ICS 143 - Principles of Operating Systems

Operating Systems - Review of content from midterm to final
Prof. Nalini Venkatasubramanian
ardalan@uci.edu
nalini@ics.uci.edu
Deadlocks

● System Model
  ● Resource allocation graph, claim graph (for avoidance)

● Deadlock Characterization
  ● Conditions for deadlock - mutual exclusion, hold and wait, no preemption, circular wait.

● Methods for handling deadlocks
  ● Deadlock Prevention
  ● Deadlock Avoidance
  ● Deadlock Detection
  ● Recovery from Deadlock
  ● Combined Approach to Deadlock Handling
Deadlock Prevention

- If any one of the conditions for deadlock (with reusable resources) is denied, deadlock is impossible.

- Restrain ways in which requests can be made
  - Mutual Exclusion - cannot deny (important)
  - Hold and Wait - guarantee that when a process requests a resource, it does not hold other resources.
  - No Preemption
    - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, the process releases the resources currently being held.

- Circular Wait
  - Impose a total ordering of all resource types.
Deadlock Avoidance

- Requires that the system has some additional *a priori* information available.
  - Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.

- Computation of Safe State
  - When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state. Sequence <P1, P2, …Pn> is safe, if for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by Pj with j<i.
  - Safe state - no deadlocks, unsafe state - possibility of deadlocks
  - Avoidance - system will never reach unsafe state.
Algorithms for Deadlock Avoidance

- Resource allocation graph algorithm
  - only one instance of each resource type
- Banker’s algorithm
  - Used for multiple instances of each resource type.
  - Data structures required
    - Available, Max, Allocation, Need
  - Safety algorithm
  - resource request algorithm for a process.
Memory Management

- Main Memory is an array of addressable words or bytes that is quickly accessible.
- Main Memory is volatile.
- OS is responsible for:
  - Allocate and deallocate memory to processes.
  - Managing multiple processes within memory - keep track of which parts of memory are used by which processes. Manage the sharing of memory between processes.
  - Determining which processes to load when memory becomes available.
Binding of instructions and data to memory

- Address binding of instructions and data to memory addresses can happen at three different stages.
  - Compile time, Load time, Execution time
- Other techniques for better memory utilization
  - Dynamic Loading - Routine is not loaded until it is called.
  - Dynamic Linking - Linking postponed until execution time
  - Overlays - Keep in memory only those instructions and data that are needed at any given time
  - Swapping - A process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution
- MMU - Memory Management Unit
  - Hardware device that maps virtual to physical address.
Contiguous Allocation

- Divides Main memory usually into two partitions
  - Resident Operating System, usually held in low memory with interrupt vector and User processes held in high memory.

- Single partition allocation
  - Relocation register scheme used to protect user processes from each other, and from changing OS code and data

- Multiple partition allocation
  - holes of various sizes are scattered throughout memory. When a process arrives, it is allocated memory from a hole large enough to accommodate it.
  - Variation: Fixed partition allocation
Dynamic Storage Allocation Problem

- How to satisfy a request of size $n$ from a list of free holes.
  - First-fit
  - Best-fit
  - Worst-fit

- Fragmentation
  - External fragmentation
    - total memory space exists to satisfy a request, but it is not contiguous.
  - Internal fragmentation
    - allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

- Reduce external fragmentation by compaction
Paging

- Logical address space of a process can be non-contiguous;
  - process is allocated physical memory wherever the latter is available.
- Divide physical memory into fixed size blocks called **frames**
  - size is power of 2, 512 bytes - 8K
- Divide logical memory into same size blocks called **pages**.
  - Keep track of all free frames.
  - To run a program of size n pages, find n free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Note:: Internal Fragmentation possible!!
Page Table Implementation

- Page table is kept in main memory
  - Page-table base register (PTBR) points to the page table.
  - Page-table length register (PTLR) indicates the size of page table.
- Every data/instruction access requires 2 memory accesses.
  - One for page table, one for data/instruction
  - Two-memory access problem solved by use of special fast-lookup hardware cache (i.e. cache page table in registers)
    - associative registers or translation look-aside buffers (TLBs)
Paging Methods

● Multilevel Paging
  • Each level is a separate table in memory
  • converting a logical address to a physical one may take 4 or more memory accesses.
  • Caching can help performance remain reasonable.

● Inverted Page Tables
  • One entry for each real page of memory. Entry consists of virtual address of page in real memory with information about process that owns page.

● Shared Pages
  • Code and data can be shared among processes. Reentrant (non self-modifying) code can be shared. Map them into pages with common page frame mappings
Segmentation

● Memory Management Scheme that supports user view of memory.
● A program is a collection of segments.
● A segment is a logical unit such as
  • main program, procedure, function
  • local variables, global variables, common block
  • stack, symbol table, arrays
● Protect each entity independently
● Allow each segment to grow independently
● Share each segment independently
Segmented Paged Memory

- Segment-table entry contains not the base address of the segment, but the base address of a page table for this segment.
  - Overcomes external fragmentation problem of segmented memory.
  - Paging also makes allocation simpler; time to search for a suitable segment (using best-fit etc.) reduced.
  - Introduces some internal fragmentation and table space overhead.
- Multics - single level page table
- IBM OS/2 - OS on top of Intel 386
  - uses a two level paging scheme
Virtual Memory

● Virtual Memory
  ● Separation of user logical memory from physical memory.
  ● Only *PART* of the program needs to be in memory for execution.
  ● Logical address space can therefore be much larger than physical address space.
  ● Need to allow pages to be swapped in and out.

● Virtual Memory can be implemented via
  ● Paging
  ● Segmentation
Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less Memory needed
  - Faster response
  - More users

- The first reference to a page will trap to OS with a page fault.

- OS looks at another table to decide
  - Invalid reference - abort
  - Just not in memory.
Page Replacement

- Prevent over-allocation of memory by modifying page fault service routine to include page replacement.
- Use modify(dirty) bit to reduce overhead of page transfers - only modified pages are written to disk.
- Page replacement
  - large virtual memory can be provided on a smaller physical memory.
Page Replacement Strategies

- **The Principle of Optimality**
  - Replace the page that will not be used again the farthest time into the future.

- **Random Page Replacement**
  - Choose a page randomly

- **FIFO - First in First Out**
  - Replace the page that has been in memory the longest.

- **LRU - Least Recently Used**
  - Replace the page that has not been used for the longest time.
  - LRU Approximation Algorithms - reference bit, second-chance etc.

- **LFU/MFU - Least/Most Frequently Used**
  - Replace the page that is used least/most often.
Allocation of Frames

- **Single user case is simple**
  - User is allocated any free frame

- **Problem: Demand paging + multiprogramming**
  - Each process needs minimum number of pages based on instruction set architecture.
  
- **Two major allocation schemes:**
  - Fixed allocation - (1) equal allocation (2) Proportional allocation.
  - Priority allocation - May want to give high priority process more memory than low priority process.
Thrashing

● If a process does not have enough pages, the page-fault rate is very high. This leads to:
  ● low CPU utilization.
  ● OS thinks that it needs to increase the degree of multiprogramming
  ● Another process is added to the system.
  ● System throughput plunges...

● **Thrashing**
  ● A process is busy swapping pages in and out.
  ● In other words, a process is spending more time paging than executing.
Working Set Model

- \( \Delta \equiv \text{working-set window} \)
  - a fixed number of page references, e.g. 10,000 instructions
  - \( \text{WSS}_j \) (working set size of process \( P_j \)) - total number of pages referenced in the most recent \( \Delta \) (varies in time)
    - If \( \Delta \) too small, will not encompass entire locality.
    - If \( \Delta \) too large, will encompass several localities.
    - If \( \Delta = \infty \), will encompass entire program.
  - \( D = \sum \text{WSS}_j \equiv \text{total demand frames} \)
    - If \( D > m \) (number of available frames) \( \Rightarrow \) thrashing
  - Policy: If \( D > m \), then suspend one of the processes.
File System Management

- File is a collection of related information defined by creator - represents programs and data.
- OS is responsible for
  - File creation and deletion
  - Directory creation and deletion
  - Supporting primitives for file/directory manipulation.
  - Mapping files to disks (secondary storage).
  - Backup files on archival media (tapes).
File Concept

- **Contiguous logical address space**
  - OS abstracts from the physical properties of its storage device to define a logical storage unit called file. OS maps files to physical devices.

- **Types**
  - Data, Program, Documents

- **File Attributes**
  - Name, type, location, size, protection etc.

- **File Operations**
  - Create, read, write, reposition, delete etc..
Directory Structure

- **Number of files on a system can be extensive**
  - Hold information about files within partitions called directories.
  - Device Directory: A collection of nodes containing information about all files on a partition. Both the directory structure and files reside on disk. Backups of these two structures are kept on tapes.

- **Operations on a directory**
  - create a file, delete a file, search for a file, list directory etc.
Logical Directory Organization

- **Goals** - Efficiency, Naming, grouping
- **Single Level Directories**
  - Single level for all users, naming and grouping problem
- **Two Level Directories**
  - first level - user directories, second level - user files
- **Tree Structured Directories**
  - arbitrary depth of directories, leaf nodes are files
- **Acyclic Graph Directories**
  - allows sharing, implementation by links or shared files
- **General Graph Directories**
  - allow cycles - must be careful during traversal and deletion.
File Protection - Access lists and groups

- Associate each file/directory with access list
  - Problem - length of access list..
- Solution - condensed version of list
  - Mode of access: read, write, execute
  - Three classes of users
    - owner access - user who created the file
    - groups access - set of users who are sharing the file and need similar access
    - public access - all other users
- In UNIX, 3 fields of length 3 bits are used.
  - Fields are user, group, others (u,g,o),
  - Bits are read, write, execute (r,w,x).
  - E.g. chmod go+rw file, chmod 761 game
File-System Implementation

● File System Structure
  • File System resides on secondary storage (disks). To improve I/O efficiency, I/O transfers between memory and disk are performed in blocks. Read/Write/Modify/Access each block on disk.
  • File System Mounting - File System must be mounted before it can be available to process on the system. The OS is given the name of the device and the mount point.

● Allocation Methods
● Free-Space Management
● Directory Implementation
● Efficiency and Performance, Recovery
Allocation of Disk Space

- Low level access methods depend upon the disk allocation scheme used to store file data
  - Contiguous Allocation
    - Each file occupies a set of contiguous blocks on the disk. Dynamic storage allocation problem. Files cannot grow.
  - Linked List Allocation
    - Each file is a linked list of disk blocks. Blocks may be scattered anywhere on the disk. Not suited for random access.
    - Variation - FILE ALLOCATION TABLE (FAT) mechanisms
  - Indexed Allocation
    - Brings all pointers together into the index block. Need index table. Can link blocks of indexes to form multilevel indexes.