BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

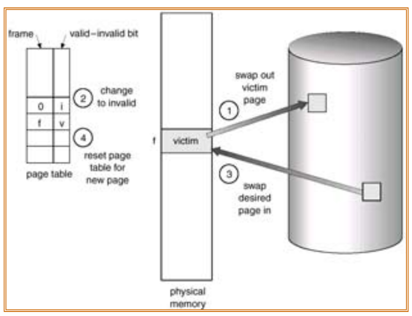
Page Replacement
-- Part 1 of 2

Notice: The slides for this lecture have been largely based on those accompanying an earlier edition of the course text *Operating Systems Concepts, 8th ed.*, by Silberschatz, Galvin, and Gagne. Many, if not all, of the illustrations contained in this presentation come from this source. Revised by X.M. Based on Professor Perrone's notes.

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

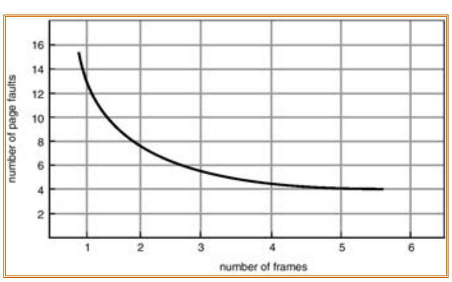


Page Replacement Algorithms

- **Goal:** Produce a low 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.
- The reference string is produced by tracing a real program or by some stochastic model. We look at every address produced and strip off the page offset, leaving only the page number. For instance:

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

Graph of Page Faults Versus The Number of Frames



FIFO Page Replacement

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

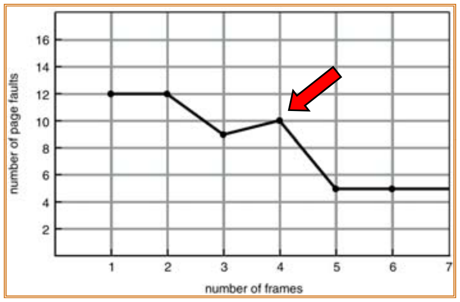
1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames

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

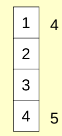
- FIFO Replacement ⇒ **Belady's Anomaly:** more frames, *more* page faults.

FIFO (Belady's Anomaly)



Optimal Algorithm

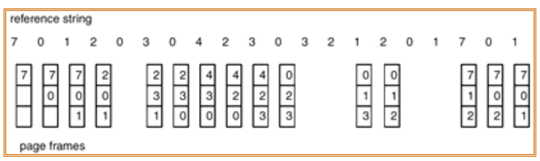
- Replace the page that will not be used for longest period of time.
- 4 frames example: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**



6 page faults

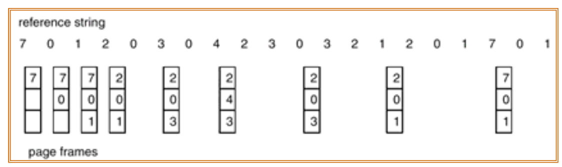
- Used for measuring how well your algorithm performs.
- How can you know what the future references will be?

Another FIFO Page Replacement Example



FIFO: 15 page faults

Optimal Page Replacement



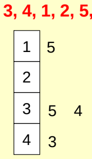
Optimal: 9 page faults

Optimal not Practical!

- Optimal page replace algorithm works great, except it is not practical!
  - Compare to optimal CPU scheduling algorithm (Shortest-Remaining-Time-First)
- We will try to approximate the optimal algorithm
  - In CPU scheduling, we try to predict the next CPU burst length and use it to approximate the SJF
- In page replacement, we use LRU (Least Recently Used) to approximate the optimal algorithm

LRU Algorithm

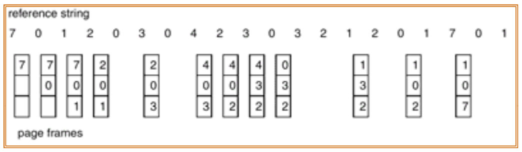
- Reference string: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**



Optimal: 6 page faults
LRU: 8 page faults

- It works great!
- But, how do we implement the LRU algorithm? (more later.)

LRU Page Replacement



Optimal: 9 page faults
LRU: 12 page faults

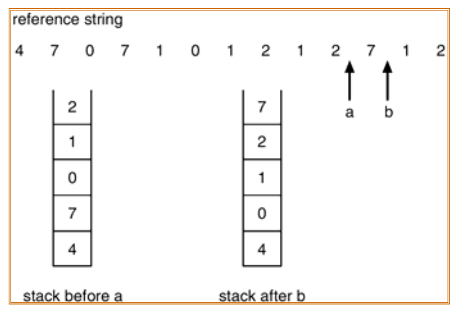
LRU Algorithm (Cont.)

- 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.

LRU and Belady's Anomaly

- LRU does not suffer from Belady's Anomaly (OPT doesn't either).
- It has been shown that algorithms in a class called **stack algorithms** can never exhibit Belady's Anomaly.
- A **stack algorithm** is one for which the set of pages in memory for *n* frames is a subset of the pages that could be in memory for *n+1* frames.

Use Of A Stack to Record The Most Recent Page References



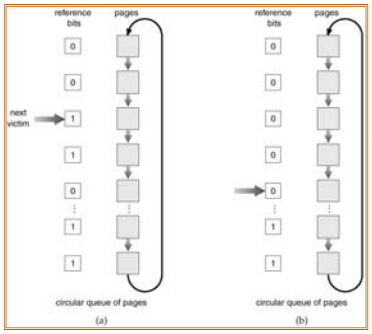
LRU Approximation Algorithms

- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1.
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- **Additional reference bits (e.g., 8 bits)**
  - Every time a page is referenced
    - Shift the reference bits to the right by 1
    - Place the reference bit (1 if being visited, 0 otherwise) into the high order bit of the reference bits
    - The page with the lowest reference bits value is the one that is Least Recently Used, thus to be replaced
  - E.g., the page with ref bits 11000100 is more recently used than the page with ref bits 01110111

LRU Approximation Algorithms

- **Second Chance**
  - If we consider the number of reference history bits to be zero, only using the reference bit itself, we have the *Second Chance* (a.k.a. *Clock*) algorithm
  - Need a pointer (clock handle) to point the next victim.
  - At each clock interruption, we check the reference bit for the victim.
  - If the victim page has reference bit = 1, then:
    - set reference bit 0.
    - leave this page in memory.
  - Else if the page reference bit is 0, this page can be replaced.

Second-Chance (Clock) Page-Replacement Algorithm



Counting Algorithms

- Keep a counter of the number of references that have been made to each page.
- **LFU Algorithm:** replaces page with smallest count.
- **MFU Algorithm:** based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Allocation of Frames

- Each process needs a **minimum** number of pages.
- There are two major allocation schemes:
  - **fixed allocation**
  - **priority allocation**

Fixed Allocation

- Equal allocation – e.g., if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation – Allocate according to the size of process.

  – s_i = size of process p_i

  – $S = \sum s_i$

  – m = total number of frames

  – a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$$m = 64$$
$$s_i = 10$$
$$s_2 = 127$$
$$a_1 = \frac{10}{137} \times 64 \approx 5$$
$$a_2 = \frac{127}{137} \times 64 \approx 59$$

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.

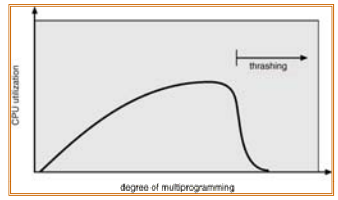
Global vs. Local Allocation

- **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

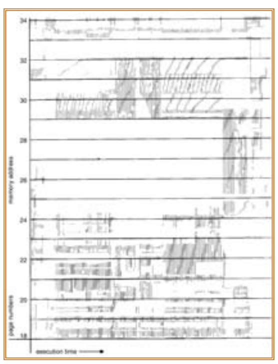
- If a process does not have "enough" pages, the page-fault rate is very high. This leads to:
  - **Low CPU utilization**.
  - Operating system thinks that it needs to increase the degree of multiprogramming.
  - Another process added to the system.
- **Thrashing** ≡ a process is busy swapping pages in and out.

Thrashing



- Why does paging work?
  Locality model
  - Process migrates from one locality to another.
  - Localities may overlap.
- Why does thrashing occur?
  Σ size of locality > total memory size

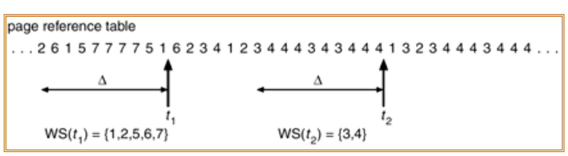
Locality in Memory-Reference Pattern



Working-Set Model

- $\Delta \equiv$ **working-set window** ≡ a fixed number of page references.
- WSS_i (working set of Process P_i) =
  total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small will not encompass entire locality.
  - if Δ too large will encompass several localities.
  - if $\Delta = \infty \Rightarrow$ will encompass entire program.
- $D = \Sigma\, WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ **Thrashing**
- Policy if D > m, then suspend one of the processes.

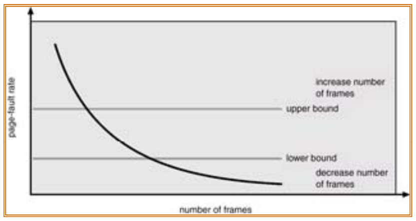
Working-set model



Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: Δ = 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

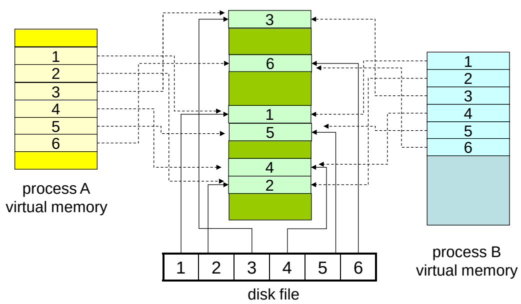


Establish "acceptable" page-fault rate.
- If actual rate too low, process loses frame.
- If actual rate too high, process gains frame.

Memory-mapped Files

- Memory mapping a file can be accomplished by mapping a disk block to one or more pages in memory.
- A page-sized portion of the file is read from the file system into a physical page. Subsequent `read()` and `write()` operations are handled as memory (not disk) accesses.
- Writing to the file in memory is not necessarily synchronous to the file on disk. The file can be committed back to disk when it's closed.

Memory-mapped Files



Prepaging

- **Prepaging**: In order to avoid the initial number of page faults, the system can bring into memory all the pages that will be needed all at once.
- This can also be applied when a swapped-out process is restarted. The smart thing to do is to remember the working set of the process.
- One question that arises is whether all the pages brought in will actually be used…
- Is the cost of prepaging less than the cost of servicing each individual page fault?