Procedures and Threads

What is a process?

- code, data, and stack
    - usually has its own address space
- program state:
    - CPU registers
    - program counter
    - stack pointer
- only one process can be running in a single CPU core at any given time

The Process Model;

- Multiprogramming of 4 programs
- conceptual model:
    - 4 independent processes
    - processes run sequentially
- only one program active at any instant
    - instant can be very short
    - only applies if there is a single CPI (with a single core) in the system
- context switches

When is a process created?

- created in 2-ways
    - system initialization: one or more processes created when the OS starts up
    - execution of a process creation system call: something explicitly asks for a new process

Processes and voluntary or involuntary:

- voluntary: normal exit
- involuntary: fatal error, killed by another process

Process Hierarchies:

- parent creates a child process
- forms a hierarchy
    - if a process terminates, its children are inherited by there terminating process's parent

Process States

- process in one of 5 states:

    - created
    - ready
    - running
    - blocked
    - exit
- transitions between states:
    - 1. Process enters ready queue
    - 2. scheduler picks this process
    - 3. scheduler picks a different process
    - 4. process waits for an event such as I/O
    - 5. event occurs
    - 6. process exits
    - 7. process ended by another process

Process in the OS:

- two layers for processes
- lowest layer of process - structured OS handles interrupts, schedulers
- above that layer are sequential processes

What's in a process table entry?

- process Management:
    - registers
    - PC
    - CPU status word
    - stack pointer
    - process state
- File Management:
    - root directory
    - current directory
    - file descriptors
    - user ID
    - group ID
- Memory Management:
    - pointers to text, data, stack
    - or
    - pointer to page table

Threads: "processes" sharing memory

- process <-> address space
- thread <-> program counter / stream of instructions

Why use Threads?

- allow a single application to do many things at once
- threads are faster to create or destory
- overlap computation and I/O

Implementing threads:

- user-level threads
- kernel-level threads