Final Review

Fall 2017

Also see Midterm Review

CS370 Operating Systems

Yashwant K Malaiya
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

At this point, two minimal cycles exist in the system:

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

Processes $P_1$, $P_2$, and $P_3$ are deadlocked.
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 apriori 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 \(<P_1, P_2, \ldots P_n>\) is safe, if for each \(P_i\), the resources that \(P_i\) can still request can be satisfied by currently available resources + resources held by \(P_j\) with \(j<i\).
  - Safe state - no deadlocks, unsafe state - possibility of deadlocks
  - Avoidance - system will never reach unsafe state.
Example: 12 Tape drives available in the system

<table>
<thead>
<tr>
<th></th>
<th>Max need</th>
<th>Current need</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>P1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

- **At time T0** the system is in a safe state
  - P1 can be given 2 tape drives
  - When P1 releases its resources; there are 5 drives
  - P0 uses 5 and subsequently releases them (# 10 now)
  - P2 can then proceed.

**At T0:**
3 drives available

Safe sequence
<P1, P0, P2>
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.

Suppose $P_2$ requests $R_2$. Although $R_2$ is currently free, we cannot allocate it to $P_2$, since this action will create a cycle getting system is in an unsafe state. If $P_1$ requests $R_2$, and $P_2$ requests $R_1$, then a deadlock will occur.
Deadlock Detection

- Allow system to enter deadlock state

Detection Algorithm
  - Single instance of each resource type
    - use wait-for graph
  - Multiple instances of each resource type
    - variation of banker’s algorithm

Recovery Scheme
  - Process Termination
  - Resource Preemption

3 cycles. Deadlock.
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.
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
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)
Effective Access Time

- **Associative Lookup** = $\varepsilon$ time unit
  - Can be < 10% of memory access time
- **Hit ratio** = $\alpha$
  - Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- Consider $\alpha = 80\%$, $\varepsilon = 20$ns for TLB search, 100ns for memory access
- **Effective Access Time (EAT)**
  
  \[
  EAT = (100 + \varepsilon) \alpha + (200 + \varepsilon)(1 - \alpha)
  \]

  Consider $\alpha = 80\%$, $\varepsilon = 20$ns for TLB search, 100ns for memory access
  - $EAT = 0.80 \times 120 + 0.20 \times 220 = 140$ns
- Consider more realistic hit ratio -> $\alpha = 99\%$, $\varepsilon = 20$ns for TLB search, 100ns for memory access
  - $EAT = 0.99 \times 120 + 0.01 \times 220 = 121$ns
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.

- Hashed page table

- 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.
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 fault:**

1. Find free frame
2. Get page into frame via scheduled disk operation
3. Reset tables to indicate page now in memory
   Set validation bit = v
4. Restart the instruction that caused the page fault
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 - Least Frequently Used
  – Replace the page that is used least often.

• NUR - Not Used Recently
  – An approximation to LRU

• Working Set
  – Keep in memory those pages that the process is actively using
Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page

Reference string

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |

Page frames

<table>
<thead>
<tr>
<th>7</th>
<th>7</th>
<th>7</th>
<th>2</th>
<th>2</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- Approximate Implementations:
  - Counter implementation time of use field
  - Stack implementation
  - Reference bit
  - Second chance

Colorado State University
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.
  - Global vs local allocation
Working-Set Model

• $\Delta \equiv \text{working-set window} \equiv$ a fixed number of page references
  Example: 10,000 instructions

page reference table

$\ldots 26157775162341234443434444413234444444 \ldots$

$WS(t_1) = \{1,2,5,6,7\}$
$WS(t_2) = \{3,4\}$

• $WSS_i$ (working set of Process $P_i$) =
  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 $\Delta = \infty \Rightarrow$ will encompass entire program

• $D = \sum WSS_i \equiv \text{total demand frames}$
  – Approximation of locality

• if $D > m \Rightarrow$ Thrashing

• Policy if $D > m$, then suspend or swap out one of the processes
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
File Systems

• Many file systems, sometimes several within an operating system
  – Each with its own format
    • Windows has FAT (1977), FAT32 (1996), NTFS (1993)
    • Linux has more than 40 types, with extended file system (1992) ext2 (1993), ext3 (2001), ext4 (2008);
    • plus distributed file systems
    • floppy, CD, DVD Blu-ray

  – New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE, xFAT
On-disk File-System Structures

1. **Boot control block** contains info needed by system to boot OS from that volume
   – Needed if volume contains OS, usually first block of volume
2. **Volume control block (superblock UFS or master file table NTFS)** contains volume details
   – Total # of blocks, # of free blocks, block size, free block pointers or array
3. Directory structure organizes the files
   – File Names and inode numbers UFS, master file table NTFS
4. Per-file **File Control Block (FCB or “inode”)** contains many details about the file
   – Indexed using inode number; permissions, size, dates UFS
   – master file table using relational DB structures NTFS

Volume: logical disk drive, perhaps a partition
4. Per-file **File Control Block (FCB or “inode”)** contains many details about the file

- Indexed using inode number; permissions, size, dates
- master file table using relational DB structures

<table>
<thead>
<tr>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>file permissions</td>
</tr>
<tr>
<td>file dates (create, access, write)</td>
</tr>
<tr>
<td>file owner, group, ACL</td>
</tr>
<tr>
<td>file size</td>
</tr>
<tr>
<td>file data blocks or pointers to file data blocks</td>
</tr>
</tbody>
</table>
In-Memory File System Structures

- An in-memory **mount table** contains information about each mounted volume.
- An in-memory **directory-structure cache** holds the directory information of recently accessed directories.
- The **system-wide open-file table** contains a copy of the FCB of each open file, as well as other information.
- The **per-process open file table** contains a pointer to the appropriate entry in the system-wide open-file table.
- Plus buffers hold data blocks from secondary storage.
- Open returns a file handle (file descriptor) for subsequent use.
- Data from read eventually copied to specified user process memory address.
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.
Combined Scheme: UNIX UFS

4K bytes per block, 32-bit addresses

Volume block: Table with file names
Points to this

Inode (file control block)

More index blocks than can be addressed with 32-bit file pointer
Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
  - (Using term “block” for simplicity)
- **Approaches:**
  i. Bit vector
  ii. Linked list
  iii. Grouping
  iv. Counting
- **Bit vector** or **bit map** ($n$ blocks)

```
0 1 2 .. n-1
```

$\text{bit}[i] = \begin{cases} 
1 & \Rightarrow \text{block}[i] \text{ free} \\
0 & \Rightarrow \text{block}[i] \text{ occupied} 
\end{cases}$

Block number calculation

$$(\text{number of bits per word}) \times (\text{number of 0-value words}) + \text{offset of first 1 bit}$$

CPUs have instructions to return offset within word of first “1” bit
Hadoop: Core components

- Hadoop (originally): MapReduce + HDFS
- MapReduce: A programming framework for processing parallelizable problems across huge datasets using a large number of commodity machines.
- HDFS: A distributed file system designed to efficiently allocate data across multiple commodity machines, and provide self-healing functions when some of them go down

- Commodity machines: lower performance per machine, lower cost, perhaps lower reliability compared with special high performance machines.
- RAID (redundant array of inexpensive disks): using multiple disks in a single enclosure for achieving higher reliability and/or higher performance.
HDFS Architecture

- HDFS Block size: 64-128 MB  ext4: 4KB
- HDFS file size: “Big”
- Single HDFS FS cluster can span many nodes possibly geographically distributed. datacenters-racks-blades
- Node: system with CPU and memory

Metadata (corresponding to superblocks, Inodes)
- **Name Node**: metadata, where blocks are physically located

Data (files blocks)
- **Data Nodes**: hold blocks of files (files are distributed)
HDFS Architecture

http://a4academics.com/images/hadoop/Hadoop-Architecture-Read-Write.jpg
HDFS Fault-tolerance

• Disks use error detecting codes to detect corruption.
• Individual node/rack may fail.
• **Data Nodes (on slave nodes):**
  – data is replicated. Default is 3 times. Keep a copy far away.
  – Send periodic heartbeat (I’m OK) to Name Nodes. Perhaps once every 10 minutes.
  – Name node creates another copy if no heartbeat.
HDFS Fault-tolerance

Name Node (on master node) Protection:

• Transaction log for file deletes/adds, etc (only metadata recorded). Creation of more replica blocks when necessary after a DataNode failure

• Standby name node: namespace backup
  – In the event of a failover, the Standby will ensure that it has read all of the edits from the Journal Nodes and then promotes itself to the Active state
  – Implementation/delay version dependent

Name Node metadata is in RAM as well as checkpointed on disk.
On disk the state is stored in two files:
• fsimage: Snapshot of file system metadata
• editlog: Changes since last snapshot
• **Average access time** = average seek time + average latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms

• **Average I/O time** = average access time + (amount to transfer / transfer rate) + controller overhead

• For example to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead =
  - 5ms + 4.17ms + 0.1ms + transfer time
  - Transfer time = 4KB / 1Gb/s = 4x8K/G = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms
Disk Scheduling

- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (cylinders 0-199)
  
  98, 183, 37, 122, 14, 124, 65, 67

  Head pointer 53 (head is at cylinder 53)
- SCAN, C-SCAN, C-LOOK,
SCAN (Elevator) Algorithm

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest.

```
queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
```

![Diagram showing the SCAN algorithm with a queue and head starting at 53]
RAID Techniques

- **Striping** uses multiple disks in parallel by splitting data: higher performance, no redundancy (ex. RAID 0)
- **Mirroring** keeps duplicate of each disk: higher reliability (ex. RAID 1)
- **Block parity**: One Disk hold parity block for other disks. A failed disk can be rebuilt using parity. Wear leveling if interleaved (RAID 5, double parity RAID 6).
- Ideas that did not work: Bit or byte level level striping (RAID 2, 3) Bit level Coding theory (RAID 2), dedicated parity disk (RAID 4).
- Nested Combinations:
  - RAID 01: Mirror RAID 0
  - RAID 10: Multiple RAID 1, striping
  - RAID 50: Multiple RAID 5, striping
  - others

Parity: allows rebuilding of a disk

Not common: RAID 2, 3,4
Most common: RAID 5