Key memory management issues

- Utilization
- Programmability
- Performance
- Protection
Memory utilization

- What decreases memory utilization?
- What is fragmentation and why does it occur?
- How can fragmentation be reduced?
  - memory compaction?
  - paging?
- What problems are introduced by compaction and paging?
  - dynamic relocation
Supporting dynamic relocation

- Can application programmers and compilers easily deal with dynamic relocation?
- What new abstraction and associated hardware support is needed?
  - virtual address spaces
  - page tables for mapping virtual pages to physical page frames
  - MMU for automatic translation of virtual addresses to physical addresses
Paged virtual memory
Memory management using paging

Logical address: \(<\text{page number}, \text{page offset}>\)

<table>
<thead>
<tr>
<th>page number</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m-n)</td>
<td>(n)</td>
</tr>
</tbody>
</table>

Virtual Address Space

Page Table

Memory
Memory management unit (MMU)

The CPU sends virtual addresses to the MMU

The MMU sends physical addresses to the memory

CPU

CPU package

Memory management unit

Memory

Disk controller

Bus
Internal operation of a MMU

Virtual page = 2 is used as an index into the page table

12-bit offset copied directly from input to output

Outgoing physical address (24580)

Incoming virtual address (8196)
Page tables

A typical page table entry
Performance of memory translation

- Why can’t memory address translation be done in software?
- How often is translation done?
- What work is involved in translating a virtual address to a physical address?
  - indexing into page tables
  - interpreting page descriptors
  - more memory references!
- Do memory references to page tables hit in the cache?
  - if not what is the impact on performance?
Memory hierarchy performance

- The “memory” hierarchy consists of several types of memory
  - L1 cache (typically on dye) 0.5 ns! 1 cycle
  - L2 cache (typically available) 0.5 ns - 20 ns 1 - 40 cycles
  - Memory (DRAM, SRAM, RDRAM,...) 40 - 80 ns 80 - 160
  - Disk (lots of space available) 8 - 13 ms 16M - 26M
  - Tape (even more space available...) longer than you want!

360 Billion
Performance of memory translation (2)

- How can additional memory references be avoided?
  - TLB - translation look-aside buffer
  - an associative memory cache for page table entries
  - if there is locality of reference, performance is good
Translation lookaside buffer

Diagram showing the flow from CPU to page frame, TLB Hit, to physical memory.
## TLB entries

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virtual page</th>
<th>Modified</th>
<th>Protection</th>
<th>Page frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140</td>
<td>1</td>
<td>RW</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>RX</td>
<td>38</td>
</tr>
<tr>
<td>1</td>
<td>130</td>
<td>1</td>
<td>RW</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>129</td>
<td>1</td>
<td>RW</td>
<td>62</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>0</td>
<td>RX</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>0</td>
<td>RX</td>
<td>45</td>
</tr>
<tr>
<td>1</td>
<td>860</td>
<td>1</td>
<td>RW</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>861</td>
<td>1</td>
<td>RW</td>
<td>75</td>
</tr>
</tbody>
</table>
Page table organization

- How big should a virtual address space be?
  - what factors influence its size?
- How big are page tables?
  - what factors determine their size?
- Can page tables be held entirely in cache?
  - can they be held entirely in memory even?
- How should page tables be structured?
  - fast TLB miss handling (and write-back)?
  - what about unused regions of memory?
Two-level page table organization
Inverted page table

Traditional page table with an entry for each of the $2^{52}$ pages

Indexed by virtual page

256-MB physical memory has $2^{16}$ 4-KB page frames

Indexed by hash on virtual page

Hash table

Virtual page

Page frame
Address space organization

- How big should a virtual address space be?
- Which regions of the address space should be allocated for different purposes - stack, data, instructions?
- What if memory needs for a region increase dynamically?
- What are segments?
- What is the relationship between segments and pages?
- Can segmentation and paging be used together?
- If segments are used, how are segment selectors incorporated into addresses?
Memory protection

- At what granularity should protection be implemented?
  - page-level?
  - segment level?
- How is protection checking implemented?
  - compare page protection bits with process capabilities and operation types on every access
  - sounds expensive!
- How can protection checking be done efficiently?
  - segment registers
  - protection look-aside buffers
Segmentation & paging in the Pentium

A Pentium segment selector

Bits

13

1 2

Index

0 = GDT/1 = LDT

Privilege level (0-3)
Segmentation & paging in the Pentium

- Pentium segment descriptor
Segmentation & paging in the Pentium

Conversion of a (selector, offset) pair to a linear address
Segmentation & paging in the Pentium

Mapping of a linear address onto a physical address
Protection levels in the Pentium

Protection on the Pentium
Other VM-related costs

- What work must be done on a context switch?
- What work must be done on process creation?
- What work must be done on process termination?
Handling accesses to invalid pages

- The page table is used to translate logical addresses to physical addresses
- Pages that are not in memory are marked invalid
- A page fault occurs when there is an access to an invalid page of a process
- Page faults require the operating system to
  - suspend the process
  - find a free frame in memory
  - swap-in the page that had the fault
  - update the page table entry (PTE)
  - restart the process
Anatomy of a page fault

Logical memory:

Page table:

O.S.

Physical memory:

Page fault

Update PTE

Find Frame

Get from backing store

Bring in page

Restart Proc.

1

off

2

3

4

V

V

V

i

i

i

A

B

C

D

E

0

1

2

3

4

0

1

2

3

4

5

6

7

8

9

10

A

B

C

D

E

0

1

2

3

4

5

6

7

8

9

10

A

B

C

D

E

OGI SCHOOL OF SCIENCE & ENGINEERING
OREGON HEALTH & SCIENCE UNIVERSITY
Page fault handling in more detail

- Hardware traps to kernel
- General registers saved
- OS determines which virtual page needed
- OS checks validity of address, seeks page frame
- If selected frame is dirty, write it to disk
Page fault handling in more detail

- OS brings new page in from disk
- Page tables updated
- Faulting instruction backed up to when it began
- Faulting process scheduled
- Registers restored
- Program continues
Complexity of instruction backup

MOVE.L #6(A1), 2(A0)

16 Bits

1000
1002
1004

MOVE
6
2

Opcode
First operand
Second operand

An instruction causing a page fault
Locking pages in memory

- Virtual memory and I/O occasionally interact
- Process issues call for read from device into buffer
  - while waiting for I/O, another processes starts up
  - has a page fault
  - buffer for the first process may be chosen to be paged out
- Need to specify some pages locked (pinned)
  - exempted from being target pages
Spare Slides
Memory management

- **Memory** – a linear array of bytes
  - Hold O.S. and programs (processes)
  - Each memory cell is accessed by a unique memory address
- Recall, processes are defined by an address space, consisting of text, data, and stack regions
- **Process execution**
  - CPU fetches instructions from memory according to the value of the program counter (PC)
  - Each instruction may request additional operands from the data or stack region
The “memory” hierarchy consists of several types of memory:
- L1 cache (typically on dye)
- L2 cache (typically available)
- Memory (DRAM, SRAM, RDRAM,...)
- Disk (lots of space available)
- Tape (even more space available...)
Memory hierarchy

- The “memory” hierarchy consists of several types of memory
  - L1 cache (typically on dye) 0.5 ns!
  - L2 cache (typically available) 0.5 ns - 20 ns
  - Memory (DRAM, SRAM, RDRAM,...) 40 - 80 ns
  - Disk (lots of space available) 8 - 13 ms
  - Tape (even more space available...) longer than you want!
Memory hierarchy

- The “memory” hierarchy consists of several types of memory
  - L1 cache (typically on dye) \(0.5\) ns! 1 cycle
  - L2 cache (typically available) \(0.5\) ns - 20 ns 1 - 40 cycles
  - Memory (DRAM, SRAM, RDRAM,...) 40 - 80 ns 80 - 160
  - Disk (lots of space available) 8 - 13 ms 16M - 26M
  - Tape (even more space available...) longer than you want! 360 Billion
Understanding the memory hierarchy

- The memory hierarchy is extremely important in maximizing system performance
  
  Ex: 2 GHz processor
  - If missing the L1 cache at all times you reduce it to a 50 MHz processor

- The biggest “hits” in the memory system are currently:
  - Memory to cache interface (Hardware)
  - Disk to memory interface (OS)
Memory management overview

- Memory management - dynamically manages memory between multiple processes
  - Keep track of which parts of memory are currently being used and by whom
  - Provide protection between processes
  - Decide which processes are to be loaded into memory when memory space becomes available
  - Allocate and deallocate memory space as needed
  - Hide the effects of slow disks
    - Consistency vs. performance

- Maximize # of processes and throughput as well as minimize response times for requests
Simple memory management

- Load process into memory
- Run Process
Memory management issues

- How should memory be partitioned?
- How many processes?
- Swapping
- Relocation
- Protection
- Sharing
- Logical vs. Physical addresses
Degree of multiprogramming

CPU utilization (in percent) vs. Degree of multiprogramming for different I/O wait percentages: 20%, 50%, and 80%.
Address generation

- Processes generate logical addresses to physical memory when they are running.

- How/when do these addresses get generated?
  - Address binding - fixing a physical address to the logical address of a process’ address space
  - Compile time - if program location is fixed and known ahead of time
  - Load time - if program location in memory is unknown until run-time AND location is fixed
  - Execution time - if processes can be moved in memory during execution
    - Requires hardware support
Relocatable address generation

Compilation → Assembly → Linking → Loading

Prog P

foo() : push ... jmp _foo

End P

P:

push ...

jmp _foo

foo: ...

P:

push ...

jmp 75

foo: ...

P:

push ...

jmp 175

foo: ...

P:

push ...

jmp 1175

foo: ...

Library Routines

0

75

100

175

1100

1175

1000

Library Routines
Making systems more usable

- **Dynamic Loading** - load only those routines that are accessed while running
  - +) Does not load unused routines

- **Dynamic Linking** - link shared code such as system libraries and window code until run-time
  - +) More efficient use of disk space

- **Overlays** - allows procedures to “overlay” each other to decrease the memory size required to run the program
  - +) Allows more programs to be run
  - +) Programs can be larger than memory
Basics - logical and physical addressing

- **Memory Management Unit (MMU)** - dynamically converts logical addresses into physical address
- **MMU** stores base address register when loading process

Relocation register for process $i$

Program generated address

MMU

Physical memory address

Max Mem

process $i$

Operating system

Max addr
Swapping - allows processes to be temporarily “swapped” out of main memory to a backing store (typically disk)

Swapping has several uses:
- Allows multiple programs to be run concurrently
- Allows O.S. finer grain control of which processes can be run
Basics - simple memory protection

- "keep addresses in play"
  - Relocation register gives starting address for process
  - Limit register limits the offset accessible from the relocation register

![Diagram showing memory protection logic]

- Logical address
- Limit register
- Relocation register
- Physical address
- Memory
- Yes or No
- Addressing error
Basics - overlays

- Overlays - allow different parts of the same program to “overlay” each other in memory to reduce the memory requirements
- Example - scanner program
Memory management architectures

- **Fixed size allocation**
  - Memory is divided into fixed partitions
  - Fixed Partitioning (partition > proc. size)
    - Different constant size partitions
  - Paging (partition < proc. size)
    - Constant size partitions

- **Dynamically sized allocation**
  - Memory allocated to fit processes exactly
    - Dynamic Partitioning (partition > proc. size)
    - Segmentation
Multiprogramming with fixed partitions

- Memory is divided into fixed size partitions
- Processes loaded into partitions of equal or greater size

Internal Fragmentation
Multiprogramming with fixed partitions

(a) Multiple input queues

(b) Single input queue

Partition 4

Partition 3

Partition 2

Partition 1

Operating system

OGI SCHOOL OF SCIENCE & ENGINEERING
OREGON HEALTH & SCIENCE UNIVERSITY
Dynamic partitioning

- Allocate contiguous memory equal to the process size
- Corresponds to one job queue for memory

External Fragmentation
128K O.S.

896K

576K

P1

320K

O.S.

128K

352K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.

128K

64K

P3

288K

P2

224K

P1

320K

O.S.
Relocatable address generation

Compilation → Assembly → Linking → Loading

Prog P:
    foo()
    End P

P:
    push ...
    jmp _foo
    foo: ...

P:
    push ...
    jmp 75
    foo: ...

P:
    push ...
    jmp 175
    foo: ...

P:
    push ...
    jmp 1175
    foo: ...

Library Routines

0

75

175

1175

0

100

1000

1100

1175

0

100
Compile Time Address Binding

Load Time Address Binding

Execution Time Address Binding

Base register

Library Routines

P:  
  :  
push ...  
jmp 175  
:  

foo: ...

Library Routines

P:  
  :  
push ...  
jmp 175  
:  

foo: ...

Library Routines

P:  
  :  
push ...  
jmp 1175  
:  

foo: ...

Library Routines

P:  
  :  
push ...  
jmp 175  
:  

foo: ...

Library Routines

P:  
  :  
push ...  
jmp 1175  
:  

foo: ...
Dealing with external fragmentation

- **Compaction** - from time to time shift processes around to collect all free space into one contiguous block

- **Placement algorithms**: First-fit, best-fit, worst-fit
Compaction examples

FIRST-FIT

BEST-FIT
Compaction algorithms

- **First-fit**: place process in first hole that fits
  - Attempts to minimize scanning time by finding first available hole.
  - Lower memory will get smaller and smaller segments (until compaction algorithm is run)

- **Best-fit**: smallest hole that fits the process
  - Attempts to minimize the number of compactions that need to be run

- **Worst-fit**: largest hole in memory
  - Attempts to maximize the external fragment sizes
Memory management using paging

- Fixed partitioning of memory suffers from internal fragmentation, due to coarse granularity of the fixed memory partitions.

- Memory management via paging:
  - Permit physical address space of a process to be noncontiguous
  - Break physical memory into fixed-size blocks called frames
  - Break a process’s address space into the same sized blocks called pages
  - Pages are relatively small compared to processes (reduces the internal fragmentation)
Memory management using paging

- Logical address: \(<\text{page number}, \text{page offset}>\)

<table>
<thead>
<tr>
<th>page number</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m-n)</td>
<td>(n)</td>
</tr>
</tbody>
</table>

![Diagram showing logical address space, page table, and memory.]
Hardware Support for Paging

- The page table needs to be stored somewhere
  - Registers
  - Main Memory
- **Page Table Base Register (PTBR)** - points to the in memory location of the page table.
- **Translation Look-aside Buffers** make translation faster

**Paging Implementation Issues**
- Two memory accesses per address?
- What if page table > page size?
- How do we implement memory protection?
- Can code sharing occur?
Paging system performance

- The page table is stored in memory, thus, every logical address access results in TWO physical memory accesses:
  - 1) Look up the page table
  - 2) Look up the true physical address for reference

- To make logical to physical address translation quicker:
  - Translation Look-Aside Buffer - very small associative cache that maps logical page references to physical page references
  - Locality of Reference - a reference to an area of memory is likely to cause another access to the same area
Translation lookaside buffer
TLB implementation

- In order to be fast, TLBs must implement an associative search where the cache is searched in parallel.
  - EXPENSIVE
  - The number of entries varies (8 -> 2048)

- Because the TLB translates logical pages to physical pages, the TLB must be flushed on every context switch in order to work
  - Can improve performance by associating process bits with each TLB entry

- A TLB must implement an eviction policy which flushes old entries out of the TLB
  - Occurs when the TLB is full
Memory protection with paging

- Associate protection bits with each page table entry
  - Read/Write access - can provide read-only access for re-entrant code
  - Valid/Invalid bits - tells MMU whether or not the page exists in the process address space

<table>
<thead>
<tr>
<th>Frame #</th>
<th>R/W</th>
<th>V/I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>R</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>R</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>W</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>W</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>I</td>
</tr>
</tbody>
</table>

- Page Table Length Register (PTLR) - stores how long the page table is to avoid an excessive number of unused page table entries
Multilevel paging

- For modern computer systems,
  - \# frames \ll \# pages

- Example:
  - 8 kbyte page/frame size
  - 32-bit addresses
  - 4 bytes per PTE

  How many page table entries?

  How large is page table?

- Multilevel paging - page the page table itself
Multilevel paging

- Page the page table
- Logical address $\rightarrow$ [Section #, Page #, Offset]

How do we calculate size of section and page?
Virtual memory management overview

- What have we learned about memory management?
  - Processes require memory to run
  - We have assumed that the entire process is resident during execution

- Some functions in processes never get invoked
  - Error detection and recovery routines
  - In a graphics package, functions like smooth, sharpen, brighten, etc... may not get invoked

- Virtual Memory - allows for the execution of processes that may not be completely in memory (extension of paging technique from the last chapter)

Benefits?
Virtual memory overview

- Hides physical memory from user
- Allows higher degree of multiprogramming (only bring in pages that are accessed)
- Allows large processes to be run on small amounts of physical memory
- Reduces I/O required to swap in/out processes (makes the system faster)

- Requires:
  - Pager - page in /out pages as required
  - “Swap” space in order to hold processes that are partially complete
  - Hardware support to do address translation
Demand paging

- Each process address space is broken into pages (as in the paged memory management technique)
- Upon execution, swap in a page if it is not in memory (lazy swapping or demand paging)
- Pager - is a process that takes care of swapping in/out pages to/from memory
Demand paging implementation

- One page-table entry (per page)
- valid/invalid bit - tells whether the page is resident in memory
- For each page brought in, mark the valid bit

<table>
<thead>
<tr>
<th>Logical memory</th>
<th>Page table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 A</td>
<td>0 9 V</td>
</tr>
<tr>
<td>1 B</td>
<td>1 i</td>
</tr>
<tr>
<td>2 C</td>
<td>2 V</td>
</tr>
<tr>
<td>3 D</td>
<td>3 i</td>
</tr>
<tr>
<td>4 E</td>
<td>4 5 V</td>
</tr>
</tbody>
</table>

Physical memory: A B C D E
Another example

Physical memory

Logical memory

Page table
Chapter 4

Memory Management

4.1 Basic memory management
4.2 Swapping
4.3 Virtual memory
4.4 Page replacement algorithms
4.5 Modeling page replacement algorithms
4.6 Design issues for paging systems
4.7 Implementation issues
4.8 Segmentation
Memory Management

- Ideally programmers want memory that is
  - large
  - fast
  - non volatile

- Memory hierarchy
  - small amount of fast, expensive memory - cache
  - some medium-speed, medium price main memory
  - gigabytes of slow, cheap disk storage

- Memory manager handles the memory hierarchy
Basic Memory Management
Monoprogramming without Swapping or Paging

Three simple ways of organizing memory
- an operating system with one user process
Analysis of Multiprogramming System Performance

- Arrival and work requirements of 4 jobs
- CPU utilization for 1 - 4 jobs with 80% I/O wait
- Sequence of events as jobs arrive and finish
  - note numbers show amount of CPU time jobs get in each interval
Relocation and Protection

- Cannot be sure where program will be loaded in memory
  - address locations of variables, code routines cannot be absolute
  - must keep a program out of other processes’ partitions

- Use base and limit values
  - address locations added to base value to map to physical addr
  - address locations larger than limit value is an error
Swapping (1)

Memory allocation changes as
- processes come into memory
- leave memory

Shaded regions are unused memory
Swapping (2)

- Allocating space for growing data segment
- Allocating space for growing stack & data segment
Memory Management with Bit Maps

- Part of memory with 5 processes, 3 holes
  - tick marks show allocation units
  - shaded regions are free
- Corresponding bit map
- Same information as a list
Memory Management with Linked Lists

Four neighbor combinations for the terminating process X

Before X terminates

(a) A X B
(b) A X
(c) X B
(d) X

After X terminates

becomes

A B
A
X B
X
Small page size

- **Advantages**
  - less internal fragmentation
  - better fit for various data structures, code sections
  - less unused program in memory

- **Disadvantages**
  - programs need many pages, larger page tables
Page Size (2)

- Overhead due to page table and internal fragmentation

- Where
  - $s =$ average process size in bytes
  - $p =$ page size in bytes
  - $e =$ page entry

\[ \text{overhead} = \frac{s \cdot e}{p} + \frac{p}{2} \]

Optimized when

\[ p = \sqrt{2se} \]
Separate Instruction and Data Spaces

- One address space
- Separate I and D spaces
Two processes sharing same program sharing its page table
Cleaning Policy

- **Need for a background process, paging daemon**
  - periodically inspects state of memory

- **When too few frames are free**
  - selects pages to evict using a replacement algorithm

- **It can use same circular list (clock)**
  - as regular page replacement algorithm but with diff ptr
Implementation Issues

Operating System Involvement with Paging

Four times when OS involved with paging

- **Process creation**
  - determine program size
  - create page table

- **Process execution**
  - MMU reset for new process
  - TLB flushed

- **Page fault time**
  - determine virtual address causing fault
  - swap target page out, needed page in

- **Process termination time**
  - release page table, pages
Back in Store

(a) Paging to static swap area
(b) Backing up pages dynamically
Separation of Policy and Mechanism

Page fault handling with an external pager

User space

Kernel space

Main memory

Disk

1. Page fault
2. Needed page
3. Request page
4. Page arrives
5. Here is page
6. Map page in
Segmentation (1)

- One-dimensional address space with growing tables
- One table may bump into another
Segmentation (2)

Allows each table to grow or shrink, independently

OGI SCHOOL OF SCIENCE & ENGINEERING
OREGON HEALTH & SCIENCE UNIVERSITY
## Segmentation (3)

<table>
<thead>
<tr>
<th>Consideration</th>
<th>Paging</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need the programmer be aware that this technique is being used?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>How many linear address spaces are there?</td>
<td>1</td>
<td>Many</td>
</tr>
<tr>
<td>Can the total address space exceed the size of physical memory?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can procedures and data be distinguished and separately protected?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Can tables whose size fluctuates be accommodated easily?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Is sharing of procedures between users facilitated?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Why was this technique invented?</td>
<td>To get a large linear address space without having to buy more physical memory</td>
<td>To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection</td>
</tr>
</tbody>
</table>

### Comparison of paging and segmentation
Implementation of Pure Segmentation

(a)-(d) Development of checkerboarding
(e) Removal of the checkerboarding by compaction
Segmentation with Paging: MULTICS (1)

- Descriptor segment points to page tables
- Segment descriptor - numbers are field lengths
A 34-bit MULTICS virtual address
Segmentation with Paging: MULTICS (3)

Conversion of a 2-part MULTICS address into a main memory address
**Segmentation with Paging: MULTICS (4)**

<table>
<thead>
<tr>
<th>Comparison field</th>
<th>Segment number</th>
<th>Virtual page</th>
<th>Page frame</th>
<th>Protection</th>
<th>Age</th>
<th>Is this entry used?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>Read/write</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>Read only</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>Read/write</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>Execute only</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>Execute only</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Simplified version of the MULTICS TLB**
- **Existence of 2 page sizes makes actual TLB more complicated**
Page Replacement Algorithms and Performance Modelling
Virtual memory performance

- What is the limiting factor in the performance of virtual memory systems?
  - In the above example, steps 5 and 6 require on the order of 10 milliseconds, while the rest of the steps require on the order of microseconds/nanoseconds.
  - Thus, disk accesses typically limit the performance of virtual memory systems.

- Effective Access Time - mean memory access time from logical address to physical address retrieval
  
  $\text{effective access time} = (1-p) \times \text{ma} + p \times \text{page\_fault\_time}$
  
  $p = \text{probability that a page fault will occur}$
The great virtual memory struggle

- Having the option to run programs that are partially in memory leads to a very interesting problem:
  - How many programs should we allow to run in memory at any given time?
  - We can make sure that all the pages of all processes can fit into memory
    - ) may be under-allocation of memory
  - We can over-allocate the memory by assuming that processes will not access (or need) all their pages at the same time
    - ) may run out of pages in memory
    + ) can increase the throughput of the system
Page replacement

- Page Replacement - is a technique which allows us to increase the degree of multiprogramming (i.e. over-allocate memory) by using the disk as “extended” memory.

- If a page fault occurs and no frames are free:
  1) Find a frame in memory not currently being used
  2) Select a victim page and swap it out to the swap-space on disk (changing its page table entry)
  3) Use the freed frame to hold the page that caused the page fault
  4) Restart the process

- Requires two page transfers to memory (one in, one out)
Page replacement performance

- Because page replacement could potentially result in two disk transfers, a small number of page faults can greatly impact system performance
  - Ex. Mem. Access. = 60ns, disk access = 10 ms, page fault rate = 0.01
  - No page replacement (just demand paging)
    - $EAT = 0.99 \times 60\text{ns} + 0.01 \times 10\text{ms} = 100.060\text{ us}$
  - Page replacement
    - $EAT = 0.99 \times 60\text{ns} + 0.01 \times 2 \times 10\text{ms} = 200.060\text{ us}$

- Page replacement is key in virtual memory performance

- Using a modified bit in the PTE helps by only swapping out pages that have been modified

- Efficient page replacement and frame allocation algorithms are required for efficient virtual memory
Page replacement algorithms

- Page replacement algorithms determine what frame in memory should be used to handle a page fault
  - May require the frame to be swapped out to the swap-space if it has been modified

- Page replacement algorithms are typically measured by their page fault rate or the rate at which page faults occur.
  - Commonly referred to algorithms:
    - First-in First-out (FIFO)
    - Optimal
    - Least-recently used (LRU)
    - Least-frequently used (LFU)
Page replacement algorithms

- Which page should be replaced?
  - Local replacement - replace page of faulting process
  - Global replacement - possibly replace page of another process in memory

- Evaluation of algorithms?
  - Record traces of pages accessed by a process

  - Example: Virtual addresses (page, offset)
    - (3,0), (1,9), (4,1), (2,1), (5,3), (2,0), (1,9), (2,4)
    - ... generate page trace
    - 3, 1, 4, 2, 5, 2, 1, 2

- Simulate behavior of process and measure the number of page faults that occur
FIFO page replacement

- Replace the page that was first brought into memory
  + Simple to implement (FIFO queue)
  - Performance not always good

Example: Memory system with 4 frames:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

| Page | 0 | a | a | a | a | a | a | a | a | a | a |
| Frames | 1 | b | b | b | b | b | b | b | b | b | b |
| 2 | c | c | c | c | c | c | c | c | c | c | c |
| 3 | d | d | d | d | d | d | d | d | d | d | d |

Page faults | X |
### FIFO page replacement

- **Replace the page that was first brought into memory**
  - Simple to implement (FIFO queue)
  - Performance not always good

- **Example: Memory system with 4 frames**:

<table>
<thead>
<tr>
<th>Time</th>
<th>Requests</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td><strong>Page 0</strong></td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>e</td>
<td>e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Frames 1</strong></td>
<td>b</td>
<td></td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Page faults</strong></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FIFO page replacement

- Replace the page that was first brought into memory
  + Simple to implement (FIFO queue)
  - Performance not always good

- Example: Memory system with 4 frames:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>Frames 1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>

Page faults | X | X | X | X | X | X | X | X | X | X | X |
Page replacement and # of pages

- One would expect that with more memory the number of page faults would decrease
- Belady's Anomaly - More memory does not always mean better performance

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>e</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames 1</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page faults
Page replacement and # of pages

- One would expect that with more memory the number of page faults would decrease.
- Belady's Anamoly - More memory does not always mean better performance.

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>e</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames 1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page faults: x
Page replacement and # of pages

- One would expect that with more memory the number of page faults would decrease
- Belady's Anamoly - More memory does not always mean better performance

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>e</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>e</td>
</tr>
</tbody>
</table>

| Page  | 0 | a | a | a | a | d | d | d | d | e | e | e | e |
| Frames | 1 | b | b | b | b | b | a | a | a | a | a | c | c |
|        | 2 | c | c | c | c | c | b | b | b | b | b | d | d |

Page faults: X X X X X X X X
Belady's anamoly

- 4 pages

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>e</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Page  | 0 | a |
| Frame | 1 | b |
|       | 2 | c |
|       | 3 |   |

Page faults
Belady's anamoly

- **4 pages**

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>e</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames 1</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>

Page faults: X
Belady's anamoly

4 pages

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>a b c d a b e a b c d e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames 1</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page faults</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page faults: x x x x x x x x x
Optimal page replacement

- Replace the page that will not be needed for the longest period of time
  + Minimum number of page faults
  - How do you foresee the future?

- Example:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
</tr>
</tbody>
</table>

Page faults: X
Optimal page replacement

- Replace the page that will not be needed for the longest period of time
  
  +) Minimum number of page faults
  
  -) How do you foresee the future?

- Example:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Frames</th>
<th>Page faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 a</td>
<td>1 b</td>
<td>X</td>
</tr>
<tr>
<td>2 c</td>
<td>d</td>
<td>X</td>
</tr>
</tbody>
</table>

OGI SCHOOL OF SCIENCE & ENGINEERING
OREGON HEALTH & SCIENCE UNIVERSITY
LRU page replacement

- Replace the page that hasn't been referenced in the longest time
  - Uses recent past as predictor for the future
  - Quite widely used

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>Page</td>
<td>0</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames</td>
<td>1</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page faults
LRU page replacement

- Replace the page that hasn't been referenced in the longest time
  - Uses recent past as predictor for the future
  - Quite widely used

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>Frames 1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td>Page faults</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
LRU implementation

- LRU requires the O.S. to keep track of the accesses to the pages in memory
  - Exact LRU implementation
    1) Counters - save “clock” for each reference smallest “clock” is the victim page
    2) Stacks - every time page referenced put at the top of the stack bottom of stack is the victim page
    - Both require keeping fairly detailed information
  - Approximate LRU implementation
    1) The clock algorithm (second-chance algo.)
    2) Two-handed clock algorithm
LRU implementation

- Take referenced and put on top of stack

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>0</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

Page faults
LRU implementation

- Take referenced and put on top of stack

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames 1</td>
<td>b</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page faults
**LRU implementation**

- Take referenced and put on top of stack

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames 1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page faults | x

```
C A D B 
A C A D
B B C A
D D B C
```

OGI SCHOOL OF SCIENCE & ENGINEERING
OREGON HEALTH & SCIENCE UNIVERSITY
# LRU implementation

- Take referenced and put on top of stack

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Frames</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>d</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Page faults: X X X

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>D</th>
<th>A</th>
<th>B</th>
<th>E</th>
<th>B</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>A</td>
<td>D</td>
<td>B</td>
<td>E</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>D</td>
<td>E</td>
<td>D</td>
<td>E</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>D</td>
<td>D</td>
<td>E</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>
Clock algorithm

- Maintain a circular list of pages in memory
- Set a clock bit for the page when a page is referenced
- Clock sweeps over memory looking for a page that does not have the clock-bit set
- Replace pages that haven’t been referenced for one complete clock revolution
**Clock algorithm**

- **Example:** clear one-page per reference

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames 1</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Page faults**

- Diagram showing page faults for each time step.
Clock algorithm

Example: clear one-page per reference

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>0</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>d</td>
</tr>
</tbody>
</table>

Page faults:

```
0
1
1
1
```

```
0
0
```

```
1
1
```

```
1
1
```

```
0
0
```

```
1
1
```

```
1
1
```

```
0
0
```

```
1
1
```

```
1
1
```
Clock algorithm

- Example: clear one-page per reference

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames 1 1</td>
<td>b</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page faults

```
  0  1  0  1
  1  1  1  1
  1  1  1  1
  0  0  0  0
```

OGI SCHOOL OF SCIENCE & ENGINEERING
OREGON HEALTH & SCIENCE UNIVERSITY
Clock algorithm

- Example: clear one-page per reference

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page faults

```
0 1 1 1 1
0 0 0 0
0 0 0 0
```
Clock algorithm

- Example: clear one-page per reference

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames 1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page faults
Clock algorithm

- Example: clear one-page per reference

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

| Page | 0   | a   | a   | a   | a   | a   |     |     |     |     |     |
| Frames | 1   | b   | b   | b   | b   | b   |     |     |     |     |     |
|       | 2   | c   | c   | c   | c   | c   | e   |     |     |     |     |
|       | 3   | d   | d   | d   | d   | d   | d   |     |     |     |     |

Page faults: X
**Clock algorithm**

- **Example: clear one-page per reference**

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page faults</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Theory and practice

- Identifying victim frame on each page fault typically requiring two disk accesses per page fault.

- Alternative → the O.S. can keep several pages free in anticipation of upcoming page faults.
  
  In Unix: low and high water marks

```
low water mark

high water mark
```

low < # free pages < high
Free pages and the clock algorithm

- The rate at which the clock sweeps through memory determines the number of pages that are kept free:
  - Too high a rate --> Too many free pages marked
  - Too low a rate --> Not enough (or no) free pages marked

- Large memory system considerations
  - As memory systems grow, it takes longer and longer for the hand to sweep through memory
  - This washes out the effect of the clock somewhat
  - Can use a two-handed clock to reduce the time between the passing of the hands
The UNIX memory model

- **UNIX page replacement**
  - Two handed clock algorithm for page replacement
    - If page has not been accessed move it to the free list for use as allocatable page
      - If modified/dirty → write to disk (still keep stuff in memory though)
      - If unmodified → just move to free list
  - High and low water marks for free pages
  - Pages on the free-list can be re-allocated if they are accessed again before being overwritten
VM and multiprogramming

- **Goal:** Maximize # processes, minimize response time

- **Measurements of real operating systems** has led to the following CPU utilization measurements:

- **Thrashing** - when the CPU is spending all of its time swapping in/out pages from/to disk
Prevention of thrashing

- In order to prevent thrashing, we really need to know how many pages of process needs at any given time.

- Given these numbers, we then allocate memory such that the sum of all the needs of the processes is less than the total memory available.
  
  Problem - each process’ set of pages required dynamically changes during its execution!

- Locality model
  - As processes execute, they move from locality to locality, each with a set of pages that are actively used together.
  - Programs consist of several localities.
Working set model

- Based on the assumption of locality
- Use parameter $\Delta$ to define working-set window
- The set of pages in the most recent $\Delta$ references is the working set

- Working sets and virtual memory
  - Working-sets change over time
  - Want to make sure that the sum of all the process' working-sets is less than the memory size
  - Prevents thrashing, while keeping the degree of multiprogramming high
  - Has a large amount of overhead
Working set modeling

- Given a fixed $D$, processes exhibit working-sets similar to the graph below:
Prevention of thrashing

- The working-set model gives a reasonably accurate measurement of the number of pages needed by a process at any given time
  - Requires keeping track of the working-set

- Another method for preventing thrashing is dynamically measuring page fault frequency
  - If the page fault frequency is high, we know that the process requires more frames
  - If the page fault is low, then the process may have too many frames

- Like the low and high water marks for memory, we can do the same for page fault frequencies
Page Replacement Algorithms

- Page fault forces choice
  - which page must be removed
  - make room for incoming page

- Modified page must first be saved
  - unmodified just overwritten

- Better not to choose an often used page
  - will probably need to be brought back in soon
Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future
  - Optimal but unrealizable

- Estimate by ...
  - logging page use on previous runs of process
  - although this is impractical
Not Recently Used Page Replacement Algorithm

- Each page has Reference bit, Modified bit
  - bits are set when page is referenced, modified
- Pages are classified
  - not referenced, not modified
  - not referenced, modified
  - referenced, not modified
  - referenced, modified
- NRU removes page at random
  - from lowest numbered non empty class
FIFO Page Replacement Algorithm

- Maintain a linked list of all pages
  - in order they came into memory

- Page at beginning of list replaced

- Disadvantage
  - page in memory the longest may be often used
Second Chance Page Replacement Algorithm

Page loaded first

- Most recently loaded page
- A is treated like a newly loaded page

Operation of a second chance
- Pages sorted in FIFO order
- Page list if fault occurs at time 20, A has R bit set
- Numbers above pages are loading times
The Clock Page Replacement Algorithm

When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

- $R = 0$: Evict the page
- $R = 1$: Clear R and advance hand
Least Recently Used (LRU)

- Assume pages used recently will be used again soon
  - throw out page that has been unused for longest time

- Must keep a linked list of pages
  - most recently used at front, least at rear
  - update this list every memory reference !!

- Alternatively keep counter in each page table entry
  - choose page with lowest value counter
  - periodically zero the counter
Simulating LRU in Software (1)

LRU using a matrix – pages referenced in order
0,1,2,3,2,1,0,3,2,3
- The aging algorithm simulates LRU in software
- Note 6 pages for 5 clock ticks, (a) - (e)
The Working Set Page Replacement Algorithm (1)

- The working set is the set of pages used by the $k$ most recent memory references.
- $w(k,t)$ is the size of the working set at time, $t$. 

![Graph](image)
The Working Set Page Replacement Algorithm (2)

The working set algorithm

Scan all pages examining R bit:
if (R == 1)
    set time of last use to current virtual time
if (R == 0 and age > τ)
    remove this page
if (R == 0 and age ≤ τ)
    remember the smallest time
The WSClock Page Replacement Algorithm

Operation of the WSClock algorithm
Review of Page Replacement Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>Not implementable, but useful as a benchmark</td>
</tr>
<tr>
<td>NRU (Not Recently Used)</td>
<td>Very crude</td>
</tr>
<tr>
<td>FIFO (First-In, First-Out)</td>
<td>Might throw out important pages</td>
</tr>
<tr>
<td>Second chance</td>
<td>Big improvement over FIFO</td>
</tr>
<tr>
<td>Clock</td>
<td>Realistic</td>
</tr>
<tr>
<td>LRU (Least Recently Used)</td>
<td>Excellent, but difficult to implement exactly</td>
</tr>
<tr>
<td>NFU (Not Frequently Used)</td>
<td>Fairly crude approximation to LRU</td>
</tr>
<tr>
<td>Aging</td>
<td>Efficient algorithm that approximates LRU well</td>
</tr>
<tr>
<td>Working set</td>
<td>Somewhat expensive to implement</td>
</tr>
<tr>
<td>WSClock</td>
<td>Good efficient algorithm</td>
</tr>
</tbody>
</table>
Modeling Page Replacement Algorithms
Belady's Anomaly

All pages frames initially empty

Youngest page

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

Oldest page

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

9 Page faults

(a)

Youngest page

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

Oldest page

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

10 Page faults

(b)

- FIFO with 3 page frames
- FIFO with 4 page frames
- P's show which page references show page faults
### Stack Algorithms

#### State of memory array, $M$, after each item in reference string is processed

| Reference string | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 |
|                  | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 |
|                  | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 7 | 3 | 3 | 5 | 3 | 3 | 3 | 1 | 7 | 1 | 3 | 4 | 1 |
|                  | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 3 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 5 | 3 | 3 | 7 | 1 | 3 | 4 |
|                  | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 6 | 6 | 6 | 4 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 5 | 5 | 7 | 7 | 7 |
|                  | 0 | 2 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 5 |
|                  | 0 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
|                  | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|                  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |


| Distance string  | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 4 | $\infty$ | 4 | 2 | 3 | 1 | 5 | 1 | 2 | 6 | 1 | 1 | 4 | 7 | 4 | 6 | 5 | 7 | 4 | 6 | 5 |

State of memory array, $M$, after each item in reference string is processed.
The Distance String

Probability density functions for two hypothetical distance strings
The Distance String

- Computation of page fault rate from distance string
  - the C vector
  - the F vector
Design Issues for Paging Systems
Local versus Global Allocation Policies (1)

- **Original configuration**
- **Local page replacement**
- **Global page replacement**

<table>
<thead>
<tr>
<th></th>
<th>Age</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>10</td>
<td>A0</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>7</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>5</td>
<td>A2</td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>4</td>
<td>A3</td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>6</td>
<td>A4</td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>3</td>
<td>A6</td>
<td></td>
</tr>
<tr>
<td>B0</td>
<td>9</td>
<td>B0</td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td>4</td>
<td>B1</td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>6</td>
<td>B2</td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>2</td>
<td>B3</td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td>5</td>
<td>B4</td>
<td></td>
</tr>
<tr>
<td>B5</td>
<td>6</td>
<td>B5</td>
<td></td>
</tr>
<tr>
<td>B6</td>
<td>12</td>
<td>B6</td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>3</td>
<td>C1</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>5</td>
<td>C2</td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>6</td>
<td>C3</td>
<td></td>
</tr>
</tbody>
</table>

(a) Original configuration
(b) Local page replacement
(c) Global page replacement
Local versus Global Allocation Policies (2)

Page fault rate as a function of the number of page frames assigned

Page fault rate as a function of the number of page frames assigned
Load Control

- Despite good designs, system may still thrash

- When PFF algorithm indicates
  - some processes need more memory
  - but no processes need less

- Solution:
  - Reduce number of processes competing for memory
    - swap one or more to disk, divide up pages they held
    - reconsider degree of multiprogramming