CS 537 Section

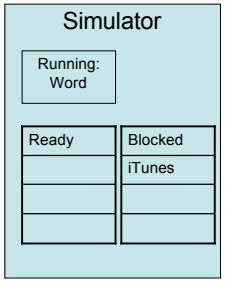
Michael Swift

Project 3: Simulating VM and Scheduling

- Your task (should you choose to accept it):
  - Read in a trace of programs and memory references
  - Simulate the behavior of an OS
  - Print out performance results

Trace based simulation (T=13)

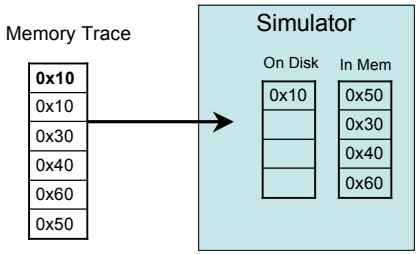
Name	Start	CPU
FireFox	1	1.3
gcc	3	0.5
iTunes	10	200
Word	11	20



Scheduling simulation

- Keep track of each process state:
  - running
  - ready
  - blocked
- Make a scheduling decision when events occur:
  - new process arrives
  - timeslice expires
  - I/O completes
- You will do simple round/robin scheduling
- There are 4 samples: sched-trace.tiny, sched-trace.small, sched-trace.medium, sched-trace.large

Trace Base Simulation



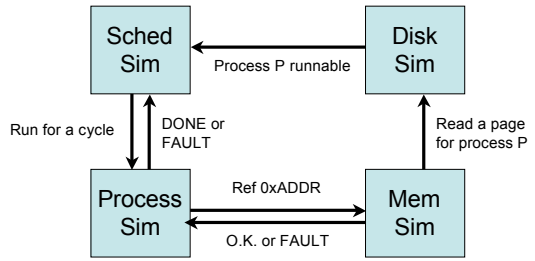
Memory Simulation

- On each memory reference
  - Check if page is in memory.
    - If so, continue
    - If not, page fault
      - suspend process
      - issue disk request
      - run next ready process
- You need to maintain
  - Page table for each process, saying where in memory its pages are (on disk or in memory)
- You do not need to maintain
  - A precise physical - virtual mapping
  - Just knowing that a page is in memory or on disk is enough
- Note: the traces are big (5MB each) -- don't copy them if you don't need to.

Disk simulation

- To handle page faults, you must be able to move data between memory and disk
- Saving a page is free for this assignment
- Reading a page takes 1000 cycles
  - Process needing a page must wait this long for the page to come off disk
  - Only one page at a time can be read (e.g. 2 pages == 2000 cycles)

Overall simulator structure



PseudoCode for Main Loop

```
while (1) {
  load_processes_that_start_this_cycle(time);
  check_for_timer_interrupt(time);
  check_for_disk_completion(time);
  running_process = check_for_context_switch_completion(time);
  if (running a process) {
    didFault = simulate_one_instruction();
    if (didFault) {
      enqueue(blocked,running process)
      start_context_switch();
    }
  }
```

•Notes:

•Sometimes nothing happens for a while, if nothing is runnable you can skip ahead

How to do this project

- Come up with a design
  - Break the problem into modules
  - Figure out what each module does
  - Figure out the interface
    - function calls in to the module
    - function calls out of the module
- Write and test modules separately
  - scheduling simulator
  - disk simulator
  - memory simulator
- Integrate them all

Schedule

- Next week: come up with a plan
- Two weeks: have standalone simulators working
- After that: integrate it all and run experiments