Threads

Thread

- A thread is a basic unit of CPU utilization.
- Thread comprises a thread ID, a program counter (PC), a register set, and a stack.
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
- A traditional process has a single thread of control.
- If a process has multiple threads of control, it can perform more than one task at a time.

Thread

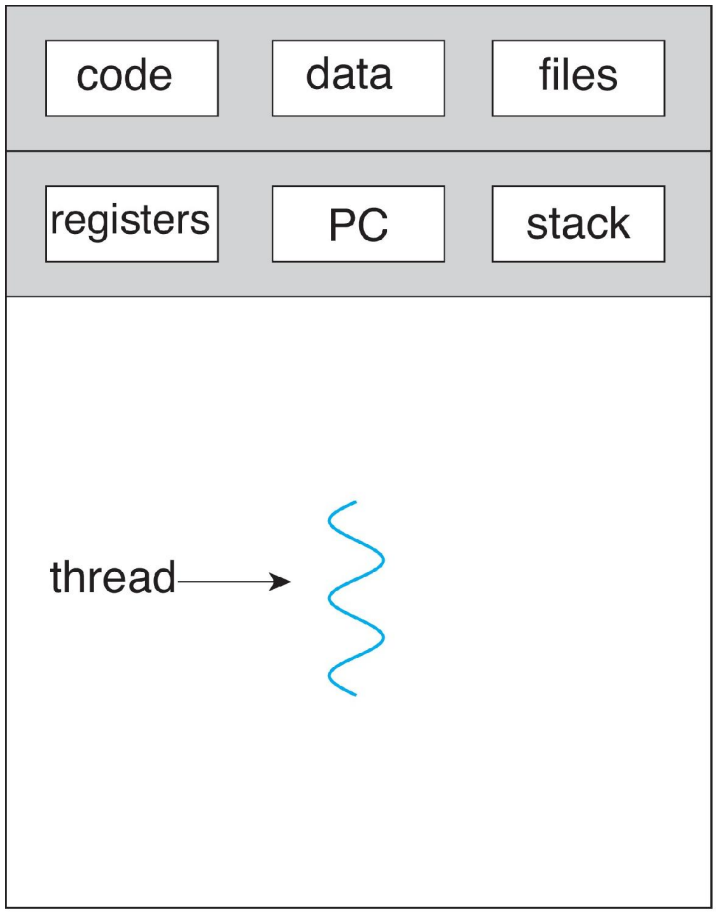
- Most software applications that run on modern computers and mobile devices are multithreaded.
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation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

Thread

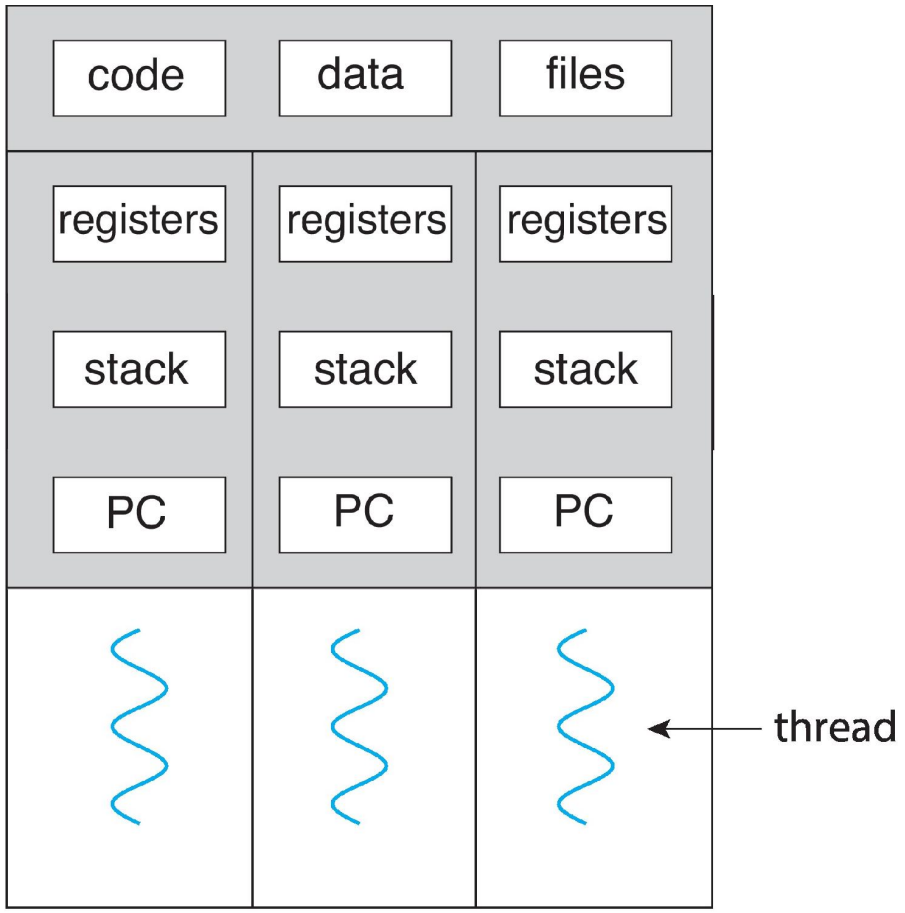
Examples of multithreaded applications:

- An application that creates photo thumbnails from a collection of images may use a separate thread to generate a thumbnail from each separate image.
- A web browser might have one thread display images or text while another thread retrieves data from the network.
- A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

Single and Multithreaded Processes



single-threaded process



multithreaded process

Benefits

Responsiveness – Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.

- This quality is especially useful in designing user interfaces.

Resource Sharing – Processes can share resources only through techniques such as shared memory and message passing.

- Such techniques must be explicitly arranged by the programmer.
- However, threads share the memory and the resources of the process to which they belong by default.

Benefits

Economy – Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong.

- It is more economical to create and context-switch threads. In general thread creation consumes less time and memory than process creation.
- Additionally, context switching is typically faster between threads than between processes.

Scalability – The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores.

- A single-threaded process can run on only one processor, regardless how many are available.

Process	Thread
Process means any program is in execution.	Thread means segment of a process.
Process takes more time to terminate.	Thread takes less time to terminate.
It takes more time for creation.	It takes less time for creation.
It also takes more time for context switching.	It takes less time for context switching.
Process is less efficient in term of communication.	Thread is more efficient in term of communication.
Process consume more resources.	Thread consume less resources.
Process is isolated.	Threads share memory.
Process is called heavy weight process.	Thread is called light weight process.
Process switching uses interface in operating system.	Thread switching does not require to call a operating system and cause an interrupt to the kernel.
If one process is blocked then it will not effect the execution of other process	Second thread in the same task couldnot run, while one server thread is blocked.
Process has its own Process Control Block, Stack and Address Space.	Thread has Parents' PCB, its own Thread Control Block and Stack and common, Address space.

Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- There are two primary ways of implementing a thread library
  - ✔ The first approach is to provide a library entirely in user space with no kernel support. All code and data structures for the library exist in user space.
  - ✔ The second approach is to implement a kernel-level library supported directly by the operating system.

Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- ***Specification***, not ***implementation***
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Linux & Mac OS X)

Java Threads

- Java threads are managed by the JVM
- Typically implemented using the threads model provided by underlying OS
- Java threads may be created by:
  - ✔ Extending Thread class
  - ✔ Implementing the Runnable interface

```
public interface Runnable
{
    public abstract void run();
}
```

  - ✔ Standard practice is to implement Runnable interface

Java Threads

Implementing Runnable interface:

```
class Task implements Runnable
{
   public void run() {
      System.out.println("I am a thread.");
   }
}
```

Creating a thread:

```
Thread worker = new Thread(new Task());
worker.start();
```

Waiting on a thread:

```
try {
   worker.join();
}
catch (InterruptedException ie) { }
```

Java Threads

```
class NewThread implements Runnable {
Thread t;
NewThread() {
     t = new Thread(this, "Demo Thread");
     System.out.println("Child thread: " + t);
     t.start();
}
public void run() {
     try {
     for(int i = 5; i > 0; i--) {
          System.out.println("Child Thread: " + i);
          Thread.sleep(500);
     }
     }
     catch (InterruptedException e) {
     System.out.println("Child interrupted.");
     }
     System.out.println("Exiting child thread.");
}
}
```

```
class ThreadDemo {
     p s v main(String args[ ] ) {
     new NewThread();
     }
}
```

Java Threads

```
class NewThread extends Thread {
NewThread() {
    super("Demo Thread");
    System.out.println("Child thread: " + this);
    start();
}
public void run() {
    try {
    for(int i = 5; i > 0; i--) {
        System.out.println("Child Thread: " + i);
        Thread.sleep(500);
    }
    } catch (InterruptedException e) {
    System.out.println("Child interrupted.");
    }
    System.out.println("Exiting child thread.");
}
}
```

```
class ThreadDemo {
    p s v main(String args[ ] ) {
    new NewThread();
    }
}
```