ETEC3702 – Concurrency
Final Exam Study Guide

Topics

You are responsible for **everything** covered in class through Thursday, 23 April 2020.

Some of the major (but not necessarily all) topics included will be:

- ❍ Introduction to Concurrency
  - ❒ Definition
  - ❒ Concurrent Versus Sequential
  - ❒ Formal notation
    - "For all"
    - "There exists"
    - "It is true that"
    - "Concurrent with" (||)
    - "Precedes" (→)
    - Formal definition of sequential
    - Formal definition of concurrent
  - ❒ Non-deterministic execution
- ❍ Dangers of Concurrency
  - ❒ Effective orderings – how to compute
  - ❒ Problematic orderings
  - ❒ Safety Property Violations
    - Concurrent update problem (turnstiles problem, bank account problem)
    - Critical Sections
    - Mutual Exclusion
    - Software Solutions
    - Dekker's Algorithm
    - Peterson's Algorithm
  - ❒ Liveness Properties Violations
    - Deadlock
    - Livelock
    - Starvation

    - Difficulty in Testing
- Motivation for Concurrent Solutions
    - Organization
        - Natural Solutions
        - Relax over-specification
        - Multiple separate tasks
    - Speed-up
        - Moore’s Law and the Historic Trend of CPU Clock Speeds
        - Multiple Core CPUs
        - Distributed Computing / “Cloud” Computing
        - Specialty Parallel Processors / GPU Computing
        - Computing Speedup to compare sequential versus concurrent solutions.
        - Computing Maximum speedup of a program with sequential and parallelizable parts.
    - Unavoidable / Inherent
        - Network / Communication Systems
        - Database Systems
        - Distributed Systems
        - Event-driven Systems
        - Physical Systems.
- Synchronization mechanisms
    - Locks
        - acquire( )
        - release ( )
        - with lock syntax.
        - Optional arguments (blocking, timeout)
        - The “Monitor” object design pattern concept.
    - Rlocks
        - motivation for / versus normal locks.
        - acquire( )
        - release( )
        - with rlock syntax
        - Optional arguments (blocking, timeout)
    - Semaphores

- As a data-type
- Set of values
- Set of permissible operations
    - Definition of P( ) or acquire( )
    - Definition of V( ) or release( )
- Types of Semaphores
    - Binary
    - Counting
- In Python
    - Semaphore objects
    - BoundedSemaphore objects

❐ Conditional Sections

- Producer / Consumer design patterns.
- The need for conditional synchronization
- Condition objects
    - acquire( )
    - release( )
    - wait( )
    - notify( )
    - notify_all( )
    - Use to implement a “Monitor” design pattern

❐ Events

- Concept and function
- As a synchronization mechanism
- As a communication mechanism
- Event objects
    - set( )
    - clear( )
    - is_set( )
    - wait( )

❐ Barriers

- Concept and function
- Uses
- Barrier objects
    - wait( )

            - abort( )
            - reset( )
            - broken
            - parties
            - n_waiting

❍ Communication Mechanisms

- ❒ Global variables
- ❒ Events
- ❒ Queues
    - Queue(), LifoQueue( ), PriorityQueue( )
    - Optional args: maxsize
    - put( )
    - get( )
    - empty( )
    - full( )
    - task_done( )
    - join( )

❍ Multiprocessing

- ❒ Differences from threading
- ❒ Advantages over threading
- ❒ Disadvantages related to threading
- ❒ Potential for speedup
- ❒ Need for the __name__==”__main__” check
- ❒ cpu_count( )
- ❒ Inter-process communication using queues.
- ❒ Using queues with processes to “return” values.
- ❒ Support for other synchronization and communication mechanisms
    - Lock
    - Rlock
    - Condition
    - Semaphore / BoundedSemaphore
    - BarriersEvent
    - Queue

❍ Inter-process Communication

❒ Queues

❒ Pipes

- Use in operating systems to allow IPC
- multiprocessing.Pipe()
- send()
- recv()
- close()
- pipes vs. queues

❒ Managers

- multiprocessing.Manager()
- manager namespaces
- adding items to a manager namespace
- immutable versus mutable objects in managers
- manager.list() type

❒ Value / Array Objects

- multiprocessing.Value() and typecodes
- multiprocessing.Array()
- The need to lock while accessing Value() / Array() data.
- Implementing a Monitor to protect Value / Array objects.

❒ Shared memory

- multiprocessing.shared_memory
- creating shared_memory, create, size, name
- advantages and disadvantages wrt other IPC mechanisms.
- Connecting a process to a named shared_memory block.
- Using struct.pack and struct.unpack to store data into shared_memory.

❒ IPC for distributed computing

- Sockets
    - creating a socket object
    - connect()
    - bind()
    - listen()
    - accept()
    - send() / recv()
    - close()

-
  - Encoding / decoding socket data
    - binary / custom
    - JSON
    - Pickle
    - XML
    - MsgPack
    - encoding / decoding with loads() / dumps()
- multiprocessing.connection objects
  - advantages over sockets
  - send() / recv() / poll()
  - send_bytes() / recv_bytes()
  - Listener() / Client()
  - sending objects over connection objects

❒ Event-driven Programming and asyncio

- cooperative multi-tasking.
- concept and use with concurrent systems
- asyncio
- event loop concepts and use with I/O-bound tasks
- async / await syntax and uses.
- Awaitable items: coroutines, tasks, futures.
- async def coroutine functions.
- Tasks and asyncio.run()
- asyncio,wait()
- asyncio.sleep()
- asyncio.gather()
- futures

❒ Liveness: Deadlock and Livelock

- Liveness definition / Safety Definition
- Deadlock
  - Deadlock definition
  - 4 required conditions
  - Deadlock Prevention
  - Deadlock Detection and Recovery
    - Resource Allocation Graphs
    - Reduction to detect deadlock

    - recovery methods
  - Deadlock avoidance
    - Safe vs. unsafe vs. deadlock states
    - Djikstra’s Banker’s Algorithm
- Livelock
  - livelock definition
  - differences from deadlock
- The dining philosophers example
  - deadlock
  - livelock

❒ Concurrency in C and C++

- Posix Threads / pthreads
  - pthread.h
  - pthread_create()
  - pthread_join()
  - pthread_exit()
  - pthread synchronization
    - mutexes
      - pthread_mutex_lock()
      - pthread_mutex_unlock()
    - condition variables
      - pthread_cond_wait()
      - pthread_cond_signal()
      - pthread_cond_broadcast()
- C++11 Concurrency
  - #include <thread>
  - creating thread object instances: thread t1()
  - .join()
  - thread synchronization
    - std::mutex
    - .lock()
    - .unlock()
    - std::condition_variable
    - Other mechanisms

Reminders/Advice:

- The entire exam will be open-notes.
- The exam will be posted on Blackboard on the day of the exam.
- The exam will have a time-limit and must be completed and submitted within that time limit.
- There may be a programming portion of the exam. You can use a computer for that portion, but you may not communicate with others during that portion. (no discord, no stack-exchange, no reddit, no facebook messenger, no zoom, no SMS messages, no whatsapp, no email, no message boards, no talking, no communicating whatsoever!) Seriously, if I find evidence that you've talked to others or been intellectually dishonest in any way you will fail the exam.
- You should complete and understand all of the assigned labs.
- You may have to write and submit some code. Be prepared to do that.
- Open notes, but don't assume open-book == no preparation/study.