Instructor:
Dr. Binoy Ravindran
Associate Professor
ECE Dept., Virginia Tech
Office: 2040-C Torgersen Hall
Blacksburg, VA 24061
540-231-3777, binoy@vt.edu
http://www.ece.vt.edu/faculty/ravindran.html
http://www.real-time.ece.vt.edu/

Meeting Times and Location:
Tuesday, Thursday, 5:00 – 6:15 PM, TORG 1020

Course Objectives:
The computer industry is undergoing a paradigm shift, as chip manufacturers are increasingly investing in, and manufacturing a new generation of multi-processor chips called multicores, as it is becoming increasingly difficult to increase clock speeds by packing more transistors in the same chip without increasing power consumption and overheating. Consequently, application software performance can no longer be improved by simply relying on increased clock speeds of single processors; software must explicitly be written to exploit the hardware parallelism of multiprocessors. Programming multi-processors is fundamentally different from programming single-processors, due to the need to understand how concurrent computations on separate processors coordinate with one another, in contrast with understanding how concurrent computations on the same processor coordinate with one another. This is a complex and intricate problem, and requires new abstractions, algorithms, and programming tools.

The course will approach multiprocessor programming from two complimentary directions: principles and practice. In the principles part, the course will illustrate how to reason about concurrency by discussing the classical mutual exclusion problem, correctness properties (e.g., fairness, deadlock, linearizability), properties of shared memory (e.g., register constructions, atomic snapshots), and synchronization primitives for implementing concurrent data structures, ranging from simple ones (e.g., consensus protocols) to powerful universal constructions (e.g., universality of consensus). The course will illustrate the practice of multiprocessor programming through programming patterns such as spin locks, monitor locks, the work-stealing paradigm, and barriers. Concurrent data structures (e.g., concurrent linked lists, queues, stacks, hash maps, skiplists) will be discussed through synchronization patterns ranging from coarse-grained locking to fine-grained locking to lock-free structures, atomic synchronization primitives, elimination, parallel search, and transactional memory.

Upon completion of the course, the student should be able to:

- Demonstrate fundamental principles of multiprocessor programming including:
  - concurrency correctness properties such as consistency, linearizability, progress, fairness, and deadlock-freedom;
  - properties of shared memory such as register constructions and atomic snapshots; and
• Synchronization primitives for implementing concurrent data structures, ranging from simple ones (e.g., atomic registers, consensus protocols, FIFO queues) to powerful universal constructions (e.g., universality of consensus).

• Write multiprocessor programs using:
  o Programming patterns including spin locks and contention, monitor locks and waiting, work-stealing and parallelism, and barriers;
  o Concurrent data structures including concurrent linked lists, concurrent queues, concurrent stacks, concurrent hash maps, and concurrent skip lists;
  o Synchronization patterns including coarse-grained locking, fine-grained locking, optimistic locking, lazy synchronization, non-blocking synchronization, atomic synchronization primitives, elimination, parallel search; and
  o Transactional memory abstractions including software transactional memory.

• Establish properties of multiprocessor programs including their correctness, fairness, consistency, linearizability, progress, and deadlock-freedom.

Prerequisites:

Graduate standing. ECE 4534 or ECE 4550. Knowledge of concurrent programming in Java, or in C++ (augmented with a threads package such as PThreads), or in C# is assumed.

Required Text:


Papers from the literature will be made available at the course website.

Software packages that implement synchronization abstractions and algorithms discussed in the course will be made available on the course website.

Grading (tentative):

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Homeworks</td>
<td>40% (about eight homeworks)</td>
</tr>
<tr>
<td>Presentation of a paper from the multiprocessor literature</td>
<td>10%</td>
</tr>
<tr>
<td>Programming Project</td>
<td>40% (one project)</td>
</tr>
<tr>
<td>Final Exam</td>
<td>10%</td>
</tr>
</tbody>
</table>

Notes:

1. The one programming project will involve (substantial and non-trivial) implementation of a concurrent data structure and/or illustration of a concurrent programming and synchronization pattern. A list of possible topics will be proposed during the semester. Students are also encouraged to propose topics for the project (e.g., related to their ongoing MS/PhD thesis research).

2. A list of papers from the multiprocessor literature will be suggested for presentation. The papers will cover topics that are not discussed in the lectures. Students are also encouraged to come up with their own selection of a paper.

3. The final exam will be open-book and open-notes, and will be comprehensive.
Honor Code Policy
Adherence to Virginia Tech's honor code is expected in all phases of this course. All graded work is expected to be the original work of the individual student unless otherwise directed by the instructor. In working on the projects and homeworks, discussion and cooperative learning are allowed and, in fact, encouraged. However, copying or otherwise using another person's solutions to the project and homework problems is an honor code violation. You may discuss general concepts, such as software libraries, Internet resources, or class and text topics, with others. However, discussion of project solutions or homework solutions, or specific program-code is an honor code violation. Any violations of the honor code will automatically be forwarded to the Office of the Honor System.

Course Website
The website of the course is [http://www.learn.vt.edu](http://www.learn.vt.edu). The website will be used to distribute lecture transparencies, programming projects, and software packages, provide solutions after exams and projects, and any modifications to the syllabus.

Special Needs or Circumstances
Any student with special needs or circumstances should feel free to meet with or otherwise contact the instructor.

Course Topic Outline (tentative):

<table>
<thead>
<tr>
<th>Topic</th>
<th>Book Chapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to shared objects and synchronization, and the mutual exclusion problem</td>
<td>1, 2</td>
</tr>
<tr>
<td>Correctness properties of concurrent programs (consistency, linearizability, progress, fairness, deadlock-freedom)</td>
<td>3</td>
</tr>
<tr>
<td>Foundations of shared memory (register constructions, atomic snapshots)</td>
<td>4</td>
</tr>
<tr>
<td>Synchronization operations for concurrent data structures (atomic registers, consensus protocols, FIFO queues, universality of consensus)</td>
<td>5, 6</td>
</tr>
<tr>
<td>Programming patterns: Spin locks and contention, monitor locks and waiting</td>
<td>7, 8</td>
</tr>
<tr>
<td>Concurrent data structure: concurrent linked lists and synchronization patterns: coarse-grained locking, fine-grained locking, optimistic locking, lazy synchronization, and non-blocking synchronization</td>
<td>9</td>
</tr>
<tr>
<td>Concurrent data structures: concurrent queues, concurrent stacks</td>
<td>10, 11</td>
</tr>
<tr>
<td>Concurrent data structures: concurrent hash maps, concurrent skiplists</td>
<td>13, 14</td>
</tr>
<tr>
<td>Programming patterns: work-stealing and parallelism, barriers</td>
<td>16, 17</td>
</tr>
<tr>
<td>Transactional memory</td>
<td>18</td>
</tr>
</tbody>
</table>