CS455 – Lab Session 06

NAMAN SHAH

02/24/2017

Agenda

- Quiz 4: review
- HW2:PC: Solution design
- HW2:PC: FAQ
- Midterm

---

Quiz 4 review

1. A program that has concurrency bugs may continue to function correctly if the rate of invocations and the number of threads are below a certain threshold. [True/False]

2. A thread-safe class does not require any additional synchronization or coordination by the calling class. [True/False]

3. Storing state variables of a particular class in public fields allows other classes within that program to reason about thread-safety. [True/False]

4. The wait()/notify() mechanism in Java has an inherent race condition that cannot be solved without deep integration with the JVM. [True/False]

5. The key to thread-safe programming is not so much what the object does, but rather how it will be accessed.
   - Unsafe publication of object
   - Improper locking to manage object’s access

6. If two operations executed by threads are atomic with respect to each other, each of these operations should map to a single underlying machine level instruction that is executed atomically. [True/False]

7. Stateless objects are always thread-safe. [True/False]

8. Consider a variable count of type long. If the mutation operation on this variable is the increment operator (++), there is no need to synchronize accesses to the mutation operation. [True/False]

9. We only need to synchronize accesses to write operations on a variable. The read operations need not be synchronized.
   - Memory visibility issues

---
Quiz-4 review

10. Which of the following constructs does not address the memory visibility issue in concurrent programming?

   a) Use of the volatile keyword
   b) Use of the synchronized keyword
   c) Using the AtomicInteger and AtomicLong variable types
   d) Using the final variable

- volatile
  - Always read from memory, don't store variable in register
  - Reads ensures up-to-date copy, all other threads' updates are performed
- final
  - Reference is final, not the object

Solution design

HW2-PC

FAQ

- accept() should not be managed using ThreadPool

- Core of HW2: Manage tasks using fix number of threads
- Extra threads:
  - ThreadPoolManager
  - Writer thread at Client side
  - Statistics printer

Things to note

- OP_ACCEPT needed to accept connections
- Channels are readable until their send buffers are full
- Even if key is not set with OP_WRITE
- Set OP_WRITE
- key.isWritable() = true in next iteration
- OP_READ needed to read data
- Otherwise received data sits idle in receive buffer
- Be careful while using ByteBuffer
  - Any read/get/put operation advances pointer
  - Leads to program/traceable behavior
  - clear() before sending data to channel
  - clear() before sending data to channel

Logistics

- Working on utility script
- Starts 100 clients and check progress
- Announcement on Piazza and Canvas after completion
- By the end of the day
Information

- Midterm is on Thursday, 2nd March in CSB 130
- 12:30-2 PM
- Everything until Lecture-13
- Questions from quizzes
- Watch for words
- Referring slides is enough
- No surprises
- Question types
  - True/False
  - Multiple choice questions
  - Answer in brief
- No surprises
- Question types
- True/False
- Multiple choice questions
- Answer in brief
- Worth 100 points (10% of your final grade)