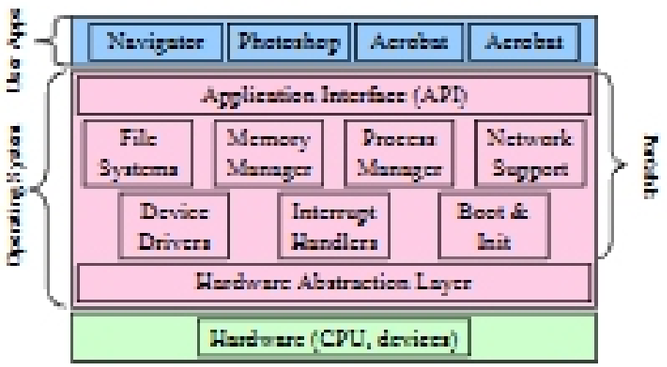
CS 537
Lecture 4: Processes

Michael Swift

Process Management

- This lecture begins a series of topics on processes, threads, and synchronization
- Today: processes and process management
  - what are the OS units of execution?
  - how are they represented inside the OS?
  - how is the CPU scheduled across processes?
  - what are the possible execution states of a process?
    - and how does the system move between them?

Example OS in operation



Why Processes? Simplicity + Speed

- Hundreds of things going on in the system

- How to make things simple?
  - Separate each to an isolated process
  - Communication
- How to speed-up?
  - Overlap I/O bursts of one process with CPU bursts of another

The Process

- The process is the OS's abstraction for execution
  - the unit of execution
  - the unit of scheduling
  - the dynamic (active) execution context
    - compared with program: static, just a bunch of bytes
- Process is often called a job, task, or sequential process
  - a sequential process is a program in execution
    - defines the instruction-at-a-time execution of a program

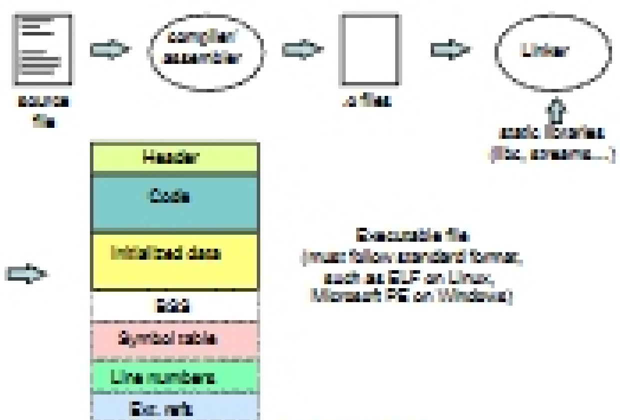
What is a program?

A program consists of:

- Code: machine instructions
- Data: variables stored and manipulated in memory
  - initialized variables (globals)
  - dynamically allocated variables (malloc, new)
  - stack variables (C automatic variables, function arguments)
- DLLs: libraries that were not compiled or linked with the program
  - containing code & data, possibly shared with other programs
- mapped files: memory segments containing variables (mmap())
  - used frequently in database programs

- What's the relationship between a program and process?
  - A process is a executing program

Preparing a Program



Running a program

- OS creates a "process" and allocates memory for it
- The loader:
  - reads and interprets the executable file
  - sets process's memory to contain code & data from executable
  - pushes "argc", "argv", "envp" on the stack
  - sets the CPU registers properly & calls "__start()" [Part of CRT0]
- Program start running at __start(), which calls main()
  - we say "process" is running, and no longer think of "program"
- When main() returns, CRT0 calls "exit()"
  - destroys the process and returns all resources

What's in a Process?

- A process consists of (at least):
  - an address space
  - the code for the running program
  - the data for the running program
  - an execution stack and stack pointer (SP)
    - traces state of procedure calls made
  - the program counter (PC), indicating the next instruction
  - a set of general-purpose processor registers and their values
  - a set of OS resources
    - open files, network connections, sound channels, …
- The process is a container for all of this state
  - a process is named by a process ID (PID)
    - just an integer

Process != Program

Executable

Program is passive
• Code + data

Process is running program
• stack, regs, program counter

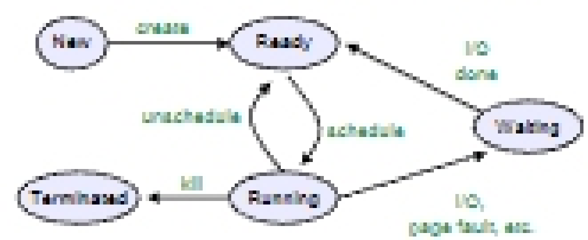
Example:
We both run IE:
- Same program
- Separate processes

Process address space

Process states

- Each process has an execution state, which indicates what it is currently doing
  - ready: waiting to be assigned to CPU
    - could run, but another process has the CPU
  - running: executing on the CPU
    - is the process that currently controls the CPU
    - pop quiz: how many processes can be running simultaneously?
  - waiting: waiting for an event, e.g. I/O
    - cannot make progress until event happens
- As a process executes, it moves from state to state
  - UNIX: run ps, STAT column shows current state
  - which state is a process in most of the time?

Process state transitions



- What can cause schedule/unschedule transitions?