Chapter 9. **Virtual Memory**

- Background
- Demand Paging
- Page Replacement

1. Background

- In Paging, Segmentation and Hybrid, the whole sections of a program is assumed to be loaded, but …
- **Virtual memory**
  - Separation of user logical memory from physical memory
  - Only part of the program needs to be in memory for execution.
- Advantages
  - Logical address space can therefore be much larger than physical address space.
  - More processes could be supported.
  - 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**

Virtual Memory That is Larger Than Physical Memory

- See Figure 9.1.

Virtual-address Space

- See Figure 9.2.
  - ☺ *What is stack?*
  - ☺ *What is heap?*

Virtual Memory has Many Uses

- It can enable processes to share memory.

Shared Library Using Virtual Memory

- See Figure 9.3.

2. 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 $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory

Valid-Invalid Bit

- With each page table entry a **valid–invalid bit** is associated. (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries.
- Example of a page table snapshot:

Frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

page table

- During address translation, if valid–invalid bit in page table entry is 0 ⇒ **page fault.**
- ☺ *Then?*

Page Table When Some Pages Are Not in Main Memory

- See Figure 9.5.
  - ☺ *What if A in logical memory is accessed?*
  - ☺ *What if D in logical memory is accessed?*

Page Fault

- When and how do we bring a page into the memory?

- If there is ever a reference to a page, first reference to the page will trap to OS
- ⇒ i.e., **page fault trap** by the paging hardware when it notices that the invalid bit is set.
- OS looks at the table to decide:
  - Invalid reference ⇒ abort
  - Just not in memory
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.
- Restart instruction.
- ☺ *How???*

Steps in Handling a Page Fault

- See Figure 9.6.

What happens if there is no free frame?

- Page replacement:
  - Find some page in memory, but not really in use, swap it out.
  - Algorithm
  - Performance – want an algorithm which will result in minimum number of page faults.
  - Same page may be brought into memory several times.

Performance of Demand Paging

- Page fault rate $0 \le p \le 1.0$
  - If $p = 0$, no page faults
  - If $p = 1$, every reference is a fault
- Effective Access Time (EAT)

$$
\begin{aligned}
\text{EAT} = (1 - p) \; * \; & \text{memory access} \\
+ p \; * \; & (\text{ page fault overhead} \\
& + [\text{ swap page out }] \\
& + \text{ swap page in} \\
& + \text{ restart overhead })
\end{aligned}
$$

Demand Paging Example

- Memory access time = 1 microsecond

- An average page-fault service time = 8 msec

$$EAT = (1 - p) * 1 + p * (8000)$$
$$= 1 + p * 7999 \qquad \text{(in } \mu\text{sec)}$$

- **=> P should be very small**

4 Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

Need For Page Replacement

- See Figure 9.9.
  - ☺ *What if B in logical memory for user 2 is accessed?*

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. Read the desired page into the (newly) free frame. Update the page and frame tables.

4. Restart the process.

Page Replacement

- See Figure 9.10.

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

First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames

1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- See Figure 9.12.

Optimal Algorithm

- Replace page that will not be used for longest period of time.
- 4 frames example

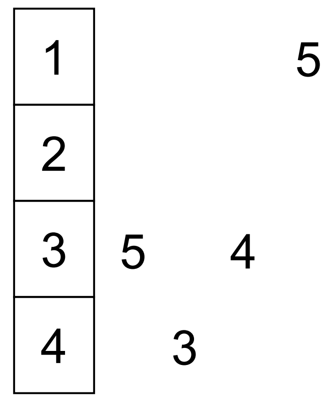
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1		4	
2			6 page faults
3			
4	5		

- ☺ *How do you know this?*
- Used for measuring how well your algorithm performs.
- See Figure 9.14.

Least Recently Used (LRU) Algorithm

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



- Counter implementation
  - Every page entry has a counter; every time 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.
  - Very expensive operation
- See Figure 9.15.