1. Describe how you implemented the program in detail. (10%)
   In main thread:

   First use getopt() in while loop to get the input arguments from command line and use a struct datatype to store the arguments. Second, create pthread array, pthread attributes and schedule parameters used for latter. Third, initializing a pthread barrier and setting up the CPU affinity. Fourth, setting the pthread priority, schedule policy and creating pthread. Fifth, the main thread call join() to wait all the threads it created.

   In worker thread:

   First, waiting at barrier to wait for other threads. Second, printing the information according to their schedule policy and priority.

2. Describe the results of sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30 and what causes that. (10%)
   The result :
   Thread 2 is starting
   Thread 2 is starting
   Thread 2 is starting
   Thread 1 is starting
   Thread 1 is starting
   Thread 1 is starting
   Thread 0 is starting
   Thread 0 is starting
   Thread 0 is starting

   Thread1 and thread 2 schedule policy are FIFO, which has higher priority than NORMAL, besides, thread 2 has higher priority than thread 1, as a result , thread 2 executes first, followed by thread 1 , finally thread 0.
3. Describe the results of sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30, and what causes that. (10%)
   The result:

   Thread 3 is starting

Thread 3 is starting
Thread 3 is starting
Thread 1 is starting
Thread 1 is starting
Thread 1 is starting
Thread 0 is starting
Thread 2 is starting
Thread 0 is starting
Thread 2 is starting
Thread 0 is starting
Thread 2 is starting

thread 1 and thread 3 schedule policy are FIFO, which has high priority than NORMAL, besides, thread 3 has higher priority than thread 1 (30 vs 10). Thread 0 and thread 2 are SCHED_NORMAL and their nice values are -1, according to CFS, they will be executed interleaving.

4. Describe how did you implement n-second-busy-waiting? (10%)
   I use the clock() function to record the current cpu clock, the variable CLOCKS_PER_SEC defines how many clocks in a second, so if the current clock() – previous clock() >= busy waiting time*CLOCKS_PER_SEC, the thread completed the busy waiting.
5. What does the kernel.sched_rt_runtime_us effect? If this setting is changed, what will happen?(10%)
   It controls how much CPU time is allocated to real time tasks in each period. If it changes to -1, the real time tasks will get 100% CPU time in each period.