OS Lab 3

*Read the lab pdf first and try to understand it as much as you can

overview

We have 3 types of threads **mCounter**, **mMonitor** and **mCollector**.

At our main function we should create one **mMonitor** thread, one **mCollector** thread but N **mCounter** threads.

You could initialize N with any number that you like or you can take it as an input from the user.

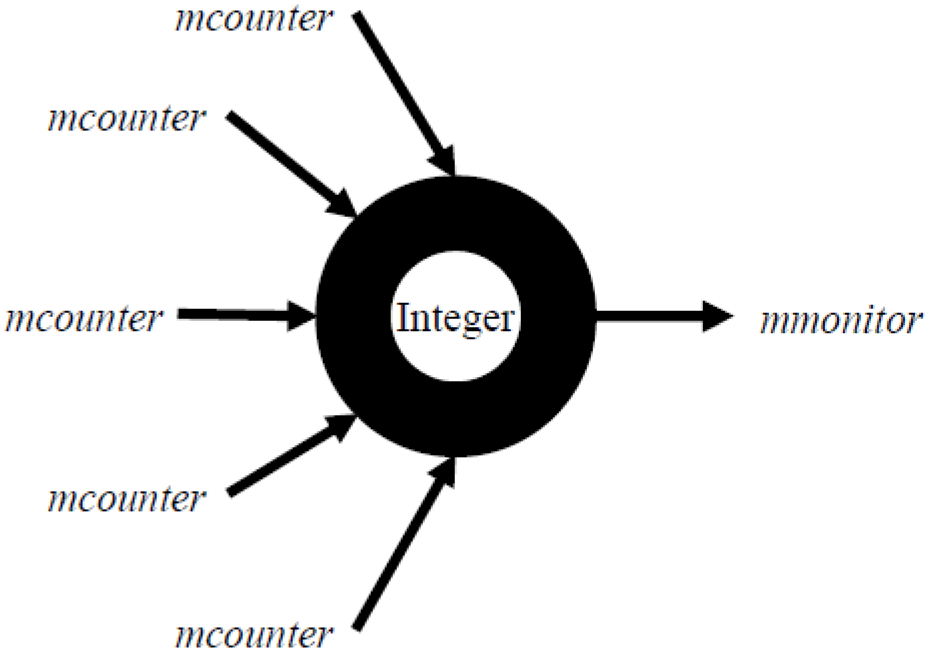
To simulate messages coming randomly to the system we can let each **mCounter** thread sleep for a random period of time before it starts.

Also, both **mCollector** and **mMonitor** thread should be activated at random time intervals.

Break it down

It becomes much easier if we follow the hint provided in the pdf and **divided up the problem into two sub problems**:

- **Problem 1:**
  - Threads included:
    - N **mCounter** threads
    - the **mMonitor** thread
  - Shared Resources:
    - an integer to count messages.
  - problem:
    - when a **mCounter** thread grand access to the counter, it should add one to it
    - when the **mMonitor** thread grand access to the counter, it should reset it to 0 and save its value to use it later
    - only one thread should be able to access the shared counter

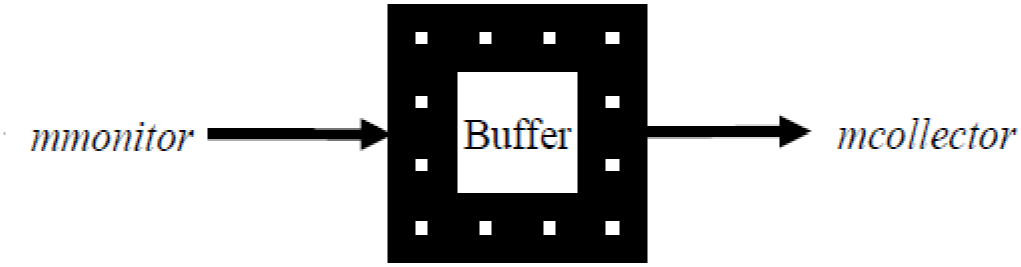


- **Problem 2:**
  - Threads included:
    - the **mMonitor** thread
    - the **mCollector** thread

  - Shared Resources:
    - A buffer can be implemented using a FIFO queue

  - problem:
    - It’s a bounded buffer producer/consumer problem
    - **mMonitor** is our producer it enqueues the value that is saved from the previous problem into the buffer
    - **mCollector** is the consumer it takes the data out of the buffer

  - **you can find the solution for this problem at chapter 5’s slides**



Output

*taken form the sample run in the pdf

The output shows the behavior of the threads so each thread should print a certain output when a particular event happens

- **mCounter**:
    - **At time of activation (sleep time end):** Counter thread %I%: received a message
    - **Before waiting:** Counter thread %I%: waiting to write
    - **After increasing the counter:** Counter thread %I%: now adding to counter, counter value=%COUNTER%
- **mMonitor**:
    - **Before waiting to read the counter:** Monitor thread: waiting to read counter
    - **After reading the counter value:** Monitor thread: reading a count value of %COUNTER%
    - **After writing in the buffer:** Monitor thread: writing to buffer at position %INDEX%
    - **If the buffer is full:** Monitor thread: Buffer full!!
- **mCollector:**
    - **After reading from the buffer:** Collector thread: reading from buffer at position %INDEX%
    - **If the buffer is empty:** Collector thread: nothing is in the buffer!