Prof. Jingke Li (FAB 120-06, lij@pdx.edu); Class: TR 16:40-17:55 @ ASRC 230; Lab: F 10:30-11:50 @ FAB 88-10.

# Assignment 1: Programming with Pthreads
## (Due Tuesday, 1/30/18)

This assignment is to practice multi-threaded programming using C with the Pthreads library. You'll write a task-queue based Pthreads program, compile and run it on the CS Linux system (`linuxlab.cs.pdx.edu`). CS515 students will further write a second program. This assignment carries a total of 10 points.

Download the file `assign1.zip` to your CS Linux account and unzip it. You'll see several program files: `task-queue.h`, `task-queue.c`, and `prime.c`, plus a set of supporting files: a `Makefile`; two run scripts `run1` and `run2`, and two sample outputs `sample1.txt` and `sample2.txt`.

### 0. Task Queue Representation and Supporting Routines

Your program will use the task-queue representation given in the file `task-queue.h`:

```
typedef struct task_ task_t;
struct task_ {
  int val;        // content
  task_t *next;   // pointer to next task
};

struct queue_ {
  task_t *head;   // head pointer
  task_t *tail;   // tail pointer
  int limit;      // queue capacity (0 means unlimited)
  int length;     // number of tasks in queue
};
```

Each task holds an integer content and a pointer, which can be linked to another task. A queue has two pointers, head and tail, and two integer fields. There are four task-queue routines, defined in the file `task-queue.c`:

```
extern task_t *create_task(int val);
extern queue_t *init_queue(int limit);
extern int add_task(queue_t *queue, task_t *task);
extern task_t *remove_task(queue_t *queue);
```

To use the task-queue code, add a header line `#include "task-queue.h"` in your program.

### 1. Producer-Consumer Program (Required for All Students)

Your task is to implement a producer-consumer program, `prodcons-pthd.c`. You may use code samples from this week's lecture as a starting template.

### Program Specifications

- The program takes an *optional* command-line argument, `numCons`, which represents the number of consumer threads. If this argument is not provided, a default value of 1 is used.

```
linux> ./prodcons-pthd 10    // 10 consumer threads
linux> ./prodcons-pthd       // 1  consumer thread
```

- The `main` routine creates threads: one producer and `numCons` consumers.
- The producer generates 100 total tasks, each identified by an unique integer `i`, where $1 \leq$ `i` $\leq 100$. This value is stored in the `val` field of a task record.
- The producer enters these tasks onto the task queue, one at a time. The task queue is set to have a fixed capacity of 20 tasks.
- The consumer threads compete to remove tasks off the task queue, one at a time. Each consumer thread keeps track of how many tasks it has successfully obtained.
- The program terminates properly after all tasks are done.

### Thread Termination

A challenging part of this program is the handling of the termination of consumer threads. The following are two possible approaches:

- The producer creates a bogus "termination" task and add `numCons` copies of it to the global queue. Upon receiving such a task, a consumer thread will termination itself by breaking out its infinite loop.
- Use a global count to keep track of the completed tasks. The consumer threads all participate in updating and monitoring the global count. When the count reaches the total number of tasks, all threads terminate.

You may use either of these approaches, or a totally different approach of your own.

### Output Requirements

- Each producer/consumer thread should print out a message at both the beginning and the end of its execution:

```
Producer starting on core 3
Consumer[1] starting on core 5
Consumer[2] starting on core 2
...
Producer ending
Comsumer[2] ending
Comsumer[1] ending
...
```

  The starting message should contain the id of the CPU core it is running on. (There is no requirement on where threads should run, *i.e.* no need to control CPU affinity.) The consumer threads should include their own id in the messages.

- The program should print out a final message showing the distribution of tasks over threads, and the total number of tasks computed from the distribution data:

```
Total tasks across threads: 100
C[ 0]:14, C[ 1]:19, C[ 2]: 9, C[ 3]:21,
C[ 4]: 6, C[ 5]: 9, C[ 6]: 5, C[ 7]:17,
```

The numbers inside the brackets are consumer ids, and the numbers outside are the number of tasks each consumer thread has obtained. The total is the sum of the individual counts. (It should match the input parameter set at the beginning of the program.) The file `sample1.txt` contains sample outputs for this program.

## 2. Prime-Finding Program (Required for CS515 Students)

**[3 extra points for CS415 students.]** This part is to implement a prime-finding program, `prime-pthd.c`, using the producer-consumer infrastructure. It is an extension to the producer-consumer program. Make sure you have that program completed first.

This new program has the following specifications:

- The `main` routine reads in two command-line arguments. The first argument, `N`, is the array size, and the second (optional) argument, `numCons`, represents the number of consumer threads (the same as in the previous program):

```
linux> ./prime-pthd 80 10    // array[80], 10 consumer threads
linux> ./prime-pthd 50       // array[50], 1  consumer thread
```

- The `main` routine prepares an array of the given size with random integer values, and creates one producer and `numCons` consumer threads. (For array initialization, you may copy the `init_array` routine from the provided program, `prime.c`.) It also initializes a `result` array for keeping primes.
- The producer thread creates and enters tasks onto the task queue. Each task represents a block of array elements; the block size is fixed at 10. Therefore, task 1 represents the array range `a[0]-a[9]`, task 2 represent the array range `a[10]-a[19]`, and so on. The array size parameter `N` is required to be a multiple of 10. Your program can reject other values.
- A team of consumer threads remove tasks off the task queue, one at a time. For each task, a consumer thread finds all the primes in the array range and adds them to the `result` array. (*Hint:* Synchronization might be needed for the prime-adding operation.)
- Queue synchronization and thread termination should be handled the same way as in the previous program. At the end of the program, all primes are printed out:

```
linux> ./prime-pthd.exe 80 5
Main: started, array[80] with 5 threads
Master starting on core 2
Worker[0] starting on core 4
...
Main: ... all threads have joined
Found 22 primes in array[80]:
67, 29, 71, 3, 73, 23, 11, 2, 37, 53, 17, 41, 43, 79, 59, 7, 31, 19, 5, 13, 61, 47,
Total tasks across threads: 8
C[ 0]: 2, C[ 1]: 2, C[ 2]: 1, C[ 3]: 1,
C[ 4]: 2,
```

A sequential prime-finding program, `prime.c`, is provided. You may copy any code to your program.

## 3. Summary and Submission

Write a short (one-page) summary (in pdf or plain text) covering your experience with this assignment. What issues did you encounter?, How did you resolve them? What is your thread termination approach? What can you say about the work distribution of your program? etc.

Make a zip file containing your program(s), two new sample outputs, and your write-up. Submit it through the Dropbox on the D2L site.