Some clarifications about OS

- OS role:
  - Properly functioning OS ensures that protection between processes are not breached. OS makes no such promise for threads within a single process.
    - Two processes cannot share memory unless explicitly allowed. Two threads can trample on the local memory. Sometimes (not always) you get segmentation violation. Regardless, you cannot rely on the OS giving you segmentation fault all the time.

```
void *ptr = random(2^32);
*ptr=1;
```

Threading Issues

1. Semantics of fork() and exec() system calls
   - Does **fork()** duplicate only the calling thread or all threads?
2. Thread cancellation
   - Asynchronous cancellation terminates the target thread immediately
   - Deferred cancellation allows the target thread to periodically check if it should be cancelled

3: Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred
  - Ctrl-C sends SIGINT (Interrupt)
  - float x=1/0; sends SIGFPE
- A signal handler is used to process signals
  - Signal is generated by particular event
  - Signal is delivered to a process
  - Signal is handled
- Options:
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the process

Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool

Thread Specific Data

- Allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the thread library
- This communication allows an application to maintain the correct number kernel threads

Windows XP Threads

- Implements the one-to-one mapping
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the context of the threads
- The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

Linux Threads

- Linux refers to them as tasks rather than threads
- Thread creation is done through clone() system call
- clone() allows a child task to share the address space of the parent task (process)

Java Threads

- Java threads are managed by the JVM
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface

Summary

- Processes are programs in execution
  - PCB encapsulates all data for processes
  - Processes go through many queues depending on the state, RUN, Ready or Wait.
  - Processes share with other processes using shared memory or message passing; by default processes are isolated
- Threads are used to assign multiple CPUs to a process
  - Maps m user level threads to n kernel level threads