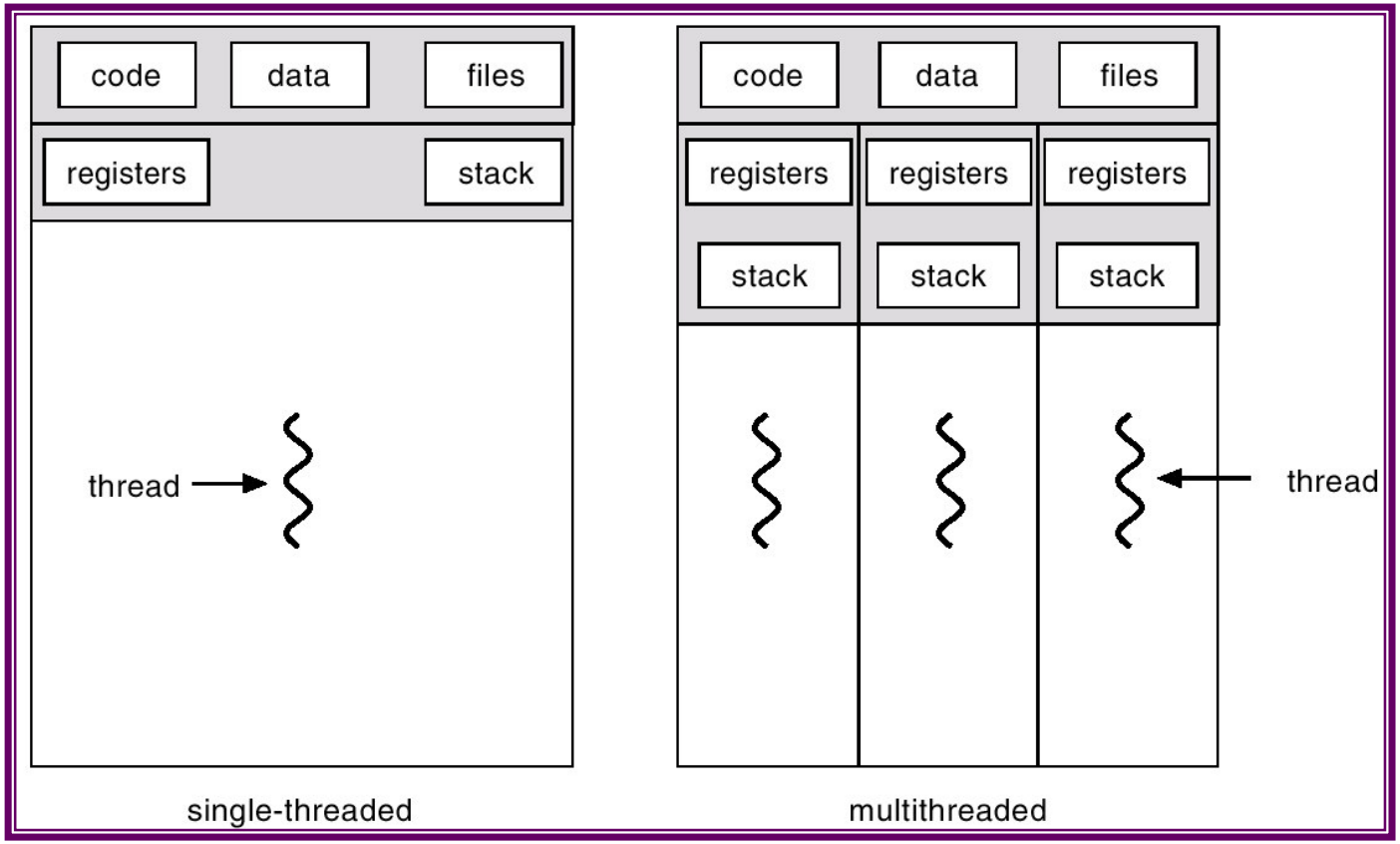
Chapter 5: Threads

- Overview.
- Multithreading Models.
- Threading Issues.
- Thread Pools.
- Java ThreadLocal.
- Pthreads.
- Solaris 2 Threads.
- Java Threads.

Single and Multithreaded Processes



Benefits

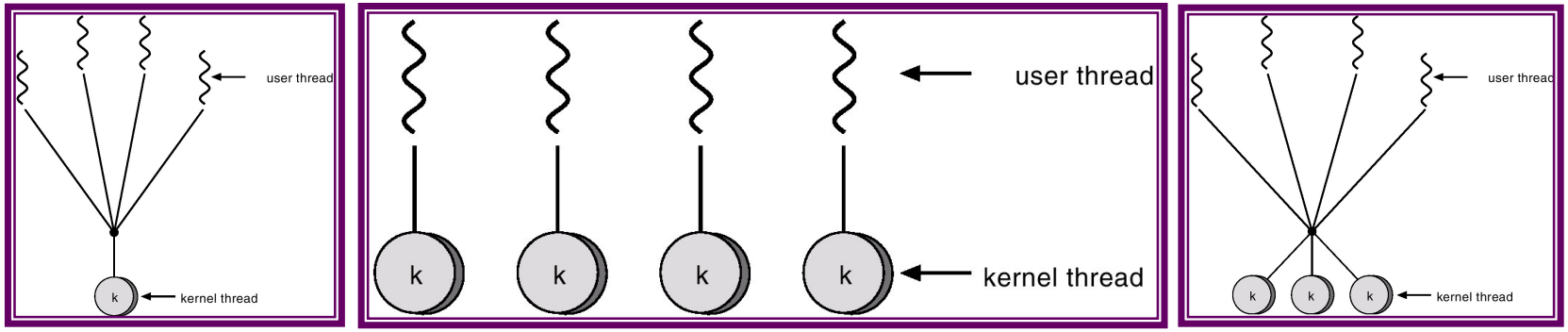
- Responsiveness: can still type in Word (main thread) while Word formats document for printing in other thread.
- Resource Sharing: all threads share same process memory.
- Economy: context-switching less expensive (e.g. page table registers need not be saved).
- Utilization of Multi-Processor Architectures: different threads, different CPUs.

User vs. Kernel Threads

- Pure user threads:
  - Thread management done by user-level library, i.e. kernel has no awareness application is multi-threaded.
  - No pre-emption: one thread has to willingly relinquish CPU and let other go ahead. To kernel, switch looks like process just did a jump.
  - If one thread blocks, kernel suspends process, so all other threads can't move forward either.
- Pure kernel threads:
  - Managed by kernel.
  - Thread can be pre-empted if overusing CPU.
  - If thread blocks; other threads can proceed.
- Pure user threads are cheap: no context-switching, no kernel data.

Multithreading Models

- OS offers both user and kernel threads.
- Apps always work with user-level thread API which may relate to kernel threads in one of three ways:
  - Many-to-One: no kernel threads, only pure user threads; see previous slide for discussion.
  - One-to-One: each user thread is a shallow shell around a single kernel thread; see previous slide for discussion.
  - Many-to-Many: use as many kernel threads as needed, e.g. create one if a user thread blocks. Most efficient.



Threading Issues

- Semantics of fork() and exec() system calls:
  - Does fork() duplicate all threads in new process?
  - Does exec() destroy all threads and overwrite complete process?
- Thread cancellation:
  - Asynchronous: Java stop().
  - Synchronous: thread checks “should I quit?” at *cancellation points*. Preferred because thread can release resources.
- Signal handling: which thread gets signal?
  - *Signal* is to user processes as are interrupts for kernels: asynchronous notification.
  - Some go to all threads (e.g. Ctrl+C to terminate).
  - UNIX: first thread that is not blocking (not masking).
  - Solaris 2: special thread to handle signals.

Threading Pools

- If database server starts one thread for each transaction, then on busy day, disk *thrashes*: disk heads spend most time moving between cylinders, not serving data.
- Solution: create *pool* with 10 threads, assign to transactions.
- If more transactions than threads, queue them up or refuse service.
- Thread reuse also reduces overhead: no thread creation/deletion (very expensive for kernel threads).

Java ThreadLocal

Local data to each thread (besides stack variables):

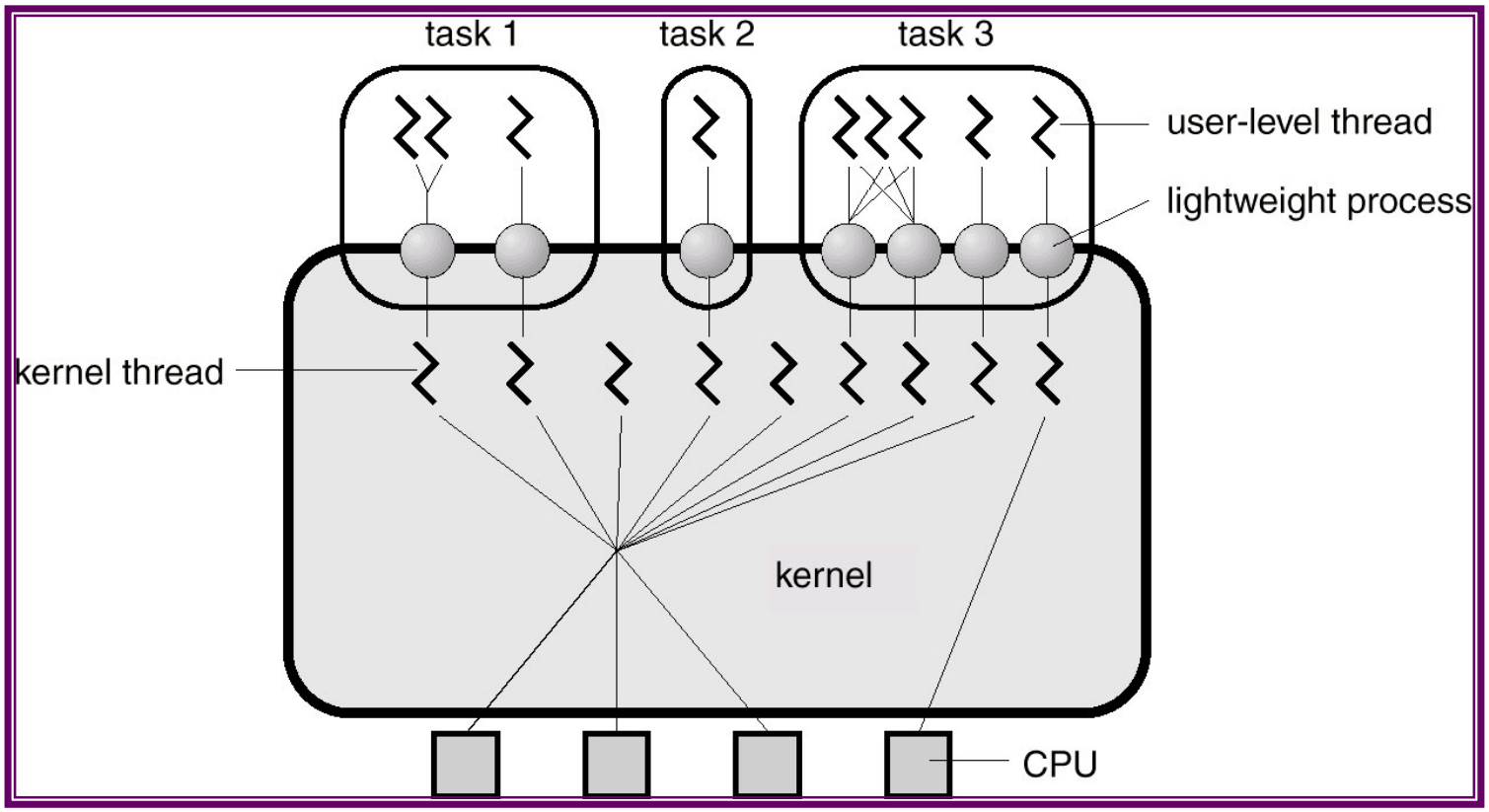
```
class TransactionRunner {
    ThreadLocal mPriority=new ThreadLocal();
    void setPriority(int p) {mPriority.set(new Integer(p)); }
    void exec(Transaction t) {
          int p=((Integer)(mPriority.get())).intValue();
          /* Transaction t is executed based on thread priority p. */ } }

class ClientSaleProcessor extends Thread {
    static Object sLock=new Object();
    static int sNext=0;
    static TransactionRunner sRunner=new TransactionRunner();
    public void run() {
          int p; synchronized (sLock) { p=sNext++; }
          sRunner.setPriority(p);
          /* All calls to sRunner.exec() by ANY method EVER called
          by this thread will receive priority p. */ } }
```

- No need to pass p around to every call during execution of this thread.
- ThreadLocal is like Hashtable with implicit key:
  - set(x) is like put(Thread.currentThread(),x);
  - get() is like get(Thread.currentThread());
  - Unlike Hashtable, entry automatically removed when thread ends.

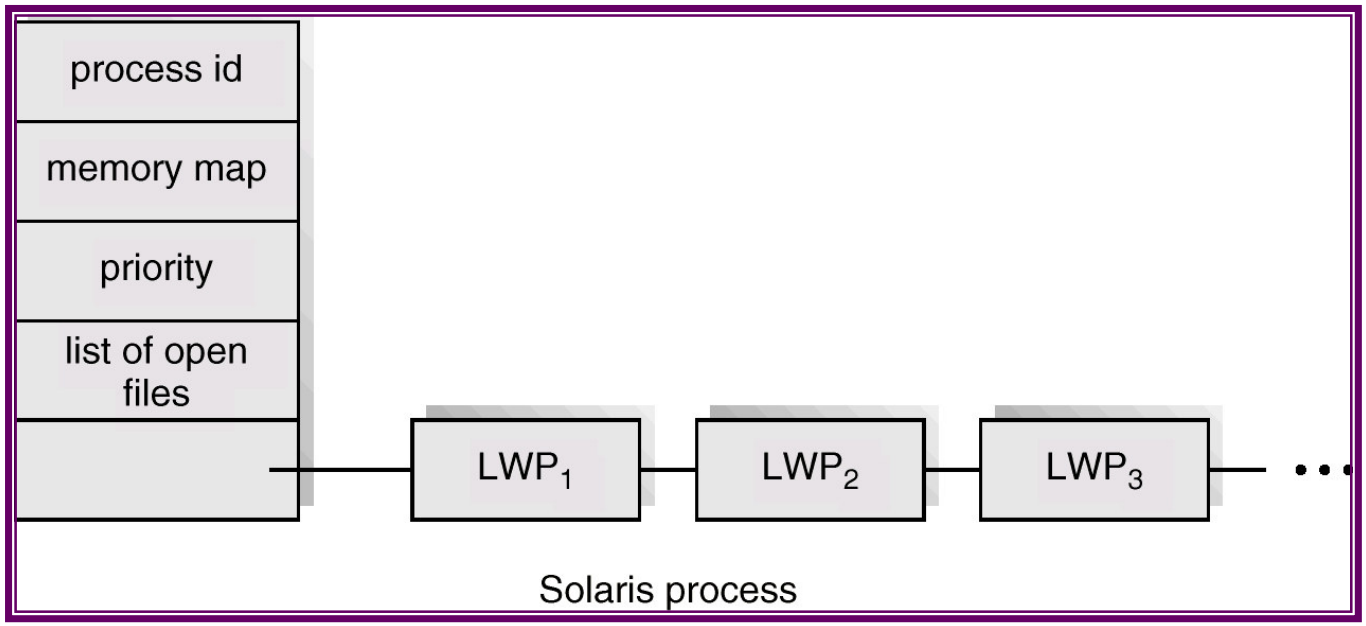
Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization.
- API specifies behavior of the thread library, implementation is up to development of the library. Often user-level.
- Common in UNIX operating systems.

Solaris 2 Threads



Solaris Process



Java Threads

- Java thread creation:
  - Extend `Thread` class:

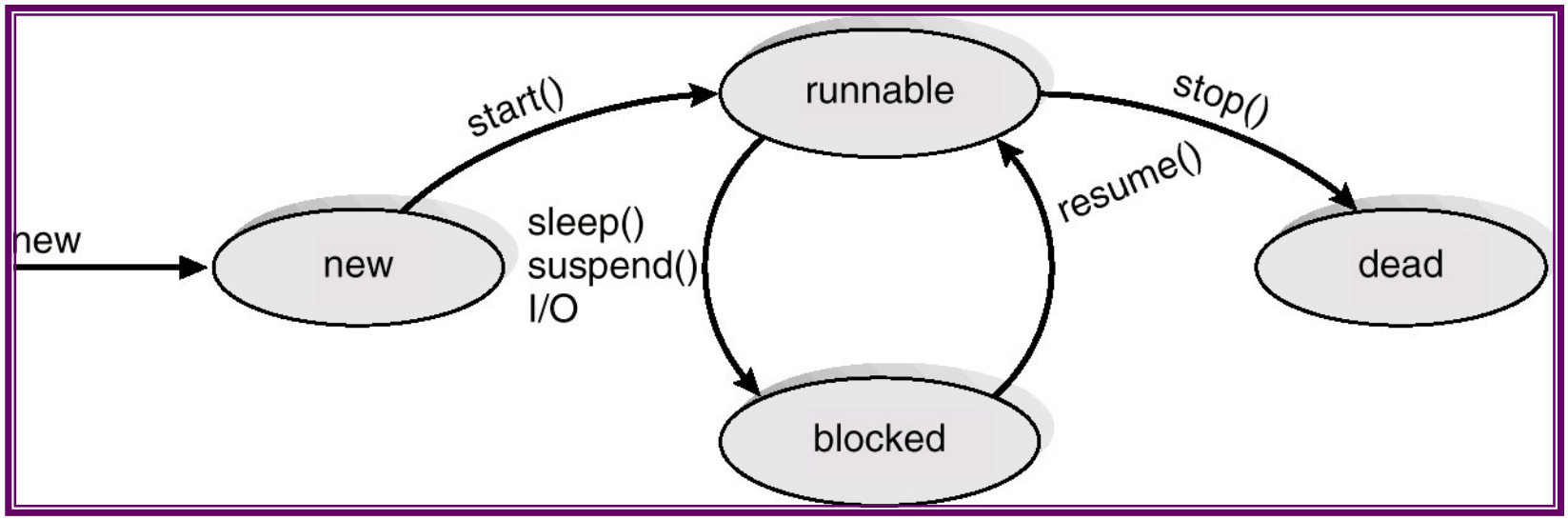
```
class MyThread extends Thread {
 public void run() { /* computation */ } }
// Creation.
(new MyThread()).start(); // NOT .run()
```

  - Implementing the `Runnable` interface

```
class MyRunnable implements Runnable {
 public void run() { /* computation */ } }
// Creation.
(new Thread(new MyRunnable())).start();
```

    - Necessary because Java disallows multiple parents.
- Java threads are managed by the JVM: can be one-to-one or any other model.
- Key element of assignment 1.

Java Thread States



- Deprecated:
  - suspend().
  - resume().
  - stop().
- Why?
  - suspend(), stop() prevent thread from releasing its resources.
  - resume() deprecated because it is inverse of suspend().
  - See Thread class javadoc.