CSE3221.3
Operating System Fundamentals

No.9

Memory Management (2)

Prof. Hui Jiang
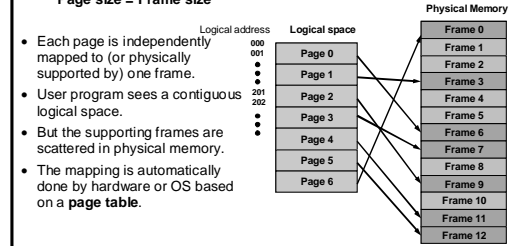*Dept of Computer Science and Engineering*
York University

Memory Management Approaches

- **Contiguous Memory Allocation**

- **Paging**

- **Segmentation**

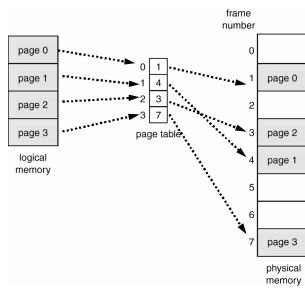- **Segmentation with paging**

Contiguous Memory Allocation suffers serious external fragmentation

---

Paging(1)

- **Logical space is contiguous and consists of pages**
- **Physical space is broken into frames**
- **Page size = Frame size**

- Each page is independently mapped to (or physically supported by) one frame.
- User program sees a contiguous logical space.
- But the supporting frames are scattered in physical memory.
- The mapping is automatically done by hardware or OS based on a **page table**.

1

Paging Example(1)

frame number

page 0
page 1
page 2
page 3
logical memory

page table

0	1
1	4
2	3
3	7

0
1 page 0
2
3 page 2
4 page 1
5
6
7 page 3
physical memory

Address Translation Architecture

- Convert logical address into page # and offset :
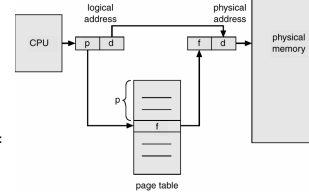 Logical address (X) = page number (p) + page offset (d).
- Assume page size k:
  p = X/k (quotient).
  d = X%k (remainder).

- p is used to index page table to find frame number or base physical address of this page.
- d is the offset in the mapped frame.

- The physical address Y:
  Y = f*k + d
  (f is frame number).

logical address

physical address

CPU   p d   f d   physical memory

p

f

page table

## Translation of logical address (for binary address)

- Page size (frame size) is typical a power of 2. (512k – 16M).
- Logical address is a concatenated bit stream of page number and page offset.
- An example: 1) logical space is 2**m: logical address is m bits.
  2) page size is 2**n: page offset is n bits.
  3) a logical space needs at most 2**(m-n) pages:
     page table contains at most 2**(m-n) elements
     page number needs (m-n) bits to index page table

page number	page offset
p	d
m-n bits	n bits
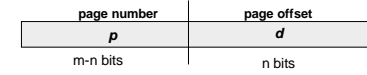
Given a binary logical address, the last n bits is page offset and the first m-n bits is page number.

2

Paging Example(2)

- Physical memory: 32-byte (2**5).
- Logical memory: 16-byte (2**4).
- Page size: 4-byte (2**2).
- Logical memory needs up to 4 pages: 4 entries in page table.
- m=4, n=2.

Logical address 9 : | 1 | 0 | 0 | 1 |
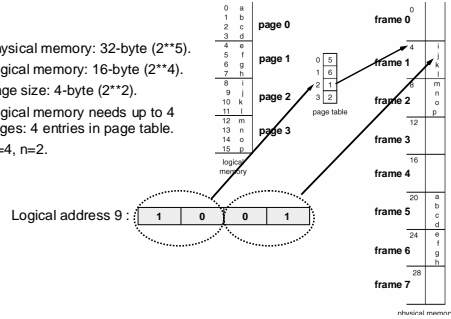
Paging (2)

- No external fragmentation in paging.
- Internal fragmentation: process size does not happen to fall on page boundaries.
  - Average one-half page per process.
- How to choose page size:
  - Smaller page size:
    - less internal fragmentation.
    - large page table (more overhead).
  - Typical 4K—8KB
- If each page table entry is 4 bytes long, it can point to one of 2**32 frames
  - Maximal physical address: frame size * (2**32)
    (from this we can deduce bit number in physical address)

Paging (3): memory allocation

- OS keeps track of all free frames.
- To run a program of size n pages, OS needs to find *n* free frames and load program.
- OS sets up a page table to translate logical to physical addresses.
- Each process has its page table and saved in memory pointed by its PCB.
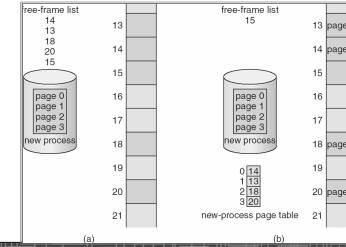
Paging(4)

- OS maintains a **frame table:**
  - One entry for each physical frame in memory.
  - To indicate the frame is free or allocated, if allocated, to which page of which process(es).
- OS maintain a copy of page table for each process in memory, pointed by PCB of this process.
  - Used to translate logical address in a process' address space into physical address.
  - Example: one process make an I/O system call and provide an address as parameter (logical address in user space). OS must use its page-table to produce the correct physical address.
- In context switch, the saved page-table is loaded by CPU dispatch to hardware page table for every memory reference. (copying page table increases context switch time)

Paging hardware

- For small page-table (<256 entries): using registers
- For large page-table: using two indexing registers
  - page table is kept in main memory.
  - *page-table base register (*PTBR) points to the page table.
  - *page-table length register* (PRLR) indicates size of the page table.
  - In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

Paging hardware: TLB

- Caching: using of a special fast-lookup hardware cache called *associative registers* or **translation look-aside buffers (TLBs)**
  - Associative registers (expensive) – parallel search
  - speedup translation from page # → frame # :
    - Assume page number is A:
      - -- If A is in associative register, get frame # out. (hit)
      - -- Otherwise get frame # from page table in memory (miss)
      - Save to TLB for next reference, replace an old if full

Page #	Frame #

Paging hardware with TLB
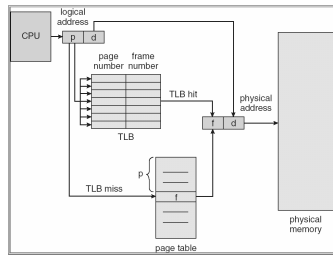
Need to flush TLB's in context switch

Effective Access Time of paging after TLB

- Assume memory cycle time is **a** time unit.
- One TLB Lookup = **b** time unit.
- Hit ratio – percentage of times that a page number is found in the associative registers; ration related to number of associative registers.
- Hit ratio = α.
- Effective Access Time (EAT):

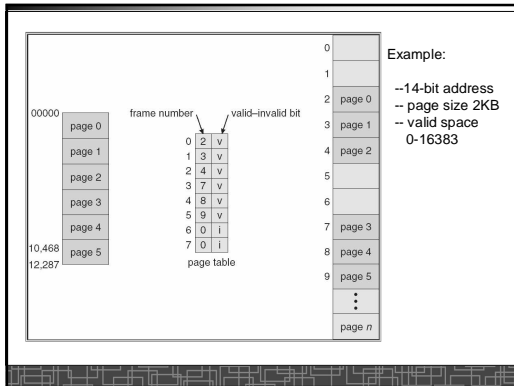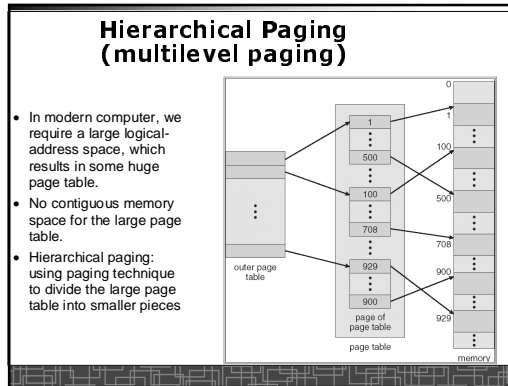$$EAT = (a + b)\,\alpha + (2a + b)(1 - \alpha)$$
$$= (2 - \alpha)a + b$$

Example: a = 100 nanoseconds, **b** = 20 nanosecond.

If α = 0.80, EAT = 140 nanoseconds (40% slower).
If α = 0.98, EAT = 122 nanoseconds (22% slower).

Memory Protection in paging

- Memory is protected among different processes.
- In paging, other process' memory space is protected automatically.
- Memory protection implemented by associating protection bits with each frame in page table
  – One bit for read-only or read-write
  – One bit for execute-only
  – One *Valid-invalid* bit
    • "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page.
    • "invalid" indicates that the page is not in the process' logical address space.
    • Use page-table length register (PTLR): to indicate the size of page table
    • *Valid-invalid* bit is mainly used for virtual memory
- In every memory reference, the protection bits are checked. Any invalid access will cause a trap into OS.

Slide 1

Example:

--14-bit address
-- page size 2KB
-- valid space
   0-16383

```
00000        frame number   valid–invalid bit
  page 0         0  2  v
  page 1         1  3  v
  page 2         2  4  v
  page 3         3  7  v
  page 4         4  8  v
                 5  9  v
10,468  page 5   6  0  i
12,287           7  0  i
              page table
```

```
0
1
2  page 0
3  page 1
4  page 2
5
6
7  page 3
8  page 4
9  page 5
   ⋮
   page n
```

Slide 2

Hierarchical Paging (multilevel paging)

- In modern computer, we require a large logical-address space, which results in some huge page table.
- No contiguous memory space for the large page table.
- Hierarchical paging: using paging technique to divide the large page table into smaller pieces

Slide 3

Address-Translation in two-level paging

- Logical address 32-bit, page size 4K, maximal physical address 2**32 frames
- A logical address is divided into 20 bits page number and 12 bits page offset.
- Since page-table is paged, the logical address is as follows:

page number		page offset
p_1	p_2	d
10	10	12

where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table.
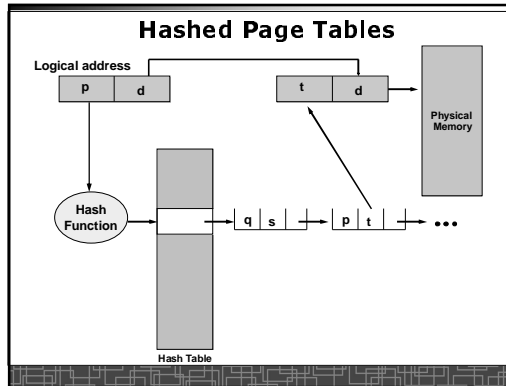
Multilevel Paging and Performance

- 64-bit logical address may require 7-level paging.
- Since each level is stored as a separate table in memory, converting a logical address to a physical one may take seven memory accesses.
- TBL-based caching permits performance to remain reasonable.
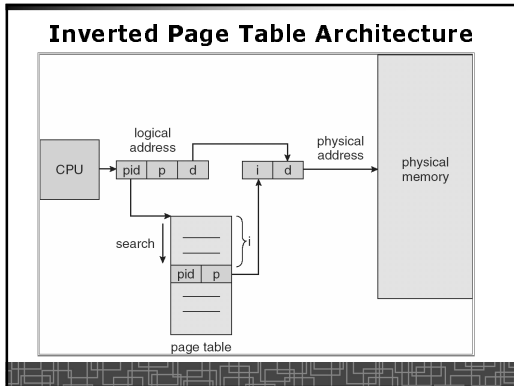- Cache hit rate of 98 percent yields:

  effective access time = 0.98 x 120 + 0.02 x 820

  = 134 nanoseconds.

  which is only 34 percent slowdown in memory access time.
- But the overhead is too high to maintain many page-tables
- For 64-bit, hierarchical page table is inappropriate.

Hashed Page Tables

Logical address

| p | d |

| t | d |

Physical Memory

Hash Function

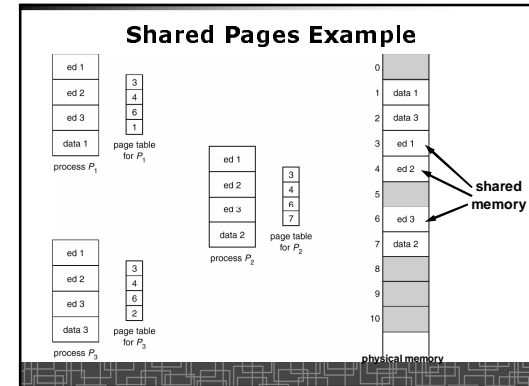| q | s | → | p | t | → ● ● ●

Hash Table

Inverted Page Table

- One entry for each real frame of memory.
- Each entry consists of the virtual page number stored in this frame, with information about the process that owns that page.
- Only one table in the system: decreases memory needed to store page tables.
- But increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one — or at most a few — page-table entries.
  - To speedup further, TLB is used.

Inverted Page Table Architecture

logical address
physical address
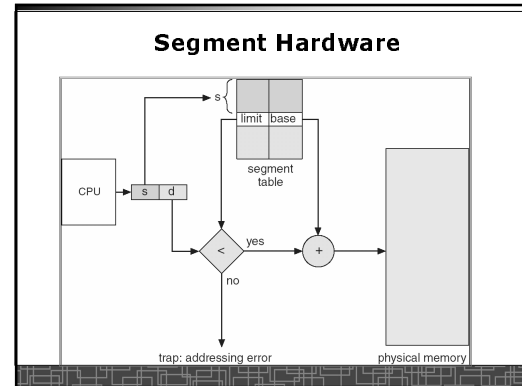
CPU → | pid | p | d | → | i | d | → physical memory

search

| pid | p |

page table

Shared Pages

- Different pages of several processes can be mapped to the same frame to let them share memory.
- Shared code
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes.
- Private code and data
  - Each process keeps a separate copy of the code and data.
  - The pages for the private code and data can appear anywhere in the logical address space.
- Shared-memory for inter-process communication
- Inverted page table has problems in sharing pages

Shared Pages Example

| ed 1 |
| ed 2 |
| ed 3 |
| data 1 |

process P_1

| 3 |
| 4 |
| 6 |
| 1 |

page table for P_1

| ed 1 |
| ed 2 |
| ed 3 |
| data 2 |

process P_2

| 3 |
| 4 |
| 6 |
| 7 |

page table for P_2

| ed 1 |
| ed 2 |
| ed 3 |
| data 3 |

process P_3

| 3 |
| 4 |
| 6 |
| 2 |

page table for P_3

0	
1	data 1
2	data 3
3	ed 1
4	ed 2
5	
6	ed 3
7	data 2
8	
9	
10	

shared memory

physical memory

8

User's view of a program

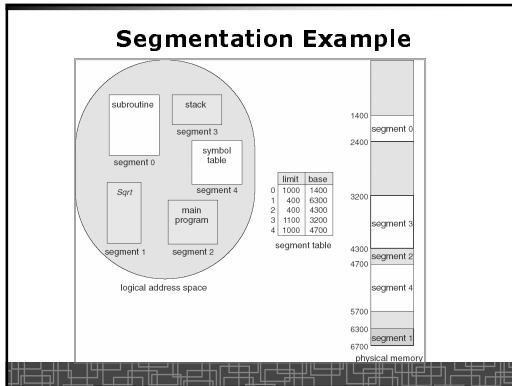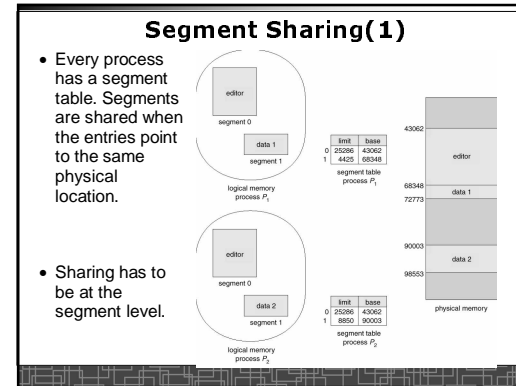logical memory space   physical memory space
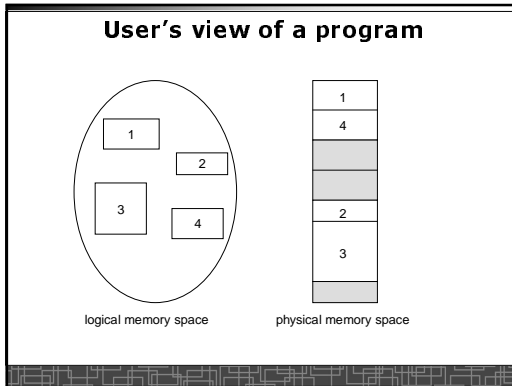
Segmentation

- A logical-address space is a collection of segments.
- Each segment has a name (segment number) and a length.
- A logical address consists of: a segment number and an offset.

  <segment-number (**s**), offset (**d**)>

- Physical memory address is still one-dimension linear array.
- Translation from logical (2-D) into physical (1-D) is based on a **segment table**.
  - Including all segments in the program.
  - Each entry has a segment base and a segment limit.
  - An array of base-limit pairs.
- The active segment table is pointed by two CPU registers:
  - *Segment-table base register (STBR)* points to the segment table's location in memory.
  - *Segment-table length register (STLR)* indicates number of segments used by a program; *segment number s is legal if s < STLR.*
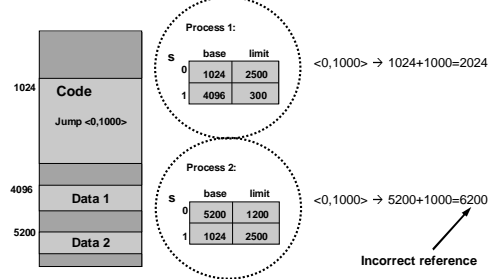
Segment Hardware

Segmentation Example

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

logical address space

physical memory

Segment Protection

- **Limit check for every memory reference.**

- **Each segment is a semantically defined portion of data. They tend to be used in the same way. We can define protection bits for every segment:**
  - **Read-only, read-write, and so on.**
  - **The segment hardware will check protection bits for each memory reference.**

Segment Sharing(1)

- Every process has a segment table. Segments are shared when the entries point to the same physical location.

- Sharing has to be at the segment level.

	limit	base
0	25286	43062
1	4425	68348

segment table process P_1

	limit	base
0	25286	43062
1	8850	90003

segment table process P_2

logical memory process P_1

logical memory process P_2

physical memory

10

Segment Sharing(2)

- How to share a code segment which has a direct address reference?

Process 1:

s	base	limit
0	1024	2500
1	4096	300

$<0,1000> \rightarrow 1024+1000=2024$

Code
1024
Jump $<0,1000>$

Data 1
4096

Data 2
5200

Process 2:

s	base	limit
0	5200	1200
1	1024	2500

$<0,1000> \rightarrow 5200+1000=6200$

Incorrect reference

Segment Sharing(3)

- If processes share a code segment with the direct address reference, all processes should have the same segment number for this segment.

- The following segments can be shared freely:
  - Read-only data segment.
  - Code segment with only indirect address reference (by offset from the current position or segment beginning).
  - Code segment with address relative to a register which contains the current segment number.

Fragmentation

- No internal fragmentation.

- External fragmentation
  - Since segments have various size.
  - Dynamic storage-allocation problem.
  - Best-fit, first-fit, worst-fit.
  - External fragmentation depends on average segment size.
  - If the average segment size is small, external fragmentation will also be small.
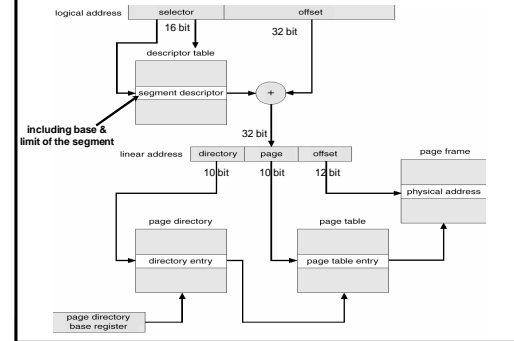
11

Segmentation with Paging

- Both segmentation and paging have advantages and disadvantages. We can combine them to improve on each.

- Two most popular CPU's, Motorola 68000 line and Intel 80x86 and Pentium uses a mixture of paging and segmentation.

- Example: Intel Pentium uses segmentation with paging for memory management.
  – Based on segmentation primarily.
  – The varying-length segments are paged into a set of fixed-sized pages.
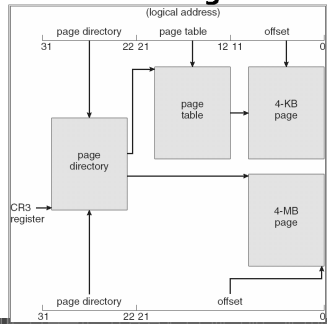
Intel Pentium addressing

- A process can have up to 16KB (2**14) segments, divided into two segment tables:
  – Local descriptor table (LDT)
  – Global descriptor table (GDT)
  – Each entry in the tables is 8 bytes (base+length+others).
- Each segment can be 4GB (2**32) in maximum.
- A logical address is 48 bits, consists of:
  – 16 bits selector: 13-bit segment number, 1-bit indicate LDT or GDT, 2-bit for protection.
  – 32 bits segment offset: a segment can be up to 2**32 bytes
  – Each segment is paged: page size 4KB & 2-level paging:
    10-bit page directory # +10-bit page # + 12-bit page offset
- CPU has six segment registers (caches), allowing 6 segments to be addressed at any time (avoid reading descriptor for each memory reference).
- In Pentium, physical address is 32-bit (max 4GB).

Pentium Addressing Architecture

12

Pentium Addressing Architecture

(logical address)

| page directory | page table | offset |

```
31        22 21        12 11        0
```

page directory

page table

4-KB page

CR3 register

4-MB page

| page directory | | offset |

```
31        22 21              0
```

Comparing Memory-Management Strategies

(1)Contiguous allocation, (2)paging, (3)segmentation, (4)Segmentation with paging

- Hardware support
- Performance
- Fragmentation
- Relocation
- Swapping
- Sharing
- Protection

13