

8. Virtual Memory

8.1 Principles of Virtual Memory

8.2 Implementations of Virtual Memory

- Paging
- Segmentation
- Paging With Segmentation
- Paging of System Tables
- Translation Look-aside Buffers

8.3 Memory Allocation in Paged Systems

- Global Page Replacement Algorithms
- Local Page Replacement Algorithms
- Load Control and Thrashing
- Evaluation of Paging

ICS 143

1

Principles of virtual memory

- system creates illusion of large contiguous memory space(s) for each process
- relevant portions of VM are loaded automatically and transparently
- address map translates virtual addresses to physical addresses

Figure 8-1

ICS 143

2

Principles of virtual memory

- single-segment VM:
 - one area of $0..n-1$ words
 - divided into fix-size pages
- multiple-segment VM:
 - multiple areas of up to $0..n-1$ (words)
 - each holds a logical segment (function, data structure)
 - each is contiguous or divided into pages

ICS 143

3

Main issues in VM design

- address mapping
 - how to translate virtual addresses to physical
- placement
 - where to place a portion of VM needed by process
- replacement
 - which portion of VM to remove when space is needed
- load control
 - how much of VM to load at any one time
- sharing
 - how can processes share portions of their VMs

ICS 143

4

Implementation using paging

- VM is divided into fix-size pages
- PM is divided into page frames (page size)
- system loads pages into frames and translates addresses
- virtual address: $va = (p,w)$
- physical address: $pa = (f,w)$
- $|p|$ determines number of pages in VM
- $|f|$ determines number of frames in PM
- $|w|$ determines page/frame size

Figure 8-2

ICS 143

5

Address translation

- given (p,w) :
 - determine f from p -- how to keep track?
 - concatenate $f+w$ to form pa
- one solution: frame table
 - one entry $FT[i]$ for each frame
 - $FT[i].pid$ records process ID
 - $FT[i].page$ records page number p
 - given (id,p,w) : search for a match on (id,p) :
index f is the frame number

ICS 143

6

Address translation

Figure 8-4

```
address_map(id, p, w) {
    pa = UNDEFINED;
    for (f = 0; f < F; f++)
        if (FT[f].pid==id && FT[f].page==p)
            pa = f*w;
    return pa;
}
```

- drawbacks of frame table implementation
 - costly: search must be done in parallel in hardware
 - sharing of pages difficult or not possible

ICS 143

7

Page tables

- PT associated with each VM (not PM)
- PTR points at PT at run time
- p'th entry holds frame number of page p:
 - *(PTR+p) points to frame f
- address translation:

```
address_map(p, w) {
    pa = *(PTR+p)*w;
    return pa;
}
```

Figure 8-5

- drawback: extra memory access

ICS 143

8

Demand paging

- all pages of VM can be loaded initially
 - simple but maximum size of VM = size of PM
- pages a loaded as needed -- on demand
 - additional bit in PT indicates presence/absence
 - page fault occurs when page is absent

```
address_map(p, w) {
    if (resident(*(PTR+p))) {
        pa = *(PTR+p)*w;
        return pa;
    } else page_fault;
}
```

ICS 143

9

VM using segmentation

- multiple contiguous spaces
 - more natural match to program/data structure
 - easier sharing (Chapter 9)
- $va = (s, w)$ mapped to pa (but no frames)
- where/how are segments placed in PM?
 - contiguous versus paged application

ICS 143

10

Contiguous allocation

- each segment is contiguous in PM
- ST keeps track of starting locations
- STR point to ST
- address translation

```
address_map(s, w) {
    if (resident(*(STR+s))) {
        pa = *(STR+s)+w;
        return pa; }
    else segment_fault; }
```
- drawback: external fragmentation

ICS 143

11

Paging with segmentation

- each segment is divided into fix-size pages
- $va = (s, p, w)$
 - |s| determines # of segments (size of ST)
 - |p| determines # of pages per seg. (size of PT)
 - |w| determines page size
- $pa = (*(STR+s)+p)+w$

Figure 8-7

- drawback: 2 extra memory references

ICS 143

12

Paging of system tables

- ST or PT may be too large to keep in PM
 - divide ST or PT into pages
 - keep track by additional page table
- paging of ST
 - ST divided into pages
 - segment directory keeps track of ST pages
 - $va = (s1, s2, p, w)$
 - $pa = *(*(STR+s1)+s2)+p)+w$

Figure 8-8

ICS 143

13

Translation look-aside buffers

- to avoid additional memory accesses
 - keep most recently translated page numbers in associative memory:
 - for any $(s,p,*)$ keep (s,p) and frame# f
 - bypass translation if match on (s,p) found
- TLB is different from cache
 - TLB only keeps frame #s
 - cache keeps data values

Figure 8-10

ICS 143

14

Memory allocation with paging

- placement policy: any free frame is ok
- replacement: must minimize data movement
- global replacement:
 - consider all resident pages (regardless of owner)
- local replacement
 - consider only pages of faulting process
- how to compare different algorithms:
 - use reference string: $r_0 r_1 \dots r_t \dots$
 - r_t is the number of the page referenced at time t
 - count # of page faults

ICS 143

15

Global page replacement

- optimal (MIN): replace page that will not be referenced for the longest time in the future

Time t	0	1	2	3	4	5	6	7	8	9	10
RS		c	a	d	b	e	b	a	b	c	d
Frame 0	a	a	a	a	a	a	a	a	a	a	d
Frame 1	b	b	b	b	b	b	b	b	b	b	b
Frame 2	c	c	c	c	c	c	c	c	c	c	c
Frame 3	d	d	d	d	d	e	e	e	e	e	e
IN						e					d
OUT						d					a

- problem: RS not known in advance

ICS 143

16

Global page replacement

- random replacement
 - simple but
 - does not exploit locality of reference
 - most instructions are sequential
 - most loops are short
 - many data structures are accessed sequentially

ICS 143

17

Global page replacement

- FIFO: replace oldest page

Time t	0	1	2	3	4	5	6	7	8	9	10
RS		c	a	d	b	e	b	a	b	c	d
Frame 0	>a	>a	>a	>a	>a	e	e	e	e	>e	d
Frame 1	b	b	b	b	b	>b	>b	a	a	a	>a
Frame 2	c	c	c	c	c	c	c	>c	b	b	b
Frame 3	d	d	d	d	d	d	d	d	>d	c	c
IN						e	a	b	c	d	
OUT						a	b	c	d	e	

- problem:
 - favors recently accessed pages but
 - ignores when program returns to old pages

ICS 143

18

Global page replacement

- LRU: replace least recently used page

Time t	0	1	2	3	4	5	6	7	8	9	10
RS		c	a	d	b	e	b	a	b	c	d
Frame 0	a	a	a	a	a	a	a	a	a	a	d
Frame 1	b	b	b	b	b	b	b	b	b	b	b
Frame 2	c	c	c	c	e	e	e	e	e	d	
Frame 3	d	d	d	d	d	d	d	d	d	c	c
IN						e				c	d
OUT					c					d	e
Q.end	d	c	a	d	b	e	b	a	b	c	d
	c	d	c	a	d	b	e	b	a	b	c
	b	b	d	c	a	d	d	e	e	a	b
Q.head	a	a	b	b	c	a	a	d	d	e	a

ICS 143

19

Global page replacement

- LRU implementation
 - software queue: too expensive
 - time-stamping
 - stamp each referenced page with current time
 - replace page with oldest stamp
 - hardware capacitor with each frame
 - charge at reference
 - replace page with smallest charge
 - n-bit aging register with each frame
 - shift all registers to right at every reference
 - set left-most bit of referenced page to 1
 - replace page with smallest value

ICS 143

20

Global page replacement

- second-chance algorithm
 - approximates LRU
 - implement use-bit u with each frame
 - u=1 when page referenced
 - to select a page:
 - if u=0, select page
 - else, set u=0 and consider next frame
 - used page gets a second chance to stay in PM
- algorithm is called “clock” algorithm:
 - search cycles through page frames

ICS 143

21

Global page replacement

- second-chance algorithm

...	4	5	6	7	8	9	10
...	b	e	b	a	b	c	d
...	>a/1	e/1	e/1	e/1	e/1	>e/1	d/1
...	b/1	>b/0	>b/1	b/0	b/1	b/1	>b/0
...	c/1	c/0	c/0	a/1	a/1	a/1	a/0
...	d/1	d/0	d/0	>d/0	>d/0	c/1	c/0
...		e		a		c	d

ICS 143

22

Global page replacement

- third-chance algorithm
 - second chance makes no difference between read and write access
 - write access more expensive
 - give modified pages a third chance:
 - u-bit set at every reference (read and write)
 - w-bit set at write reference
 - to select a page, cycle through frames, resetting bits, until $uw=00$:

$uw \rightarrow uw$	
11	01
10	00
01	00* (remember modification)
00	select

ICS 143

23

Global page replacement

- third-chance algorithm

...	0	1	2	3	4	5
...		c	a	d	b	e
...	>a/10	>a/10	>a/11	>a/11	>a/11	a/00*
...	b/10	b/10	b/10	b/10	b/11	b/00*
...	c/10	c/10	c/10	c/10	c/10	e/10
...	d/10	d/10	d/10	d/10	d/10	>d/00
...						e

ICS 143

24

Local page replacement

- measurements indicate that every program needs a “minimum” set of pages
 - if too few, thrashing occurs
 - if too many, page frames are wasted
- the “minimum” varies over time
- how to determine and implement this “minimum”?

ICS 143

25

Local page replacement

- optimal (VMIN)
 - define a sliding window $(t, t+\tau)$
 - τ is a parameter (constant)
 - at any time t , maintain as resident all pages visible in window
- guaranteed to generate smallest number of page faults

ICS 143

26

Local page replacement

- optimal (VMIN) with $\tau=3$

Time t	0	1	2	3	4	5	6	7	8	9	10
RS	d	c	c	d	b	c	e	c	e	a	d
Page a	-	-	-	-	-	-	-	-	-	x	-
Page b	-	-	-	x	-	-	-	-	-	-	-
Page c	-	x	x	x	x	x	x	x	-	-	-
Page d	x	x	x	x	-	-	-	-	-	-	x
Page e	-	-	-	-	-	-	x	x	x	-	-
IN		c			b		e			a	d
OUT				d	b			c	e		a

- guaranteed optimal but
- unrealizable without RS

ICS 143

27

Local page replacement

- working set model: use principle of locality
 - use trailing window (instead of future window)
 - working set $W(t, \tau)$: all pages referenced during $(t-\tau, t)$
 - at time t :
 - remove all pages not in $W(t, \tau)$
 - process may run only if entire $W(t, \tau)$ is resident

ICS 143

28

Local page replacement

- working set model

Time t	0	1	2	3	4	5	6	7	8	9	10
RS	a	c	c	d	b	c	e	c	e	a	d
Page a	x	x	x	x	-	-	-	-	-	x	x
Page b	-	-	-	-	x	x	x	x	-	-	-
Page c	-	x	x	x	x	x	x	x	x	x	x
Page d	x	x	x	x	x	x	x	-	-	-	x
Page e	x	x	-	-	-	-	x	x	x	x	x
IN		c			b		e			a	d
OUT			e		a			d	b		

- drawback: costly to implement
- approximate (aging registers, time stamps)

ICS 143

29

Local page replacement

- page fault frequency
 - main objective: keep page fault rate low
 - basic principle of pff:
 - if time between page faults $\geq \tau$, grow resident set: add new page to resident set
 - if time between page faults $< \tau$, shrink resident set: add new page but remove all pages not referenced since last page fault

ICS 143

30

Local page replacement

- page fault frequency

Time t	0	1	2	3	4	5	6	7	8	9	10
RS		c	c	d	b	c	e	c	e	a	d
Page a	x	x	x	x	-	-	-	-	-	x	x
Page b	-	-	-	x	x	x	x	x	-	-	-
Page c	-	x	x	x	x	x	x	x	x	x	x
Page d	x	x	x	x	x	x	x	x	x	-	x
Page e	x	x	x	x	-	-	x	x	x	x	x
IN		c			b		e			a	d
OUT					ae						bd

ICS 143

31

Load control and thrashing

- main issues:
 - how to choose degree of multiprogramming
 - when level decreased, which process should be deactivated
 - when new process reactivated, which of its pages should be loaded

ICS 143

32

Load control and thrashing

- choosing degree of multiprogramming
- local replacement:
 - working set of any process must be resident
 - this automatically imposes a limit
- global replacement
 - no working set concept
 - use CPU utilization as a criterion
 - with too many processes, thrashing occurs

Figure 8-11

ICS 143

33

Load control and thrashing

- how to find N_{\max} ?
 - L=S criterion:
 - page fault service S needs to keep up with mean time between faults L
 - 50% criterion:
 - CPU utilization is highest when paging disk 50% busy (found experimentally)

ICS 143

34

Load control and thrashing

- which process to deactivate
 - lowest priority process
 - faulting process
 - last process activated
 - smallest process
 - largest process
- which pages to load when process activated
 - prepage last resident set

Figure 8-12

ICS 143

35

Evaluation of paging

- experimental measurements:
 - Figure 8-13
 - (a): prepaging is important
 - initial set can be loaded more efficiently than by individual page faults
 - (b,c): page size should be small, however small pages require
 - larger page tables
 - more hardware
 - greater I/O overhead
 - (d): load control is important

ICS 143

36
