Virtual Memory (2)

Dr. Jun Zheng

CSE325 Principles of Operating Systems

11/6/2019

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)

$$
\begin{aligned}
EAT = (1 - p) &\times \text{memory access} \\
&+ p\,(\text{page fault overhead} \\
&\quad + \text{swap page out} \\
&\quad + \text{swap page in} \\
&\quad + \text{restart overhead})
\end{aligned}
$$

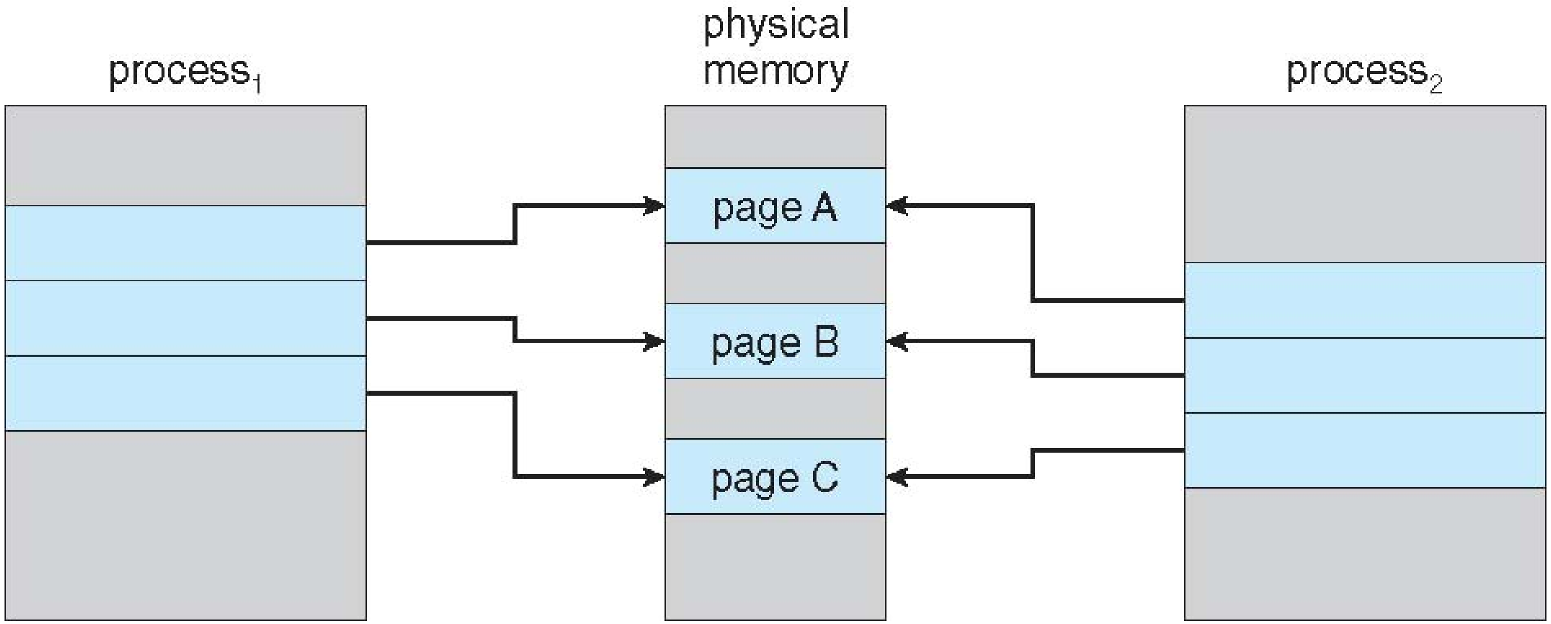
Demand Paging Example

- Memory access time = 200ns
- Average page fault service time = 8ms
- `EAT = (1-p)*200 + p*8,000,000 = 200 + p*7,999,800`
- If one access out of 1000 causes a page fault, then `EAT = 8.2us`
  - A slowdown by a factor of 40

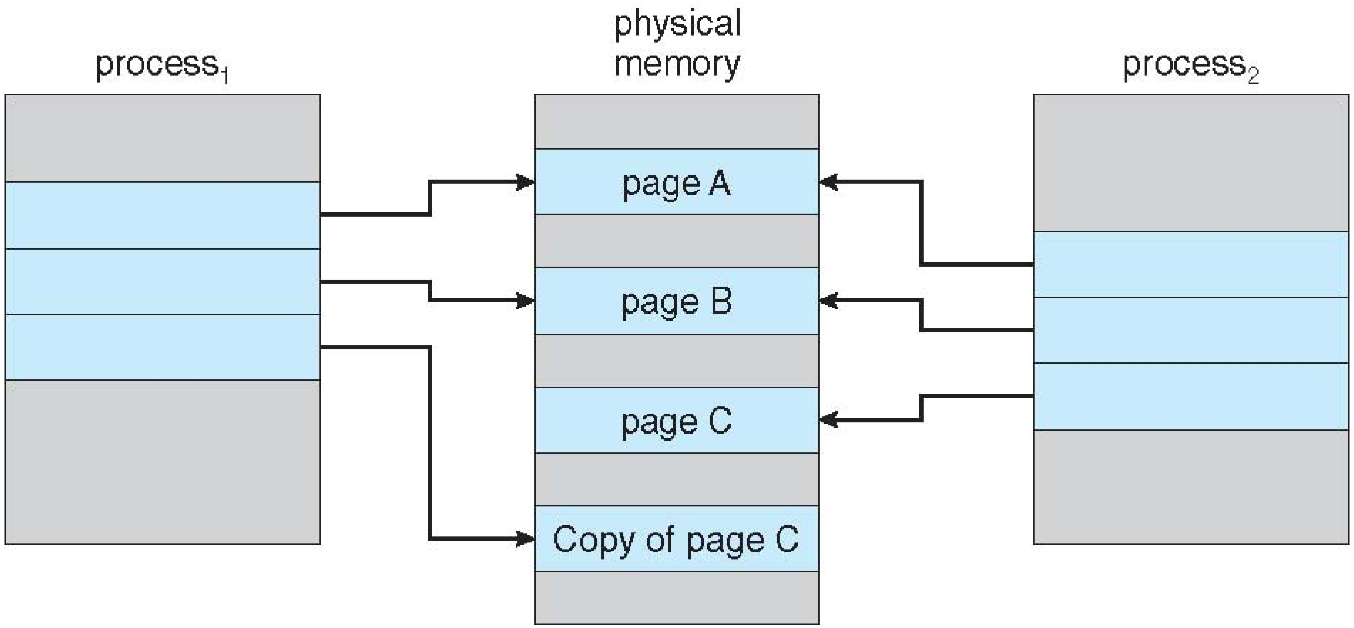
Copy-on-Write

- **Copy-on-Write** (COW) allows both parent and child processes to initially ***share*** the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- `vfork()` variation on `fork()` system call has parent suspend and child using copy-on-write address space of parent
  - Designed to have child call `exec()`
  - Very efficient

Before Process 1 Modifies Page C



After Process 1 Modifies Page C



What Happens if There is no Free Frame?

- Used up by process pages
- Also in demand from the kernel, I/O buffers, etc
- How much to allocate to each?
- Page replacement – find some page in memory, but not really in use, page it out
  - Performance – want an algorithm which will result in minimum number of page faults

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

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

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**
   - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

Page Replacement Algorithms

- Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In the examples if not specified, the **reference string** of referenced page numbers is

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

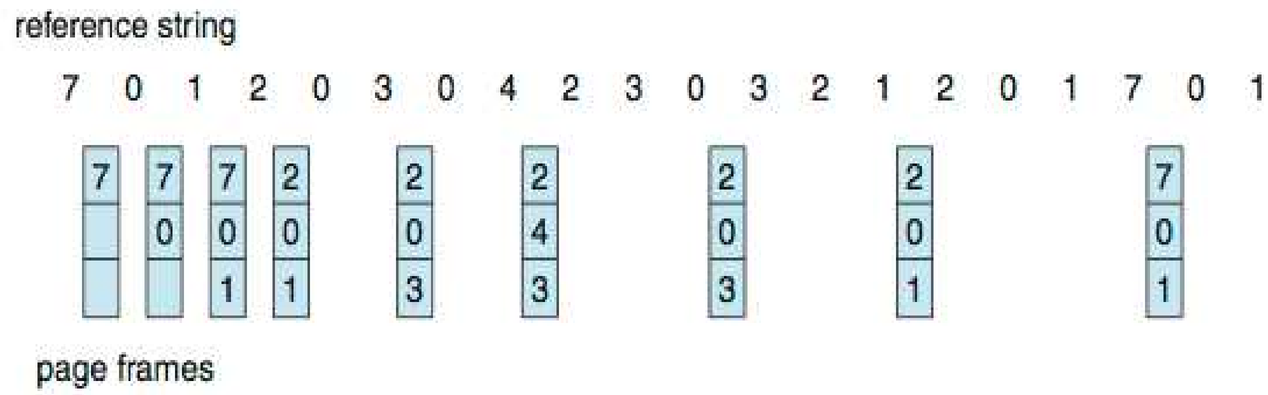
NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Random Replacement

- Replaced page is chosen from *m* loaded frames with probability $1/m$
- Easy to implement but does not perform well

Optimal Algorithm

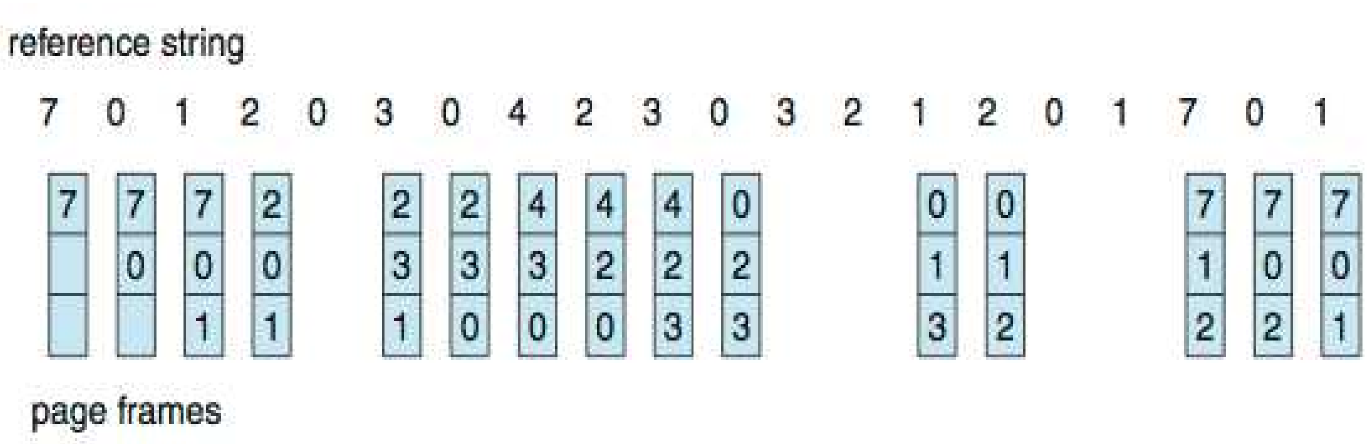
- Replace page that will not be used for longest period of time



- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs

First-In-First-Out (FIFO) Algorithm

- ❑ Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- ❑ 3 frames (3 pages can be in memory at a time per process)



15 page faults

- ❑ What if we have 4 frames?
- ❑ Consider reference string: 1 2 3 4 1 2 5 1 2 3 4 5

Belady’s Anomaly

- Adding more frames can cause more page faults!