchronization primitives.

When a thread is created, you are creating a new context to be scheduled. You provide a function to be run in this context as an argument to `thread_create()`. The first time the thread is scheduled and runs, it starts from the beginning of that function and executes in that context. When the function returns, the thread terminates. Each thread, therefore, acts like a mini-program running inside PintOS.

At any given time, exactly one thread runs and the rest, if any, become inactive. The *scheduler* decides which thread to run next. (If no thread is ready to run at any given time, then the *idle* thread runs.) Synchronization primitives can force context switches when one thread needs to wait for another thread to do something.

Use semaphores, locks, and condition variables to solve your synchronization problems. For the alarm clock, the timer interrupt needs to wake up sleeping threads. In the mlfq scheduler implementation, the timer interrupt needs to access a few global and per-thread variables.

**Begin: Crux of the project (20th June, 2018)**

---

**Sub-checkpoint (1/3): Alarm clock**
*Design Doc Deadline*: **25th June** *Code Deadline*: **28th June**

Design Doc template – https://jerryajay.com/wp-content/uploads/2018/06/design-doc-1.txt

Re-implement timer_sleep(), defined in “devices/timer.c”. Although a working implementation is provided, it "busy waits", that is, it spins in a loop checking the current time and calling thread_yield() until enough time has gone by. Re-implement it to avoid busy waiting.

End point: All 6 alarm clock tests passed. ***(18 points)***

---

**Sub-checkpoint (2/3): Priority Scheduling**
*Design Doc Deadline*: **2th July** *Code Deadline*: **9th July**

Design Doc template – https://jerryajay.com/wp-content/uploads/2018/06/design-doc-2.txt

Implement priority scheduling in PintOS. When a thread is added to the ready list that has a higher priority than the currently running thread, the current thread should immediately yield the processor to the new thread. Similarly, when threads are waiting for a lock, semaphore, or condition variable, the highest priority waiting thread should be awakened first. A thread may raise or lower its own priority at any time, but lowering its priority such that it no longer has the highest priority must cause it to immediately yield the CPU. Make sure to address *priority inversion*- solve this by implementing *priority donation*. Your code would need to account for all different situations in which priority donation is required.

End point: All 12 priority scheduling tests passed. ***(38 points)***

---

**Sub-checkpoint (3/3): Multilevel Feedback Queue Scheduler**
*Design Doc Deadline*: **6th July** *Code Deadline*: **15th July**

Design Doc template – https://jerryajay.com/wp-content/uploads/2018/06/design-doc-3.txt

Implement a multilevel feedback queue scheduler to reduce the average response time. Have the priority scheduler working, except possibly for priority donation, before you start work on the mlfq scheduler.

You must write your code to allow us to choose a scheduling algorithm policy at PintOS startup time. By default, the priority scheduler must be active, but we must be able to choose the mlfq scheduler with the “-mlfqs” kernel option. Passing this option sets thread_mlfqs, declared in “threads/thread.h”, to true when the options are parsed by parse_options() in main().

When the mlfq scheduler is enabled, threads no longer directly control their own priorities. The *priority* argument to thread_create() should be ignored, as well as any calls to thread_set_priority(), and thread_get_priority() should return the thread's current priority set by the scheduler.

End Point: All 9 tests of mlfq scheduling tests passed. ***(37 points)***

---

**End: Crux of the Project (15th July, 2018)**

**Appendix:**

## A. Submission Procedure

Navigate to your pintOS repository folder (i.e. ~/pintos/pa1-foobar/)

1. Run the following commands:
   ```
   cd src; make clean; cd ..; tar -czf pa1.tar.gz src/
   ```
2. This will clean your src directory, and provide you with an archive file that you will need to submit.
3. Open your browser, and navigate to: https://autograder.cse.buffalo.edu/
4. Sign in using your UB credentials.
5. Under Courses choose CSE421/521: Operating Systems (u18).
6. Complete your account information if necessary.
7. Now go to the course page and choose Project 1.
8. Under Options, click on Group options.
9. Here you should form your groups by creating it, inviting your members, and then confirming the invitation.
10. All members of a group should confirm their membership, before any of them can submit the assignment.
11. Hit SUBMIT and upload pa1.tar.gz file that you created. Note that once a member submits, the submission and score will be applied to all group members automatically.
12. Once the job is processed (might take some time), your submission and score will be shown in Project 1 page. When you click on the score, it will show you the full result from the job.

Please make an early submission, so that you can make sure everything is right, and the submission works well. Note that you can submit as many times as you would like, but only your last submission will define your score. In case your last submission is not the best one, you can access all of your submission files, their scores, and their feedback messages from the Project 1 history page, you can also download them and resubmit if needed.

## B. Credits

Prof. Tevfik Kosar (University at Buffalo, NY)
Mr. Farshad Ghanei (University at Buffalo, NY)