Comp Review Spring 1996: Operating Systems


Chapter 1: Introduction

A. 4 components of a computer

1) hardware - CPU, memory, I/O
2) operating system - controls and coordinates hardware
3) application programs - compiler, debuggers, etc.
4) users

B. Various types of systems

1. Batch - jobs were batched to reduce set up time
2. Interactive - I/O bound; immediate response is required
3. Time Sharing - allows many users to utilize 1 CPU, by letting the CPU
   swap among processes.
4. Parallel - more than one CPU; can share memory, bus, peripherals.
5. Distributed - collection of processors that don't share memory or
   clock;
   uses local memory.

Chapter 2: Computer-System Structures

A. Storage

1. The main difference among storage systems lie in: speed, cost, size
   and volatility
2. Storage Hierarchy
   a. higher levels are expensive but fast
   b. As you go lower, cost is cheaper but access time increases
      Registers
      Cache
      Main Memory
      Electronic Disk
      Magnetic Disk
      Optical Disk
      Magnetic tapes

B. Hardware Protection - to ensure correct operation of the system:
1. I/O protection
   I/O and halt instructions are privileged and can only be executed in
   monitor mode
2. Memory protection
   The memory in which the OS resides must be protected from modification
   by the user.
3. CPU protection
   A timer prevents infinite loops
4. Dual Mode (monitor or privileged mode vs. user mode)
   Dual mode to keep user programs from interfering with the operation of the system.

Chapter 3: Operating-System Structures

A. System Components that the OS is responsible for:

1. Process Management - creation and deletion of user and system processes,
   deadlock handling, etc.
2. Main-Memory Management - keep track of which parts of memory are being used,
   allocate/deallocate memory space as required, etc.
3. Secondary-Storage Management - Free-space management, disk scheduling,
   storage allocation
4. I/O System Management - deals with hardware specific drivers for devices,
   keeps it all hidden from the rest of the system.
5. File Management - creating/deleting files and directories, backup, etc.
6. Protection System - controlling access to programs, processes, or users
7. Networking - generalizes network access
8. Command-Interpreter System - interface between the user and the OS

B. Policy vs. Mechanism

1. Policies decide what will be done (ex. how long to set a timer)
2. Mechanisms determine how to do something (ex. actual implementation of the timer)
Chapter 4: Processes

A. Process - a program in execution

B. Process States:

1. New: is being created
2. Running: instructions are being executed
3. Waiting: waiting from some event to occur (such as I/O completion)
4. Ready: waiting to be assigned to a processor
5. Terminated: has finished execution

C. Process Control Block - what a process is represented in the OS as

1. Process state
2. Program counter
3. CPU registers - all data here needs to be saved when interrupts occur
4. CPU scheduling information - includes priority, pointers to scheduling queues
5. Memory-management information - may include value of page or segment tables
6. Accounting information - amount of CPU used, etc.
7. I/O status information - list of devices allocated to process, list of open files, etc.

D. Process Scheduling

The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.

1. Schedulers - the primary distinction is the frequency of execution
   (ST is more frequent)
   a. Long term scheduler - selects processes from the pool of processes that have been submitted and loads them into memory for execution.
   i. The goal of the LTS should be to look for a good mix of CPU-bound
and I/O bound processes (if all CPU, devices go unused, if all I/O then ready queue is always empty and STS is wasted).

b. Short term scheduler - selects processes from the ones that are ready to execute and allocates the CPU to one of them.

2. Context Switch - switching the CPU from one process to another; requires saving the old process state, and loading the state of the new process.

E. Threads (pp. 111-116)

Also called a lightweight process, is a basic unit of CPU utilization and consists of:
1) PC
2) register set
3) stack space
4) code section, data section and OS resources - all shared with peer threads

Chapter 5: CPU Scheduling

A. Scheduling Criteria

1. (maximize) CPU utilization - how often the CPU is busy
2. (maximize) Throughput - how many jobs are processed
3. (minimize) Wait Time - total time spent in the ready queue
4. (minimize) Response Time - amount of time to start responding
5. (minimize) Turnaround Time - total time it takes from the time of submission to completion.

B. Scheduling Algorithms

1. **FCFS (First Come First Served)**
   a. non-preemptive
   b. very simple
   c. can be implemented with a FIFO queue.
   d. Not a good choice when there are variable burst times (CPU bound and I/O bound)
   e. drawback: causes short processes to wait for longer ones
   f. Example:
2. **SJF (Shortest Job First)**
   a. based on burst times (not total job time)
   b. difficult to get this information - must approximate it for short term scheduling
   c. Used frequently for long term scheduling
   d. preemptive or non-preemptive
   e. Special case of priority scheduling
   f. example:

3. **Priority Scheduling**
   a. preemptive or non-preemptive
   b. interal priorities can be used (ie, time limits, memory requirements, etc)
   c. external priorities can be used (ie, user defined priority, funds, dept., etc)
   d. Major problem is starvation or indefinite blocking (a solution to this is aging)
   e. example

4. **Round-Robin (RR) Scheduling**
   a. designed especially for time-sharing systems
   b. depends heavily on the size of the time quantum
   c. preemptive (since processes switch in and out)
   d. if t.q. is too large, this just becomes FCFS
   e. Since context-switching costs time, the rule of thumb is 80% of CPU bursts should be shorter than the time quantum.
   f. example

5. **Multi-level Queue Scheduling**
   a. several subqueues set up, which can each run their own scheduling algorithm
   b. ex. foreground process queue uses RR, background process queue uses FCFS

6. **Multilevel Feedback Queue Scheduling**
   a. like the above but allows process to move between subqueues
   b. most general but most complex scheme

**C. Priority Inversion Problem (p. 151)**
When a higher priority process needs to read or modify data/resources that is currently being accessed by a lower priority process

Chapter 6: Process Synchronization

required when there is concurrent access to shared data

A. The Critical-Section Problem

1. Critical section: segment of code that contains common variables, updates to tables, writes to files, etc.
2. Race condition: outcome of the execution depends on particular order in which access takes place.
3. A solution to the critical section problems must satisfy the following 3 requirements:
   a. Mutual Exclusion - only one process can execute in its critical section at a time.
   b. Progress - if no process is executing in its critical section, and one is waiting, it must be allowed to immediately enter. If there are multiple processes waiting, only they must decide and selection can not be postponed indefinitely.
   c. Bounded Waiting - there must be a bound on the number of times other processes can enter their cs after a process can requested to enter its cs and before that request is granted. (so a process doesn't get left out).

B. Semaphores (p. 175)

1. a synchronization tool; basically an integer variable that can only be accessed through atomic wait (P) and signal (W) operations.
2. use of these avoid busy waiting

C. Classic synchronization problems (pp. 181 - 182)
1. Bounded-Buffer (like producer-consumer)
2. Readers and Writers
3. Dining Philosophers

D. Monitors (p. 191)

1. high level synchronization construct to allow sharing of abstract data types.
2. Calls to critical section have to go through the monitor (programmer can't just bypass it by avoiding the lock.

Chapter 7: Deadlocks

A. Definition

A deadlock occurs when a process is waiting for resources that other waiting processes have and they are waiting for resources that it has.

B. 3 methods for dealing with deadlocks

1. Prevention - allocate resources at the start
2. Avoidance - only do "safe" allocations at a time
   a. Banker's Algorithm
3. Detection/Recovery - search for cycles in wait-for graphs, then rollback, redo, preempt, etc.

C. A deadlock situation may occur iff 4 necessary conditions hold simultaneously:

1. Mutual Exclusion
   a. avoided if resources are shareable
2. Hold and Wait
To prevent deadlocks, we ensure that at least one of the necessary conditions never holds.

Chapter 8: Memory Management

A. Address Binding - binding of instructions and data to memory addresses

1. Compile time - if process location is known then absolute code can be generated.
2. Load time - compiler generates relocatable code which is bound at load time
3. Execution time - if a process can be moved from one memory segment to another then binding must be delayed until run time.

B. Dynamic Loading - routine is not loaded until it is called

1. Advantage: unused routines are never loaded; useful when large amounts of code are needed to handle infrequent cases.

C. Logical Vs. Physical Address Space
1. An address generated by the CPU is a logical address.
2. An address seen by the memory unit is the physical address.
3. CPU -> generate logical address -> relocation register
   adds base -> physical address

D. Memory Management Algorithms

1. Contiguous Allocation
   a. space is allocated directly on physical memory

2. Paging
   a. maps logical addresses to physical memory.
   b. Memory space can then be non-contiguous, which helps
      eliminate fragmentation issues.
   c. Each address generated by the CPU has page number and
      page offset.
      i. Page number maps to page table to get base address to physical
      memory
      ii. Offset is added to this base.
   d. To figure out how much space a page table takes:
      Given the address size (n), and the page size (p), you can find
      out how many entries can be specified in physical memory:
      \[ \text{# entries} = \frac{2^n}{2^p} \]
      \[ \text{# of entries} \times \text{size of each entry (p)} = \text{page table size} \]

3. Segmentation
   a. a memory management scheme that supports a user view of memory
      with modularizations (collection of segments).
   b. Advantages:
      i. protection for the segments; ie, data and instructions can
         be put in different segments
      ii. sharing of code or data - entries in segment table
         for 2 different processes can both point to the same physical memory location.

E. Dynamic storage allocation problem:

how to satisfy a request of size n from a list of free holes.
(holes appear because processes that were allocated that memory free it up).
1. First Fit - allocate the first hold that is big enough. Search can stop when found
2. Best-fit - allocate the smallest hole that is big enough. Have to search the whole list. Adv.: produces the smallest leftover hole.
3. Worst-fit - allocate the largest hold. Have to search the whole list (unless sorted by size). Adv.: produces large leftover holes, which may be easier to fill.

F. 50% Rule

Given N allocated blocks, another .5N blocks may be unusable (1/3 of memory) due to fragmentation.

G. Considerations when comparing strategies:

1. Hardware support - CA only needs 1 or 2 registers; others need mapping tables
2. Performance - P or S can be fast if table is implemented in fast registers
3. Fragmentation
4. Relocation
5. Swapping - requires P or S.
6. Sharing - requires P or S
7. Protection - different sections of a user program can be set
   (S works for this)

H. To increase multiprogramming level

1. pack more processes into memory by reducing fragmentation, waste.
2. allow sharing of code and data

Chapter 9: Virtual Memory
A. Definition

A technique that allows the execution of processes that may not be completely in memory.

B. Demand Paging – paging with swapping

C. Belady's Anomaly

reflects the fact that, for some page-replacement algorithms, the page fault rate may increase as the number of allocated frames increases.

D. Advantages of VM

1. allows really large processes to be run
2. allows degree of multiprogramming to be increased
3. frees application programmers from worrying about memory

E. Page Replacement Algorithms

1. FIFO
2. Optimal Algorithm – replace the page that will not be used for the longest period of time. (difficult to implement since it requires future knowledge!)
3. LRU Algorithm (Least Recently Used) – replace page that hasn't been used for the longest period of time.
4. LFU (Least Frequently Used)
5. Random

F. Thrashing

high paging activity, indicating that a process is spending more time paging than executing.
OS Questions from Previous Comps:

Winter 1993, Q2: Give 3 examples of sharing in an OS, and show what protection mechanisms might be necessary to control sharing.

Winter 1993, Q3: You are asked to state whether the given description is an example of deadlock; and asked to explain if the OS can detect it.

Spring 1993, Q4: State 4 factors that might cause thrashing in a multiprogramming OS w/ virtual memory, and explain what the memory manager could do about each one.

prepared by: Shikha Ghosh Gottfried, April 19, 1996