CS 471 Operating Systems

Yue Cheng

George Mason University
Fall 2017
Midterm Exam

○ Wednesday, October 11, 1:30pm – 2:45pm
  – 75 min, closed book, closed note

○ Cover topics from lec-1a to lec-4b
  – OS basics
  – Process abstraction
  – Threads
  – CPU scheduling
  – Synchronization
OS Overview

- Operating Systems: What is an OS?
- Different types of systems/OS
  - Desktop/parallel/distributed OS etc.
- OS as a resource manager
  - Virtualizing resources e.g., CPU, memory, etc.
  - How does OS provide services? -- Interrupts/syscalls
- OS structure and basic components
  - Process, memory, storage
OS Interfaces:
APIs and System Calls

- Application programming interface (API)
  - Runtime systems provide APIs which invoke necessary system calls with OS

- System calls provide the interface between a running program and the OS
  - Generally available in routines written in C/C++
  - Certain low-level tasks may have to be written using assembly language
Interrupt Mechanism

- Interrupt service routines (ISR) – Interrupt handler
  - A piece of OS code that determine what action should be taken for each type of interrupt
  - E.g., reading data and returning err code of device
- Once interrupt has been served, OS returns control over to the interrupted program
  - Context switch occurs

- Modern OS are interrupt-driven
Basic Interrupt Processing

1. The interrupt is issued
2. Processor finishes execution of current instruction
3. Processor signals acknowledgement of interrupt
4. Processor pushes PSW and PC onto control stack
5. Processor loads new PC value through the interrupt vector
6. ISR saves remainder of the process state information
7. ISR executes
8. ISR restores process state information
9. Old PSW and PC values are restored from the control stack

✓ What if another interrupt occurs during interrupt processing?
   Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
Process Management

- OS goal: Allow (safe) multitasking

- Process management
  - Process concept, process control block (PCB), address space
  - Process creation and termination
  - Process state transition
  - Context switch

- Inter-process communication
  - Message passing
  - Memory sharing
PCB in Linux (task_struct)

- **Process Management**
  - Registers
  - Program Counter
  - Stack Pointers
  - Process State
  - Priority
  - Scheduling Parameters (slice)
  - Process ID
  - Parent process
  - Process group
  - Time when process started
  - CPU time used

- **Memory Management**
  - Pointer to text (code) segment
  - Pointer to data segment
  - Pointer to stack segment

- **File Management**
  - Root directory
  - Working directory
  - User Id
  - Group Id
  - List of open files
Process Creation

- **Address space**
  - Child duplicates parent’s address space (data, code) using fork()
  - OR, child has a program loaded into it (exec())

- **Execution**
  - Parent and child execute concurrently

- **Process tree**

- **Process vs. program**
  - Program is a passive entity stored as files on disk
  - Process is an active entity, a program in execution
Threads

- Processes vs. threads
  - Multiple threads created in a process share the same:
    - Address space -- code, data, heap segments
    - Open files
    - A bunch of other resources
  - Benefits of multithreading against multi-processes

- Thread implementation
  - User-level threads
  - Kernel-level threads
  - Hybrid thread implementation

- Pthread POXIS APIs
Process Execution Patterns

- **CPU-bound process** spends more time doing computations; few very long CPU bursts

- **I/O-bound process**
  spends more time doing I/O than computations, many short CPU bursts
CPU Scheduling Algorithms

- Preemptive vs. non-preemptive scheduling
- FIFO: non-preemptive, processes run based on arrival time
  - Long waiting time
- SJF: preemptive vs. non-preemptive
  - Optimal in terms of waiting time
- RR: preemptive
  - Processes take turns with fixed time quantum
- Priority scheduling
Synchronization

- Basic concepts
  - Race condition, mutual exclusion, critical section, atomicity
  - Deadlock, starvation

- Locks
  - Basic operations: lock(), unlock()
  - Goals of a lock: mutex, fairness, performance

- Busy waiting and spin locks

- Sync hardware: e.g., TAS
Semaphores

- Motivation: avoid busy waiting by blocking a process execution until some condition is met

- Two operations (executed atomically)
  - wait(s): caller decrements s by doing s.val--
    - Wait/block if s.val < 0
  - post(s): caller increments s by doing s.val++
    - If one or more processes/threads waiting, wake one

- Semaphore initialization
Condition Variables (CV)

- An explicit queue where threads can put themselves when some condition is not as desired
  - Caller threads wait on the condition
- Other thread can wake up one of the waiting threads to allow them to continue
- Basic operations
  - wait(cv, mutex)
  - signal(cv)
- Pitfalls of using CV
Monitors

- To hide mutual exclusion within defined procedures of the monitor construct
- Ensures that at most one process is active within the monitor at a given time
- CVs declared within a monitor to allow processes to wait on certain conditions
  - CV wait, and CV signal
  - Hence, a monitor’s wait and signal are not the same as the wait and signal (post) operations of a semaphore
Classical Sync Problems

- Producer-consumer problem
  - Semaphore-based version
  - CV-based version
  - Monitor-based version

- Readers-writers problem
  - Multiple readers single writer

- Dining-philosophers problem
  - Semaphore-based version
    - Prone to deadlock and starvation
  - Monitor-based version
Types of Questions in Midterm

I. True/false questions

II. Multi-choice questions

III. Question answering
   – E.g., what is a process?

IV. Problem analysis
   – E.g., last few big questions on HW1 and HW2