2018/4/9 Monday

Printer

- Machine: reliability
- Human: faster

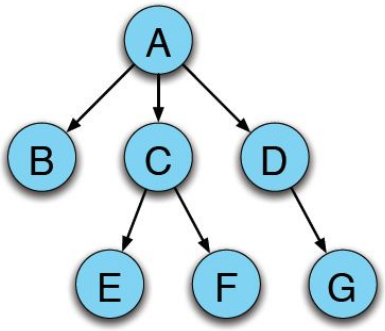
Interrupt

- Interrupt vector table
- Happens when a device needs something (e.g. a timer ran out)

Privilege state

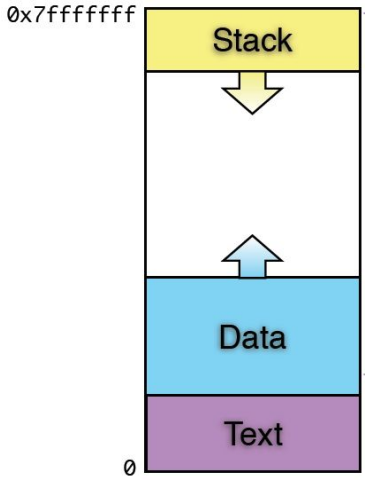
Processes

- Who makes the first processes?
- When is a process created?
    - 2 ways: system initialization and system calls
    - System calls come from: user requests, user programs, system daemons
- When do they end?
    - Voluntary: normal exit, error exit (could be “happy” (return 0) or “sad’ (return 1))
    - Involuntary: fatal error, killed by another process
- OS keeps track of all processes in a process table
    - Processes can produce other processes: parent and child relationship (sibling)



- what would happen to E&F if we take away C
    - Link them back to A, C’s parent
    - Kill E&F (orphan relationship)
- An efficiency way to keep the parent and child relationship: have the child point to parent (linked list)

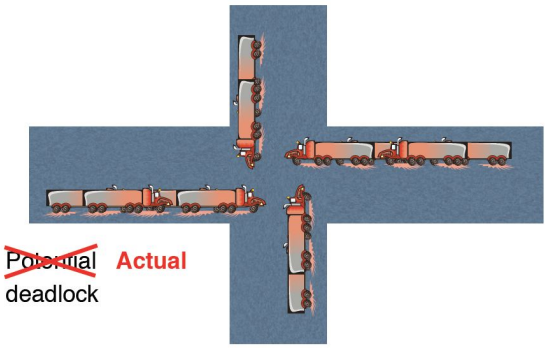
Inside a (Unix) process



- Why is stack upside down?
- Can you predict you are going to use more local space or space before running the program?
- Only 1 processes can happen at the 0th location of memory

Deadlock

- They need not happen



- Potential deadlock: when they are some distance apart and not moving
- Actual deadlock: when they are moving

Hierarchical file systems

- How many directories can we have?

Interprocess communication

- How can we talk to each other? network (), pipe (launch this program on the left side, with a standard out

System calls

- Is not a function call: when you make a function call, the caller of the function decides where you go
- Is a request from OS: is a request from a user program that request
- E.g. check file permission:
- Have to ask for permission (hi OS can you pls take me to…?)
- Don't know about the location of the things I want to know
- We use something that jumps you to a subject, you can use interrupt

- Have the choice to ask for request, but not the choice to go to the location u intend to

Making a system call

- Interrupt the flow that the CPU is currently doing
- The process is quite “confusing”
- Is really slow

System calls that you should be aware of

- When is the fewer the system calls the better?
    - When you want a smaller OS
    - Always
- Why am i not updating IOS 11?
    - It’s running on a very old system
    - Apple has very strict control over everything
- Windows have more system calls than IOS

Systems call related to processes

- s = kill(pid, signal): send the signal to a process
- Pid = fork(): create a child process identical to the parent, except the parent, in order not to confuse the “clone” with the origin. If parent is still cloned, OS can tell which one is original
- s = execve(name, argv, environp): replace a process’ core image. When you want to become something else

A simple shell

fork(): one process calls, two processes are returned
execve(): one process call, nothing return
If you wanna duplicate something, create a new “node”, make this mode point to the same thing
If execve() fails, it will just exit right after it

2018/4/11 Wednesday

Process hierarchies & states

- Parent, child (process group)
- Process either is: created, ready, running, blocked, exit
- Transitions: enter queue, scheduler picks process, process waits for events , event occurs, process exits, process ended by another
- Two layers: lower level of process OS handles interrupts and scheduling
    - Above level is the sequential process (tracked in a process table with an entry)

Process table

- Process management: registers, CPU status, process state, etc
- File management: root directory, current directory, User ID, etc
- Memory management: pointers to text, data, stack & page table (whichever process needs the CPU the most usually gets less of it)

OS: Structures + processes & threads

OS structure:

- Does the OS always delegate tasks to CPU?
    - Not always (be careful for definite terms: ALWAYS, NEVER, ABSOLUTELY, etc.)
    - 3 kinds of structures, monolithic, microkernel
        - Monolithic OS: OS is one big program “onion architecture” → layered (is dynamic)
        - Microkernel (client-server): clients and servers don’t share memory (take pieces out of the kernel: like ability to delegate task, create a “microkernel” with less roles, user mode + kernel mode)
        - Virtual machines: a program with a program (pretending to be another hardware emulator), guest OS can crash without harming actual OS, guest can even use raw hardware (V.M. keeps things separate), runs at higher state than current oS

What’s a process?

- Code, data, state
- Program state → CPU registers, program counter, stack pointer (just take an RAM and know its current state)
- Only one process at a time in the CPU

Process model

- Processes that are independent but run “sequentially” (only one program active at a time, but only for a short time very quickly (for one CPU))