Distributed Systems

COMP90015 2018 Semester 2
Tutorial 03

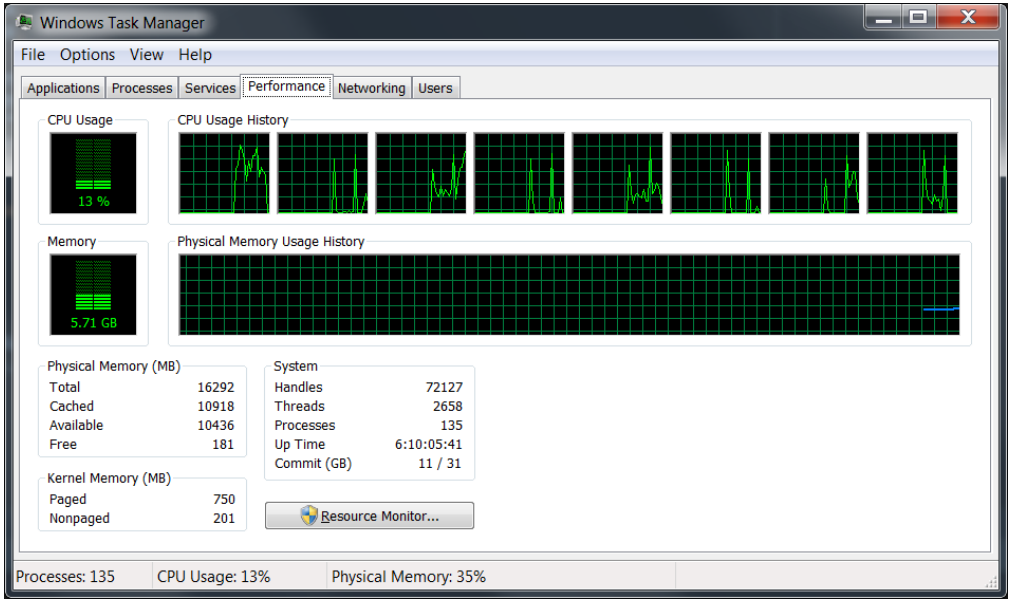
Today’s Agenda

- Quickly go through the concepts related to multi-threading
- Concept/question discussion
    1. What is a thread and life cycle of a thread
    2. Synchronous access to shared resources
    3. Comparison of worker pool multi-threading architecture with the thread-per-request architecture
    4. What is deadlock and how to avoid it
- Code demonstration of thread Sleep, Join and Synchronization.

When do we need multi-threading?

When do we need multi-threading?

- Allow multiple operations performed at the same time to achieve concurrency at the application level
- Improve application performance
- Improve resource utilization rate



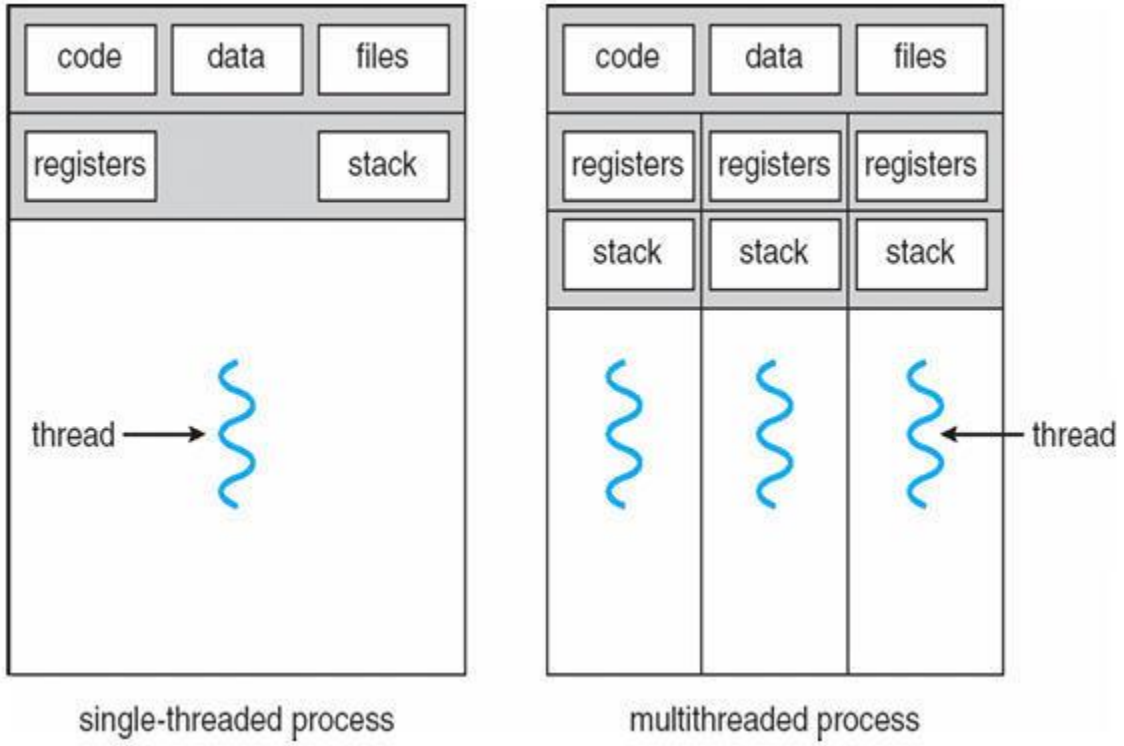
When do we need multi-threading?

- Application scenarios
  - Implementation of reactive user interfaces.
  - Handling concurrent requests and serving multiple clients
  - Non-blocking I/O operations.
  - Asynchronous behaviour.
  - Timer and alarms implementation.

What is the relationship between thread and process?

What is the relationship between thread and process?

- Threads are objects of process in Java consisting of a statically ordered sequence of instructions.
- Threads share code, data, files, and I/O in heap memory
- Threads have their independent stack memory and registers



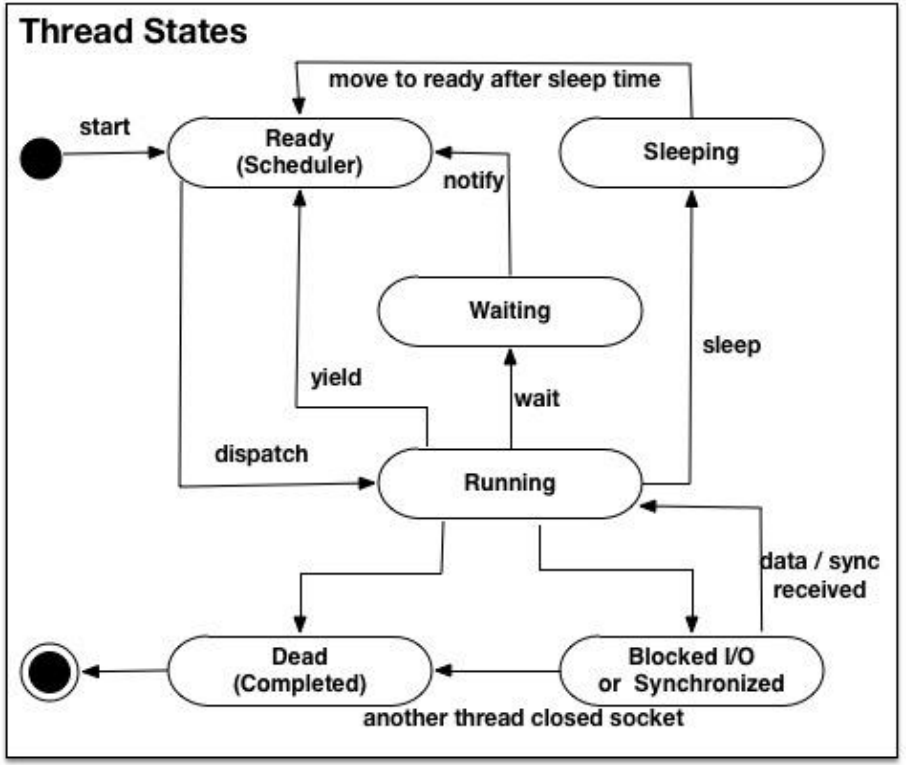
How do threads run on CPUs?

How do threads run on CPUs?

- Threads have the illusion of their own CPU
  - But in reality, they have to share one (or more) CPUs
  - This is handled by the OS with its thread scheduling module (time-sharing)
  - The OS can run or suspend a thread at any time

How do threads run on CPUs?

- Dispatch
  - Thread gets the CPU resources to run
- Yield
  - Thread gives hint to the thread scheduler that it is ready to pause its execution.
- Sleep
  - Sleep for a period
- Wait
  - Causes current **thread** to **wait** until another **thread** invokes the notify() method for this object or *notifyAll()*.
- Notify
  - To wake a thread up

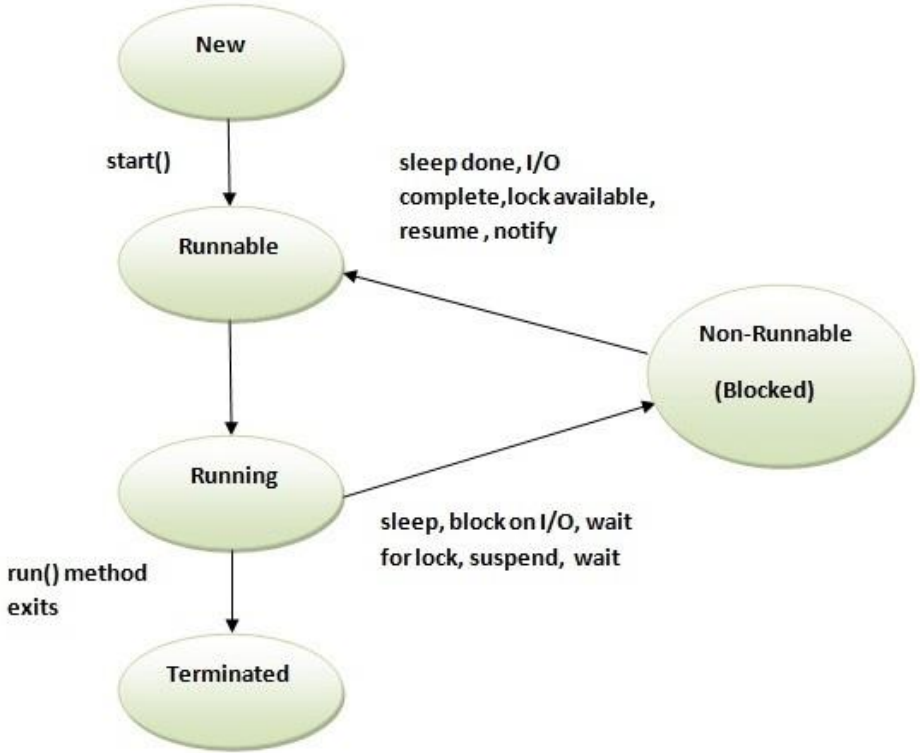


Q1. What is a thread and life cycle of a thread?

Q1. What is a thread and life cycle of a thread?

- A Thread is a piece of code that runs in concurrent with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are used to express concurrency on both single and multiprocessors machines.
- Threads share the same address space and therefore can share both data and code

Thread Lifecycle



Q2. What is synchronous access to shared resources and how can we achieve it?

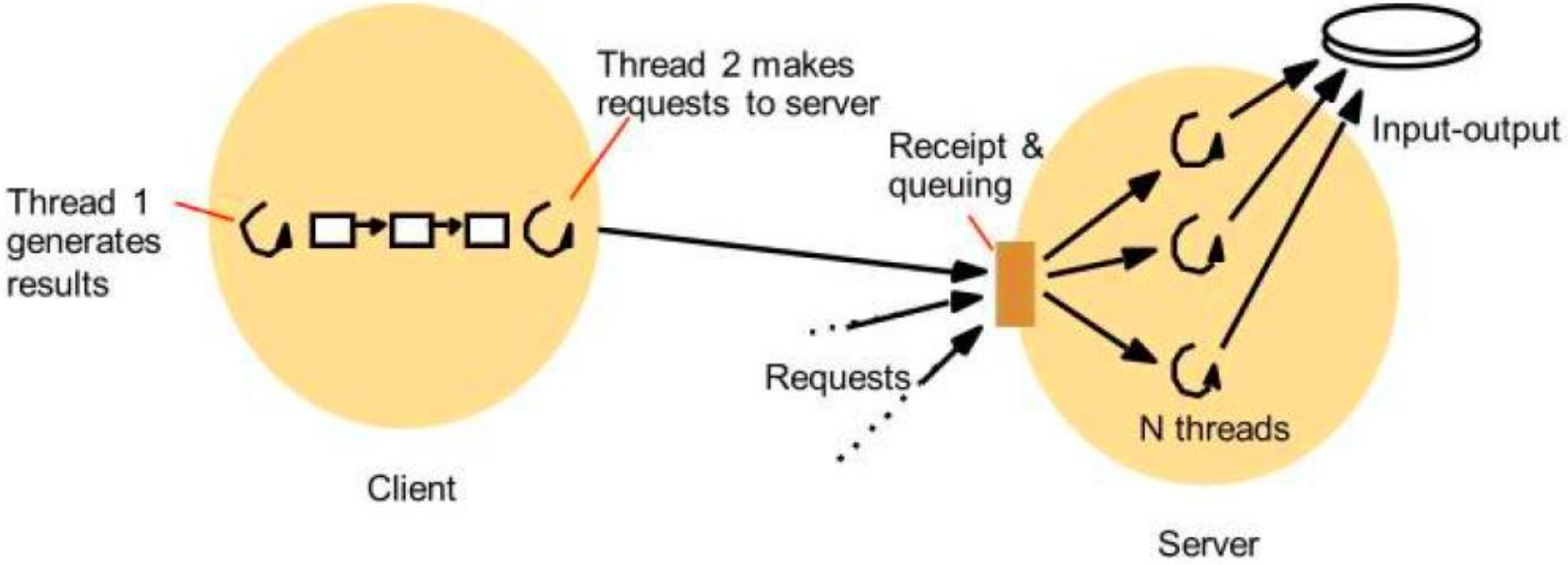
Q2. What is synchronous access to shared resources and how can we achieve it?

- If one thread tries to read the data and other thread tries to update the same data, it leads to inconsistent state.
- This can be prevented by synchronising access to the data.
- Use “synchronized” method:

```
public synchronized void update()
{
                    …
}
```

Q3. Compare the worker pool multi-threading architecture with the thread-per-request architecture.

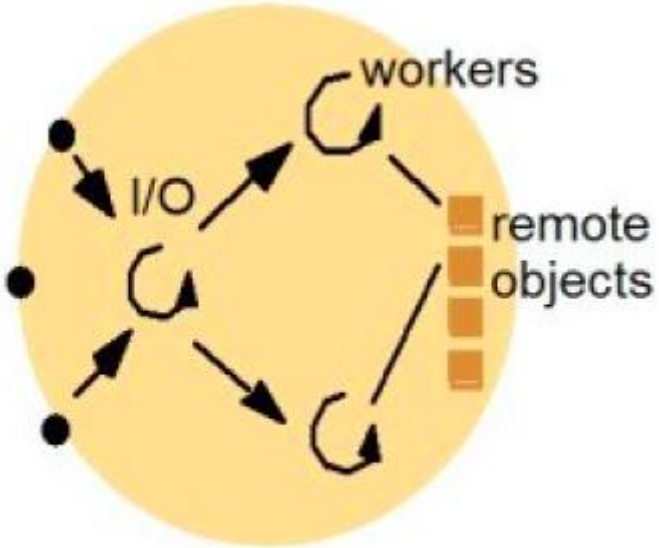
Worker pool architecture



Worker pool architecture

The server creates a fixed number of threads called a worker pool. As requests arrive at the server, they are put into a queue by the I/O thread and from there assigned to the next available worker thread.

Server creates worker pool ➜ request comes in, put into a queue ➜ assigned to an available worker thread.



a. Thread-per-request

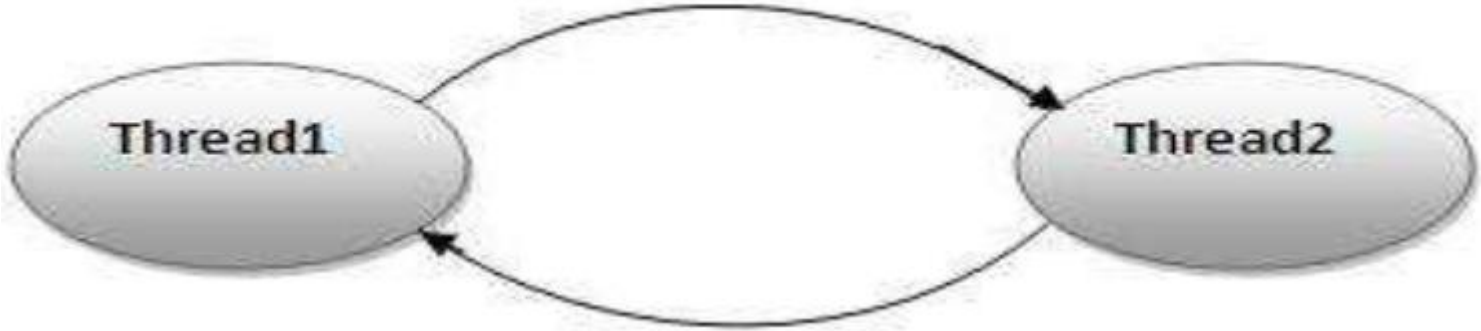
Thread-per-request architecture

Thread created for each request, when the request is finished, the thread is deallocated.

Q4. What is deadlock and how to avoid it?

Q4. What is deadlock and how to avoid it?

- Deadlock can occur when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread



Q4. What is deadlock and how to avoid it?

- Deadlock can be easily avoided by resource ordering.
- With this technique, assign an order on all the objects whose locks must be acquired and ensure that the locks are acquired in that order.
- Example:
  - Thread 1:
    lock A lock B
  - Thread 2:
    wait for A lock C (when A is locked)
  - Thread 3:
    wait for A wait for B wait for C

An example application to illustrate wait and notify

- Thinking of an implementation of blocking queue that has a fixed size
- The queue needs to be accessed by multiple threads i.e. thread-safe
  - Remember, two unsynchronized threads accessing the queue at the same time may cause conflict.
- What methods are needed for a queue abstraction?

```java
public class BlockingQueue<T> {

    private Queue<T> queue = new LinkedList<T>();
    private int capacity;

    public BlockingQueue(int capacity) {
        this.capacity = capacity;
    }

    public synchronized void put(T element) throws InterruptedException {
        while(queue.size() == capacity) {
            wait();
        }

        queue.add(element);
        notify(); // notifyAll() for multiple producer/consumer threads
    }

    public synchronized T take() throws InterruptedException {
        while(queue.isEmpty()) {
            wait();
        }

        T item = queue.remove();
        notify(); // notifyAll() for multiple producer/consumer threads
        return item;
    }
}
```

Code Demonstration – Multi-threading in Java

- Sleep
  - Pauses the execution of a thread
- Join
  - Allows one thread to wait for the completion of another
- Synchronized methods
  - Intrinsic ***object locks***
  - When a thread starts executing an object's synchronized method, it obtains the object lock, when it finishes executing the synchronized method, it releases the lock
  - Only one thread at a time can hold the object's lock
  - When one thread is executing a synchronized method for an object, all other threads that invoke synchronized methods for the same object block (suspend execution) until the first thread is done with the object

End of Tutorial