Lecture 30 - Paging and Page Faults

CprE 308

March 27, 2015

Paging

Review: Scenario

Ideal World (for the programmer)

- I'm the only process in the world
- I have more memory than I need at my disposal

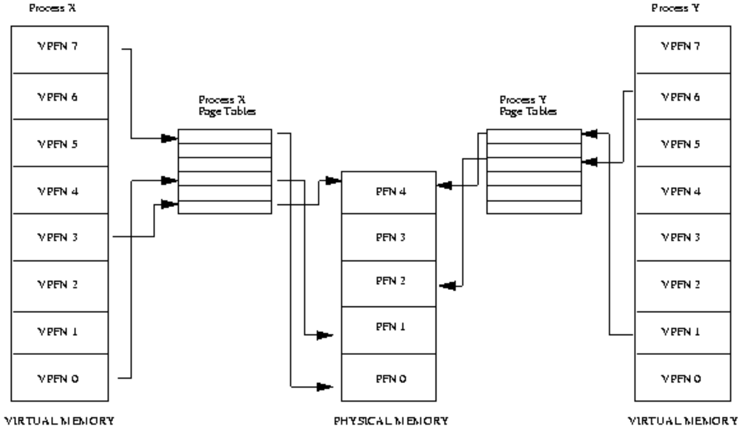
Real World

- Many processes in the system
- Not enough memory for them all
- Not all processes play nicely

Review: Goal of Memory Management

- Present the ideal world view to the programmer, yet implement it on a real system
- Add memory protections without getting in the way of the programmer

Review: Virtual Memory



Structuring Virtual Memory

- Paging
  - Divides the address space into fixed-sized pages
  - Reduces fragmentation, increases efficiency
- Segmentation
  - Divides the address space into variable-sized segments
  - Enables memory protections (Example: data, code, uninitialized, shared memory, etc.)
- Modern OS's use a mixture of both schemes (paged segmentation)

Typical Page Table Entry

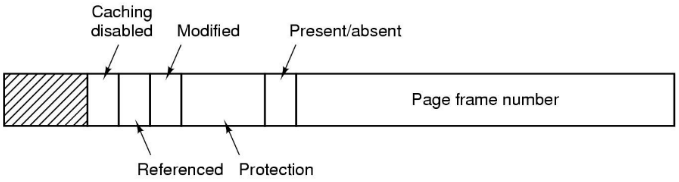


Figure 2: Page Table Entry Fields

Paging Example

- Consider a virtual memory system with two processes
  - Let the physical memory consist of 24 words and the page frame size of four words
  - Process 1 consists of 16 words (a through p)
  - Process 2 consists of 12 words (A through L)

Paging Example (Process 1 Virtual Memory)

Process 1 Virtual Memory

Virtual Address	Memory Contents
0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Paging Example (Process 1 Virtual Memory)

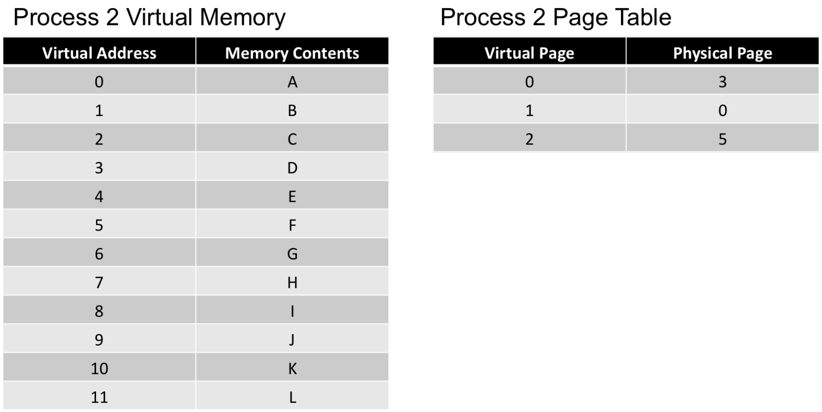
Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Paging Example (Process 2 Virtual Memory)



Process 2 Virtual Memory

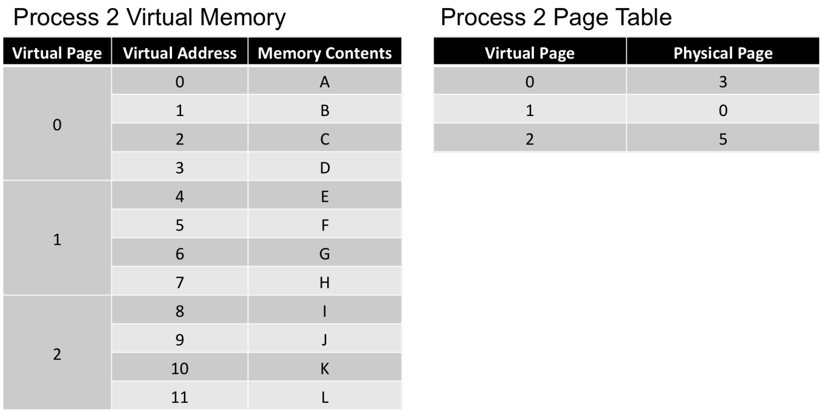
Virtual Address	Memory Contents
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I
9	J
10	K
11	L

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Figure 5: Process 2 Virtual Memory

Paging Example (Process 2 Virtual Memory)



Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Figure 6: Process 2 Virtual Memory

Paging Example (Physical Memory)

Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Physical Memory

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

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Paging Example (Physical Memory)

Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Physical Memory

Physical Page	Physical Address	Memory Contents
0	0	E
	1	F
	2	G
	3	H
1	4	
	5	
	6	
	7	
2	8	
	9	
	10	
	11	
3	12	
	13	
	14	
	15	
4	16	
	17	
	18	
	19	
5	20	
	21	
	22	
	23	

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Paging Example (Physical Memory)

Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Physical Memory

Physical Page	Physical Address	Memory Contents
0	0	E
	1	F
	2	G
	3	H
1	4	e
	5	f
	6	g
	7	h
2	8	
	9	
	10	
	11	
3	12	
	13	
	14	
	15	
4	16	
	17	
	18	
	19	
5	20	
	21	
	22	
	23	

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Paging Example (Physical Memory)

Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Physical Memory

Physical Page	Physical Address	Memory Contents
0	0	E
	1	F
	2	G
	3	H
1	4	e
	5	f
	6	g
	7	h
2	8	a
	9	b
	10	c
	11	d
3	12	
	13	
	14	
	15	
4	16	
	17	
	18	
	19	
5	20	
	21	
	22	
	23	

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Paging Example (Physical Memory)

Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Physical Memory

Physical Page	Physical Address	Memory Contents
0	0	E
	1	F
	2	G
	3	H
1	4	e
	5	f
	6	g
	7	h
2	8	a
	9	b
	10	c
	11	d
3	12	A
	13	B
	14	C
	15	D
4	16	
	17	
	18	
	19	
5	20	
	21	
	22	
	23	

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Paging Example (Physical Memory)

Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Physical Memory

Physical Page	Physical Address	Memory Contents
0	0	E
	1	F
	2	G
	3	H
1	4	e
	5	f
	6	g
	7	h
2	8	a
	9	b
	10	c
	11	d
3	12	A
	13	B
	14	C
	15	D
4	16	m
	17	n
	18	o
	19	p
5	20	
	21	
	22	
	23	

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Paging Example (Physical Memory)

Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Physical Memory

Physical Page	Physical Address	Memory Contents
0	0	E
	1	F
	2	G
	3	H
1	4	e
	5	f
	6	g
	7	h
2	8	a
	9	b
	10	c
	11	d
3	12	A
	13	B
	14	C
	15	D
4	16	m
	17	n
	18	o
	19	p
5	20	I
	21	J
	22	K
	23	L

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Paging Example (Physical Memory)

- Suppose process 1 is running and it tries to access the contents of the virtual address **15**, what is the result?

Paging Example (Physical Memory)

Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Physical Memory

Physical Page	Physical Address	Memory Contents
0	0	E
	1	F
	2	G
	3	H
1	4	e
	5	f
	6	g
	7	h
2	8	a
	9	b
	10	c
	11	d
3	12	A
	13	B
	14	C
	15	D
4	16	m
	17	n
	18	o
	19	p
5	20	I
	21	J
	22	K
	23	L

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Paging Example (Page Faults)

- Suppose process 1 is running and it tries to access the contents of the virtual address **15**, what is the result?
  - Virtual address **15** is in process 1's virtual page **3**. According to the page table for process 1, the virtual page **3** is paged in physical memory as page **4**, which means the value **p** will be immediately fetched from memory.

Paging Example (Page Faults)

- Suppose process 1 is running and it tries to access the contents of the virtual address **9**, what is the result?

Paging Example (Page Faults)

Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Physical Memory

Physical Page	Physical Address	Memory Contents
0	0	E
	1	F
	2	G
	3	H
1	4	e
	5	f
	6	g
	7	h
2	8	a
	9	b
	10	c
	11	d
3	12	A
	13	B
	14	C
	15	D
4	16	m
	17	n
	18	o
	19	p
5	20	I
	21	J
	22	K
	23	L

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Paging Example (Page Faults)

- Suppose process 1 is running and it tries to access the contents of the virtual address **9**, what is the result?
  - Virtual address **9** is in process 1's virtual page **2**. According to the page table for process 1, virtual page **2** is not paged in physical memory (flagged as invalid in the page table). A **page fault** occurs, and physical memory will need to be swapped before the value **j** can be fetched from memory.

Paging Example (Address Translation)

Process 1

- Virtual Address **2** to Physical Address
- Physical Address **5** to Virtual Address

Process 2

- Virtual Address **2** to Physical Address
- Physical Address **22** to Virtual Address

Paging Example (Address Translation)

Process 1 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	a
	1	b
	2	c
	3	d
1	4	e
	5	f
	6	g
	7	h
2	8	i
	9	j
	10	k
	11	l
3	12	m
	13	n
	14	o
	15	p

Process 2 Virtual Memory

Virtual Page	Virtual Address	Memory Contents
0	0	A
	1	B
	2	C
	3	D
1	4	E
	5	F
	6	G
	7	H
2	8	I
	9	J
	10	K
	11	L

Physical Memory

Physical Page	Physical Address	Memory Contents
0	0	E
	1	F
	2	G
	3	H
1	4	e
	5	f
	6	g
	7	h
2	8	a
	9	b
	10	c
	11	d
3	12	A
	13	B
	14	C
	15	D
4	16	m
	17	n
	18	o
	19	p
5	20	I
	21	J
	22	K
	23	L

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Paging Example (Address Translation)

Process 1

- Virtual Address **2** to Physical Address
  - **10**
- Physical Address **5** to Virtual Address
  - **5**

Process 2

- Virtual Address **2** to Physical Address
  - **14**
- Physical Address **22** to Virtual Address
  - **10**

Implementation Notes

- Virtual memory is just a concept
  - It's addresses/values are always contiguous
  - It's values only really exist in physical memory
  - Page frames are just logical groupings (that can be calculated on the fly)
- Only need to store page tables

Implementation Notes

Process 1 Page Table

Virtual Page	Physical Page
0	2
1	1
2	invalid
3	4

Process 2 Page Table

Virtual Page	Physical Page
0	3
1	0
2	5

Physical Memory

Physical Address	Memory Contents
0	E
1	F
2	G
3	H
4	e
5	f
6	g
7	h
8	a
9	b
10	c
11	d
12	A
13	B
14	C
15	D
16	m
17	n
18	o
19	p
20	I
21	J
22	K
23	L

Implementation Notes

- Virtual page frames are always in order starting at 0
  - No need to store virtual page numbers in page table (just store physical page numbers in order)
- Techinically we don't “store” addresses either

Implementation Notes

Process 1 Page Table

Physical Page
2
1
invalid
4

Process 2 Page Table

Physical Page
3
0
5

Physical Memory

Memory Contents
E
F
G
H
e
f
g
h
a
b
c
d
A
B
C
D
m
n
o
p
I
J
K
L

Implementation Notes

- If our page table stores 4 virtual pages mappings how many bits do we need to represent each page?
- If our page size is 4 words, how many bits do we need to represent each possible page offset?

Implementation Notes

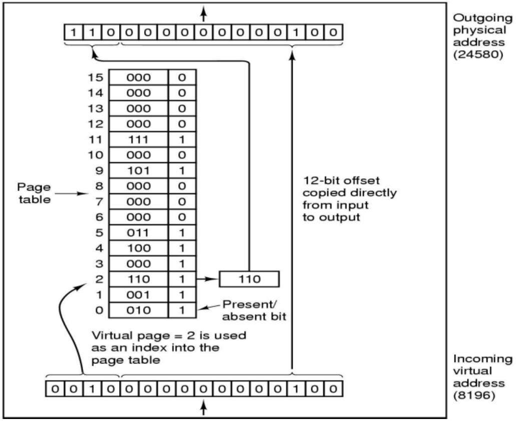


Figure 19: Address Translation

Page Faults

Page Fault

What happens if the required page is not in memory?

"Page-fault" trap is initiated, OS gets control

1. Find a free page frame
2. Read the desired page from disk into memory
3. Modify the page tables
4. Restart the interrupted instruction

OS Issues

- Fetch policy - when to fetch pages into memory?
- Placement policy - where to place pages?
- Replacement policy
- All combined in the handling of a page fault

A Simple Paging Scheme

Fetch policy

- start process off with no pages in primary storage
- bring in pages on demand (and only on demand)
  - this is known as demand paging

Placement policy

- it doesn't matter - put the incoming page in the first available page frame

Replacement policy

- replace the page that has been in primary storage the longest (FIFO policy)

Improving the Fetch Policy

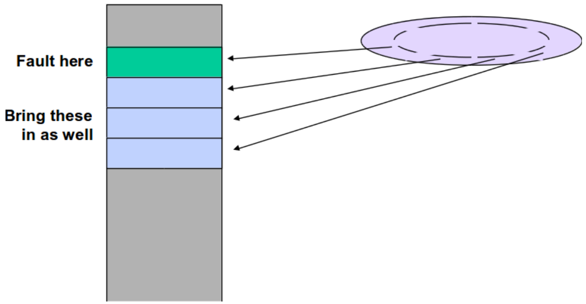


Figure 20: Fetch Policy

Page Replacement

Improving the Replacement Policy

- When is replacement done?
  - doing it “on demand” causes excessive delays
  - should be performed as a separate, concurrent activity
- Which pages are replaced?
  - FIFO policy is not good
  - want to replace those pages least likely to be referenced soon

The “Pageout Daemon”

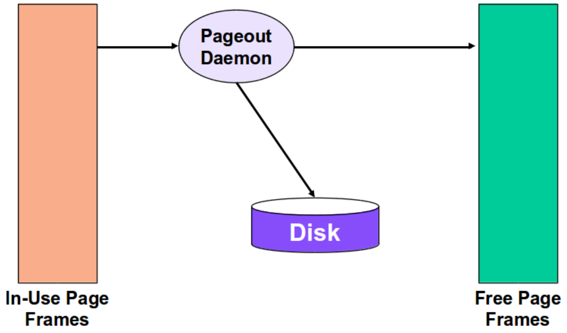


Figure 21:

Page Replacement

Problem Statement:

> *A page is being brought into memory which has no free space. Which page should we replace to make space?*

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests
- Policy: Choose the page which will be referenced farthest in the future

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests
- Policy: Choose the page which will be referenced farthest in the future
- However, we don't know the future

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests
- Policy: Choose the page which will be referenced farthest in the future
- However, we don't know the future
    - Hope that the next few references will be for pages that were recently referenced

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests
- Policy: Choose the page which will be referenced farthest in the future
- However, we don't know the future
  - Hope that the next few references will be for pages that were recently referenced
- What's the use of knowing about this policy?

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests
- Policy: Choose the page which will be referenced farthest in the future
- However, we don't know the future
  - Hope that the next few references will be for pages that were recently referenced
- What's the use of knowing about this policy?
  - Will help us access the performance of a real algorithm

Choosing the Page to Remove

Policies:

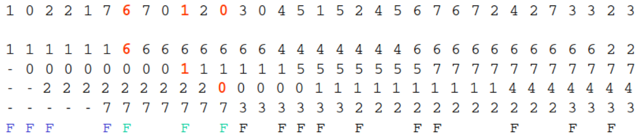
- FIFO (First-In-First-Out)
- NRU (Not-Recently-Used)
- Second Chance
- LRU (Least-Recently-Used)
- Clock Algorithm(s)
- Working Set Algorithm

Two issues:

- How good is the decision?
- Overhead?
  - Cost per memory access - should be very small
  - Cost per replacement - can be larger

FIFO

Example: 8 pages, 4 page frames



1	0	2	2	1	7	6	7	0	1	2	0	3	0	4	5	1	5	2	4	5	6	7	6	7	2	4	2	7	3	3	2	3
1	1	1	1	1	1	6	6	6	6	6	6	6	6	4	4	4	4	4	4	4	6	6	6	6	6	6	6	6	6	6	2	2
-	0	0	0	0	0	0	0	0	1	1	1	1	1	1	5	5	5	5	5	5	5	7	7	7	7	7	7	7	7	7	7	7
-	-	2	2	2	2	2	2	2	2	2	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
-	-	-	-	-	7	7	7	7	7	7	7	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3
F	F	F			F	F			F		F	F		F	F	F		F			F	F				F			F		F	

Figure 22:

Hit ratio: 16/33

Help from Hardware

For each page frame:

- Referenced Bit(R) - 1 if page frame has been referenced recently
- Modified Bit(M) - 1 if page has been modified since it has been loaded
    - Also known as “dirty bit”

Not Recently Used Algorithm (NRU)

Pages are classified into 4 classes:

- Class 0: not referenced, not modified (R=0, M=0)
- Class 1: not referenced, modified (R=0, M=1)
- Class 2: referenced, not modified (R=1, M=0)
- Class 3: referenced, modified (R=1, M=1)

NRU removes page at random from lowest number non empty class

The R bit is cleaned periodically (based on a timer)

Second Chance

- Based on FIFO
- Old pages are inspected for replacement
    - But are given a “second chance” if they have been used recently

Second Chance Algorithm

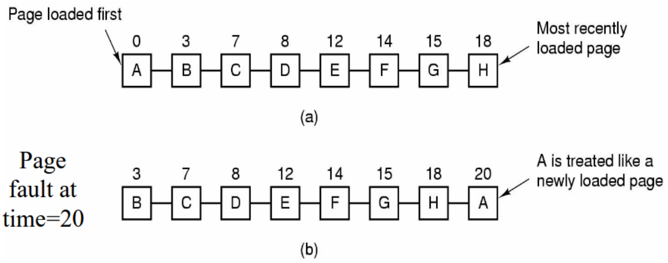


Figure 23:

- Pages sorted in FIFO order (time of arrival)
- If earliest page has R=1, then give it a second chance by moving it to the end of the list

Clock Algorithm - Another Implementation of Second Chance

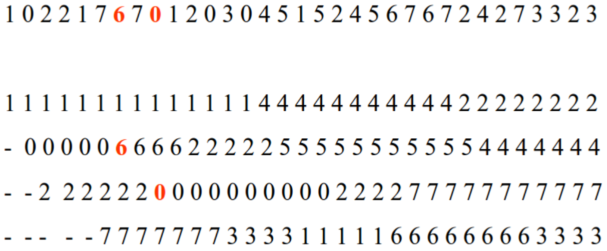
- Order pages in circular list
- "Hand" of the clock points to the page to be replaced currently
- When required to evict a page
  - If page pointed to has R=0, then evict it
  - If R=1, then reset R and move hand forward
- Clock algorithm can be used with NRU (decision based on both R and M bits)

Least Recent Used (LRU)

- Replace the page in memory which has been unused for the longest time
- Locality of Reference: pages used in the near past will be used in the near future
  - True in typical cases

Least Recently Used (LRU)

Example: 8 pages, 4 page frames



1 0 2 2 1 7 **6** 7 **0** 1 2 0 3 0 4 5 1 5 2 4 5 6 7 6 7 2 4 2 7 3 3 2 3

1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4 4 4 4 4 2 2 2 2 2 2 2 2

- 0 0 0 0 0 **6** 6 6 6 2 2 2 2 2 5 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4

- - 2 2 2 2 2 2 **0** 0 0 0 0 0 0 0 0 0 2 2 2 2 7 7 7 7 7 7 7 7 7 7 7

- - - - - 7 7 7 7 7 7 7 3 3 3 3 1 1 1 1 1 6 6 6 6 6 6 6 6 3 3 3 3

Figure 24:

LRU Implementation

- Think of how you would implement it

LRU Implementation

- Think of how you would implement it
- One possible implementation:

LRU Implementation

- Think of how you would implement it
- One possible implementation:
  - list of pages, most recently used at front, least at rear

LRU Implementation

- Think of how you would implement it
- One possible implementation:
  - list of pages, most recently used at front, least at rear
  - update this list every memory reference

LRU Implementation

- Think of how you would implement it
- One possible implementation:
  - list of pages, most recently used at front, least at rear
  - update this list every memory reference
  - when required to evict a page, choose the one at the rear of the list

LRU Implementation

- Think of how you would implement it
- One possible implementation:
  - list of pages, most recently used at front, least at rear
  - update this list every memory reference
  - when required to evict a page, choose the one at the rear of the list
- Way too expensive!

Not Frequently Used (NFU)

- Requires a software counter associated with each page, initially zero
- At each clock interrupt, OS scans all the pages in memory
- For each page, the R bit is added to the counter
- The page with the lowest counter is chosen

Aging - Approximating LRU

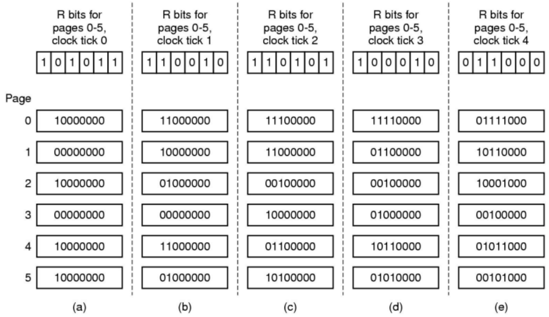
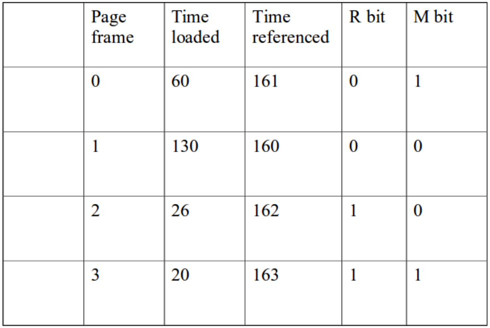


Figure 25:

Example



	Page frame	Time loaded	Time referenced	R bit	M bit
	0	60	161	0	1
	1	130	160	0	0
	2	26	162	1	0
	3	20	163	1	1

Figure 26:

Questions:

Which page frame will be replaced?

- FIFO

Questions:

Which page frame will be replaced?

- FIFO
    - PFN 3 since loaded longest ago at time 20

Questions:

Which page frame will be replaced?

- FIFO
  - PFN 3 since loaded longest ago at time 20
- LRU

Questions:

Which page frame will be replaced?

- FIFO
  - PFN 3 since loaded longest ago at time 20
- LRU
  - PFN 1 since referenced longest ago at time 160

Questions:

Which page frame will be replaced?

- FIFO
  - PFN 3 since loaded longest ago at time 20
- LRU
  - PFN 1 since referenced longest ago at time 160
- Clock

Questions:

Which page frame will be replaced?

- FIFO
    - PFN 3 since loaded longest ago at time 20
- LRU
    - PFN 1 since referenced longest ago at time 160
- Clock
    - Clear R in PFN 3 (oldest loaded), clear R in PFN 2 (next oldest loaded), victim PFN is 0 since R=0