C++ Multithreading

Fall 2018

CSC 362

instructor:	Ed Keenan
email:	ekeenan2@cdm.depaul.edu
office hours:	Wed 2:30 - 4pm, 9-10pm in classroom or by email appointment
office:	CDM 830
phone:	(312)362-6747
website:	piazza.com/depaul/fall2018/csc462362 (Preferred communication)
lecture:	CDM 200, 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

Applied C++ concurrency and multithreading fundamentals. Basics threading concepts: process model, threads, stacks, fibers, mutexes, semaphores, atomics and events. Understanding synchronous / asynchronous interactions and behavior of threads. Using managed thread pools and queues in applications. Understanding advanced C++ language features relating to the memory model and the threading support.

Prerequisites

- GAM 371 - Optimized C++ or CSC 361 Optimized C++
  - Implies CSC 393 & CSC 374

Reference Material

- **C++ Concurrency in Action: Practical Multithreading**, 2012 by Anthony Williams,
  - ISBN: 978-1933988771
- **Windows System Programming** (4th Edition), 2010, Johnson Hart,
  - ISBN: 978-0321657749
- ***The C++ Programming Language:*** Stroustrup
  - ***4th Edition 2013 (new edition) or 3rd Edition (either are acceptable)***
  - Stroustrup Addison-Wesley Longman/Pearson, 2014. ISBN: 978-0321563842

Learning Objectives

- Students will be able to demonstrate basic thread knowledge (threads, mutexes, semaphores, atomics, events)
- Students will be able to create design multithread applications using Lock-Based concurrent data structures.
- Students will be able to debug multi-threaded environment (Deadlocks, Starvation, Priority contention)
- Students will refactor single threaded applications into a multi-threaded solutions

Grading

- 30 % - Simplified Threading program
  - Program using basic multithreading primitives
    - Several fixed threads, Mutexes, Callbacks, Critical Sections
- 30% - 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% - Linked List Conditioning for multithreaded
  - Create a process for deep copy that is O(2N)
  - Allows for multithreading
- 20 % - Final Exam

Software

- ***Microsoft Visual Studio 2017 Enterprise Edition (not Community)***
  - MSDNAA Depaul – Visual Studio 2017 Enterprise
    - C++ and C# install (future classes)
  - Any other variants are not used in this class
  - Students are responsible keeping their development tools working
- ***Perforce Server***
  - Download and configuration instructions will be provided in class
  - ***Perforce – Helix Visual Client (p4v)***
    - https://www.perforce.com/downloads/helix-visual-client-p4v
    - Server address: ***140.192.39.61:1666***

Readings

Textbooks are used for references and learning new topics. It is suggested that you research and investigate material and ancillary topics covered in the class through these books as needed. High performance programming requires both breadth and depth knowledge in the C++ language, therefore everyone's needs and research will vary based on their own experience and evolving mastery of the material.

Grading Scale:

93-100: A	87-89: B+	77-79: C+	67-69: D+	0-59: F
90-92: A-	83-86: B	73-76: C	60-66: D	
	80-82: B-	70-72: C-		

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***
  - (30%) 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***
  - (10%) Linked List Conditioning for multithreaded
    - Create a process for deep copy that is O(2N)
    - Allows for multithreading
- 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 (20%)

Final exam covering the concepts and the material of the class

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

Basics Threading Programs (10%)

- There will be at least 5-10 weekly proficiency problems, validating your threading fundamental knowledge and advanced C++
- 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 of ***five*** significant (real) submissions on ***three*** 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 five 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 will 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 time when you will need help from the class (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 or website
  - 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 or Project setting 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.
  - By the process of tuition, you "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."***

Miscellaneous

- ***Late Policies***
    - Due dates and times are verified by the submission record on the Perforce Server
        - No extensions are allowed
    - All assignments need to compile without warnings
        - Failure to compile "as-is" results in a 0 for the grade
- ***Memory Leaking***
    - For assignments that have memory tracking enabled
        - If an assignment is determined that its leaking memory
            - A deduction of 20% is applied to the grade of that assignment
    - Leaking status is provided during development
- ***Crashing***
    - Assignments are expected to work for a set duration
      (long enough to demo all the features)
        - A grade of 0 is given to any project that throws an exception, ends unexpectedly, crashes or hangs (not proceeding forward).
        - Crash – program locking up or quitting unexpectedly

Tentative Schedule:

Week	Lecture	Work
1	Overview C++ 11 Review Thread Creation	Chapter 1-2
2	Sharing Data Deadlocks Mutex	Chapter 3 PA1 – Audio Playback
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 - Maze
6	Lock-Free concurrent structures Non-Blocking ABA problem	Chapter 7 PA4 – Linked List Conditioning
7	Dividing work between threads Oversubscription excessive switching Improving responsiveness	Chapter 8
8	Thread Pools Interrupting Threads Blocking calls **PA4 - due before class**	Chapter 9
9	Debugging multithreading Race conditions Unwanted blocking	Chapter 10
10	Performance tuning Contest **PA2 - due before class**	
11	**Final Exam**	

University Dates (Drop, Withdrawal, Audit, Exam)

- https://academics.depaul.edu/calendar/Pages/default.aspx

University 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. Please see https://resources.depaul.edu/teaching-commons/teaching/Pages/online-teaching-evaluations.aspx for additional information.

Academic Integrity and Plagiarism

This course will be subject to the university's academic integrity policy. More information can be found at https://resources.depaul.edu/teaching-commons/teaching/academic-integrity/Pages/default.aspx.

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:
http://www.cdm.depaul.edu/Current%20Students/Pages/PoliciesandProcedures.aspx

Incomplete Grades

An incomplete grade is a special, temporary grade that may be assigned by an instructor when unforeseeable circumstances prevent a student from completing course requirements by the end of the term and when otherwise the student had a record of satisfactory progress in the course. All incomplete requests must be approved by the instructor of the course and a CDM Associate Dean. Only exceptions cases will receive such approval. Information about the Incomplete Grades policy can be found at
http://www.cdm.depaul.edu/Current%20Students/Pages/Grading-Policies.aspx

Students with Disabilities

Students seeking disability-related accommodations are required to register with DePaul's Center for Students with Disabilities (CSD) enabling them to access accommodations and support services to assist with their success. There are two office locations:

- Loop Campus – Lewis Center #1420 – (312) 362-8002
- Lincoln Park Campus – Student Center #370 – (773) 325-1677

Students who register with the Center for Students with Disabilities are also invited to contact Dr. Gergory Moorhead, Director of the Center, privately to discuss how he may assist in facilitating the accommodations to be used in a course. This is best done early in the term. The conversation will remain confidential to the extent possible.

Please see https://offices.depaul.edu/student-affairs/about/departments/Pages/csd.aspx for Services and Contact Information.

Proctored exams for OL courses (if applicable)

If you are an online learning student living in the Chicagoland area (within 30 miles of Chicago), you will need to come to the Loop campus to take an exam. Online learning students outside of the Chicagoland area are required to locate a proctor at a local library, college or university. You will need to take the exam within the window your instructor gives. Students should examine the course syllabus to find exam dates and the instructor's policy on make-up exams. Detailed information on proctored exams for online learning students can be found at http://www.cdm.depaul.edu/onlinelearning/Pages/Exams.aspx

Online office hours for OL courses (if applicable)

Faculty should be accessible to online students via phone, email and/or Skype.