Today

- IPC Wrap-up
- Threads

Review

- What are ways that processes communicate?

Tradeoffs

- What are some of the relative advantages and disadvantages of these approaches to IPC?
  - Shared memory
  - Message passing
  - Pipes
  - Sockets

Tradeoffs Summary

- Communication overhead
  - Higher with message passing, sockets
- Amount of cooperation/collaboration
  - Higher with shared memory, pipes
- Amount of protection
  - Higher with sockets → no direct access

THREADS

Parallel Execution

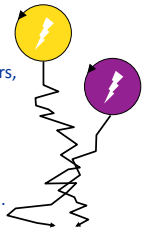
- When two or more execution events are being carried out simultaneously.
- Examples:
  - A Disk I/O and a CPU operation
  - Several CPU operations on a multiprocessor system

Concurrent Execution

- When two or more execution events either appear to or actually do occur simultaneously.
- A superset of parallel execution

Threads

- A thread is a stream of control....
  - Executes a sequence of instructions.
  - Thread identity is defined by CPU register context (PC, SP, ..., page table base registers, ...)
  - Generally, "context" is the register values and referenced memory state
- Multiple threads can execute independently:
  - They can run in parallel on multiple cores...
    - physical concurrency
  - ...or arbitrarily interleaved on some single core.
    - logical concurrency

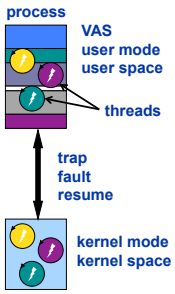


Threads vs Processes

- Threads executing within the same process share *most* of their address space.
- All threads in a process share the same:
  - Code segment
  - Data segment
  - Heap
- Each thread must have its own:
  - Program counter
  - Register values
  - Stack segment (i.e., local variables and parameters)

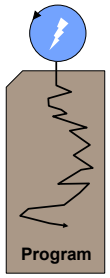
Threads and the kernel

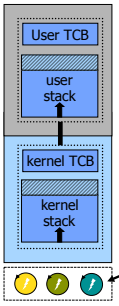
- Modern operating systems have multi-threaded processes.
- A program starts with one main thread, but once running, it may create more threads.
- Threads may enter the kernel (e.g., via syscall).
- Threads are known to the kernel and have separate kernel stacks, so they can block independently in the kernel.
  - Kernel has syscalls to create threads (e.g., Linux clone).
- Implementations vary.
  - This model applies to Linux, MacOS-X, Windows, Android, and pthreads or Java on those systems.

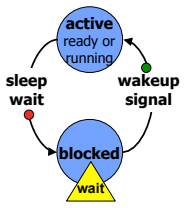


A thread

Looks familiar because applies to the ***process*** abstraction too, or, more precisely, to a process's **main thread.**







When a thread is **blocked,** its TCB is placed on a **sleep queue** of threads waiting for a specific wakeup event.

Java Threads: The Basics

- Extend the Java Thread class.

```
class MyThread extends Thread {
    public void run() {
        // do task: your code here
    }
}

…

Thread t1 = new MyThread();
t1.start();
```

Thread Methods

Name	Purpose
start	Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.
yield()	A hint to the scheduler that the current thread is willing to yield its current use of a processor.
join(long millis)	Waits at most millis milliseconds for this thread to die.

Among others….

Java Threads: The Basics

```
public class RunnableTask implements Runnable {

    public RunnableTask(…) {
        // save any arguments or input for the task (optional)
    }

    @Override
    public void run() {
        // required to implement for Runnable interface
        …
    }

}

…

RunnableTask task = new RunnableTask();
Thread t1 = new Thread(task, "thread1");
t1.start();
```

Example: Jabber

```
class Jabber implements Runnable {
      String str;
      public Jabber(String s){ str = s; }
      public void run() {
            while (true) {
                  System.out.print(str);
                  System.out.println();
            }
      }
}
```

```
public class JabberTest {
      public static void main(String[] args) {
            Jabber j = new Jabber("1");
            Jabber k = new Jabber("2");
            Thread t = new Thread(j);
            Thread u = new Thread(k);
            t.start();
            u.start();
      }
}
```


Looking Ahead

- Wednesday:
  - Midterm
- Two Wednesdays
  - Project 2 due
- Monday
  - Challenges in multi-threaded programming
  - Synchronizing threads