Assignment 3 - Semaphores

Objectives

1. Familiarizing with concurrent programming.
2. Handling races, synchronization, and deadlock conditions.

Problem statement

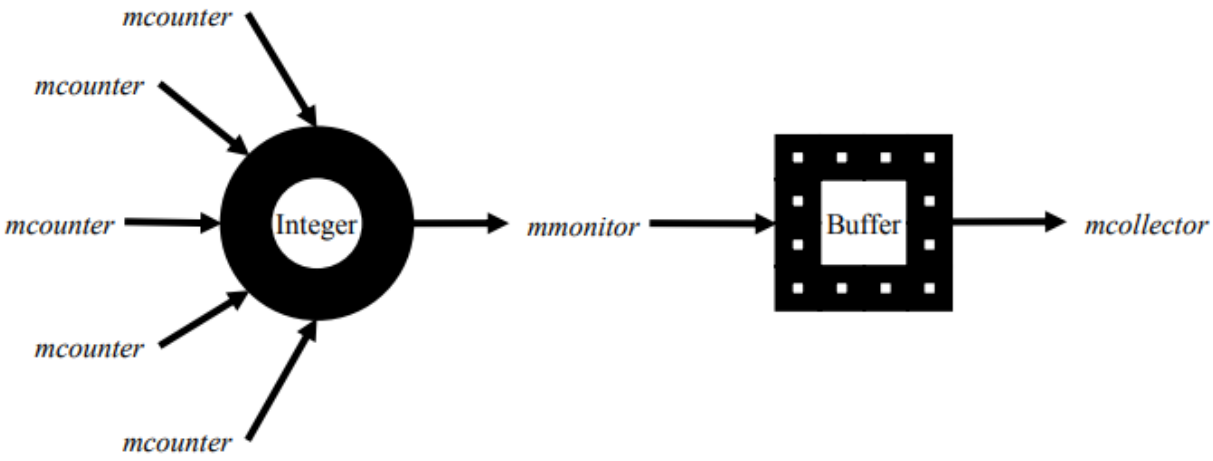
You are required to write a C program to solve the following synchronization problem using
POSIX and `“semaphore.h”` libraries.

N `mCounter` threads count independent incoming messages in a system and another thread
`mMonitor` gets the count of threads at time intervals of size t1, and then resets the counter to
0. The `mMonitor` then places this value in a buffer of size b, and a `mCollector` thread reads
the values from the buffer.

Any thread will have to wait if the counter is being locked by any other thread. Also, the `mMonitor` and `mCollector` threads will not be able to access the buffer at the same time or to
add another entry if the buffer is full.

Assume that the messages come randomly to the system, this can be realized if the `mCounter`
threads sleep for random times, and their activation (sleep time ends) corresponds to an email
arrival. Similarly, the `mMonitor` and `mCollector` will be activated at random time intervals.

The following figure represents the shared resources (the integer and the buffer) and which threads are interacting with each resource:



Breaking the problem down

overview

The problem statement may seem a little bit intimidating but once you divide the problem into smaller sub-problems; It'll become easier to approach.

We have 3 types of threads `mCounter`, `Monitor`, and `mCollector`.

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

You should define N as a constant at the beginning of your code with any number you like

> 💡 5 to 10 threads are sufficient but you may decide to go higher or lower than that to test your code

As mentioned above, 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 the `mCollector` and `mMonitor` thread should be activated at random time intervals.

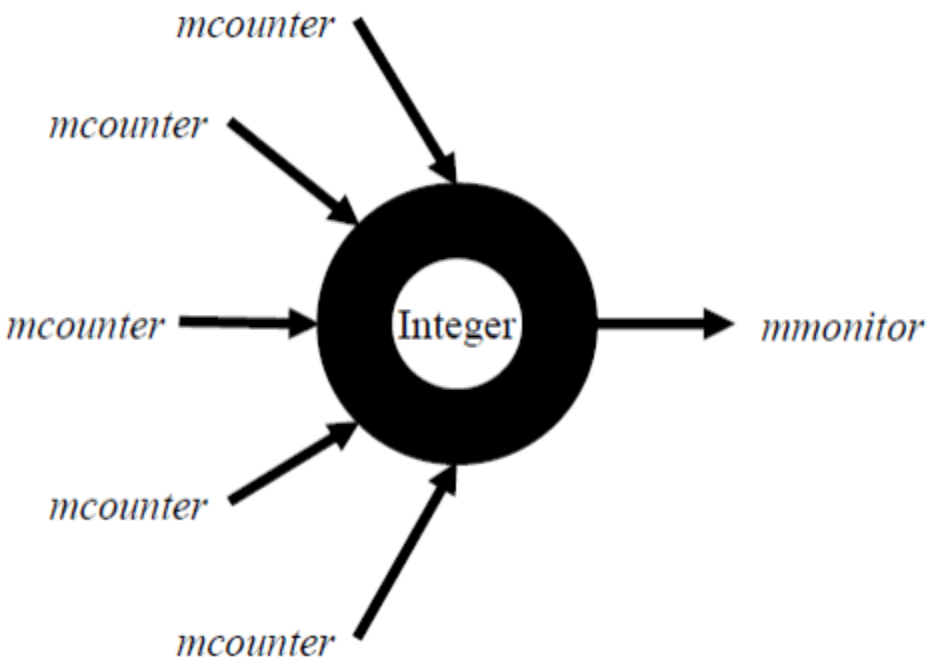
Check `srandom`, `random`, and `sleep`.

Dividing the problem

It becomes much easier if we divided up the problem into two sub-problems

Problem 1

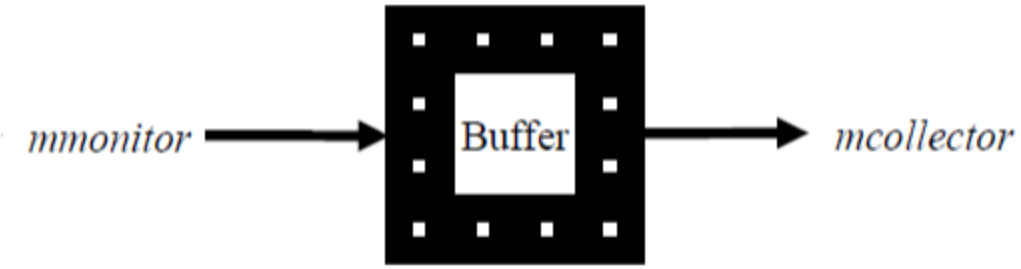
- Threads included:
  - The N `mCounter` threads.
  - The `mMonitor` thread.
- Shared resources:
  - An integer to count messages.
- Problem:
  - When a `mCounter` thread grants access to the counter, it should add one to it.
  - When the `mMonitor` thread grants access to the counter, it should reset 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 that should be implemented using a FIFO queue.
- Problem:
  - It's a bounded buffer producer/consumer problem.
  - `mMonitor` is the producer. It enqueues the value that was 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 to this problem in chapter 5's slides.



Program output

The output shows the behavior of the threads. 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 the buffer at position %INDEX%
  - **If the buffer is empty:** Collector thread: nothing is in the buffer!

Sample run

Your sample run will show a sequence of the behavior of each thread at the times of their activation (at random intervals), for example

Counter thread 1: received a message

Counter thread 1: waiting to write

Counter thread 2: now adding to counter, counter value=??

Collector thread: nothing is in the buffer!

Monitor thread: waiting to read counter

Monitor thread: reading a count value of ??

Monitor thread: writing to buffer at position ??

Monitor thread: reading a count value of ??

Monitor thread: writing to buffer at position ??

Counter thread 3: received a message

Counter thread 1: now adding to counter, counter value=??

Monitor thread: Buffer full!!

Collector thread: reading from buffer at position ??

Materials

1. Reference functions and samples: http://www.csc.villanova.edu/~mdamian/threads/posixthreadslong.html
2. A solved producer-consumer problem for a buffer of size 1: http://www.cs.arizona.edu/~greg/mpdbook/programs/pc.sems.c

Deliverables

- Complete source code in **ONE FILE** commented thoroughly and clearly.

- Name your file as your ID (e.g., 5237.c, 5237.cpp, 5237.C, 5237.cc, …)

Notes

1. Language used: C/C++, Operating System: Linux.
2. You should work individually.
3. You may talk together about the algorithms or functions being used but are not allowed to look at **anybody's** code.
4. Revise the academic integrity note found on the class web page