CS370 Operating Systems

Colorado State University
Yashwant K Malaiya
Spring 2018 L21
Virtual Memory

Slides based on
• Text by Silberschatz, Galvin, Gagne
• Various sources
• Can more than one page loaded into memory when a process starts?  
• Does Belady’s Anomaly affect all page replacement algorithms?  
• When Belady’s Anomaly is present in an algorithm, will adding an additional frame always cause more page faults?  
• Paging algorithms seem to assume both temporal and spatial locality... how true is this?
“Optimal” Algorithm

- Replace page that will not be used for longest period of time

- 4th access: replace 7 because we will not use if got the longest time...
- 9 page replacements is optimal for the example

- But how do we know this?
  - Can’t read the future in reality.

- Used for measuring how well an algorithm performs
Least Recently Used (LRU) Algorithm

• Use past knowledge rather than future
• Replace page that has not been used in the most amount of time (4th access – page 7 is least recently used ...)
• 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 |

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

page frames

• 12 faults – better than FIFO (15) but worse than OPT (9)
• Generally good algorithm and frequently used
• But how to implement it by tracking the page usage?
FAQ: Predictions in Computing

- LRU can be considered to be an attempt to predict the future.
- Attempt is useful if the prediction is successful in some cases.
- Other example: speculative execution of instructions.
  - If speculation was incorrect, the resulting information is discarded, but some may stay in the cache memory. This can include information accessed in violation of access settings.
  - Resulting in Meltdown/Specter vulnerabilities.
LRU Algorithm: Implementations

Possible implementations

• Counter implementation
  – Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  – When a page needs to be changed, look at the counters to find smallest value
    • Search through table needed

• Stack implementation
  – Keep a stack of page numbers in a double link form:
  – Page referenced:
    • move it to the top
    • requires 6 pointers to be changed
  – Each update expensive
  – No search for replacement needed (bottom is least recently used)

LRU and OPT are cases of stack algorithms that don’t have Belady’s Anomaly
Enhanced Second-Chance Algorithm

- Improve algorithm by using reference bit and modify bit (if available) in concert
  clean page: better replacement candidate
- Take ordered pair (reference, modify)
  1. (0, 0) neither recently used not modified – best page to replace
  2. (0, 1) not recently used but modified – not quite as good, must write out before replacement
  3. (1, 0) recently used but clean – probably will be used again soon
  4. (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement
- When page replacement called for, use the clock scheme but use the four classes replace page in lowest non-empty class
  - Might need to search circular queue several times
Counting Algorithms

- Keep a counter of the number of references that have been made to each page
  - Not common

- **Least Frequently Used (LFU) Algorithm**: replaces page with smallest count

- **Most Frequently Used (MFU) Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used
Clever Techniques for enhancing Perf

• Keep a buffer (pool) of free frames, always
  – Then frame available when needed, not found at fault time
  – Read page into free frame and select victim to evict and add to free pool
  – When convenient, evict victim

• Keep list of modified pages
  – When backing store is otherwise idle, write pages there and set to non-dirty (being proactive!)

• Keep free frame previous contents intact and note what is in them
  – If referenced again before reused, no need to load contents again from disk
  – Generally useful to reduce penalty if wrong victim frame selected
• Some applications (like databases) often understand their memory/disk usage better than the OS
  – Provide their own buffering schemes
  – If both the OS and the application were to buffer
    • Twice the I/O is being utilized for a given I/O
  – OS may provide “raw access” disk to special programs without file system services.
Allocation of Frames

How to allocate frames to processes?

- Each process needs *minimum* number of frames
  Depending on specific needs of the process
- *Maximum* of course is total frames in the system

- Two major allocation schemes
  - fixed allocation
  - priority allocation

- Many variations
Fixed Allocation

• **Equal allocation** – For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20 frames
  – Keep some as free frame buffer pool

• **Proportional allocation** – Allocate according to the size of process (need based)
  – Dynamic as degree of multiprogramming, process sizes change

\[
s_i = \text{size of process } p_i \\
S = \sum s_i \\
m = \text{total number of frames} \\
a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m
\]

\[
m = 64 \\
s_1 = 10 \\
s_2 = 127 \\
a_1 = \frac{10}{137} \times 62 \approx 4 \\
a_2 = \frac{127}{137} \times 62 \approx 57
\]
Priority Allocation

• Use a proportional allocation scheme using priorities rather than size

• If process $P_i$ generates a page fault,
  – select for replacement one of its frames or
  – select for replacement a frame from a process with lower priority number
Global vs. Local Allocation

• **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
  – But then process execution time can vary greatly
  – But greater throughput, so more common

• **Local replacement** – each process selects from only its own set of allocated frames
  – More consistent per-process performance
  – But possibly underutilized memory
Problem: Thrashing

• If a process does not have “enough” pages, the page-fault rate is very high
  – Page fault to get page
  – Replace existing frame
  – But quickly need replaced frame back
  – This leads to:
    • Low CPU utilization, leading to
    • Operating system thinking that it needs to increase the degree of multiprogramming leading to
    • Another process added to the system

• Thrashing $\equiv$ a process is busy swapping pages in and out
Thrashing (Cont.)

![Graph showing CPU utilization vs. degree of multiprogramming with a peak indicating thrashing.]

*Colorado State University*
Demand Paging and Thrashing

• Why does demand paging work?
  **Locality model**
  – Process migrates from one locality to another
  – Localities may overlap

• Why does thrashing occur?
  
  size of locality > total memory size allocated
  
  – Limit effects by using local or priority page replacement
Locality In A Memory-Reference Pattern
Working-Set Model

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

$\Delta = 10$ page references

- page reference table
  - $\ldots \ 2 \ 6 \ 1 \ 5 \ 7 \ 7 \ 7 \ 5 \ 1 \ 6 \ 2 \ 3 \ 4 \ 1 \ 2 \ 3 \ 4 \ 4 \ 3 \ 4 \ 4 \ 4 \ 3 \ 4 \ 4 \ \ldots$

- $\Delta$ working-set window
  - $\Delta = 10$

- $WSS_i$ (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
    - Approximation of locality

- $D = \Sigma WSS_i$ total demand for frames for all processes
  - if $D > m \Rightarrow$ Thrashing
  - Policy if $D > m$, then suspend or swap out one of the processes
Page-Fault Frequency Approach

- More direct approach than WSS
- Establish “acceptable” page-fault frequency (PFF) rate and use local replacement policy
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame
Working Sets and Page Fault Rates

• Direct relationship between working set of a process and its page-fault rate
• Working set changes over time
• Peaks and valleys over time

Peaks occur at locality changes: 3 working sets
Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by **mapping** a disk block to a page in memory.
- **File is then in memory instead of disk.**
- A file is initially read using demand paging:
  - A page-sized portion of the file is read from the file system into a physical page.
  - Subsequent reads/writes to/from the file are treated as ordinary memory accesses.
- **Simplifies and speeds file access by driving file I/O through memory rather than `read()` and `write()` system calls.**
- Also allows several processes to map the same file allowing the pages in memory to be shared.
- But when does written data make it to disk?
  - Periodically and/or at file `close()` time.
  - For example, when the pager scans for dirty pages.
Memory Mapped Files

Disk File uses 6 blocks
Page tables used for mapping
Allocating Kernel Memory

• Treated differently from user memory
• Often allocated from a free-memory pool
  – Kernel requests memory for structures of varying sizes
    • Process descriptors, semaphores, file objects etc.
    • Often much smaller than page size
  – Some kernel memory needs to be contiguous
    • I.e. for device I/O
  – approaches (skipped)
Other Considerations -- Prepaging

• Prepaging
  – To reduce the large number of page faults that occurs at process startup
  – Prepage all or some of the pages a process will need, before they are referenced
  – But if prepaged pages are unused, I/O and memory was wasted
  – Assume $s$ pages are prepaged and fraction $\alpha$ of the pages is used
    • Is cost of $s \times \alpha$ saved pages faults > or < than the cost of prepaging $s \times (1 - \alpha)$ unnecessary pages?
    • $\alpha$ near zero $\Rightarrow$ greater prepaging loses
Other Issues – Page Size

• Sometimes OS designers have a choice
  – Especially if running on custom-built CPU
• Page size selection must take into consideration:
  – Fragmentation
  – Page table size
  – I/O overhead
  – Number of page faults
  – Locality
  – TLB size and effectiveness
• Always power of 2, usually in the range $2^{12}$ (4,096 bytes) to $2^{22}$ (4,194,304 bytes)
• On average, growing over time
Page size issues – TLB Reach

• TLB Reach - The amount of memory accessible from the TLB

• TLB Reach = (TLB Size) X (Page Size)

• Ideally, the working set of each process is stored in the TLB
  – Otherwise there is a high degree of page faults
Other Issues – Program Structure

• Program structure
  – int[128,128] data; i: row, j: column
  – Each row is stored in one page
  – Program 1
    for (j = 0; j <128; j++)
      for (i = 0; i < 128; i++)
        data[i,j] = 0;
    128 x 128 = 16,384 page faults

  – Program 2  inner loop = 1 row = 1 page
    for (i = 0; i < 128; i++)
      for (j = 0; j < 128; j++)
        data[i,j] = 0;
    128 page faults
Other Issues – I/O interlock

- **I/O Interlock** – Pages must sometimes be locked into memory
- Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm
- **Pinning** of pages to lock into memory
Windows

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page.
- Processes are assigned **working set minimum** and **working set maximum**.
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory.
- A process may be assigned as many pages up to its working set maximum.
- When the amount of free memory in the system falls below a threshold, **automatic working set trimming** is performed to restore the amount of free memory.
- Working set trimming removes pages from processes that have pages in excess of their working set minimum.
CS370 Operating Systems

Colorado State University
Yashwant K Malaiya
Spring 2018  Lecture T

File-system Interface

Slides based on
• Text by Silberschatz, Galvin, Gagne
• Various sources
Chapter 11: File-System Interface

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- File Sharing
- Protection
File Systems

"MS. GRIMMETT, I SORT OF LIKED THE OLD FILING SYSTEM...IN THE FILE CABINETS."
Outline

• File Concept, types
• Attributes, Access Methods, operations, Protection
• Directory Structure, namespace, File-System Mounting, File Sharing
• Next in File System Implementation
  – **Storage abstraction**: File system metadata (size, freelists), File metadata(attributes, disk block maps), datablocks
  – **Allocation of blocks to files**: contiguous, sequential, linked list allocation, indexed
  – **In memory info**: Mount table, directory structure cache, open file table, buffers
  – **Unix**: inodes numbers for directories and files
### File types

<table>
<thead>
<tr>
<th>file type</th>
<th>usual extension</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>executable</td>
<td>exe, com, bin or none</td>
<td>ready-to-run machine-language program</td>
</tr>
<tr>
<td>object</td>
<td>obj, o</td>
<td>compiled, machine language, not linked</td>
</tr>
<tr>
<td>source code</td>
<td>c, cc, java, pas, asm, a</td>
<td>source code in various languages</td>
</tr>
<tr>
<td>batch</td>
<td>bat, sh</td>
<td>commands to the command interpreter</td>
</tr>
<tr>
<td>text</td>
<td>txt, doc</td>
<td>textual data, documents</td>
</tr>
<tr>
<td>word processor</td>
<td>wp, tex, rtf, doc</td>
<td>various word-processor formats</td>
</tr>
<tr>
<td>library</td>
<td>lib, a, so, dll</td>
<td>libraries of routines for programmers</td>
</tr>
<tr>
<td>print or view</td>
<td>ps, pdf, jpg</td>
<td>ASCII or binary file in a format for printing or viewing</td>
</tr>
<tr>
<td>archive</td>
<td>arc, zip, tar</td>
<td>related files grouped into one file, sometimes compressed, for archiving or storage</td>
</tr>
<tr>
<td>multimedia</td>
<td>mpeg, mov, rm, mp3, avi</td>
<td>binary file containing audio or A/V information</td>
</tr>
</tbody>
</table>
File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the **directory structure**, which is maintained on the disk
- Many variations, including extended file attributes such as file **checksum**
Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Partition can be **formatted** with a file system
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system’s info in **device directory** or **volume table of contents**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer
Directory Structure

- A collection of nodes containing information about all files

Both the directory structure and the files reside on disk
Operations Performed on Directory

• Traverse the file system
• List a directory
• Search for a file
• Create/Delete/Rename a file
The directory is organized logically to obtain

- **Efficiency** – locating a file quickly
- **Naming** – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games, ...)

*Directory Organization*
Directory Organization

- Single level directory
- Two-level directory
- Tree-structured directories:
  - efficient grouping, searching,
  - absolute or relative path names
- Acyclic graph directories
  - Shared sub-directory, files
File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system is mounted at a **mount point**
- **Merges the file system**

(a) 

(b)
File Sharing

• Sharing of files on multi-user systems is desirable
• Sharing may be done through a protection scheme
• On distributed systems, files may be shared across a network
• Network File System (NFS) is a common distributed file-sharing method
• If multi-user system
  – User IDs identify users, allowing permissions and protections to be per-user
  – Group IDs allow users to be in groups, permitting group access rights
  – Owner of a file / directory
  – Group of a file / directory
Protection: Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux
  a) owner access
     7 \( \Rightarrow \) 1 1 1
  b) group access
     6 \( \Rightarrow \) 1 1 0
  c) public access
     1 \( \Rightarrow \) 0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

```
chmod 761 game
```

Attach a group to a file

```
chgrp G game
```
Windows 7 Access-Control List Management
### A Sample UNIX Directory Listing

<table>
<thead>
<tr>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>Size</th>
<th>Date/Time</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rw-rw-r--</td>
<td>1 pbg</td>
<td>staff</td>
<td>31200</td>
<td>Sep 3 08:30</td>
<td>intro.ps</td>
</tr>
<tr>
<td>drwx-------</td>
<td>5 pbg</td>
<td>staff</td>
<td>512</td>
<td>Jul 8 09:33</td>
<td>private/</td>
</tr>
<tr>
<td>drwxrwxr-x</td>
<td>2 pbg</td>
<td>staff</td>
<td>512</td>
<td>Jul 8 09:35</td>
<td>doc/</td>
</tr>
<tr>
<td>drwxrwx-----</td>
<td>2 pbg</td>
<td>student</td>
<td>512</td>
<td>Aug 3 14:13</td>
<td>student-proj/</td>
</tr>
<tr>
<td>-rw-r--r--</td>
<td>1 pbg</td>
<td>staff</td>
<td>9423</td>
<td>Feb 24 2003</td>
<td>program.c</td>
</tr>
<tr>
<td>-rwxr-xr-x</td>
<td>1 pbg</td>
<td>staff</td>
<td>20471</td>
<td>Feb 24 2003</td>
<td>program</td>
</tr>
<tr>
<td>drwx--x--x</td>
<td>4 pbg</td>
<td>faculty</td>
<td>512</td>
<td>Jul 31 10:31</td>
<td>lib/</td>
</tr>
<tr>
<td>drwx-------</td>
<td>3 pbg</td>
<td>staff</td>
<td>1024</td>
<td>Aug 29 06:52</td>
<td>mail/</td>
</tr>
<tr>
<td>drwxrwxrwx</td>
<td>3 pbg</td>
<td>staff</td>
<td>512</td>
<td>Jul 8 09:35</td>
<td>test/</td>
</tr>
</tbody>
</table>