

# CMPS 3390 Homework 9

*Spring 2026*

## Concurrency and Multithreading

Concurrency and multithreading are essential concepts in modern application development, enabling programs to perform multiple tasks simultaneously, improve responsiveness, and efficiently utilize system resources.

## Concurrency vs. Parallelism

- Concurrency involves managing multiple tasks at the same time, not necessarily simultaneously. It is about dealing with multiple things at once conceptually.
- Parallelism, a subset of concurrency, involves executing tasks simultaneously, typically on multi-core processors.

## Threads

- A thread is the smallest unit of execution in a process.
- A single process can have multiple threads, sharing the same memory and resources, but operating independently.
- Multithreading allows applications to perform complex tasks like downloading a file, updating a UI, and handling user input simultaneously.

## Key Advantages of Multithreading

- Responsiveness: Keeps the application responsive by offloading time-consuming operations to background threads.
- Resource Utilization: Makes better use of multi-core CPUs by distributing workloads across multiple threads.
- Scalability: Supports efficient handling of multiple requests, such as in web servers or APIs.

## Challenges of Multithreading

- Race Conditions: Occurs when threads access shared data simultaneously without proper synchronization, leading to unpredictable behavior.
- Deadlocks: Happens when two or more threads are waiting indefinitely for each other to release resources.
- Context Switching Overhead: Frequent switching between threads can reduce performance.

## Synchronization

To prevent issues like race conditions, multithreading employs synchronization techniques:

- Locks: Ensure that only one thread accesses a resource at a time.
- Semaphores: Control access to a pool of resources.
- Atomic Operations: Perform thread-safe operations without locks.

## Popular Libraries and Frameworks

- Java: Provides the Thread class, ExecutorService, and synchronized keyword for multithreading.
- Python: The threading and concurrent.futures modules support concurrent programming.
- C#: Offers the Task class and async/await keywords for asynchronous operations.
- Golang: Allows for lightweight user-level threads called goroutines.
- Javascript: Uses internal threads for async operations, and recently added Web Workers

## Best Practices

- Minimize shared resources to reduce the risk of synchronization issues.
- Use thread pools to manage the number of concurrent threads efficiently.
- Test and debug multithreaded programs thoroughly, as concurrency bugs can be intermittent and hard to reproduce.

## Real World Application

Concurrency and multithreading are used extensively in real-world applications:

- Web servers handle thousands of simultaneous client requests.
- Mobile apps remain responsive while performing background tasks like syncing data.
- Games and simulations use threads to manage AI, rendering, and user inputs simultaneously.

## Additional Resources

- Java Concurrency and Multithreading - Introduction (14 minutes)  
<https://www.youtube.com/watch?v=mTGdtC9f4EU>
- Java Threads - Creating, starting and stopping threads in Java (17 minutes)  
<https://www.youtube.com/watch?v=eOk5AWcTS8w>
- Race Conditions in Java Multithreading (23 minutes)  
<https://www.youtube.com/watch?v=RMR75VzYoos>
- Concurrency vs Parallelism (10 minutes)  
<https://www.youtube.com/watch?v=Y1pgpn2gOSg>