l 6% of your final grade

l Individual lab, not a team lab. Collaboration is not allowed.

l Comply with Academic Integrity of University of Pittsburgh. Similar lab assignments are found, zero for the lab score will be applied to every student involved!

Video: https://canvas.pitt.edu/files/8858892/download?download_frd=1
Slides: https://canvas.pitt.edu/files/8858893/download?download_frd=1
Example Code:
https://canvas.pitt.edu/files/8858895/download?download_frd=1
https://canvas.pitt.edu/files/8858894/download?download_frd=1

Lab 4 consists of 3 sections.

Section 1 (30%)

You are required to realize rate monotonic scheduling algorithm.

1. Please read rms.c file in detail. Try to use following example as input to see how sample code will work:

	Execution	Period
P1	10	25
P2	15	60

The sample code does not consider preemptive scenarios. Your task is to modify the sample code so that it could schedule well while preemptive scenarios occur. You can use the following example to test if your modified code could handle preemptive situation.

	Execution	Period
P1	20	50

P2	35	100

Section 2 (30%)

You are required to realize earliest deadline first scheduling algorithm.

1. Please also read rms.c file in detail. Based on rms.c file, think about how to modify the judgement so as to realize EDF algorithm. You can try the following sample to see if your modified version works well:

	Execution	Period
P1	10	20
P2	25	50

Please double check with the following example to see if your algorithm could handle preemptive scenario:

	Execution	Period
P1	25	50
P2	35	80

Section 3 (40%)

You are required to implement a program that can demonstrate priority inheritance.

1. In this section, you should create at least three threads to realize priority inheritance. The design could be as following

```
void function()
{
  //do some work here.
}

void function1()
{
  //function 1 for thread 1.
  //do some work here
}

void function2()
{
//function 2 for thread 2.
//do some work here and call function()
}

void function3()
{
//function 3 for thread 3.
//do some work here
}

int main()
{
   //create mutex and initialize it.
//create thread 1, thread 2 and thread 3
```

//set different priority for each thread

//set handler for thread 1, thread 2 and thread 3 respectively. For instance, function 1 for thread 1, function 2 for thread 2 and function 3 for thread 3

}

1. For "do some work" in design chart, you may choose to print some messages to terminal. For example: "Thread 1 in", "Thread 1 starts", "Thread 1 ends" and "Thread 1 out" are used to simply describe the work each thread will do.
2. Please think carefully on what priority inheritance is, e.g., the advent order of each thread. For example, if thread 1 has the lowest priority and thread 3 has the highest priority. The definition of priority inheritance is that thread 1 should be earlier than thread 3 and thread 2 is later than thread 3.
3. General recommendations:
4. Run your program with privilege (run as "root" or using "sudo"). This may circumvent permission issues (if any) when configuring thread priorities.
5. Consider whether limiting the number of CPU cores your program may use is necessary. You may limit it by setting CPU affinity and disallow placing multiple threads on different CPU cores.
6. The following libraries, data types and functions may be useful when writing your code:
7. Library and header files:

"stdio.h", "stdlib.h", "unistd.h", "pthread.h" and "sched.h"

1. Data types:

pthread_t: thread variable

pthread_mutex_t: mutex variabl

pthread_mutexattr_t: mutex attribute variable

sched_param: structure variable for buffering thread priority

1. Functions:

pthread_mutexattr_setprotocol(): set protocol for mutex attribute variable

pthread_mutex_init(): initialize mutex variable

pthread_mutex_lock(): lock mutex variable

pthread_mutex_unlock():unlock mutex variable

pthread_create(): create a thread

pthread_setschedparam(): set priority, policy for a thread

pthread_getschedparam(): get priority, policy of a thread

pthread_join(): wait for all threads to finish before main() can exit

pthread_setaffinity_np, pthread_getaffinity_np: get/get CPU affinity of a thread

sched_setaffinity, sched_getaffinity - set and get a thread's CPU affinity mask