Objectives

- To describe the benefits of a virtual memory system
- To explain the concepts of:
  - demand paging
  - page-replacement algorithms
  - allocation of page frames
- To discuss the principle of the working-set model
Chapter Outline

- Background
- Demand Paging
- Copy-on-Write
- Page Replacement
- Allocation of Frames
- Thrashing
- Background

**Virtual memory** - separation of user logical memory from physical memory.

- Only part of the program needs to be in memory for execution
- Logical address space can therefore be much larger than physical address space
- Allows address spaces to be shared by several processes
- Allows for more efficient process creation

**Virtual memory can be implemented via:**

- Demand paging
- Demand segmentation (Skip)
Virtual Memory That is Larger Than Physical Memory
- Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users

- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

- Lazy swapper – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a pager
Transfer of a Paged Memory to Contiguous Disk Space

Diagram showing the transfer of memory pages between a program's main memory and contiguous disk space.

Program A

Program B

Main Memory

Swap out

Swap in

Disk space with pages 0 to 23.
Page Table When Some Pages Are Not in Main Memory
**-- Page Fault**

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = \( \text{v} \)
6. Restart the instruction that caused the page fault
-- Steps in Handling a Page Fault

1. Load M
2. Trap
3. Page is on backing store
4. Bring in missing page
5. Reset page table
6. Restart instruction
-- Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- Effective Access Time (EAT)
  
  $$EAT = (1 - p) \times \text{memory access} + p (\text{page fault overhead} + \text{swap page out} + \text{swap page in} + \text{restart overhead})$$
-- Demand Paging Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- \[ EAT = (1 - p) \times 200 + p \times 8,000,000 \] = 200 + 7,999,800 \times p

If one access out of 1,000 causes a page fault, then
- \[ EAT = 8.2 \text{ microseconds} \]
  This is a slowdown by a factor of 40!!
What happens if there is no free frame?

- Find a page in memory, but not really in use, and swap it out.

For better performance:

- Find an algorithm which will result in minimum number of page faults.
- Prevent over-allocation of memory.
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk.
-- Basic Page Replacement

1. Find the location of the desired page on disk

2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a victim frame

3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Restart the process
-- Page Replacement

1. Swap out victim page
2. Change to invalid
3. Swap desired page in
4. Reset page table for new page

frame valid-invalid bit
page table

physical memory

victim
Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- In all our examples, the reference string is

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
Page replacement Algorithms

- First-In-First-Out (FIFO)
- Optimal
- Least Recently Used (LRU)
-- First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

- 3 frames (3 pages can be in memory at a time per process)

| 1 | 1 | 5 | 4 |

- 4 frames

<table>
<thead>
<tr>
<th>2</th>
<th>2</th>
<th>1</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

- Belady’s Anomaly: more frames ⇒ more page faults
### FIFO Page Replacement

<table>
<thead>
<tr>
<th>reference string</th>
<th>page frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1</td>
<td>7 7 7 2 2 2 4 4 4 0 0 0 7 7 7</td>
</tr>
<tr>
<td></td>
<td>0 0 0 3 3 2 2 2 1 1 1 1 0 0 2 2 1</td>
</tr>
</tbody>
</table>
--- FIFO Illustrating Belady’s Anomaly

![Diagram showing number of page faults vs. number of frames](image-url)
-- Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
  
  | 1 | 4 |
  | 2 |
  | 3 |
  | 4 | 5 |

  6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs
- Optimal Page Replacement

reference string

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

page frames

<table>
<thead>
<tr>
<th>7</th>
<th>7</th>
<th>7</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
-- Least Recently Used (LRU) Algorithm ...

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- Counter implementation
  - Every page entry has a counter; every time a page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change
--- LRU Page Replacement

<table>
<thead>
<tr>
<th>Reference string</th>
<th>Page Frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1</td>
<td></td>
</tr>
<tr>
<td>7 7 7 2 2 4 4 4 0 1 1 1</td>
<td>0 0 0 0 0 3 3 3 3 0 0 0</td>
</tr>
<tr>
<td>1 1 3 3 2 2 2 2 2 2 2 7</td>
<td></td>
</tr>
</tbody>
</table>
-- LRU Algorithm Implementation

- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement
Use Of A Stack to Record The Most Recent Page References

reference string

<table>
<thead>
<tr>
<th>4</th>
<th>7</th>
<th>0</th>
<th>7</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>7</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
</table>

stack before a

2
1
0
7
4

stack after b

7
2
1
0
4

a
b
- Allocation of Frames

- Each process needs *minimum* number of pages
  - Example: IBM 370 – 6 pages to handle SS MOVE instruction:
    - instruction is 6 bytes, might span 2 pages
    - 2 pages to handle *from*
    - 2 pages to handle *to*

- Two major allocation schemes
  - fixed allocation
    - Equal allocation
    - Proportional allocation
  - priority allocation
--- Fixed Allocation

- Equal allocation – For example, if there are 100 frames and 5 processes, give each process 20 frames.

- Proportional allocation – Allocate according to the size of process
  
  - \( s_i = \text{size of process } p_i \)
  
  - \( S = \sum s_i \)
  
  - \( m = \text{total number of frames} \)
  
  - \( a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m \)

\[
\begin{align*}
m & = 64 \\
s_1 & = 10 \\
s_2 & = 127 \\
a_1 & = \frac{10}{137} \times 64 \approx 5 \\
a_2 & = \frac{127}{137} \times 64 \approx 59
\end{align*}
\]
- Priority Allocation

- Use a proportional allocation scheme using priorities rather than size.

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

- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another

- **Local replacement** – each process selects from only its own set of allocated frames
- Thrashing ...

If a process does not have “enough” pages, the page-fault rate is very high. This leads to:

- low CPU utilization
- operating system thinks that it needs to increase the degree of multiprogramming
- another process added to the system

Thrashing ≡ a process is busy swapping pages in and out
... - Thrashing

![Graph showing CPU utilization vs. degree of multiprogramming]

- Thrashing
**Demand Paging and Thrashing**

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

- Why does thrashing occur?
  \[
  \sum \text{size of locality} > \text{total memory size}
  \]
-- Locality In A Memory-Reference Pattern
-- Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  
  Example: 10,000 memory references

- $WSS_i$ (working set of Process $P_i$) = total number of pages referenced in the most recent $\Delta$ memory references (varies in time)
  
  - if $\Delta$ too small will not encompass entire locality
  - if $\Delta$ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program

- $D = \sum WSS_i \equiv$ total demand frames
  
  if $D > m \Rightarrow$ Thrashing

- Policy if $D > m$, then suspend one of the processes
-- Working-set model

page reference table

\[ \ldots 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 3 4 3 4 4 4 4 1 3 2 3 4 4 4 3 4 4 4 \ldots \]

\[ \Delta \]

\[ t_1 \]

WS\((t_1) = \{1, 2, 5, 6, 7\} \]

WS\((t_2) = \{3, 4\} \]

\[ \Delta \]

\[ t_2 \]
-- Keeping Track of the Working Set

- Approximate with interval timer + a reference bit

- Example: $\Delta = 10,000$
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0
  - If one of the bits in memory $= 1 \Rightarrow$ page in working set

- Why is this not completely accurate?

- Improvement $= 10$ bits and interrupt every 1000 time units
- Page-Fault Frequency Scheme

- Establish “acceptable” page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame
End of Chapter 9