Operating Systems

Virtual Memory

Virtual Memory

Topics
1. Memory Hierarchy
2. Why Virtual Memory
3. Virtual Memory Issues
4. Virtual Memory Solutions
5. Locality of Reference
6. Virtual Memory with Segmentation
7. Page Memory Architecture

Topics (continued)
8. Algorithm for Virtual Paged Memory
9. Paging Example
10. Replacement Algorithms
11. Working Set Characteristics
Why Virtual Memory?

Users frequently have a speed vs. cost decision.

Users want:
1. To extend fast but inexpensive memory with slower but cheaper memory.
2. To have fast access.

Virtual Memory Issues

1. Fetch Strategy - Retrieval from secondary storage
   (a) Demand - on an as needed basis
   (b) Anticipatory - aka Prefetching or Prepaging

2. Placement Strategies - Where does the data go?
Virtual Memory Issues (continued)

3. Replacement Strategies - Which page or segment to load and which one gets swapped out?

We seek to avoid overly frequent replacement (called thrashing).

Virtual Memory Solutions

Virtual memory solutions employ:

1. Segmentation
2. Paging
3. Combined Paging and Segmentation
   - Current trend.

Locality of Reference

Locality of reference by observing that process that process tend to reference storage in nonuniform, highly localized patterns. Locality can be:

1. Temporal - Looping, subroutines, stacks, counter variables.
2. Spatial - Array traversals, sequential code, data encapsulation.
Locality of Reference (continued)

Things that hurt locality:

1. Frequently taken conditional branches
2. Linked data structures
3. Random Access patterns in storage

Virtual Memory with Segmentation
Only

Segmentation provides protection and relocation so:

1. The operating system can swap segments out not in use.
2. Data objects can be assigned their own data segments they do not fit in an existing data segment.
3. Sharing can be facilitated using the protection mechanism.

Virtual Memory with Segmentation
Only (continued)

4. No internal fragmentation.
5. Main memory has external fragmentation.

Pure segmented solutions are not currently in fashion. These techniques combine well with paging (as shown later)
Virtual Paged Memory Architecture

- Paging is frequently used to increase the logical memory space.
- Page frames (or frames for short) are the unit of placement/replacement.
- A page which has been updated since it was loaded is dirty, otherwise it is clean.
- A page frame may be locked in memory (not replaceable). (e.g. O/S Kernel)

Virtual Paged Memory Architecture (continued)

Some architectural challenges include:
1. Protection kept for each page.
2. Dirty/clean/lock status kept for each page.
3. Instructions may span pages.
4. An instructions operands may span pages.
5. Iterative instructions (e.g Intel 80x86) may have data spanning many pages.

The Algorithm for Virtual Paged Memory

For each reference do the following:

1. If the page is resident, use it.
2. If there is an available page, allocate it and load the required nonresident page.
The Algorithm for Virtual Paged Memory (continued)

3. If there is no available frames, then:
   (a) Select a page to be removed (the victim).
   (b) If the victim is dirty, write to secondary storage.
   (c) Load the nonresident page into the victim’s frame.

Stallings [3] uses the term page fault to mean replacement operations. Many others (including your instructor) use page fault to refer to loading any pages (not just replacement).

Demand Page Replacement

Let $M_t$ be the set of resident pages at time $t$. For a program that runs $T$ steps the memory state is:

$M_0, M_1, M_2, \ldots, M_T$

Memory is usually empty when starting a process, so $M_0 = \emptyset$. 

Demand Page Replacement (continued)

Assume that real memory has \( m \) page frames, so \(|M_t| \leq m\). Let \( X_t \) be the set of newly loaded pages and \( Y_t \) be the set of newly replaced pages at time \( t \). Let \( y \in M_t \) be some page in \( M_t \) (if nonempty).

Demand Page Replacement (continued)

Each reference updates memory:\n\[
M_t = M_{t-1} \cup X_t - Y_t
\]
\[
= \begin{cases} 
  M_{t-1} & \text{if } r_t \in M_{t-1} \\
  M_{t-1} + r_t & \text{if } r_t \in M_{t-1} \land |M_{t-1}| < m \\
  M_{t-1} + r_t + y & \text{if } r_t \in M_{t-1} \land |M_{t-1}| = m
\end{cases}
\]

Prefetching has had limited success in practice.

Page Fault Rate and Locality

The reference string, denoted \( w \), is the sequence of page frames referenced by a process. \( w \) is indexed by time:
\[
w = r_1, r_2, r_3, \ldots, r_{|w|} \quad (1)
\]
Cost Measures For Paging

For each level of hierarchy, a page is either:

1. is resident with the probability 1 - p and can be referenced with cost 1
2. is not resident with probability p and must be loaded with a cost $F$

Cost Measures For Paging Cont’d.

The *Effective Access Time* (EAT) for a process is the average time to access a single memory location:

$$EAT = 1 + pF$$  \hspace{1cm} (2)

Cost Measures For Paging (continued)

The *Duty Factor* of a process measures the efficiency of the processor use by the process given its page fault pattern:

$$DF = \frac{|w|}{|w|(1+pF)} = \frac{1}{1+pF} = \frac{1}{EAT}$$  \hspace{1cm} (3)
A Paging Example

Consider the following example [?]:

\[
\text{STEPSIZE} = 1; \\
\text{for (} i = 1; i <= n; i = i + \text{STEPSIZE} \text{) } \\
A[i] = B[i] + C[i]; \\
\]

A Paging Example (continued)

With the pseudo assembly code:

\[
4000 \text{ MOV STEPSIZE, 1 } \quad \# \text{ STEPSIZE = 1} \\
4001 \text{ MOV R1, STEPSIZE } \quad \# i = 1 \\
4002 \text{ MOV R2, n } \quad \# R2 = n \\
4003 \text{ CMP R1, R2 } \quad \# \text{ test } i > n \\
4004 \text{ BGT 4009 } \quad \# \text{ exit loop} \\
4005 \text{ MOV R3, B(R1) } \quad \# R3 = B[i] \\
4006 \text{ ADD R3, C(R1) } \quad \# R3 = B[i] + C[i] \\
4007 \text{ ADD R1, STEPSIZE } \quad \# \text{ increment } R1 \\
4008 \text{ JMP 4002 } \quad \# \text{ back to the test} \\
4009 \ldots \quad \# \text{ after the loop} \\
\]

The Example (continued)

<table>
<thead>
<tr>
<th>Storage Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6000 - 6FFF</td>
<td>Storage for A</td>
</tr>
<tr>
<td>7000 - 7FFF</td>
<td>Storage for B</td>
</tr>
<tr>
<td>8000 - 8FFF</td>
<td>Storage for C</td>
</tr>
<tr>
<td>9000</td>
<td>Storage for n</td>
</tr>
<tr>
<td>9901</td>
<td>Storage for STEPSIZE</td>
</tr>
</tbody>
</table>

Table 1: Reference Locations
The Example (continued)

In this example:

\[ w = 494944 \times (4748649444)^n \] (4)

Locality in Paged Memory

If we have at least 5 frames of 64KB then we know that the program does 5 page loads and no swaps. Otherwise, how many page faults are there?

Locality in Paged Memory

Access Patterns in: \[ A[i] = B[i] + C[i] \] for \( i = 1 \to 100 \)

Page Trace of Example for \( n=100 \)
FIFO Replacement

Suppose that we try FIFO replacement (the oldest page gets replaced). Consider the reference string [1]:
\[ w = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 \] (5)

Derive the number of replacements done when:
- 3 frames are used
- 4 frames are used

FIFO Replacement with 3 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>
</tr>
</thead>
<tbody>
<tr>
<td>r_i</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Page 1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Page 2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page 3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fault?</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>Faults</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FIFO Replacement with 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>
</tr>
</thead>
<tbody>
<tr>
<td>r_i</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Page 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page 2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page 3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page 4</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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fault?</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>Faults</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Belady’s (aka FIFO) Anomaly

Recall that when we used a FIFO replacement scheme with 3 frames we got 9 faults and for 4 frames we got 10 faults. Usually, one would expect fewer faults for a larger number of frames. That did not occur, what happened? Belady discovered this (counter intuitive) problem.

The Optimal (OPT) Replacement Algorithm

The optimal (OPT) is to select the victim such that it is the page that will be referenced the longest time into the future.

Note that on the following table at $t=9$, Page 1 could have been the victim too.
Stack Page Replacement Algorithms

Let $M(m,w)$ represent the set of pages in real memory after processing reference string $w$ in $m$ frames.
The inclusion property is satisfied if for a given page replacement algorithm:
$$M(m,w) \subseteq M(w, m+1)$$
Stack replacement algorithms [?] satisfy the inclusion property

Stack Page Replacement Algorithms (continued)

e.g. FIFO is NOT a stack page replacement algorithm (inclusion is NOT satisfied), while OPT is.

Least Recently Used (LRU)

LRU is a popular stack page replacement strategy. Pages are ordered by time of last access, with the page used furthest into the past being removed.
Clock Replacement Algorithms

Clock replacement algorithms are stack algorithms. The simplest style marks pages with \(use=1\) when referenced and \(use=0\) upon a circular scan.
Simple Clock Replacement

The first page encountered with use=0 is selected. The final position of the previous scan is used to start the next scan.

Simple Clock (CLOCK)

CLOCK is a popular stack page replacement strategy, pages in use are underlined.
Gold’s Clock Algorithm

Another popular type of clock algorithm exploits the fact that dirty pages require an additional write, and as such make poor victims.

This algorithm tracks keeps track of recent modification, with $m=1$ if updated, otherwise $m=0$, use bits indicate read accesses and are denoted $u=1$ for recent read otherwise $u=0$.

Gold’s Clock Algorithm (continued)

The steps are:
1. Scan for a page with $u=0$, $m=0$. If one is found, stop otherwise after all pages have been scanned, continue.
2. Scan for a page with $u=0$, $m=1$, setting $u=0$ as the scanning. If one is found stop, otherwise continue.
3. Repeat step 1, all pages had $u=1$ before now $u=0$ so step 1 or step 2 will be satisfied.
   (as per CLOCK) and then if that fails, a mod bit scan is done.
Working Set (WS)

Working set algorithms are stack algorithms using a parameter $\Delta[3, 1]$. All pages within the last $\Delta$ reference remain resident. The number of pages allocated varies over time (upper bound is $\Delta$).

Working Set (continued)

The pages referenced by the process during this time interval constitute the process's working set $W(t, w)$.
Working Set Characteristics

Working sets can be characterized as being stable most of the time, and growing larger as program makes transitions between locals of reference. Choosing $\Delta$ or $\omega$ is hard.

Working Set Management

Since each process has dynamically sized working set, and the window, $\Delta$, is too loose an upper bound on working set size, the OS may have to select a process to deactivate.
Working Set Management

Selection Criteria:
1. The lowest priority process
2. The process with the largest fault rate
3. The process with the largest working set.
4. The process with the smallest working set.
5. The most recently activated process
6. The largest remaining quantum.

Lifetime Curves for Memory Access

Similar to inter I/O event times, there are inter page fault times. The inter page fault time is called the lifetime of the page. Lifetime curves are plotted as a function of number of pages of memory allocated \( (m) \) and typically have a “knee” (performance region) where \( L(m)/m \) is maximal (a good value of \( m \)).

Lifetime Curves (continued)

Knee

\[ L(m) \]
Page Fault Frequency (PFF)

Page fault frequency measures the time since the last page and tracks whether each page allocated to a process (via a use bit) has been accessed since the last page fault.

This information is used, processes with high fault rates are allocated more pages, while jobs with lower fault rates release pages.