COMSW4118-4-1 9/14/2017

- Last time: **fork()**, **exec()** – to create/fork and exec ; **wait()/waitpid()**
  ← copies file descriptors as well!
- **exit()** – kills the process and exits w/ return code. [read, write, open, close, getpid]

- **Pipes** – connections b/n different processes, created by a system call:
  int pipe(int fd[2]) where fd[0] is to read and fd[1] is to write
  each process needs to close the end it is not using.

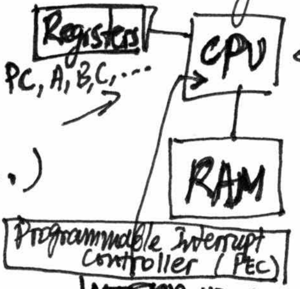
NOTE: anonymous pipes like the above can only be used b/n parent and child.

- **dup()** takes a file descriptor and mirrors it into another one (duplicate it) either to the next available or, in the case of **dup2()** to a provided one
  closing it first, if necessary
- **chdir()** to change working directories

- How do the OS services provided (system calls) work behind the covers? Going back to structure, the kernel is where the system calls are implemented.

APPS
— system calls —
KERNEL (OS)
———
HARDWARE (CPU...)

Taking a Bottom-up view:



← runs instruction in sequence has program counter (PC) to store next instruction address

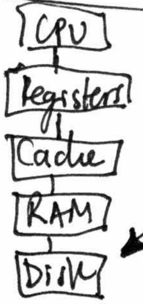
Ex. Instructions: add, subtract, load, ...

Branch and Loop instructions change the PC to sth. diff. Than ++PC.

Interrupting an interrupt? → depends on hardware: some allows only 1 interrupt, some allows nested ones.

INTERRUPTS

- To stop what CPU is doing & take care of urgent requests.
- In hardware, b/n each instruction the CPU checks for interrupts
- If one is found, CPU stops, remembers where it left off (saving PC & whole CPU state) and processes the interrupt
- Once done, The PC & state are restored & CPU goes on.
- Examples: keyboard input, devices in general (primary source), mouse move, hard drive, etc.
- Devices interface w/ a Programmable Interrupt Controller (PIC) which actually forwards interrupts to the CPU.



← Memory Hierarchy

Slower, so we use interrupts when read done, so we can keep going meanwhile

---

- Interrupts get handled by an Interrupt Handler, code defined by the OS this is installed by the OS in a place where the hardware knows to find it.
  - The interrupt handler is software, hence it uses the CPU (PC, etc.)
  - Most interrupt handlers are short: need to work quickly to not interrupt for long.
  - The interrupt handler saves the state and subsequently restores it.
    - What if it gets interrupted while processing an interrupt? (it runs on CPU, CPU checks for interrupts)
      A lot of systems just do not allow this ...