Operating Systems - Assignment 4.

Threads Synchronization

General Description

The purpose of this assignment is to gain experience with concurrent programming and synchronization mechanisms. Careful design BEFORE the actual implementation of the task is essential. Otherwise, the implementation may be very painful.

The scenario is also intended to give a feel for the performance impact of synchronization constructs under different loads.

The overall scenario we are simulating is that of news broadcasting. Different types of stories are produced and the system sorts them and displays them to the public.

In this assignment, the 'new stories' are simulated by simple strings which should be displayed to the screen in the order they arrive..

In the scenario that you should implement, there are 4 types of active actors:

Producer

Each producer creates a number of strings in the following format:

"producer <i> <type> <j>"

where 'i' is the producers ID, 'type is a random type it chooses which can be 'SPORTS', 'NEWS', WEATHER', and 'j' is the number of strings of type 'type' this producer has already produced. The number of such products that a producer produces is received via its constructor.

For example if producer 2 should create 3 strings a possible outcome of its products would be:

Producer 2 SPORTS 0

Producer 2 SPORTS 1

Producer 2 WEATHER 0

Each of the producers passes its information to the Dispatcher (introduced below) via its own private queue. Each of the Producers private queue is shared between the Producer and the Dispatcher. Each of the string products is inserted by the Producer to its 'producers queue'. After inserting all the products, the Producer sends a 'DONE' string through its Producers queue.

Dispatcher

The Dispatcher continuously accepts messages from the Producers queues. It scans the Producers queue using a Round Robin algorithm. **The Dispatcher does not block when the queues are empty**. Each message is "sorted" by the Dispatcher and inserted to a one of the Dispatcher queues which includes strings of a single type. When the Dispatcher receives a "DONE" message from all Producers, it sends a "DONE" message through each of its queues.

SPORTS	Inserted to the “S dispatcher queue”
NEWS	Inserted to the “N dispatcher queue”
WEATHER	Inserted to the “W dispatcher queue”

Co-Editors

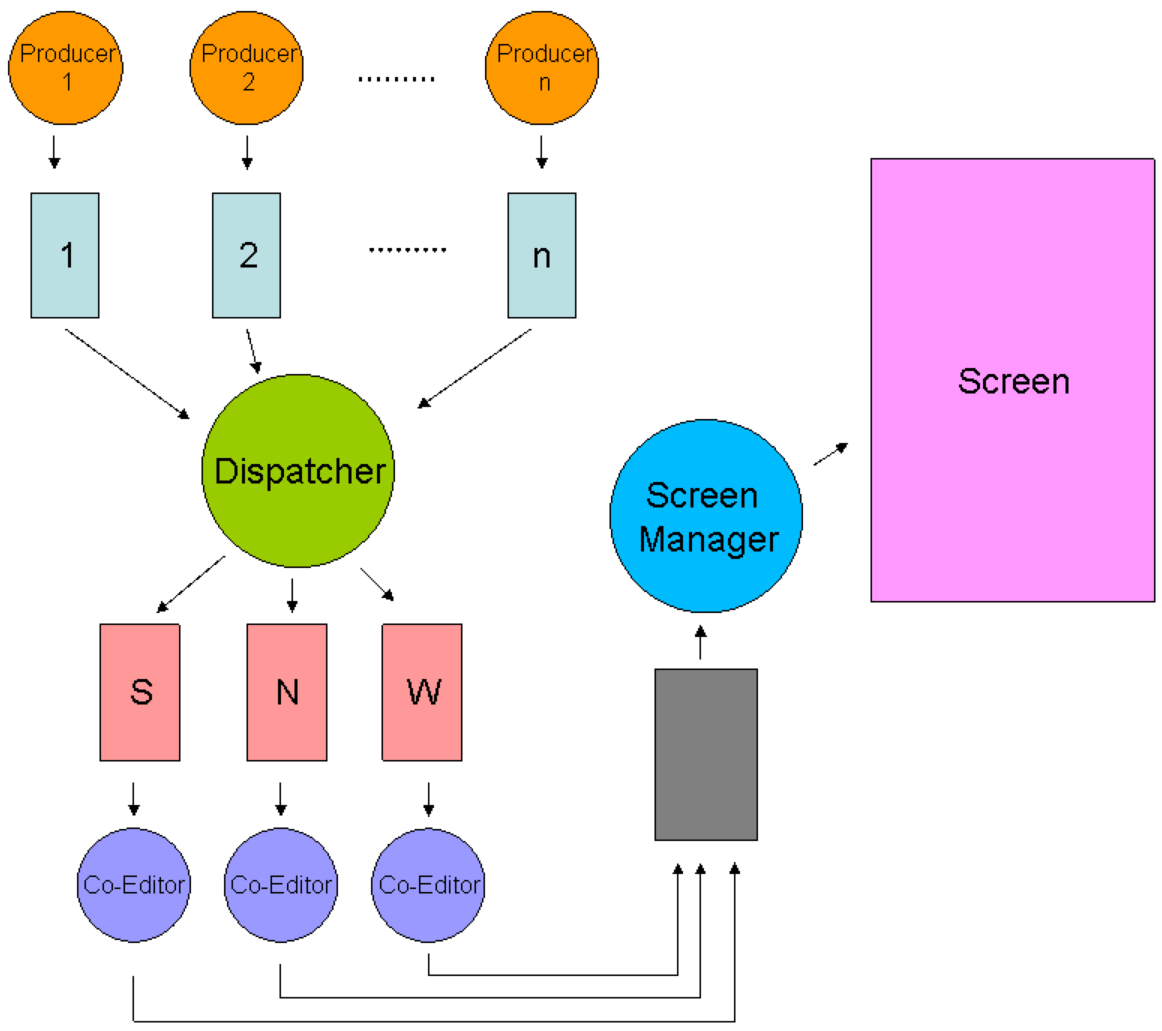
For each type of possible messages there is a Co-Editor that receives the message through the Dispatchers queue, "edits" it, and passes it to the screen manager via a single shared queue. The editing process will be simulated by the Co-Editors by blocking for one tenth (0.1) of a second. When a Co-Editor receives a "DONE" message, it passes it without waiting through the shared queue.

Screen-manager

The Screen-manager displays the strings it receives via the Co-Editors queue to the screen (std-output). After printing all messages to the screen and receiving three "DONE" messages, the Screen manager displays a ‘DONE’ statement.

System Design

The system should be implemented according to the following chart:



Three producers communicate with the dispatcher via their Producer queues. The Dispatcher communicates with the Co-Editors via three queues corresponding to the three types of messages. The Co- Editors communicate with the Screen-Manager via a single shared queue, and the Screen manager displays the systems output.

Bounded Buffer

The Producer queues in this assignment and the Co-Editors shared queue are a bounded buffer that supports the following operations.

- Bounded_Buffer (int *size*) – (constructor) create a new bounded buffer with *size* places to store objects.
- void insert (char * s) – insert a new object into the bounded buffer.
- char * remove ( ) - Remove the first object from the bounded buffer and return it to the user.

You must implement a thread safe bounded buffer. In order to do so you will be supplied with a binary semaphore (mutex) and create a counting semaphore with two binary semaphores as studied in class. The implementation of a 'bounded buffer' synchronization mechanism, with two counting semaphores and one binary semaphore was presented in class.

For more details consult your course book or material learned in class.

The dispatcher’s queues are unbounded buffers.

Configuration File

The Configuration file has the following format:

```

PRODUCER 1
[number of products]
queue size = [size]


PRODUCER 2
[number of products]
queue size = [size]

…
…
…
PRODUCER n
[number of products]
queue size = [size]

Co-Editor queue size = [size]
```

For example, the following is a legal configuration file:

```
PRODUCER 1
30
queue size = 5

PRODUCER 2
25
queue size = 3

PRODUCER 3
16
queue size = 30

Co-Editor queue size =
17
```

How to get started (suggestions):

1. Implement a Queue object, which it's 'insert' and 'remove' procedures, implement the requirements of a bounded buffer.

2. Implement the different threads: Producer and Dispatcher and Screen manager.
3. Write the main function which reads the configuration file and initiates the system.
4. Debug first with one producer, than debug the general case.

Pay attention: Some of the Threads need to access the same bounded buffer.

Two ideas to solve this problem:

1. Declare a global bounded buffer.
2. Pass the pointer to the bounded buffer to the Thread at creation as a parameter.

Good Luck.

:c)