Thread and Java Thread Programming

Sirak Kaewjamnong

517-312 Operating Systems

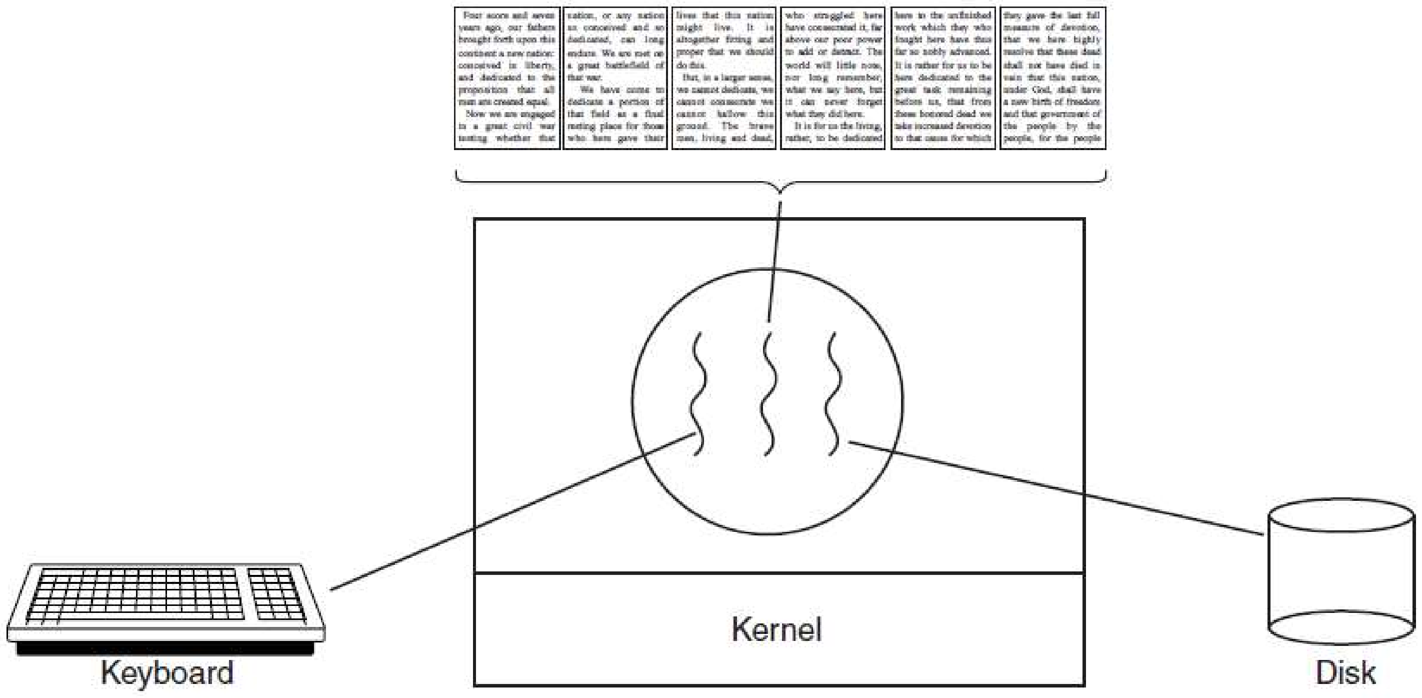
What is thread?

- Thread is lightweight process
- processes are typically independent, while threads exist as subsets of a process
- processes carry considerably more state information than threads, whereas multiple threads within a process share process state as well as memory and other resources
- processes have separate address spaces, whereas threads share their address space
- processes interact only through system-provided inter-process communication mechanisms
- context switching between threads in the same process is typically faster than context switching between processes

Motivation for using threads

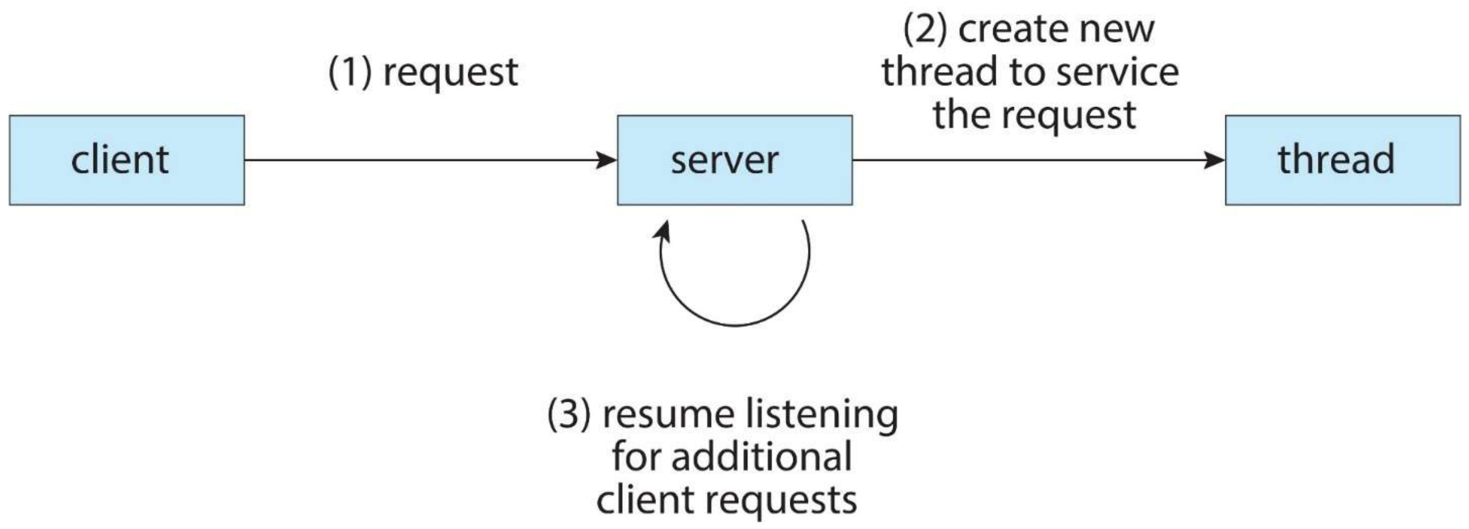
- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
    - Update display
    - Fetch data
    - Spell checking
    - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

Thread example



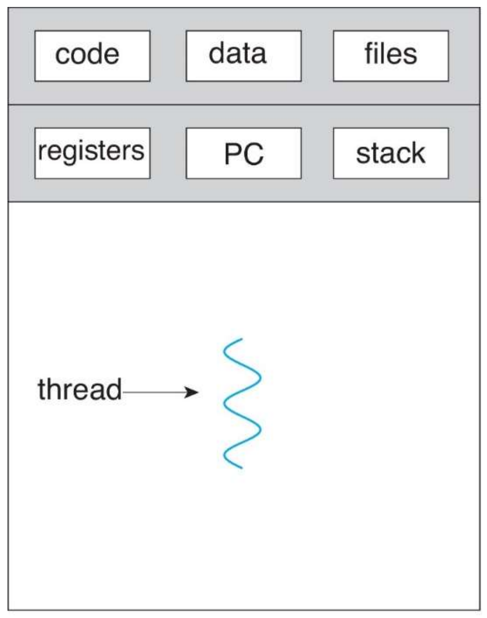
A word processor with three threads

Multithreaded Server Architecture

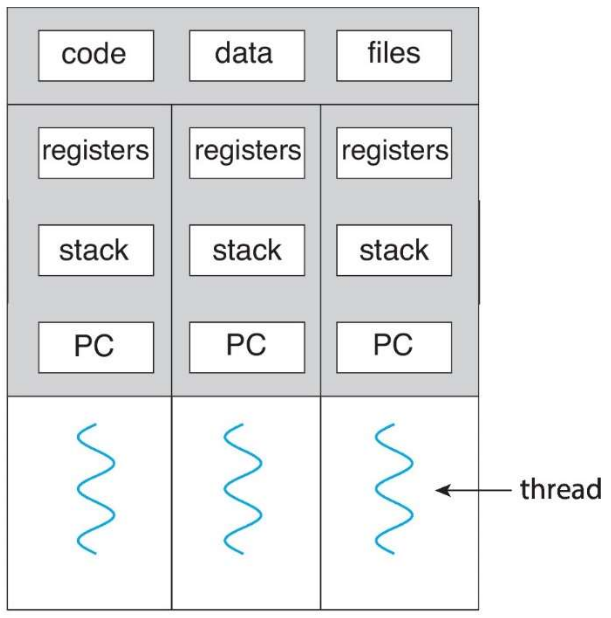


client
(1) request
server
(2) create new thread to service the request
thread
(3) resume listening for additional client requests

Single and Multithreaded Processes



single-threaded process

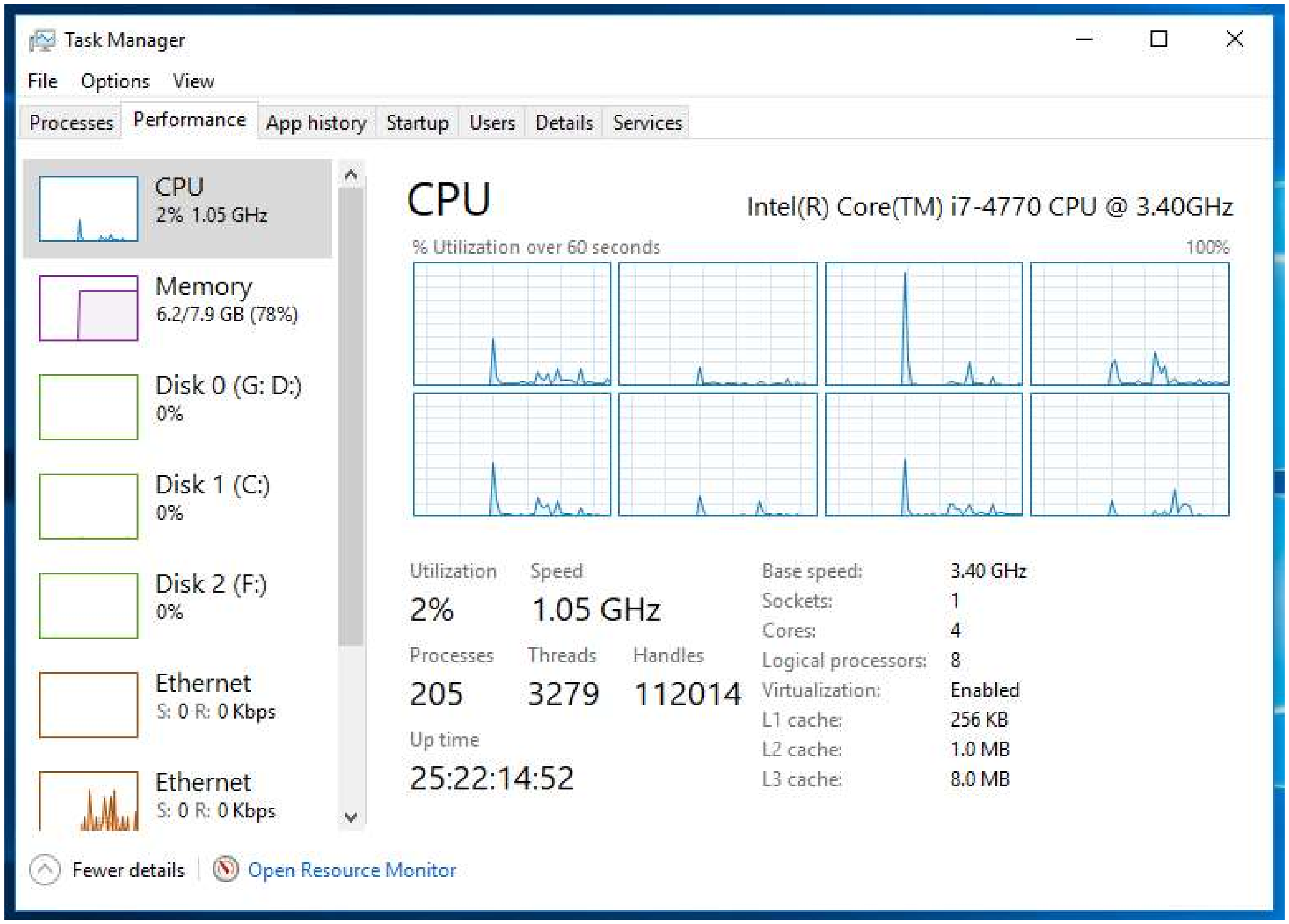


multithreaded process

Benefits

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching
- **Scalability –** process can take advantage of multicore architectures

How many threads in a system?



CPU Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz

Utilization	2%
Speed	1.05 GHz
Processes	205
Threads	3279
Handles	112014
Up time	25:22:14:52
Base speed:	3.40 GHz
Sockets:	1
Cores:	4
Logical processors:	8
Virtualization:	Enabled
L1 cache:	256 KB
L2 cache:	1.0 MB
L3 cache:	8.0 MB

Java Thread

There are two ways to create tasks and threads in Java

1. Implement Runnable interface
2. Implement by extending Thread class

Create by using Runnable interface

To create tasks:

- Declare a class for task, the class must implement the Runnable interface
- Implementing the run() method in the task class. This method tell the system how thread is going to run
- Once a task class have declared, creating a task using its constructor
- A task must be executed in a thread by invoking start() method

Declare a class

```
// task class
public class TaskClass implement Runnable{
            . . .
            // constuctor
            public TaskClass(...){
                        . . .
            }
        // implement the run method in Runnable
            public void run(){
            // Tell system how to perform
                        . . .
            }
}
```

```
// client class
public class Client{
            . . .
            public void someMethod(){

               //create an instance of Taskclass
               TaskClass task = new
TaskClass(...);

            //create a thread
               Thread thread = new Thread(task);

               //start a thread
               thread.start();
            }
}
```

Create by using Extending Java Thread

- Create a new class that extends Thread
- Override the run() method
- Create an instance of that class
- A task must be executed in a thread by invoking start() method

Declare a class

```java
// task class
public class ThreadClass extends Thread{
            . . .
            // constuctor
            public ThreadClass(...){
                        . . .
            }
        // Override the run method in Runnable
            public void run(){
            // Tell system how to perform
                        . . .
            }
}
```

```java
// client class
public class Client{
            . . .
            public void someMethod(){

               //create a thread

               ThreadClass thread1 = new
ThreadClass(...);


               //start a thread
               thread.start();
            }
}
```

Example: Create thread class

```
class ThreadPrintChar extends Thread{
    private char charToPrint;
    private int times;

    public ThreadPrintChar(char c, int t){
        charToPrint = c;
        times = t;
    }
    public void run(){
        for(int i=0;i<times;++i){
            System.out.print(charToPrint);
        }
    }
}
```

```
class ThreadPrintNum extends Thread{
    private int lastNum;
    public ThreadPrintNum(int n){
        lastNum = n;
    }
    public void run(){
        for(int i=0;i<lastNum;++i){
            System.out.print(i+" ");
        }
    }
}
```

Example: Create and run instance of the threads

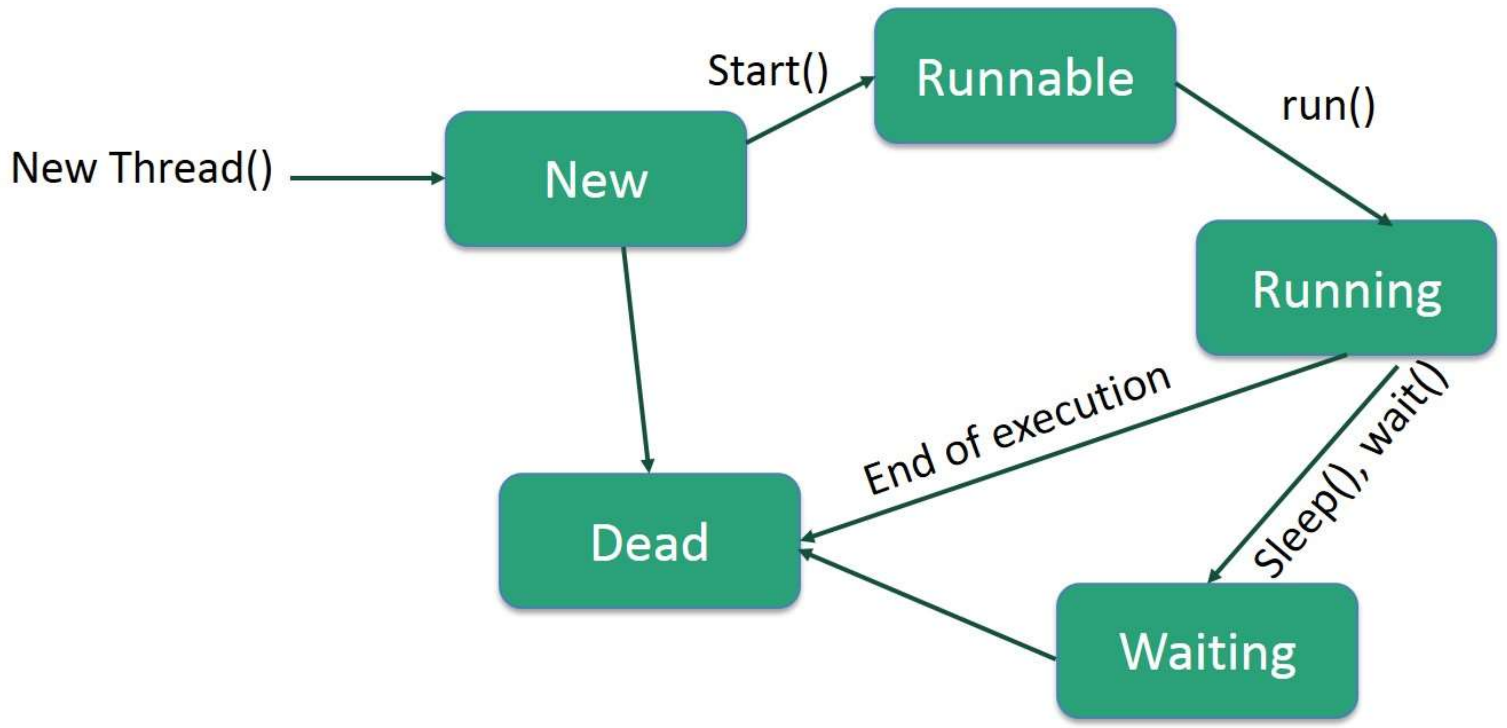
```java
public class ThreadDemo {
    public static void main(String[] args) {
        ThreadPrintChar thread1 = new ThreadPrintChar('A', 100);
        ThreadPrintChar thread2 = new ThreadPrintChar('B', 100);
        ThreadPrintNum  thread3 = new ThreadPrintNum(100);

        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

Thread life cycle

- New - When we create an instance of Thread class, a thread is in a new state.
- Running – Java thread is in running state.
- Suspended - A running thread can be suspended, which temporarily suspends its activity. A suspended thread can then be resumed, allowing it to pick up where it left off.
- Waiting or Blocked - A java thread can be blocked when waiting for a resource.
- Dead or Terminated - A thread can be terminated, which halts its execution immediately at any given time. Once a thread is terminated, it cannot be resumed.

Thread life cycle



https://www.tutorialspoint.com/java/java_multithreading.htm

Methods for controlling threads

- void start() : start the thread that causes the run() method to be invoked by the JVM
- boolean isAlive() : Tests whether the thread is currently running
- void setPriority(p: int) : set priority p (from 1 to 10) for this thread
- void join() :waits for this thread to finish
- void sleep(millis: long) : put the runnable thread to sleep for a specified time in ms
- void yield() : causes this thread to pause temporarily and allow other threads to execute
- void interrupt() : interrupts this thread

Example: sleep method

- sleep method may throw InterruptedException which is a checked execution
- Such an exception may occur when a sleeping thread's interrupt() method is called

```
public void run(){
        try{
          for(int i=0;i<times;++i){

          System.out.print(charToPrint);
                              Thread.sleep(10);
          }
        }catch(InterruptedException ex){
        }
    }
```

Example: join method

- join() method forces one thread to wait for another thread to finish

```
class ThreadPrintNum extends Thread{
    private int lastNum;
    public ThreadPrintNum(int n){
        lastNum = n;
    }
    public void run(){
        Thread thread4 = new Thread(new ThreadPrintChar('C', 150));
        thread4.start();
        try{
            for(int i=0;i<lastNum;++i){
                System.out.print(i+" ");
                if(i == 50) thread4.join();
             }
        }catch(InterruptedException ex){
      }
    }
}
```

Thread pools

- How to create a large number of thread? It is inconvenient for create a number of threads one by one
- A thread pool is ideal to manage the number of tasks executing concurrently
- To create an Executor object, use the static method in the Executors class
  - The newFixedThreadPool(int) method creates a fixed number of threads in a pool
  - The newCachedThreadPool() method creates a new thread if all the threads in the pool are not idle and there are tasks waiting for execution, A thread in a cached pool will be terminated if it has not been used for 60 seconds

Thread pools example

```java
import java.util.concurrent.*;
public class ExecutorDemo {
    public static void main(String[] args){
        //ExecutorService executor = Executors.newCachedThreadPool();
        // try changing parameter from 3 to 1 and see what happen
        ExecutorService executor = Executors.newFixedThreadPool(3);
        executor.execute(new ThreadPrintChar('a',100));
        executor.execute(new ThreadPrintChar('B',100));
        executor.execute(new ThreadPrintNum(100));
        executor.shutdown();
    }
}
```

Thread synchronization

- A shared resource may be corrupted if it is accessed simultaneously by multiple threads
- The synchronized keyword can be used to synchronize the method so that only one thread can access the method at a time

```
public synchronized void xMethod(){
        // method body
}
```

- The synchronized keyword can also be used to synchronize an object

```
synchronized (objectA){
     objectA.methodOfA();
}
```

Example: synchronizeation

- Code without synchronization

```
import java.util.concurrent.*;
public class AccountWithThread {
    private static Account account = new Account();
    public static void main(String[] args){
        ExecutorService executor = Executors.newCachedThreadPool();
        for(int i=0;i<100;i++){
            executor.execute(new AddAPennyTask());
        }
        executor.shutdown();
        while (!executor.isTerminated()){
        }
        System.out.println("What is balance? " + account.getBalance());
    }
```

Example: synchronizeation (cont)

```
private static class AddAPennyTask extends Thread{
    public void run(){
             account.deposit(1);
    }
}
private static class Account{
    private int balance = 0;
    public int getBalance(){
        return balance;
    }
    public void deposit(int amount){
        int newBalance = balance + amount;
        try{
            Thread.sleep(5);
        }catch(InterruptedException ex){

        }
        balance = newBalance;
    }
}
}
```

Run and see result

- The code creates 100 threads that execute deposit method
- Balance is initially 0 and it has been added one by one by deposit method
- Result should be 100 but…….
- What caused the error in the program?

Synchronizing it..

- By adding synchronized keyword, the result should change

```
private static class AddAPennyTask extends Thread{
    public void run(){
        synchronized(account){
            account.deposit(1);
        }
    }
}
```

- There are also other way to synchronize method or object, ReentrantLock class can also be used
- Sometimes two or more threads need to acquire the locks on several shared objects and cause a deadlock which each thread has the lock on one object and is waiting for the lock on the other object. This will be discussed later.

References

- Operating System Concepts 10th book official slides by Abraham Silberschatz, Greg Gagne, Peter B. Galvin
- Andrew S. Tanenbaum, Herbert Bos, Modern Operating 4th edition, Pearson, 2015
- Introduction to Java programming 7th edition by Y. Daniel Liang
- https://www.tutorialspoint.com/java/java_multithreading.htm