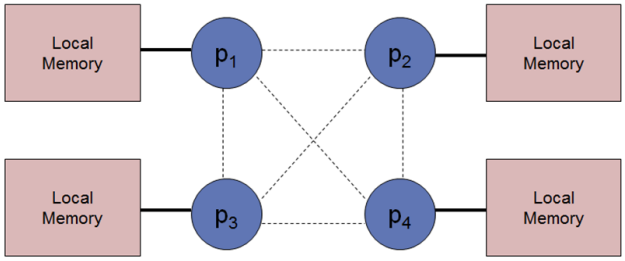
CPSC 457 – Winter 2017
Assignment 4
Critical Sections and Java Threads
Due: March 31st @11:59PM
Worth 8% of total mark

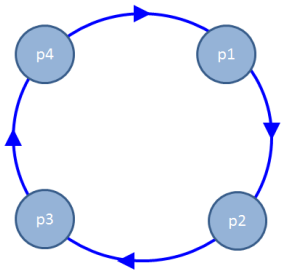
Objective: The main objective of this assignment is to work with Java threads in order to gain experience with concurrent programs.

Outcomes: In this assignment, you will write a simple simulator for a simple Distributed-Shared Memory (DSM) multiprocessor. DSM is a popular way to build high-performance machines. You will use the simulator to demonstrate how DSM can break the correctness of software solutions, using Peterson's critical section algorithm. You will introduce extra synchronization measures to re-establish correctness and perform a comparative analysis.

Background
Distributed-Shared Memory: DSM is used in order to create a common virtual address space for a multiple-processor architecture that does not have a shared physical memory. Each processor has its own physical memory that is only directly accessible by that processor. In this assignment, you will work with a fully-replicated DSM. That is, each processor has the same exact replica of memory as the rest of the processors. To perform a LOAD, a processor simply performs the LOAD on its local replica. To perform a STORE, a processor (1) performs the STORE on its local replica and (2) broadcasts the a message to all other processors to apply tat STORE to their own replicas.



Token Ring: A token ring is a synchronization mechanism used in distributed systems. The processes (or processors) are arranged on a logical ring. A unique special message is created, called the *token*, and then passed along the ring. When a process receives the token, it can execute privileged operations. These operations cannot be executed if a process does not have the token. Since there is a single token in the system, only one process can be executing the privileged operations at a given time. A process passes the token to its successor in the ring when it is done executing privileged operations or if it does not need the token. Note that there can be several token rings with the same set of processors. Since the tokens are different for each logical ring, one processor can be performing a set of privileged operations while another processor can be performing another set of such operations.
An example token ring for 4 processes is shown next:



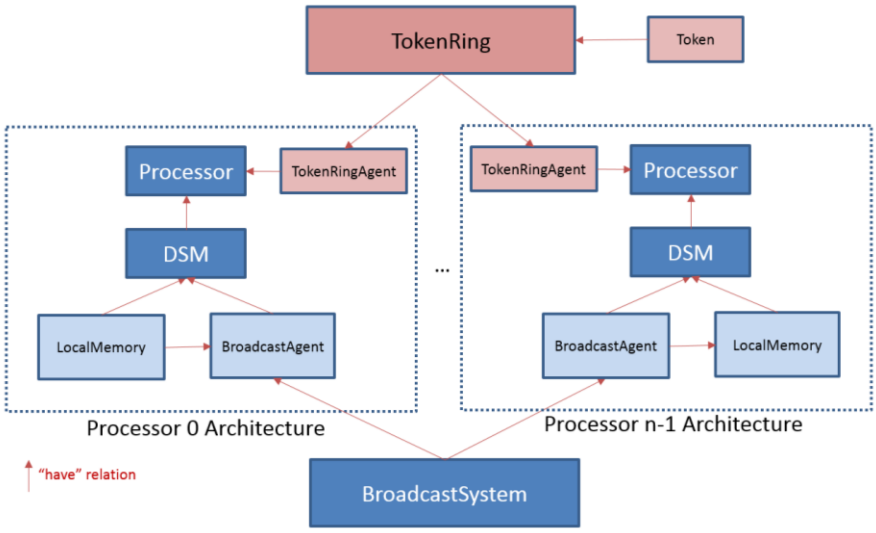
Note that a token ring can be used in order to implement critical sections in a distributed system. When a process has the token, it will enter the critical section. When a process exists the critical section, it passes the token to its successor on the ring.

Implementation

Write a simulator using the following classes. You must use the new Java Concurrency package. We also specified some of the variables and methods needed. You will need to define more methods and properties.

- LocalMemory: this class represents the local replica. It has the following methods:
  - load(x): returns the value of x read from memory
  - store(x,v): stores the value v of x to memory
- DSM: this class represents the DSM layer. It has the following methods:
  - load(x): returns the value of x read from the local memory
  - store(x,v): writes v into x in the local memory and broadcasts a message to all other DSMs to apply the write locally in their replicas

  DSM has an object of type LocalMemory and one of type BroadcastAgent. DSM executes in a separate thread.
- Processor: this class represents a processor. Each processor is executed in a separate thread. It has an object of type DSM.
- BroadcastAgent: this class provides the implementation of the broadcast mechanism needed by DSM. Each agent executes in a separate thread. It has the methods:
  - Broadcast(message): send a store
  - receive(): receive a store
- BroadcastSystem: This is the implementation of the broadcasting mechanism between processors. You may implement this using sockets, but this is not required. It is sufficient to implement some delays to simulate sending and receiving messages. One BroadcastSystem object is shared between all BroadcastAgents. The broadcastSystem executes in a separate thread.



In addition, you will need the following three classes to implement a token ring:

- Token: represents a token message.
- TokenRingAgent: Each Processor has a TokenRingAgent object. It has the properties:
  - ID, a unique identifier for the token
  - Active (if false, then the token ring is non-existent; the value will be true if the token ring is active)
  - ID (logical id for the processor on the ring)
  - RingPredecessor (logical id of the predecessor on the ring)
  - RingSuccessor (logical is of the successor)

  TokenRingAgent has the following methods:
  - RecieveToken(): returns the unique identifier for the token received from the predecessor
  - SendToken(Token t): sends the token to the successor

  Each agent executes in a separate thread.

- TokenRing: This is the arrangement of the token ring. The token ring consists of individual TokenRingAgents. The TokenRing, if active, creates the necessary token and passes it to an initially designated TokenRingAgent. There can be more than one TokenRing instances, with different token messages.

Questions

1. Write an implementation of Peterson's n-process algorithm (for n = 10) with the token ring turned off. This implementation is written in Processor and must use the available DSM methods. For instance,
   `turn = i`
   must be written as
   `dsm.store(turn,i)// store it locally and broadcast`
   Which entails:
   `localMemory.store(turn,i)// store it locally`

```
broadcastAgent.broadcast(turn,i)// broadcast
```
Other broadcast agents need to do the following upon receiving the broadcast:
```
localMemory.store(turn,i)
```

2. Demonstrate experimentally (by showing an appropriate output) that several processes can enter the critical section simultaneously.
3. Repeat 1&2 with the token ring turned on with a single token on the ring. The DSM store() implementations are revised as follows.
```
store(var,val){
      while (not have the token) ; // wait for the token
      localMemory.store(var,val);
}
```
If your implementation is correct, then Peterson's correctness is restored.
4. Repeat 3 with a number of tokens, one for each variable `turn`. That is, writing `flags` does not require acquiring the token. To write `turn[x]`, you need to acquire token x. Writes to different turns can be done simultaneously by different processors since each require a different token. You need to have a different token ring for each such token. Each ring consists of the same processors; the only difference is that for each of these rings, there is a unique token message. If your implementation is correct, this should also maintain the algorithm correctness.
5. Compare the implementations of 3 and 4 by measuring the throughput of each implementation. The throughput is measured as the number of critical sections completed per unit time. Each process must enter the critical section 100 times (a for loop containing Peterson's algorithm will do). Generate a graph for both implementations with the number pf processes (in Peterson's) on the x-axis and the throughput on the y axis. Start with n = 10 on the x axis and increment by 10 for up to at least 500 processes. Comment on your chart.

Notes

- You must use the new Java Concurrency package.
- Shared objects' implementations must be thread safe.
- Use the synchronized construct only when it is necessary.
- Use the sleep() method as you please (especially with the BroadcastSystem and Processor) in order to generate the desired output (e.g., breaking Peterson's algorithm).
- Slow down your BroadcastSystem (using sleep()) as much as possible to see the effects (after all, you are simulating a channel and random delays are common).

Deliverables and Marking Scheme
(Please submit your program to D2L. Submit the following in hard copy in the exact order as mentioned. Marks may be deducted if the ordering is not followed in the submission.)

1. Each of the 8 class' implementation (5 marks; total 40 marks)
2. Question 1 (Program) (15 marks)
3. Question 2 (Program, output) (15 marks)
4. Question 3 (Program, output) (7+3 marks)
5. Question 4 (Output and chart) (7+3 marks)
6. Question 5 (Chart, and comments) (7+3 marks)

Administrative Information

Teams

You are advised, but not required, to work in a team of two members. Teams of more than 2 students are not allowed. You and your partner must be in tutorials that are taught by the same TA. One submission per team is required.

Academic misconduct

Any similarities between assignments will be further investigated for academic misconduct. While you are encouraged to discuss the assignment with your colleagues, your final submission must be your own original work. Any re-used code of excess of 10 lines must be cited and have its source acknowledged. Failure to credit the source will also result in a misconduct investigation.

Late submission policy

Late submissions are penalized with 12.5% deduction for each late day or portion of a day. Hence no submission are accepted 8 days after the deadline

D2L Marks

Any marks posted on D2L or made available using any other mean are tentative are subject to change (after posting). They can go UP or DOWN due to necessary corrections.

Happy Coding :)