Virtual Memory

Today

- Virtual memory
- Page replacement algorithms
- Modeling page replacement algorithms
Virtual memory

- Handling processes >> than allocated memory
- Keep in memory only what’s needed
  - Full address space does not need to be resident in memory
    - Leave it on disk
  - OS uses main memory as a cache

- Overlay approach
  - Implemented by user
  - Easy on the OS, hard on the programmer

Overlay for a two-pass assembler:

<table>
<thead>
<tr>
<th>Pass</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td>70KB</td>
</tr>
<tr>
<td>Pass 2</td>
<td>80KB</td>
</tr>
<tr>
<td>Symbol Table</td>
<td>20KB</td>
</tr>
<tr>
<td>Common Routines</td>
<td>30KB</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>200KB</strong></td>
</tr>
</tbody>
</table>

Two overlays: 120 + 130KB
Virtual memory

- Hide the complexity – let the OS do the job
- Virtual address space split into pages
- Physical memory split into page frames
- Page & page frames = size (512B … 64KB)
- Map pages into page frames
  - Doing the translation – OS + MMU
Pages, page frames and tables

A simple example with
- 64KB virtual address space
- 4KB pages
- 32KB physical address space
- 16 pages and 8 page frames

Try to access:
- MOV REG, 0
  Virtual address 0
  Page frame 2
  Physical address 8192

- MOV REG, 8192
  Virtual address 8192
  Page frame 6
  Physical address 24576

- MOV REG, 20500
  Virtual address 20500 (20480 + 20)
  Page frame 3
  Physical address 20+12288
Since virtual memory >> physical memory

- Use a present/absent bit
- MMU checks –
  - If not there, “page fault” to the OS (trap)
  - OS picks a victim (?)
  - … sends victim to disk
  - … brings new one
  - … updates page table

```
MOVE REG, 32780
Virtual address 32780
Virtual page 8, byte 12 (32768+12)
Page is unmapped – page fault!
```
Details of the MMU work

- MMU with 16 4KB pages
- Page # (first 4 bits) index into page table
- If not there
  - Page fault
- Else
  - Output register +
  - 12 bit offset →
  - 15 bit physical address
Page table entry

- Page frame number – the most important field
- Protection – 1 bit for R&W or R or 3 bits for RWX
- Present/absent bit
  - Says whether or not the virtual address is used
- Modified (M): dirty bit
  - Set when a write to the page has occurred
- Referenced (R): Has it being used?
- To ensure we are not reading from cache (D)
  - Key for pages that map onto device registers rather than memory
Page replacement algorithms

- OS uses main memory as (page) cache
  - If only load when reference – demand paging
- Page fault – cache miss
  - Need room for new page? Page replacement algorithm
  - What’s your best candidate for removal?
    • The one you will never touch again – duh!
- What do you do with victim page?
  - Modified page must first be saved
  - Unmodified one just overwritten
  - Better not to choose an often used page
    • It will probably need to be brought back in soon
- Try to avoid thrashing
  - OS wastes most of the time moving pages around
  - Fix the algorithm, swap out somebody, get more memory
How can any of this work?!?!

- **Locality**
  - Temporal locality – location recently referenced tend to be referenced again soon
  - Spatial locality – locations near recently referenced are more likely to be referenced soon

- **Locality means paging could be infrequent**
  - Once you brought a page in, you’ll use it many times
  - Some issues that may play against you
    - Degree of locality of application
    - Page replacement policy and application reference pattern
    - Amount of physical memory and application footprint
## Optimal algorithm (Belady’s algorithm)

- For now, assume a process pages against itself, using a fixed number of page frames
- Best page to replace – the one you’ll never need again
  - Replace page needed at the farthest point in future
  - Optimal but unrealizable
- Estimate by …
  - Logging page use on previous runs of process
  - Although impractical, useful for comparison

<table>
<thead>
<tr>
<th>1, 2, 3, 4, 1, 2</th>
<th>1, 2, 3, 4, 5</th>
<th>Need room for this one</th>
<th>Your ideal victim!</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, 3, 4, 5</td>
<td>1, 2, 3, 4, 5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EECS 343 Operating Systems
Northwestern University
FIFO algorithm

- Maintain a linked list of all pages – in order of arrival
- Victim is first page of list
  - Maybe the oldest page will not be used again …
- Disadvantage
  - But maybe it will – the fact is, you have no idea!
  - Increasing physical memory might increase page faults
    (Belady’s anomaly, we’ll come back to this)

A, B, C, D, A, B, E, A, B, C, D, E

```
A
A
A

A
B
C

A
B
C

A
B
C

A
B
C

A
B
C

A
B
C

A
B
C

A
B
C

A
B
C
```
Least recently used (LRU) algorithm

- Pages used recently will be used again soon
  - Throw out page unused for longest time
  - Idea: past experience is a decent predictor of future behavior
    - LRU looks at the past, Belady’s wants to look at the future
    - *how is LRU different from FIFO?*

- Must keep a linked list of pages
  - Most recently used at front, least at rear
  - Update this list every memory reference!!
  - Too expensive in memory bandwidth, algorithm execution time, etc

- Alternatively keep counter in page table entry
  - Choose page with lowest value counter
  - Periodically zero the counter
A second HW LRU implementation

- Use a matrix – $n$ page frames – $n \times n$ matrix
- Page $k$ is reference
  - Set all bits of row $k$ to 1
  - Set all bits of column $k$ to 0
- Page of lowest row is LRU

0,1,2,3,2,1,0,3,2
Simulating LRU in software

- Not Frequently Used
  - Software counter per page
  - At clock interrupt – add R to counter for each page
  - Problem - it never forgets!

- Better – Aging
  - Push R from the left, drop bit on the right
  - How is this not LRU? One bit per tick & a finite number of bits per counter
Not recently used (NRU) algorithm

- Each page has *Reference* and *Modified* bits
  - Set when page is referenced, modified
  - R bit set means recently referenced, so you must clear it every now and then
- Pages are classified

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Not referenced, not modified (0,0 → 0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Not referenced, modified (0,1 → 1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Referenced, but not modified (1,0 → 2)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Referenced and modified (1,1 → 3)</td>
</tr>
</tbody>
</table>

- NRU removes page at random
  - from lowest numbered, non-empty class
- Easy to understand, relatively efficient to implement and sort-of OK performance
Second chance algorithm

- Simple modification of FIFO
  - Avoid throwing out a heavily used page – look at the R bit

- Operation of second chance
  - Pages sorted in FIFO order
  - Page list if fault occurs at time 20, A has R bit set
    (time is loading time)

<table>
<thead>
<tr>
<th>Page</th>
<th>Time</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>18</td>
<td>X</td>
</tr>
<tr>
<td>G</td>
<td>15</td>
<td>X</td>
</tr>
<tr>
<td>F</td>
<td>14</td>
<td>X</td>
</tr>
<tr>
<td>E</td>
<td>12</td>
<td>X</td>
</tr>
<tr>
<td>D</td>
<td>8</td>
<td>X</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Most recently loaded

Oldest page

<table>
<thead>
<tr>
<th>Page</th>
<th>Time</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>18</td>
<td>X</td>
</tr>
<tr>
<td>G</td>
<td>15</td>
<td>X</td>
</tr>
<tr>
<td>F</td>
<td>14</td>
<td>X</td>
</tr>
<tr>
<td>E</td>
<td>12</td>
<td>X</td>
</tr>
<tr>
<td>D</td>
<td>8</td>
<td>X</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Clock algorithm

- Quit moving pages around – move a pointer?
- Same as Second chance but for implementation
  - When page fault
  - Look at page pointed at by hand
    - If R = 0, evict page
    - If R = 1, clear R & move hand

```
R: 0
A
R: 0
B
R: 0
C
R: 0
D
R: 0
E
R: 0
F
R: 0
G
R: 0
H
R: 0
I
R: 0
J
R: 0
K
R: 0
L
R: 0
M
R: 0
N
R: 0
O
R: 0
P
R: 0
Q
R: 0
R
R: 0
S
R: 0
T
R: 0
U
R: 0
V
R: 0
W
R: 0
X
R: 0
Y
R: 0
Z
```
Demand paging
  - Simplest strategy, load page when needed

Most programs show *locality of reference*
  - Over a short time, just a few common pages

Working set
  - Set of pages used by $k$ most recent memory references
  - $ws(k, t)$ – size of working set at time $t$ ($k$ is WS window size)
  - *What bounds $ws(k, t)$ as you increase $k$?*
  - *How could you use this to reduce turnaround time?*

Clearly $ws(k_i, t) \leq ws(k_j, t)$ for $i < j$

\[ ws(k, t) \]

\[ k \]
## Working set algorithm

- **Working set and page replacement**
  - Victim – a page *not* in the working set

- **At each clock interrupt** – scan the page table
  - $R = 1$? Write Current Virtual Time (CVT) into *Time of Last Use*
  - $R = 0$? CVT – *Time of Last Use > Threshold* ? out! else see if there’s someone and evict oldest (w/ R=0)
  - If all are in the working set (all $R = 1$) random
WSClock algorithm

- Problem with WS algorithm – Scans the whole table
- Combine clock & working set
  - If R = 1, same as working set
  - If R = 0, if age > T and page clean, out
  - If dirty, schedule write and check next one
  - If loop around, There’s 1+ write scheduled – you’ll have a clean page soon
    There’s none, pick any one

R = 0 & 2204 – 1213 > T
Belady's anomaly

- The more page frames the fewer page faults, right?
  - FIFO with 3 page frames
  - FIFO with 4 page frames

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

All page frames initially empty

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

P P P P P P
Belady's anomaly

The more page frames the fewer page faults, right?

- FIFO with 3 page frames
- FIFO with 4 page frames

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

9 page faults

| P | P | P | P | P | P | P | P |

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

10 page faults

EECS 343 Operating Systems
Northwestern University
Modeling page replacement algorithms

- Paging system can be characterized by
  - Page replacement algorithm
  - a reference string
  - # page frames

- Abstract interpreter with
  - Internal array, M, to keep track of memory state
    - Size of (M) = # virtual pages, n
  - Split in two parts
    - Top m entries, for m pages frame
    - The bottom part (n – m) for pages that have been referenced but eventually paged out
  - Initially M is empty
An example using LRU

Reference to a page (5) out of the blue box → page fault

| 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 |
| Pages in page frames |
| Pages paged out to disk |
| Distance string | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 4 | ∞ | 4 | 2 | 3 | 1 | 5 | 1 | 2 | 6 | 1 | 1 | 4 | 2 | 3 | 5 | 3 |
Stack algorithms

Model works well with other algorithms. Particularly interesting …

Stack algorithm: \( M(m,r) \subseteq M(m+1,r) \)

Distance string – each page reference denoted by the distance from top of the stack where the page was located (if not yet referenced: \( \infty \))

Pages in memory with \( m \) pages frames and after \( r \) memory references

Probability density function of two distance strings
Distance string & page faults

Computation of page fault rate from distance string

\[ C_i \] – number of occurrences of \( i \) in distance string

\[ F_m \] – number of page faults with \( m \) frames

\[
F_m = \sum_{k=m+1}^{n} C_k + C_\infty
\]

<table>
<thead>
<tr>
<th>Reference string</th>
<th>0</th>
<th>2</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>4</th>
<th>6</th>
<th>3</th>
<th>7</th>
<th>4</th>
<th>7</th>
<th>3</th>
<th>5</th>
<th>5</th>
<th>3</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>7</th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Distance string</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>4</td>
<td>\infty</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Number of times 1 occur in distance string

Number of page faults with 6 frames

\[ C_7 + C_\infty \]
## Distance string & page faults

### Computation of page fault rate from distance string

- $C_i$ – number of occurrences of i in distance string
- $F_m$ – number of page faults with m frames

### Table

| Reference string | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 |
|                  | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 |
|                  | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 |
|                  | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 |
|                  | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 6 | 6 | 6 | 6 | 6 | 4 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 7 | 7 |
|                  | 0 | 2 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
|                  | 0 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 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 |
|                  | 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  | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 4 | 4 | 2 | 3 | 1 | 5 | 1 | 2 | 6 | 1 | 1 | 4 | 2 | 3 | 5 | 3 |

### Equation

$$F_m = \sum_{k=m+1}^{n} C_k + C_\infty$$
Next time …

- You now understand how things work, i.e. the mechanism …
- We’ll now consider design & implementation issues for paging systems
  - Things you want/need to pay attention for good performance