C++ Multithreading

Spring 2017

CSC 462

instructor:	Ed Keenan
email:	ekeenan2@cdm.depaul.edu
office hours:	Thursday 3-5pm, 9-10pm in classroom or by email appointment
office:	CDM 830
phone:	(312)362-6747
website:	piazza.com/depaul/spring2017/csc362 (Preferred communication)
lecture:	CDM 220, Wednesday 5:45-9:00pm
Desired to Learn (D2L):	d2l.depaul.edu (Grades, Viewing lectures, Announcements)
Version Control:	perforce: ***140.192.39.61:1666***

Description

Software architecture of applied C++ concurrency and multithreading fundamentals. Basics threading concepts: process model, threads, stacks, fibers, mutexes, semaphores, atomics and events. Leveraging advanced C++ language features relating to the memory model and the threading support in large multithreaded architectures. Architecting lock-based and lock-free concurrent data structures in applications. Designing a threaded management system to control the access and reuse of threads in applications. Designing multithread architecture for real-time performance.

Prerequisites

- GAM 471 - Optimized C++ or CSC 461 Optimized C++
    - Implies CSC 403 & CSC 407

Reference Material

- **C++ Concurrency in Action: Practical Multithreading**, 2012 by Athony Williams
- **Windows System Programming** (4th Edition), 2010, Johnson Hart, ISBN: 978-0321657749
- **The C++ Programming Language, 4th Edition** - Bjarne Stroustrup

Learning Objectives

- Students will be able to demonstrate basic thread knowledge (threads, mutexes, semaphores, atomics, events)
- Students will be able to optimally design applications to minimize stalls due to thread waits and dependencies.
- Students will be able to design a thread management system to control the access and lifecycles of threads in an application.
- Students will design multithreaded applications to improve its run-time performance and resource usage.

Grading

- 30 % - Simplified Threading program
  - Program using basic multithreading primitives
    - Several fixed threads, Mutexes, Callbacks, Critical Sections
- 20 % - Conversion problem
  - Puzzle program or something similar
    - Converting a single thread system into multiple threaded solution
- 20% - Managing multiple thread in pools
  - Creating a threading pool system.
    - Manage and control thread execution and priorities.
  - Spawning multithreaded program
    - Creating a dynamically spawning application of autonomous agents
    - Design and implemented using the thread pool system
- 10% - Basics threading assignments
  - 5-10 basics threading assignments to reinforce concepts
- 10% - Lock Free Demo
  - Create a lock free data structure design and implementation
  - Stack using 2 different techniques
- 10 % - Final Exam

Software

- ***Microsoft Visual Studio 2015 Enterprise Edition***
  - Visual Studio Enterprise 2015 with update 3 32/64-bit
    - C+ and C# install (for libraries)
  - Microsoft Visual Studio Community edition is not used in this class.
  - Microsoft Visual Studio 2017 edition is not used in this class.
- ***Perforce - Visual Client (p4v)***
  - www.perforce.com
- Download and configuration instructions will be provided in class
  - Server address: ***140.192.39.61:1666***

Topics:

Understanding the multi-threading primitives / basic concepts

- Managing Threads
- Sharing Data between Threads
- Synchronizing concurrent operations
- C++ Memory Model
- Lock-based current data structures
- Lock Free concurrent data structures

Design Software

- Dividing work between threads
- Thread Management
- Debugging

- Testing
- C++ 11 Language Features

Primitives

- Threads
- Fibers
- Atomics
- Mutexes
- Semaphores
- Events

Phase 1 : Basics Concepts

- ***Programming Assignment***
  - (30%) PA1: Simplified Threading program
    - Program using basic multithreading primitives
      - Several fixed threads, Mutexes, Callbacks, Critical Sections
- Week 1
  - Introduction to Concurrency
  - Launching a thread
  - Waiting for a thread to complete
  - Passing arguments to a thread
  - Transferring ownership of a thread
- Week 2
  - Sharing Data between thread
  - Race conditions
  - Protecting shared data with mutexes
  - Deadlocks
  - Flexible locking (C++ specifics)
  - Transferring mutex ownership
  - Recursive locking
- Week 3
  - Synchronization concurrent operations
  - Building a thread-safe queue
  - Waiting for an event or other condition
  - Waiting for one-off events with futures
  - Return from background tasks, promise - future
  - Waiting with a time limit
  - Clocks, Timepoints
- Week 4
  - C++ memory model
  - Atomic Types in C++
  - Synchronizing operations and enforcing ordering using Atomics
  - Fences

Phase 2: Refactoring existing solutions

- ***Programming Assignment***
  - (20%) PA2: Conversion problem
    - Converting a single thread system into multiple threaded solution
      - Example problems:
        - NxN sliding puzzle
        - Maze program
- Week 5
  - Lock-based concurrent data structures
  - A thread-safe stack using locks
  - A thread-safe queue using locks and condition variables
  - A thread-safe queue using fine-grained locks and condition variable
  - Writing a thread-safe list using locks
- Week 6
  - Designing lock-free concurrent data structures
  - Non blocking data structures
  - Lock-free data structures
  - Writing a thread-safe stack without locks
  - ABA problem
- Week 7
  - Designing concurrent code
  - dividing work between threads
  - Dividing work by task
  - performance of concurrent code
  - Oversubscription and excessive task switching
  - Hiding latency with multiple threads
  - Improving responsiveness with concurrency

Phase 3: Threading system / management

- ***Programming Assignment***
  - (20%) PA3: Managing multiple thread in pools
    - Creating a threading pool system.
      - Manage and control thread execution and priorities.
    - Spawning multithreaded program
      - Creating a dynamically spawning application of autonomous agents
      - Design and implemented using the thread pool system
    - Examples:
      - **Interactive spawning / dispatch / threading system**
        - Genetic bug spawn / growth / death - Gui application
- Week 8
  - Advanced thread management
  - Thread pools
  - Interrupting threads
  - Launching and interrupting another thread
  - Detecting that a thread has been interrupted

    - Interrupting other blocking calls
- Week 9
    - Debugging multithreaded applications
    - Types of concurrency-related bugs
    - Race conditions
    - Unwanted blocking
    - Locating concurrency-related bugs by testing
- Week 10
    - Performance tuning
    - Contest

Final Exam (10%)

Final exam covering the concepts and the material of the class

- In class exam
- Multithreading concepts
- Diagramming communication between threads
- Applied problems

Lock Free Demo (10%)

- Read the material from the class reference books and external materials
    - journals, white papers, technology specifications
- Write a lock free data structure demo
    - Using push() model described in the book
    - Using alternative with atomics

Basics Threading Programs (10%)

- There will be at least 5-10 weekly proficiency problems, validating your threading fundamental knowledge
- The programs are easy to implement, but is a motivator for those who need encouragement to learn more thoroughly the threading material.
- Topics range from:
    - Threads, Mutexes, Futures, Promise
- If you already know the material, the assignments are a way to validate that knowledge

Perforce Submissions

- Everyone is expected to submit several submissions to perforce a week.
    - Minimum 5 significant (real) submissions on 3 separate days.
    - To promote incremental development and prevent last day rush.
    - Grade deduction will occur if not followed
- The biggest reason students get into trouble with software design:
    - Not starting the project early

    - Not working on the material frequently enough
    - Taking too large of a bite(byte) of the design
- Both are minimized with this *Perforce RULE*
- Even my simplest programs take 10-20 submissions.
    - For these project assignments my average is 40-400 submissions, so 5 will be no problem.
- Detailed perforce changelist comments are expected

Piazza Discussion forum

- Statistics show: students who participate more and help other students do better!
    - The correlation is ridiculous!
        - Poor understanding / poor participation.
        - Great understanding / Great participation
    - As you master the material, help others learn!
        - Want to be a Master programmer so master it!
- Everyone is ***expected*** and encouraged to participate on the Piazza discussion forum. All class-related discussion here this term.
    - At least one real question or response per week from every student.
- Everyone is expected to keep up with the material on Piazza and are responsible for its content. Critical class updates and directions will be presented there.
    - Not participating or reading the material on Piazza is not an excuse.
- All correspondence that is not personal in nature should be vectored through Piazza
    - Sensitive material, use Piazza private note, not email.
- The quicker you begin asking questions on Piazza (rather than via emails), the quicker you'll benefit from the collective knowledge of your classmates and instructors. I encourage you to ask questions when you are struggling to understand a concept.
- Keep the forum professional and positive, help each other out.
    - Karma really pays off here.
    - Help each other whenever you can.
        - There will be a section where you'll need help (trust me).

NOTE: Do ***NOT*** post until you have watched the entire lecture ***FIRST*** (in class or online)

This will prevent frustration on all sides (members asking or answering questions)

Collaborating together on programming assignments

- You are encouraged to work together
  - Use the Piazza forums heavy
  - Even share your material with others in the common directory
    - Obviously not the answers
- Everyone is 100% responsible for the work they do.
  - If you get help with a section of code,
  - Please refactor the code the ***snot out of it***
    - Comment and understand that material
    - Transform the code to ***make it yours***
  - Be able to answer ***any*** question regarding the code you commit
- System for Detecting Software Plagiarism
  - We will be using MOSS - Measure of Software Similarity (Stanford University)
    - Indicates possible code infringements (plagiarism)
    - MOSS - will detect the similarity independent of naming convention, indentation style or formatting, it compares abstract syntax tree of your code.
  - I will pursue any plagiarism/integrity violations aggressively, arguing for full expulsion from the university for the offenders.
    - Don't put me or you in this scenario
- If you gain significant support / help from another student
  - Fully disclose the support / help you had in a Readme.txt file submitted with your assignments.
    - Disclosing the help, is ***not permission*** for copying the code.
    - Only there to clarify and acknowledge help you were given from a fellow student.
- Modifying any Unit Test to alter the outcome results is also an ***Academic Integrity Violation***
- If you are stuck and find yourself even tempted to plagiarize
  - Ask for help !!!!
    - Use on Piazza -> Visit during offices hours, make an appointment
    - ***Don't ever compromise your integrity!***
- Material was uniquely created for this Class.
  - You indirectly by the process of tuition, "paid" for the contents and material of this class.
    - Do not share this ***copyrighted*** material in any form
    - It is design for your personal use, while enrolled in the Class.
  - Do ***NOT*** post any content or revealing material to any external website or forum outside of this class.
    - The Class Piazza forum is provided for this service, ask questions there, not on the internet (i.e. StackOverflow and other software forums)
- After you leave this class
  - You are expressly ***FORBIDDEN*** to provide or share the content with others.
  - Academic Integrity Violations can still be applied to students who provide material support to other students even after completion of the class.
- Just follow the golden rule:
  - ***"I have neither given, nor received, nor have I tolerated others' use of unauthorized aid."***

Schedule:

Week	Lecture	Work
1	Overview C++ 11 Review Thread Creation	Chapter 1-2 PA1 - assignment
2	Sharing Data Deadlocks Mutex	Chapter 3
3	Synchronization Waiting for Events Clocks	Chapter 4
4	C++ Memory Model Atomic Types Fences	Chapter 5
5	Lock-Base concurrent structures Thread-safe Queues **PA1 - due before class**	Chapter6 PA2 - assignment
6	Lock-Free concurrent structures Non-Blocking ABA problem	Chapter 7
7	Dividing work between threads Oversubscription excessive switching Improving responsiveness	Chapter 8
8	Thread Pools Interrupting Threads Blocking calls **PA2 - due before class**	Chapter 9 PA3 - assignment
9	Debugging multithreading Race conditions Unwanted blocking	Chapter 10
10	Performance tuning Contest	
11	**Final Exam** **PA3 - due before exam** **Lock Free Demo**	

March 31, 2017	Last day to add classes to SQ2017 schedule
April 7, 2017	Last day to drop classes with no penalty, Last day to select pass/fail option
April 8, 2017	Grades of "W" assigned for SQ2017 classes dropped on or after this day
April 13, 2017	Last day to select auditor status
May 12, 2017	Last day to withdraw from SQ2017 classes

Course Policies

Changes to Syllabus

This syllabus is subject to change as necessary during the quarter. If a change occurs, it will be thoroughly addressed during class, posted under Announcements in D2L and sent via email.

Online Course Evaluations

Evaluations are a way for students to provide valuable feedback regarding their instructor and the course. Detailed feedback will enable the instructor to continuously tailor teaching methods and course content to meet the learning goals of the course and the academic needs of the students. They are a requirement of the course and are key to continue to provide you with the highest quality of teaching. The evaluations are anonymous; the instructor and administration do not track who entered what responses. A program is used to check if the student completed the evaluations, but the evaluation is completely separate from the student's identity. Since 100% participation is our goal, students are sent periodic reminders over three weeks. Students do not receive reminders once they complete the evaluation. Students complete the evaluation online in CampusConnect.

Academic Integrity and Plagiarism

This course will be subject to the university's academic integrity policy. More information can be found at http://academicintegrity.depaul.edu/. If you have any questions be sure to consult with your professor.

Academic Policies

All students are required to manage their class schedules each term in accordance with the deadlines for enrolling and withdrawing as indicated in the University Academic Calendar. Information on enrollment, withdrawal, grading and incompletes can be found at: cdm.depaul.edu/enrollment.

Students with Disabilities

Students who feel they may need an accommodation based on the impact of a disability should contact the instructor privately to discuss their specific needs. All discussions will remain confidential.
To ensure that you receive the most appropriate accommodation based on your needs, contact the instructor as early as possible in the quarter (preferably within the first week of class), and make sure that you have contacted the Center for Students with Disabilities (CSD) at: csd@depaul.edu.

Lewis Center 1420, 25 East Jackson Blvd.
Phone number: (312)362-8002
Fax: (312)362-6544
TTY: (773)325.7296

Retroactive withdrawal

This policy exists to assist students for whom extenuating circumstances prevented them from meeting the withdrawal deadline. During their college career students may be allowed one medical/personal administrative withdrawal and one college office administrative withdrawal, each for one or more courses in a single term. Repeated requests will not be considered. Submitting an appeal for retroactive withdrawal does not guarantee approval. Information on enrollment, withdrawal, grading and incompletes can be found at: http://www.cdm.depaul.edu/Enrollment-Policies.aspx