Prof. Jingke Li (FAB 120-06, lij@pdx.edu); Class: MW 14:00-15:15 @ FAB 47; Lab: F 10:00-11:30 @ FAB 88-10.

Assignment 1: Shared-Memory Programming with Explicit Threads

(Due Wednesday, 1/30/19)

This assignment is to practice shared-memory programming using Pthreads and Java threads. There are four programs in total, all implementing the producer-consumer problem. Requirements for CS 415 and 515 students are different; details are at the end of this document. This assignment carries a total of 20 points.

0. The Producer-Consumer Problem

The producer-consumer problem is a classic example of multi-threaded synchronization problem. The base version of the problem describes two threads, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to repeatedly generate data and add it to the queue. At the same time, the consumer is consuming the data (i.e., removing it from the queue), one piece at a time. The producer won't try to add data into the queue if it's full and the consumer won't try to remove data from an empty queue.

The extended version of the problem allows multiple producer and/or consumer threads.

1. Base Pthread Version

Implement a base version of the producer-consumer program with Pthreads, `prodcons-pthd1.c`. Your program will use the queue representation given in the file `queue.h`:

```
typedef struct queue_ queue_t;
struct queue_ {
  int *buffer;   // a circular buffer
  int capacity;  // buffer capacity
  int head;      // head of queue
  int tail;      // tail of queue
  int size;      // current size of queue
};

extern queue_t *create_queue(int bufsize);
extern void add_item(queue_t *q, int val);
extern int remove_item(queue_t *q);
```

A queue has a fixed-size circular array buffer, whose elements are integers. The three queue routines are implemented in the file `queue.c`. To use the queue representation, add a header line `#include "queue.h"` in your program.

Your program should follow the following requirements:

- The program parameters are: queue capacity = 20; total number of items to produce/consume = 100.
- The program defines three functions, `producer()`, `consumer()`, and `main()`.
- The `main` routine create a queue (with a capacity of 20). It then creates a single thread for running the `consumer()` routine; the main thread itself runs the `producer()` routine.

- Both the `producer()` and the `consumer()` routines print out a message at the beginning of their execution, with information about where it is executed:

  ```
  Producer starting on core 0
  Comsumer starting on core 1
  ```

- The producer adds 100 integer values (1 to 100) to the queue, one at a time. It prints out a message for each addition, showing the value and the (after-addition) queue size:

  ```
  Producer added value 23 (qsize = 8)
  ```

- The consumer removes 100 integer values from the queue, one at a time. It prints out a message for each removal, showing the value and the (after-removal) queue size:

  ```
  Comsumer removed value 54 (qsize = 3)
  ```

- The program terminates properly after all of the work is done. A final message shows:

  ```
  Main: all done!
  ```

Note that you need to figure out where and what kinds of synchronization are needed. Remember that the producer can't add any item to a full queue and the consumer can't remove any item from an empty queue. In either case, a while-loop waiting is not acceptable.

2. Base Java Version

Implement a base version of the producer-consumer program using Java threads, `ProdCons1.java`. The requirements are similar to the above Pthreads version. The differences are:

- There is no need to implement a queue. You may use Java's Queue interface:

  ```
  Queue<Integer> queue = new LinkedList<>();
  ```

  It comes with three useful methods: `add()`, `remove()`, and `size()`. The last can be used to check if a queue is empty or at capacity. Java's queue does not have a capacity attribute, so you should keep the capacity parameter as an independent constant.

- There is no easy way to show CPU info from a Java thread. So the printout messages do not need to include that info.

3. Extended Pthread Version

Implement an extended version of the producer-consumer program with Pthreads, `prodcons-pthd2.c`. This version supports one producer thread with multiple consumer threads. The queue representation and the program parameters stays the same.

Your new program should follow the following requirements:

- (*New requirement*) 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-pthd2 10    // 10 consumer threads
  linux> ./prodcons-pthd2       // 1  consumer thread
  ```

- The `main` routine creates `numCons` consumer threads, each running a copy of the `consumer()` routine; it then runs the `producer()` routine itself.

- The producer behaves the same as in the base version.

- The `consumer()` routine needs to change. The consumer threads compete to remove items off the queue, one at a time. Each consumer thread keeps track of how many items it has successfully obtained. The printout messages are similar to the base version, except that each consumer show's its ID:

  ```
  Comsumer[4] removed value 54 (qsize = 3)
  Comsumer[6] removed value 55 (qsize = 2)
  ```

- (*New requirement*) The program should print out a final message before termination, showing the item counts of the consumer threads, and the total sum of these counts (which should be 100 if the program is correct):

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

  The numbers inside the brackets are consumer IDs, and the numbers outside are the item counts.

An additional new challenge to this program is the handling of the proper termination of consumer threads, since they don't have a fixed number of items to remove from the queue.

The following are two possible approaches:

- The producer creates a bogus "termination" value (say `-1`) and add `numCons` copies of it to the queue right after the actual items. Upon receiving such an item, a consumer thread will termination itself.
- Use a global count to keep track of the removed items. The consumer threads all participate in updating and monitoring this global count. When the count reaches the total number of items, all threads terminate.

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

4. Extended Java Version

Implement the same extended version of the producer-consumer program using Java threads, `ProdCons2.java`.

415 vs. 515 Students

CS515 students are required to complete all four programs. CS415 students are required to complete the two base versions of programs and one of the two extended versions. You decide which of the two to implement. If you choose to implement both, you will receive up to 5 extra points.

Summary and Submission

Save an execution script for each of your programs. A sample script is provided in `sample-script.txt`. (*Hint:* You can use the linux command `script` to create such scripts.)

Write a short (one-page) summary (in plain text or pdf) 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 programs, the execution scripts, and your write-up. Submit it through the "Assignment 1" dropbox on D2L.