§ LFU

- Least Frequently Used (LFU): select for replacement the page that is least frequently used
  - needs counter mechanism
**LFU**

- **Least Frequently Used (LFU):** select for replacement the page that is least frequently used
  - needs counter mechanism
- **consider this scenario:**
  - executing program cross page boundary to \( P_j \) from \( P_i \)
  - timeslice switch to another process
  - that other process now has page fault
  - frame holding \( P_j \) is least frequently used so is replaced
- LFU only slightly better than…

**Random**

- **Random:** select a page at random for replacement
  - poor performance, high page fault rate
“Clock”

- let each frame have a use bit
- when page first loaded into frame, set use bit to 1
- each subsequent reference set use bit to 1
- also known as second chance
  - will become apparent why

“Clock”

- organize frames into circular buffer with next-frame pointer:
- when a page is loaded, pointer advances to next frame
to select page for replacement:
• scan buffer for frame with $U = 0$
• while doing this, clear any $U = 1$ to 0
• stop at first frame with $U = 0$

what’s worst case?
if ignored $U$ bits, what would this be?

---

**Example of Clock (Second Chance)**

```
0 7 0 1 2 0 3 0 4 2 3 0 3 2 1 0 1 7 0 1

→

PF?
```

```
0 7
1
2
PF?
```

PFs: 1
### Example of Clock (Second Chance)

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>4</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>7</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>7</td>
<td>7</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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>0</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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PF?</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFS:</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Example of Clock (Second Chance)

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>4</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>7</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>7</td>
<td>7</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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>0</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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PF?</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFS:</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Charles M Snow 5
Example of Clock (Second Chance)

1. Start with the initial state: 0 7 0 0 0 0 1 7 0 1
2. Perform the first operation: 0 7 7 7 2
3. Perform the second operation: 0 0 0 0 0 0 1 7 0 1
4. Perform the third operation: 0 1 1 1 1 1 0 0 0 0

PFs: 4
Example of Clock (Second Chance)

|   | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 0 |   |   |   |   |   |   |   |   | 7 | 7 | 7 | 2 | 2 | 2 |   |   |   |   |   |   |   |
| 1 |   |   |   |   |   |   |   |   | 0 | 0 | 0 | 0 | 0 |   |   |   |   |   |   |   |   |
| 2 |   |   |   |   |   |   |   |   | 1 | 1 | 1 | 3 |   |   |   |   |   |   |   |   |   |   |
| PF? |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

PFs: 5

 Example of Clock (Second Chance)

|   | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 0 |   |   |   |   |   |   |   |   | 7 | 7 | 7 | 2 | 2 | 2 |   |   |   |   |   |   |   |
| 1 |   |   |   |   |   |   |   |   | 0 | 0 | 0 | 0 | 0 |   |   |   |   |   |   |   |   |
| 2 |   |   |   |   |   |   |   |   | 1 | 1 | 1 | 3 |   |   |   |   |   |   |   |   |   |   |
| PF? |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

PFs: 5

© Charles M Snow 7
### Example of Clock (Second Chance)

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

PFS: 6

### Example of Clock (Second Chance)

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

PFS: 14
Second Chance: Improved Version

- "clock" with 0 bits is simple FIFO
- with 1 bit is second chance we just saw
- add another bit: M for modified

- sometimes called "dirty" bit

<table>
<thead>
<tr>
<th>M</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>U→</td>
<td>not recently accessed, not modified</td>
<td>accessed recently, not modified</td>
</tr>
<tr>
<td></td>
<td>recently accessed, modified</td>
<td>recently accessed, modified</td>
</tr>
</tbody>
</table>
Second Chance: Improved Version

- enhanced replacement policy:
  - scan buffer, make no changes to bits
  - first (0,0) frame is chosen for replacement

- scan again, for (U=0,M=1) clearing each frame’s U bit to 0; pick first (0,1) frame encountered
Second Chance: Improved Version

- enhanced replacement policy:
  - scan buffer, make no changes to bits
  - first (0,0) frame is chosen for replacement
  - scan again, for (U=0,M=1) clearing each frame’s U bit to 0; pick first (0,1) frame encountered
  - scan again: look for U=M=0; if none, scan one last time for U=0, M=1: pick first encountered
- this version used in (pre X) Macintosh OS

Page Buffering

- used by VMS
- page replacement is FIFO but:
  - replaced page is added to the end of 1 of 2 lists (FIFO):
    (1) free page list if was unmodified,
    (2) modified page list if was modified
  - free page list is list of frames available to receive pages
- VMS trying to keep a small number of frames free at all times
Page Buffering

- Moving pages to lists handled by pointer manipulation: fast
- Page moved to free list is still 'resident', so page-in is fast (switch some pointers)
- Modified pages can be written out in batches: more efficient use of I/O
- Simplified version used in Mach
  - No distinction between modified/unmodified

Replacement Policies: Comparison

after results in Baer, J.L., Computer Systems Architecture, 1980
Resident Set Management

- how small or big should resident set be?

less memory allocated

more processes fit in main memory

more memory allocated

fewer processes fit in main memory

but

higher incidence of page faults per process

after certain point, more memory doesn’t noticeably help
Resident Set Management: Fixed RS

- fixed: each process given fixed number of page frames process may have resident
- determined by:
  - type of process (e.g., interactive, batch)
  - suggestion of programmer and/or sys mgr

Resident Set Management: Variable RS

- variable: number of page frames process may have resident varies during life of program
  - e.g., if have high PFR, may get more frames
  - e.g., if have low PFR, may lose frames
- more powerful but:
  - requires OS overhead to monitor PFR behaviour
Resident Set Management: Scope

- when there’s a page fault, and you need to get a frame, where do you look?
  - local: select only from frames holding pages of the process that had the page fault
Resident Set Management: Scope

- when there’s a page fault, and you need to get a frame, where do you look?
  - local: select only from frames holding pages of the process that had the page fault
  - global: select from any frame in memory (of any process)

- size and scope are related…

RS Management: Size/Scope Combos

- fixed + local:
  - can use any replacement policy already discussed
  - tricky to ‘guess’ right number of frames to allocate for good performance across all processes
RS Management: Size/Scope Combos

- fixed + local
- variable + global:
  - easiest to implement
  - OS maintains “free frame list”: upon PF, attach a free frame to process’ list of frames
    - processes that PF a lot get more frames
  - when no free frames, pick using any policy already discussed
  - may degrade overall performance
    - possible fix?

- variable + local:
  - when new proc starts, give it some frames
  - page faults fill up initial frame allocation
  - occasionally re-evaluate the allocation of frames to this process, adjust as necessary
  - an example of this scheme...
Denning’s Working Set

- working set = $W(t, \tau)$

Some virtual time, $t$, is the set of pages of the process that have been referenced in the last $\tau$ time units.

- Look at example from Stallings...

<table>
<thead>
<tr>
<th>$t$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Denning's Working Set

<table>
<thead>
<tr>
<th>A →</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>24,15</td>
<td>24,15</td>
<td>24,15</td>
<td>24,15</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Denning’s Working Set

<table>
<thead>
<tr>
<th>Δ</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>24,15</td>
<td>24,15</td>
<td>24,15</td>
<td>24,15</td>
</tr>
<tr>
<td>18</td>
<td>15,18</td>
<td>24,15,18</td>
<td>24,15,18</td>
<td>24,15,18</td>
</tr>
<tr>
<td>23</td>
<td>18,23</td>
<td>18,18,23</td>
<td>24,15,18,23</td>
<td>24,15,18,23</td>
</tr>
<tr>
<td>18</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Diagram

- **Δ →**
- **t →**
## Denning’s Working Set

<table>
<thead>
<tr>
<th>Δ →</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>24,15</td>
<td>24,15</td>
<td>24,15</td>
<td>24,15</td>
</tr>
<tr>
<td>18</td>
<td>15,18</td>
<td>24,15,18</td>
<td>24,15,18</td>
<td>24,15,18</td>
</tr>
<tr>
<td>23</td>
<td>18,23</td>
<td>15,18,23</td>
<td>24,15,18,23</td>
<td>24,15,18,23</td>
</tr>
<tr>
<td>24</td>
<td>23,24</td>
<td>18,23,24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>24,17</td>
<td>23,24,17</td>
<td>18,23,24,17</td>
<td>15,18,23,24,17</td>
</tr>
<tr>
<td>18</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t→</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>24,15</td>
<td>24,15</td>
<td>24,15</td>
<td>24,15</td>
</tr>
<tr>
<td>18</td>
<td>15,18</td>
<td>24,15,18</td>
<td>24,15,18</td>
<td>24,15,18</td>
</tr>
<tr>
<td>23</td>
<td>18,23</td>
<td>15,18,23</td>
<td>24,15,18,23</td>
<td>24,15,18,23</td>
</tr>
<tr>
<td>24</td>
<td>23,24</td>
<td>18,23,24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>17,24</td>
<td>23,24,17</td>
<td>18,23,24,17</td>
<td>15,18,23,24,17</td>
</tr>
<tr>
<td>18</td>
<td>17,18</td>
<td>24,17,18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>18,24</td>
<td></td>
<td>24,17,18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>18,17</td>
<td>24,18,17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>18,17</td>
<td>24,18,17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>17,15</td>
<td>17,15</td>
<td>18,17,15</td>
<td>24,18,17,15</td>
</tr>
<tr>
<td>24</td>
<td>15,24</td>
<td>17,15,24</td>
<td>17,15,24</td>
<td></td>
</tr>
</tbody>
</table>
Using WS

- each process uses pages in its WS
- goal of virtual memory manager:
  keep working set resident
  - so allocate/remove frames as program runs
- what to do if we run out of free frames?
Using WS

- system wide, $S_{WSS_1}$ is demand for frames
- if $>$ number of frames:
  - suspend a process using some criterion
  - make its frames available for re-allocation
- if $<$ number of frames by some threshold $\sigma$,
admit another process

WSS provides implicit load control in addition to
WS management
Implementing WS

- need timer
  - so can examine WS every t time units
- need to mark pages
  - so know if has been referenced during window

- but:
  - timer/marking impractical
  - no determinable optimum for $\Delta$
- so how do we do this in “real life”?
Really Implementing WS

- instead of watching WS, look only at rate of PFs
- what if PFR for a process is above a threshold $\theta_U$?

• process having too many PFs so needs more frames allocated to it

- what if PFR for a process is below a threshold $\theta_L$?
Really Implementing WS

- instead of watching WS, look only at rate of PFs
- what if PFR for a process is above a threshold $\theta_U$?
  - process having too many PFs so needs more frames allocated to it
- what if PFR for a process is below a threshold $\theta_L$?
  - process has more than enough frames, so can steal some for allocation elsewhere
Really Implementing WS

Page Fault Frequency: PFF

- PFF: really just the time between page faults
- need a Use bit per frame: set to 1 on each access
- whenever PF happens, log virtual time since last PF for this process
- define threshold $F$:
  - if time since last PF is $< F$, add a frame to resident set
  - if time since last PF is $> F$, discard all pages in frames where $U=0$ (reduce resident set size) and clear all $U$ bits on remaining page frames
- can use two thresholds for finer control
PFF Performance

- good in “steady-state”
- what about in locality transitions?

- characterized by sudden high demand for new pages
- PFF becomes short (i.e., PFR is high)
  - so is \( F \), so we are adding frames
  - but frames from this process are not released until PFs are far enough apart in time (PFR is low)
- so PFF is quick to give frames, slow to reclaim them.
Improving PFF

- want way to be able to get rid of pages fast when “lots is happening”
  - so need F to be smaller when “it should be”
- try variable–interval sampled working set (VSWS)
  - evaluate WS at sampling interval (elapsed virtual time)

VSWS

- at start of interval:
  - clear all use bits
- at end of interval:
  - only referenced pages have U=1: keep these
  - release all other pages
- RS size decreases only at end of an interval
- RS size can increase during interval as faulted pages are added to RS
VSWS

- uses 3 parameters:
  - M: minimum duration of sampling interval
  - L: maximum duration of sampling interval
  - Q: # of PFs allowed between successive samples
- if virtual time, t, since last sampling reaches L:
  - suspend process
  - scan use bits
- if Q PFs occur before L has elapsed:
  - if t < M, wait until M then suspend and scan use bits
  - if t ≥ M, suspend process and scan use bits

VSWS

- goal: set parameters so sampling occurs on Qth page fault after last scan
- VSWS trying to reduce peak memory demands during transients by increasing sampling frequency
  - hence rate at which unused pages drop out of WS
- used in GCOS 8 (Honeywell)
- more effective than PFF
  - as easy to implement
Cleaning Policy

- policy for how to ‘clean’ memory frames
- demand-cleaning: write frame out only when it has been selected for replacement
- pre-cleaning: write out frame before needed for replacement
  - can batch write several frames
- which is better? downsides?

Cleaning Policy

- precleaning:
  - ✔ efficient I/O by use of batch writing
  - ✔ leaves unmodified frames, faster to replace
  - ✗ modified frame may be modified again before needed for replacement
  - ✗ need more writes, can clog disk ↔ memory bandwidth
- demand cleaning:
  - ✗ a process may wait for 2 disk xfers before resuming
- compromise?
**Practical Cleaning Technique**

- use pre-cleaning with page buffering
  - writes out only modified pages from list
  - moves them to unmodified list
  - subsequent modification moves it back to modified list

**Load Control**

- virtual memory lets us increase multiprogramming
Load Control

- virtual memory lets us increase multiprogramming
- “Too much of a good thing…” isn’t good
  - [J.T. Kirk, commenting on tribbles]
- if multiprogramming level too low: CPU is underutilized
  - why might this be?

- if multiprogramming level too low: CPU is underutilized
  - high probability of all processes being blocked
  - e.g., try "top" on UNIX/Linux: see what CPU idle % is
- if multiprogramming level too high: CPU is
Load Control

- if multiprogramming level too low: CPU is underutilized
  - high probability of all processes being blocked
  - e.g., try "top" on UNIX/linux: see what CPU idle % is

- if multiprogramming level too high: CPU is underutilized
  - each process has small RSS
  - so PFR goes up

Load Control

- high PFR ⇒ processes blocked waiting for pages
  ⇒ idle time ⇒ CPU underutilized

- but wait: it can get worse!
Thrashing

Thrashing: point where most (eventually, all) of CPU time is spent waiting for and handling page faults, none for actual processing of jobs.

Multiprogramming without Thrashing

1. PFF- or WS-based techniques implicitly include load control
   - by controlling size of RS
2. Denning’s “L=S” criterion:
   - adjust multiprogramming level so that:
     - mean time between faults ≈ mean time to service faults
   - this is point of highest CPU utilization
3. "50% Rule": keep use of paging mechanism at 50%
   - again, point at which CPU utilization highest
Multiprogramming without Thrashing

(4) derived from Clock algorithm: monitor how fast cycle through list (pointer circles ring): if rate is low then either
   i. few PFs happening (few requests to advance ptr)
   ii. on each request, average number of frames scanned is small ⇒ many unreferenced resident frames
      either way: can increase multiprogramming

But, if rate is high then either
   i. high PFR
   ii. hard to find replaceable frames
      then reduce multiprogramming

Reducing Multiprogramming

- how reduce multiprogramming?
  - get rid of, i.e., suspend, some programs
- how to choose...
Reducing Multiprogramming

1. lowest priority process
2. faulting process
   - faulting because doesn’t have RS resident
   - so suspending it causes little impact
3. last process activated
   - is process least likely to have WS resident
4. process with smallest RS
   - easiest to reload later
   - unfair to programs with very good locality
5. largest process:
   - frees up largest # frames with least effort

Secondary Memory

- made necessary by volatility of primary memory
- made software “portable”
- early systems (as still used today) based on magnetic recording
  - on drums
  - on tape
  - on disks
- we’ll look mostly at disks
  - still mostly magnetic recording technology
  - optical coming into widespread use, but slower
Magnetic Recording

- “ferrous” particles randomly distributed over recording medium (tape, disk)

- in presence of electromagnetic field, can rearrange the particles into an orderly pattern
Magnetic Recording

- “ferrous” particles randomly distributed over recording medium (tape, disk)
- in presence of electromagnetic field, can rearrange the particles into an orderly pattern
- imposing that magnetic field on the medium is done by a head

in tape recorder, medium (tape) moves past stationary Read/Write head

Magnetic Disks

- have ≥ 1 disk coated with magnetic material
  - may be made of mylar: floppy or flexible disk
  - may be made of aluminum: hard or rigid disk (called platter)
- disk rotates on center axis
- head is fixed onto arm that moves radially across surface of disk
- with floppy, head is in contact with medium
  - like with tape recorder
- with hard disk, head never contacts medium
  - floats above it on air cushion (e.g., 30 µm)
Magnetic Disks

- Disk rotates at, e.g., 3600 RPM
  (1 rev takes 16.667 ms)

- Move arm to first radial position over surface
  ‘Write’ a signal onto disk at 2 ms intervals
Magnetic Disks

- Move arm to 2\textsuperscript{nd}, 3\textsuperscript{rd}, etc., radial position over surface
- ‘Write’ a signal onto disk at 2 ms intervals

- Join ‘dots’ axially to make rings called \textit{tracks}
Magnetic Disks

Each sector holds (typically) 512 bytes of data.
May have one “runt” sector at end of track.
Track/sector layout handled by “hard format.”
- Drive usually hard formatted at factory.
Some typical numbers:
- Tracks: 80 on floppy disk, 9732 on hard disk.
- Sectors per track: 63 on hard disk.
**Disk Layout**

- hard disks can have multiple platters per spindle
  - and use both top and bottom surfaces of each platter
- track j on top surface of platter i is also track j on bottom surface of platter i
  - same for all n platters because all arms move together
- think of projecting (geometrically) all tracks j: you’d have a cylinder
- CHS=6296/16/63, 512 bytes per sector ⇒ 3,249,340,416 bytes raw capacity

**Using Disks**

- what are timing considerations?
- data rate: how fast do bits move by the head?
  - 3600 RPM, 60 sectors/track ⇒ ≈ 1.75 Mbytes/sec
  - drives typically have buffer memory to smooth-out data rate
  - have different data xfer rates:

<table>
<thead>
<tr>
<th></th>
<th>Peak medium xfer</th>
<th>Sustained read</th>
<th>Sustained write</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>54.2</td>
<td>44.2</td>
<td>24.2</td>
</tr>
</tbody>
</table>

- but head has to be in right place...
Using Disks

- data rate: how fast do bits move by the head?
- moving the head called a **seek**:
  - from one track to an adjacent track, typically 5 ms
  - from one edge of the disk to the other (max. seek distance) typically 20 ms
- usually work with **average seek time** as time to seek half-way across the disk
  - typically 8.5 ms
- and disk has to be in right place...

Using Disks

- data rate: how fast do bits move by the head?
- moving the head called a **seek**
- once head is “on station” have to wait for **rotational delay**: get the correct sector to where head is
  - determined by rotation speed
  - at 3600 RPM full rotation is 16.67 ms
  - average rotational delay is half a rotation: 8.33 ms
Using Disks

- so average latency to access data is
  - average seek time + average rotational delay
- these delays totally overwhelm data transfer delay times
- anything we can do to mitigate these delays goes long way toward improving efficiency...

Optical Disks

- delays for CDROM/CDRW optical disks are same as for magnetic disks, but worse:
  - rotation delay: disk spins more slowly and
  - change rotation motor speed: varies from 400 to 200 RPM (nominally, for audio CD)
  - seeks are slower (typ. average 100 – 120 ms)
  - another possible delay: focus time once head “on station”
- and, for DVD:
  - layer change time