Week 2 (Jan 11 - 15)

Process

- Process - Program in execution
  - Has address space the program can use
  - Has State (registers, including program counter and stack pointer)
- OS tracks processes in process table
- Processes can create other processes
  - Process tree tracks these relationships

Process = Address Space + Thread of Control

Inside a UNIX Process

- Three Segments:
  - Text: contains the program code
  - Data: contains the program data
    * static variables
    * malloc, new
  - stack:
    * Automatic variables
    * Procedure call info
- Address space Growth
  - Text: No growth
  - Data and Stack grow towards each other

Deadlocks are caused by Circular dependencies

Interprocess Commmunication

- Processes work to exchange info with eachother
- Many ways
  - Network
  - Pipes

System Calls

- OS runs in priveledged (supervisor) mode
  - User Processes do not
- Programs want OS to perform service
  - Access a file
  - Create process
- Accomplished with System Calls

How System Calls Work

- User program enters priveledged mode through well defined entrypoint
- Program passes relevant info into OS
- OS performs service
  - Able to do so
  - Service is permitted for user process
- Example system calls
  - fork
  - waitpid
  - execvp

Operating System Types

- Monolithic
  - All just a program
    - Any piece has access to other pieces
  - Sometimes Modular
    - Extra pieces dynamically added
    - Extra pieces part of whole
- Layered
  - CPU Supports layers of priveledge
    - Intel supports 4
    - User at outermost layer
  - Use priveledge layers to enforce separation and provide security
    - Outer layers can only move inward at certain gates
  - Not widely used, but a good idea
- Micro-Kernels
  - Bare minimum in kernel
  - Everything else in User-space
  - Inefficient

Scheduling

- How are processes scheduled?
  - When they enter system
    - Batch Systems
      - New job runs scheduler
      - Scheduling done when job voluntarily gives up time
  - Fixed interval scheduling
    - Necessary for interactive systems
- Goals of Scheduling
  - Fairness: Fair chance at CPU
  - Enforcement: Ensure policy is carried out
  - Balance: Keep all parts of system busy

  - Batch
    * Throughput
      - Work/time
    * Turnaround time
      - Like response time, but for batch jobs
    * CPU utilization
  - Interactive
    * Response time
      - Can measure avg or other metrics
      - Good to measure # times faster than certain time (measure # of outliers)
    * Proportionality: meet user expectations
  - Real time
    * meet deadlines
    * predictability

Types of Schedulers

- Batch Schedulers
  - First come, first served
    * Jobs in order they arrive
    * Simple
    * Problem: long jobs delay those behind
  - Shortest First
    * Short jobs finish first, so long jobs do not cause delays
    * Shortest remaining time first
      - reevaluate when new job is finished
    * Problem
      - How to know how long a job is
  - Three level scheduling
    * Jobs in input queue until moved to mem
      - Pick complementary Jobs: small/large; cpu heavy/io heavy
    * CPU scheduler picks next job
    * mem sched picks jobs to move from mem if out of memory
- Interactive Schedulers
  - Round Robin
    * Each process gets fixed time slot (quantum)
    * Rotate through ready processes
    * Each makes progress
    * What is a good amount of time?
      - Too short: Wasteful
      - Too long: not good for interactive
      - Typically: 10-100ms
  - Priority Scheduling
    * Assign priority to each process
      - "Ready" process with highest priority runs next
    * Dynamic priorities

            - Reduce when using lots of CPU time
            - Increase on wait for I/O
        - Often grouped into multiple queues based on priority
            - Round robin within queues
    - Shortest process next
        - Run process that will finish soonest
        - Guess time based on previous runs
        - Not used because round robin works so well
    - Lottery Scheduling
        - Each process gets tickets for CPU time
            - More tickets -> More time
        - Each quantum, a random ticket is picked
            - Can be implemented efficiently by using integer ranges for tickets
        - Over time each process gets gets CPU m/n of the time
            - m = # of tickets process has
            - n = total # of tickets
        - Tickets can be transferred
            - Processes exchange tickets
            - Clients to server
            - Parents to children

Scheduling in FreeBSD

Called ULE scheduler because file is sched_ule.c

- Two Parts
    - Low level
        - Selects next thread from run queue
        - Very fast
    - High level
        - Sets thread priority
        - Selects processor to use
        - Slower
        - Runs infrequently
        - Adjusts quantum and priority
- Run queues only have ready threads
    - Blocked threads placed in
        - Turnstile: short-term
        - sleep queue: med/long term
    - Runnable thread that consumes quantum placed at end of queue

Policy VS Mechanism

- Separate what must be done (policy) from how it is done (mechanism)