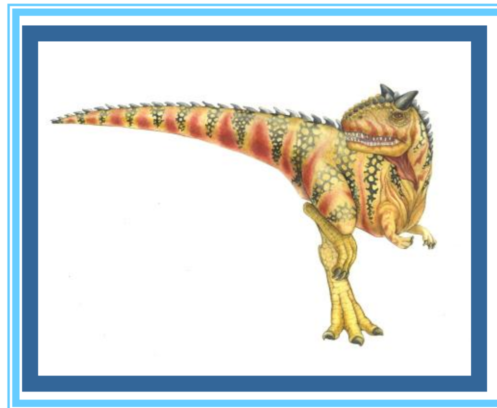
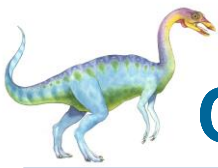
Chapter 8: Virtual-Memory Management

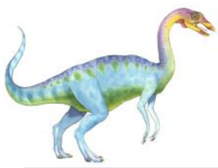


Silberschatz, Galvin and Gagne ©2009



Chapter 8: Virtual-Memory Management

- Background
- Demand Paging
- Copy-on-Write
- Page Replacement
- Allocation of Frames
- Thrashing
- Memory-Mapped Files
- Allocating Kernel Memory
- Other Considerations
- Operating-System Examples



Objectives

- To describe the benefits of a virtual memory system
- To explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames
- Apply the FIFO, optimal, and LRU page-replacement algorithms.

Background

- Code needs to be in memory to execute, but entire program rarely used
  - Error code, unusual routines, large data structures
- Entire program code not needed at same time
- Consider ability to execute partially-loaded program
  - Program no longer constrained by limits of physical memory
  - Each program takes less memory while running -> more programs run at the same time
    - Increased CPU utilization and throughput with no increase in response time or turnaround time
  - Less I/O needed to load or swap programs into memory -> each user program runs faster

Background

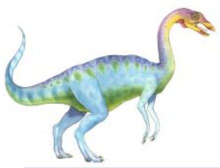
- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
  - More programs running concurrently
  - Less I/O needed to load or swap processes
- **Virtual address space** – logical view of how process is stored in memory
  - Usually start at address 0, contiguous addresses until end of space
  - Meanwhile, physical memory organized in page frames
  - MMU must map logical to physical
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Virtual Memory That is Larger Than Physical Memory

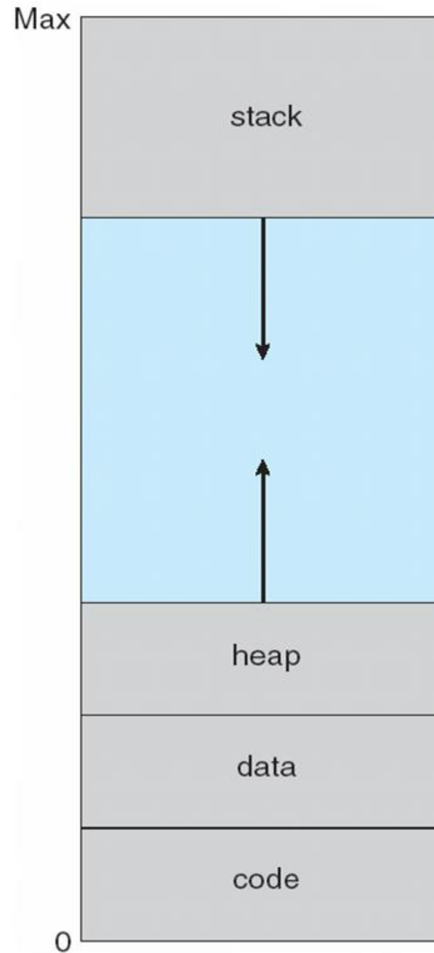
page 0
page 1
page 2
•
•
•
page *v*
virtual memory

memory map

physical memory



Virtual-address Space



Shared Library Using Virtual Memory

stack

shared library

heap

data

code

shared pages

stack

shared library

heap

data

code

Demand Paging

(การจัดสรร **Paging** ตามความต้องการที่ร้องขอ)

- 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 (หากต้องการใช้ page ให้อ้างตำแหน่งถึง page ที่ต้องการ)
  - invalid reference ⇒ abort (หากอ้างตำแหน่งไม่ถูกต้องให้ยกเลิก)
  - not-in-memory ⇒ bring to memory (หากอ้างแล้วไม่มีใน memory ให้นำเข้ามาไว้ใน 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

มี pager เป็นตัวย้าย page
เข้า (swap in) หรือออก (swap out)

program A

program B

swap out

swap in

0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23

main memory

Valid-Invalid Bit

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

Frame #	valid-invalid bit
	v
	v
	v
	v
	i
….	
	i
	i

page table

page fault

- During address translation, if valid–invalid bit in page table entry is **I** ⇒ page fault (การอ้างอิงผิดหน้าหรือไม่พบหน้าที่ต้องการ).

Page Table When Some Pages Are Not in Main Memory

logical memory

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

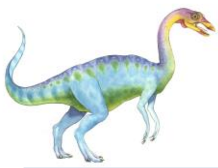
page table (frame, valid–invalid bit)

	frame	valid–invalid bit
0	4	v
1		i
2	6	v
3		i
4		i
5	9	v
6		i
7		i

physical memory

0	
1	
2	
3	
4	A
5	
6	C
7	
8	
9	F
10	
11	
12	
13	
14	
15	

A B
C D E
F G H



Page Fault

เมื่อมีการอ้างอิงผิดหน้าหรือไม่พบหน้าที่ต้องการในหน่วยความจำหลัก **(page fault)** จะมีขั้นตอนดำเนินการดังนี้

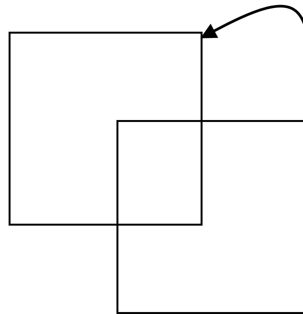
- If there is a reference to a page, first reference to that page will trap to operating system:

  page fault

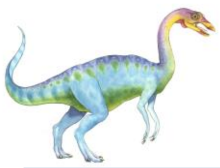
1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

Page Fault (Cont.)

- Restart instruction (ทำต่อจากจุดที่ได้ทำมาแล้วล่าสุด หรือ ทำต่อจากจุดที่เกิด page fault ขึ้น)
  - block move



  - auto increment/decrement location



Steps in Handling a Page Fault

operating system

3 page is on backing store

2 trap

reference

1

load M

i

page table

6 restart instruction

free frame

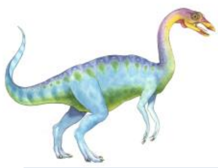
physical memory

5 reset page table

4 bring in missing page

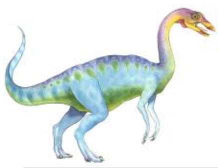
What happens if there is no free frame?

- **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

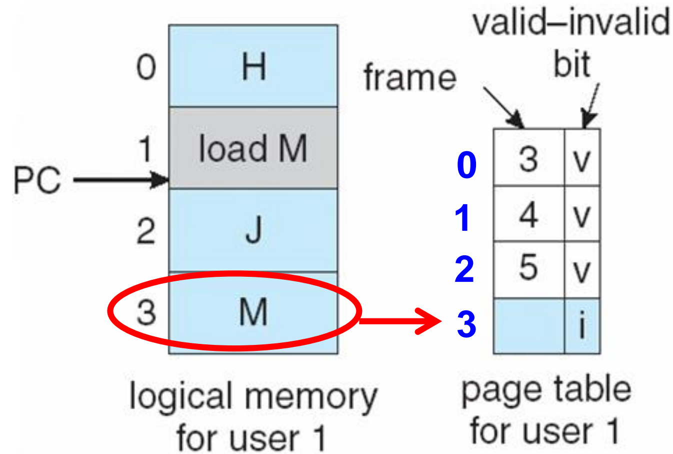


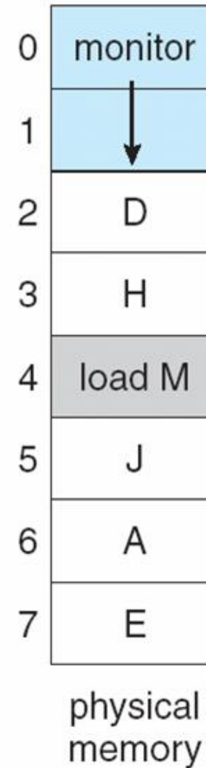
Page Replacement

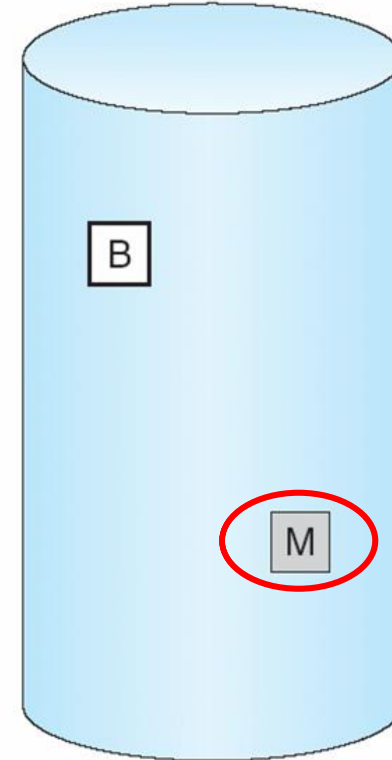
- **Prevent over-allocation of memory** by modifying page-fault service routine to include page replacement
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory –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

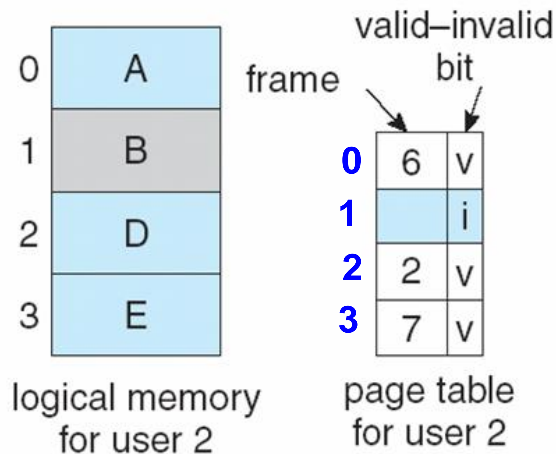


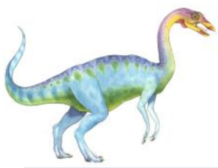
Need For Page Replacement





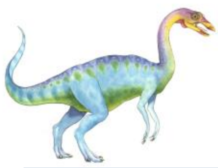




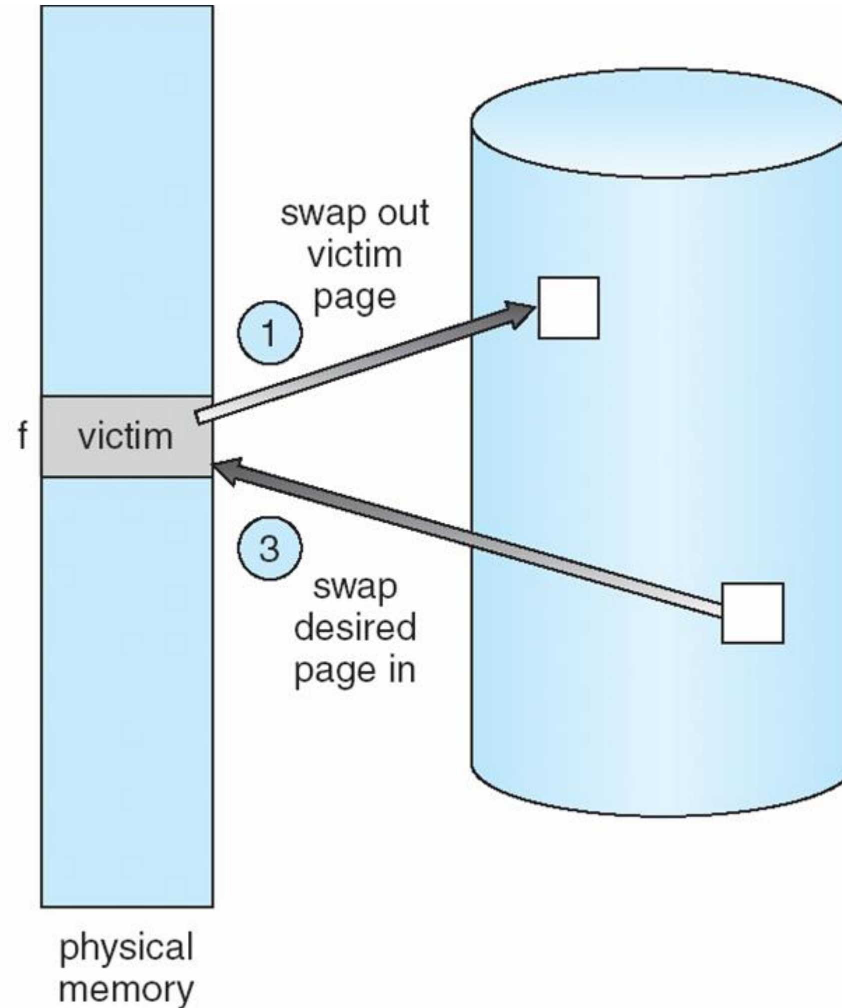


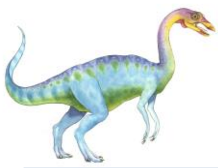
Basic Page Replacement

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





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

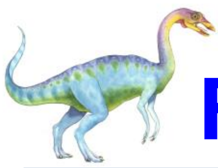
Graph of Page Faults Versus The Number of Frames

number of page faults

16
14
12
10
8
6
4
2

1 2 3 4 5 6

number of frames



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
3	3	2	4

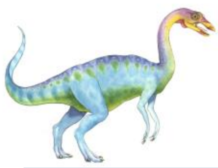
9 page faults

- 4 frames

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

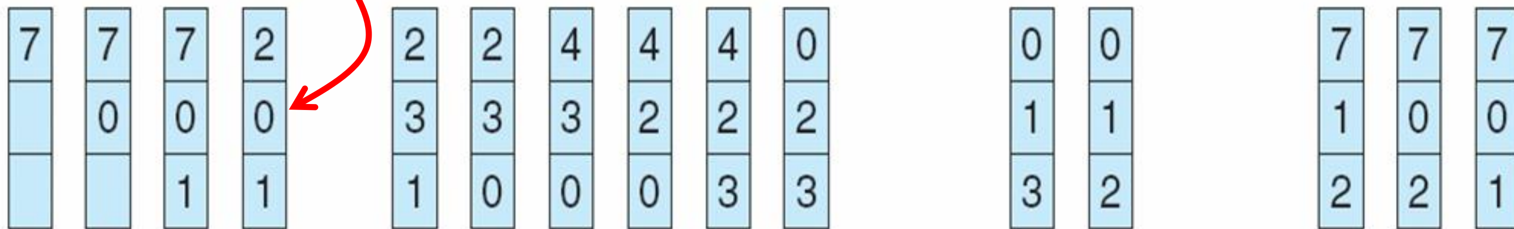
10 page faults

- Belady's Anomaly: more frames $\Rightarrow$ more page faults



FIFO Page Replacement

มี 0 อยู่แล้วใน memory จึงไม่ต้องไปดึงข้อมูลมาใหม่อีก จึงไม่เกิด page fault สำหรับ page 0



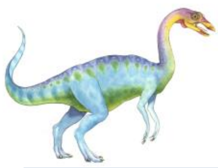
เมื่อ physical memory 3 frames และใช้ reference string ตามที่กำหนดให้ จะเกิด page fault ทั้งหมด 15 page faults

- ครั้งแรกไม่มี page อยู่ใน Physical Memory จะเกิด page fault

FIFO Illustrating Belady's Anomaly

number of page faults
16
14
12
10
8
6
4
2
1
2
3
4
5
6
7
number of frames

Belady's Anomoly : เป็นเหตุการณ์ที่เมื่อมี Physical Memory เพิ่มขึ้น แต่จะเกิด page fault เพิ่มขึ้นด้วย (เป็นข้อยกเว้นสำหรับ FIFO Algorithm)



Optimal Algorithm

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

(มองใน **list** ของ **reference string** ถัดไปในอนาคตว่า **page** ใดอีกนานที่จะถูกใช้งาน จะโดน **replace**)

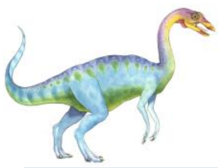
- 4 frames example

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

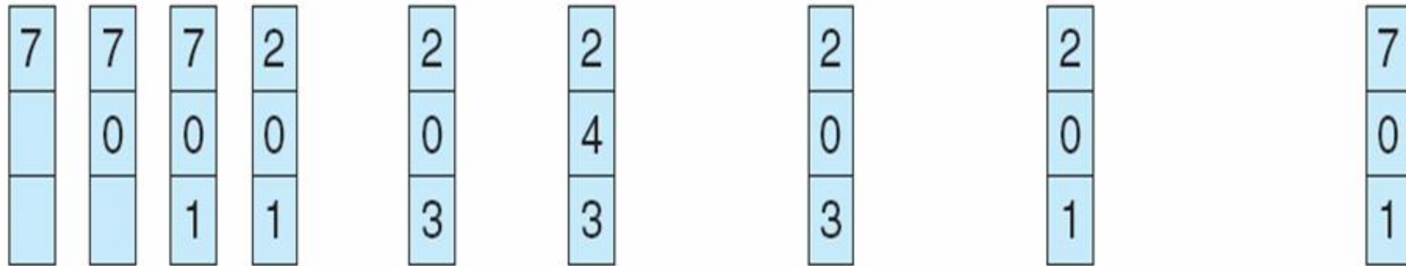
1	4
2	
3	
4	5

6 page faults

- How do you know this? ไม่ **make sence** เราไม่สามารถรู้อนาคตได้
- Used for measuring how well your algorithm performs



Optimal Page Replacement



เกิดกี่ page fault ???

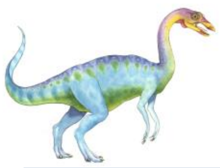
Least Recently Used (LRU) Algorithm

(มองย้อนกลับไปในอดีตว่า **page** ใดไม่ได้ถูกใช้มานานที่สุดจะถูก **replace)**

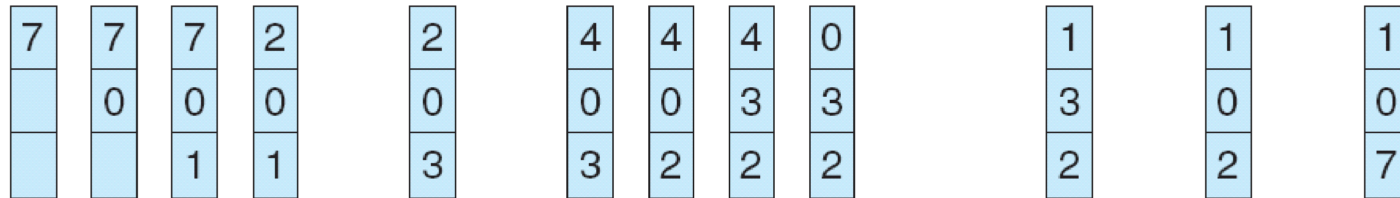
- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

1	1	1	1	**5**
2	2	2	2	2
3	**5**	5	**4**	4
4	4	**3**	3	3

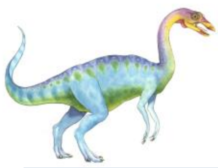
- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change



LRU Page Replacement



เกิดกี่ page fault ???



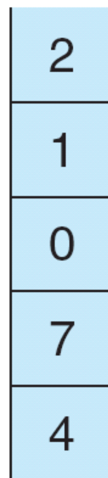
LRU Algorithm (Cont.)

- 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 find smallest value
    - Search through table needed
- 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 OPT are cases of **stack algorithms** that don't have Belady's Anomaly

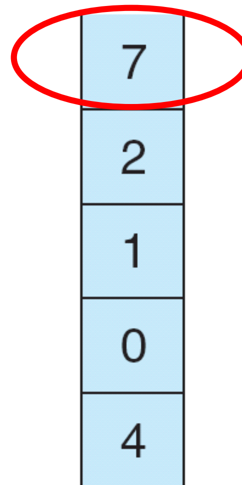
Use Of A Stack to Record The Most Recent Page References

reference string

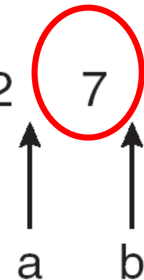
4 7 0 7 1 0 1 2 1 2 7 1 2

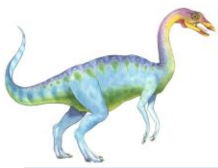


stack before a



stack after b





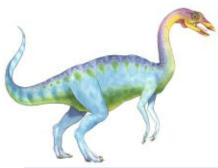
LRU Approximation Algorithms

- LRU needs special hardware and still slow
- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace any with reference bit = 0 (if one exists)
    - We do not know the order, however
- **Second-chance algorithm**
  - Generally FIFO, plus hardware-provided reference bit
  - **Clock** replacement
  - If page to be replaced has
    - Reference bit = 0 -> replace it
    - reference bit = 1 then:
      - set reference bit 0, leave page in memory
      - replace next page, subject to same rules

Second-Chance (clock) Page-Replacement Algorithm

reference bits
pages
next victim
0
0
1
1
0
1
1
circular queue of pages
(a)

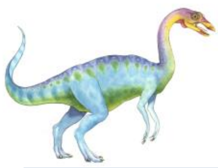
reference bits
pages
0
0
0
0
0
1
1
circular queue of pages
(b)



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

LFU : Least Frequently Used (ความถี่ในการถูกใช้น้อยที่สุด)
MFU: Most Frequently Used (ความถี่ในการถูกใช้มากที่สุด)



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
  - priority allocation

Fixed Allocation

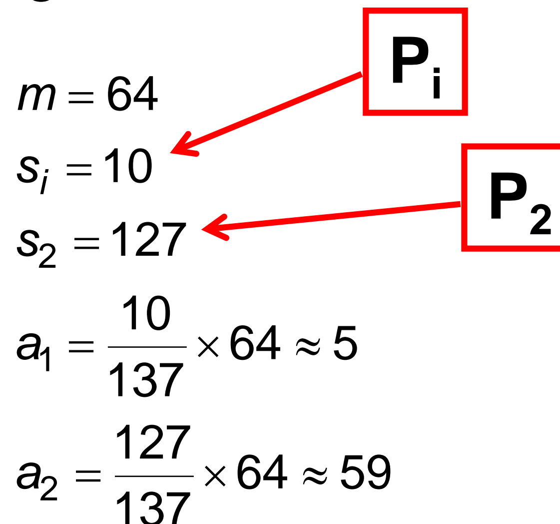
- Equal allocation – For example, if there are 100 frames and 5 processes, give each process 20 frames. (ได้จาก (frame / process) = (100 /5)
- Proportional allocation – Allocate according to the size of process

  – $s_i = \text{size of process } p_i$

  – $S = \sum s_i$

  – $m = \text{total number of frames}$

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

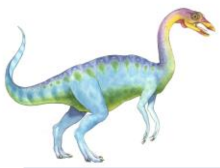


P_i ได้ 5 pages frame

P_2 ได้ 59 pages frame

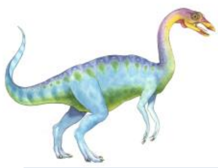
Priority Allocation

- 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

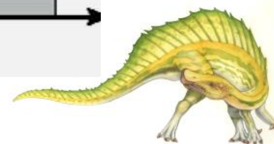
- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
- **Local replacement** – each process selects from only its own set of allocated frames

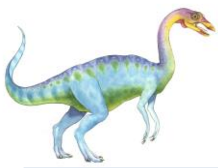


Page-Fault Frequency Scheme

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

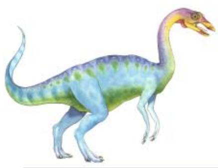
page-fault rate
increase number
of frames
upper bound
lower bound
decrease number
of frames
number of frames





Other Issues -- Prepaging

- Prepaging
  - To reduce the large number of page faults that occurs at process startup
  - Prepage all or some of the pages a process will need, before they are referenced
  - But if prepaged pages are unused, I/O and memory was wasted
  - Assume s pages are prepaged and α of the pages is used
    - Is cost of $s * \alpha$ save pages faults > or < than the cost of prepaging
      $s * (1- \alpha)$ unnecessary pages?
    - α near zero $\Rightarrow$ prepaging loses

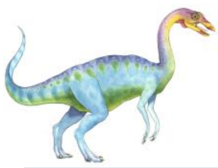


Other Issues – Page Size

- Page size selection must take into consideration:

  (เมื่อพิจารณาขนาดของ page size เป็นประเด็นหลัก)

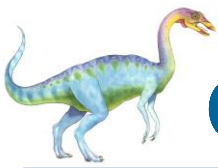
  - fragmentation
  - table size -> หาก ขนาด page เล็ก ก็จะต้องมี page table ที่ใหญ่
  - I/O overhead -> จะพิจารณาที่จำนวน page fault
  - locality



Other Issues – TLB Reach

- TLB Reach - The amount of memory accessible from the TLB
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB
  - Otherwise there is a high degree of page faults
- Increase the Page Size
  - This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes
  - This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation

TLB : Translation Look-aside Buffer



Other Issues – Program Structure

i อ้างถึง row, j อ้างถึง column

- Program structure
  - `Int[128,128] data;`
  - Each row is stored in one page
  - **Program 1**

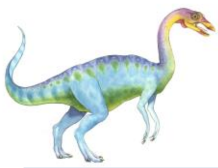
```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 page faults

  - **Program 2**

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

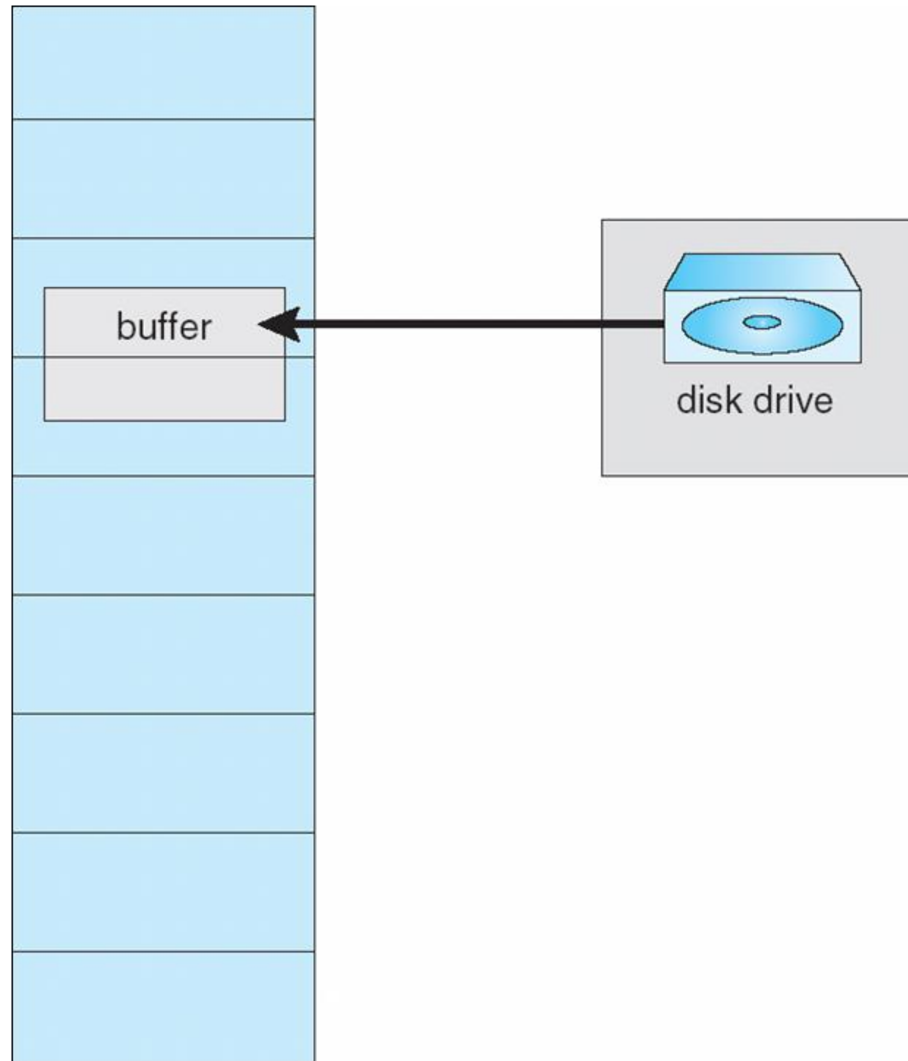
128 page faults

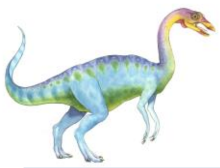


Other Issues – I/O interlock

- **I/O Interlock** – Pages must sometimes be locked into memory
- Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm

Reason Why Frames Used For I/O Must Be In Memory



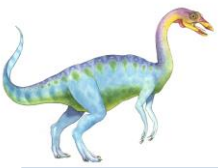


Operating System Examples

- Windows XP
- Solaris

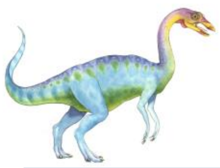
Windows XP

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page
- Processes are assigned **working set minimum** and **working set maximum**
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory
- A process may be assigned as many pages up to its working set maximum
- When the amount of free memory in the system falls below a threshold, **automatic working set trimming** is performed to restore the amount of free memory
- Working set trimming removes pages from processes that have pages in excess of their working set minimum

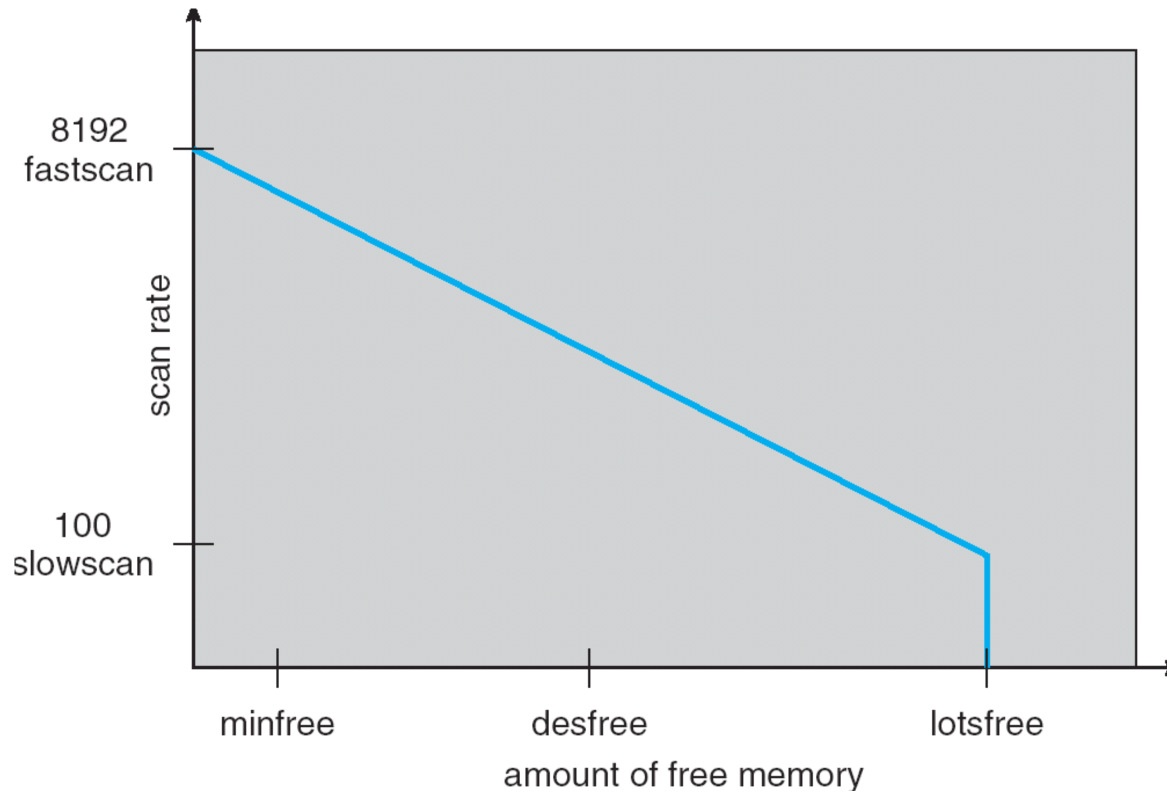


Solaris

- Maintains a list of free pages to assign faulting processes
- *Lotsfree* – threshold parameter (amount of free memory) to begin paging
- *Desfree* – threshold parameter to increasing paging
- *Minfree* – threshold parameter to being swapping
- Paging is performed by *pageout* process
- Pageout scans pages using modified clock algorithm
- *Scanrate* is the rate at which pages are scanned. This ranges from *slowscan* to *fastscan*
- Pageout is called more frequently depending upon the amount of free memory available



Solaris 2 Page Scanner



End of Chapter 8