4 The Abstraction: The Process

* The process is just a running program, but also a fundamental abstraction from the OS.

* The OS takes the bytes of a program & helps give them meaning by executing them.

* Typical OS may be running few/hundreds of processes @ once.
    ↳ don't have to worry if a CPU is available.

CRUX: How to provide the illusion of many CPU?

↳ do so by virtualizing the CPU, i.e., running one process, stopping it, and running another.

↳ this technique is known as time-sharing the CPU.
    ↳ performance cost of timesharing is that processes will run more slowly

* A mechanism is low-level methods/protocols that implement a needed piece of functionality.
    ↳ necessary for implementation of CPU virtualization
    ↳ A context-switch gives the OS the ability to stop running one program & start running another on a given CPU.

Tip: Space Sharing

* Space sharing is when a resource, i.e. disk space, is divided (in space) among those who wish to use it.

* OS implements policies (Algorithms) for making decisions.
    ↳ Scheduling Policy determines which program to run on a given CPU.

4.1 Abstraction: A Process

* we can summarize a process by the different parts of the system it access/affects.

* A process' **machine state** is what a program can read/update & what parts of the machine are important for its execution
  1. Process' memory (**address space**) are the locations (addresses) in memory that the program reads/write/ instructions are.
  2. Registers are included as many instructions access them
     - ↳ program counter
     - ↳ stack & frame pointer (func. params, local vars, return addresses)

Tip: Separate Policy & Mechanism

* Typical design paradigm of OS is to separate high-level policies from low level mechanisms.
  (which) (how)
  ↳ allows for modularity — don't have to change mechanisms if want to change policies

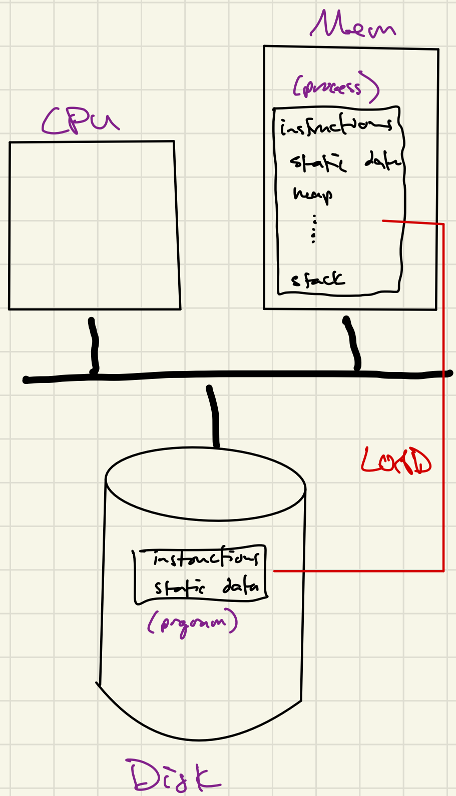
4.2 Process API

* following are descriptions of common functionality in a process API:

  1. **Create**: OS must have some method to create a process
  2. **Destroy**: Many processes terminate themselves when done running, but there should be a method for a user to kill them just in case
  3. **Wait**: Can be useful to wait for a process to stop running
  4. **Miscellaneous**:
     ↳ suspend?
  5. **Status**: Ready, Run, Wait, etc.
     ↳ runtime duration, state of process

4.3 PROCESS CREATION

* Q. How are programs transformed into processes?



① OS must load instructions & static data into memory (address space of process)
↳ OS must read bytes of an executable from disk/SSD and load to mem.
↳ Old OS load programs eagerly, i.e. all at once before executing
↳ Modern OS load lazily, i.e. load instructions/data as needed during execution.

② Allocate memory for the program's stack
↳ local vars, function params, return addrs
↳ initialize the stack of arguments (fill in argv & argc)

③ Initialization related to I/O
↳ setup file descriptors?

④ Start execution @ entry point
↳ relinquish control of CPU to the running process.

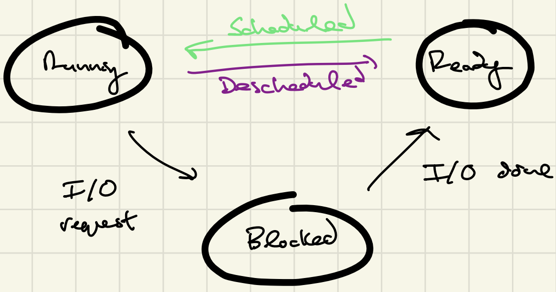
4.4 PROCESS STATES

* States that a process can be in @ any given time:

  * Running : A process is running on a processor
  * Ready : A process is ready to run but OS has chosen not to run it yet.

* Blocked : process performed some operation s.t. it is not ready to run until some other event takes place
  ↳ e.g. I/O request to a disk
  ↳ this means some other process can use the processor

* A process can be moved between Ready ⇄ Running @ discretion of OS.
  ↳ Ready → Running ⟹ Process was scheduled
  ↳ Running → Ready ⟹ Process was descheduled



describes how OS transitions process states.

NOTE : when I/O done doesn't mean OS will immediately run the process that requested!

4.5 Data Structures

* OS maintains some kind of process list to track the state of each process.
  ↳ track ready processes & info for running processes.
  ↳ must have a way of tracking blocked processes to know which ones to wake when their event completes.

* Xv6 Kernel tracks :
  * Register context : the state of the registers when a process becomes blocked.
    ↳ Allows for a process to resume exactly where it left off
  * pid, parent process, cwd, start & size of address space in mem, etc.
  * process' state
    ↳ Zombie State : used by some OS to indicate a process has finished but info hasn't been cleaned up.
      ↳ maybe parent process doesn't need to wait for child so doesn't need to examine & clean its final state.

    Note: Sometimes parent needs to wait for child process to examine its exit code, and will signal to OS to clean up its info after finished examining it!

Aside: The Process List

* Something similar is necessary for any time-sharing OS to complete context switches.
* Often refer to individual entries in the list as Process Control Blocks (PCB)

4.6 Summary

* will learn:
  1. Low-level mechanisms to implement processes
  2. High-level policies to schedule processes

→ These will teach us how OS virtualizes CPU