



Main Memory (II)

Operating Systems

Yu-Ting Wu

(with slides borrowed from Prof. Jerry Chou)

1

Operating Systems 2022

Outline

- Paging (cont.)
- Segmentation
- Segmentation with Paging

2

2

Operating Systems 2022

Paging (cont.)

3

3

Operating Systems 2022

Memory Protection

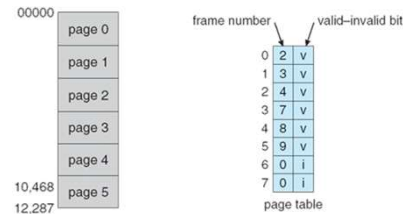
- Each page is associated with a set of **protection bit** in the page table
 - E.g., a bit to define read/write/execution permission
- Common use: **valid-invalid bit**
 - **Valid**: the page / frame is in the process' logical address space, and is thus a legal page
 - **Invalid**: the page / frame is not in the process' logical address space (e.g., virtual memory)

4

4

Valid-Invalid Bit Example

- Potential issues
 - Un-used page entry cause memory waste
→ use **page table length register (PTLR)**
 - Process memory may not be on the boundary of a page
→ memory limit register is still needed



5

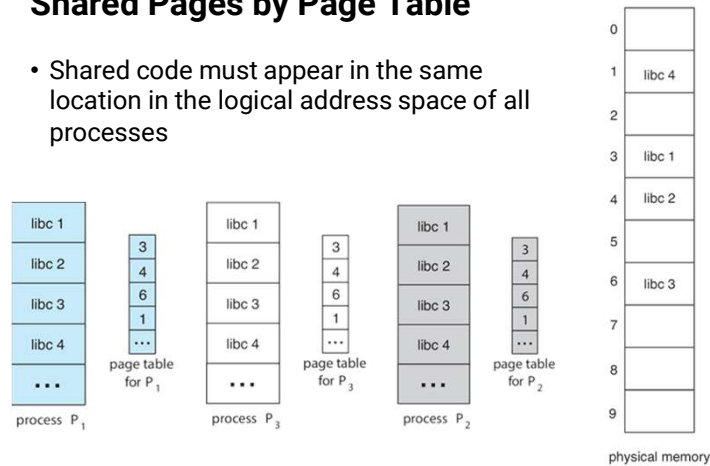
Shared Pages

- Paging allows processes **share** common code, which must be **reentrant**
- Reentrant code (pure code)
 - It never change during execution
 - Text editors, compilers, web servers, etc.
- Only one copy** of the shared code needs to be kept in physical memory
- Two (several) virtual addresses** are mapped to one physical address

6

Shared Pages by Page Table

- Shared code must appear in the same location in the logical address space of all processes



7

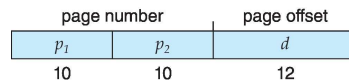
Page Table Memory Structure

- Page table could be huge and difficult to be loaded
 - 4GB (2^{32}) logical address space with 4KB (2^{12}) page
→ 1 million (2^{20}) page table entry
 - Assume each entry need 4 bytes (32 bits)
→ Total size = **4MB**
 - Need to break it into several smaller page tables, better within a single page size (i.e. 4KB)
 - Or reduce the total size of page table
- Solutions
 - Hierarchical paging
 - Hash page table
 - Inverted page table

8

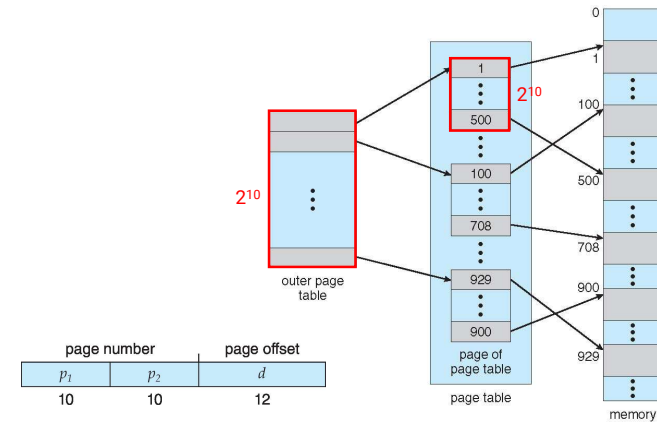
Hierarchical Paging

- Break up the logical address space into **multiple page tables**
 - Paged the page table
 - i.e. n -level page table
- Two-level paging (32-bit address with 4KB (2^{12}) page size)
 - 12-bit offset (d) \rightarrow 4KB (2^{12}) page size
 - 10-bit **outer** page number \rightarrow 1K (2^{10}) page table entries
 - 10-bit **inner** page number \rightarrow 1K (2^{10}) page table entries
 - 3 memory accesses



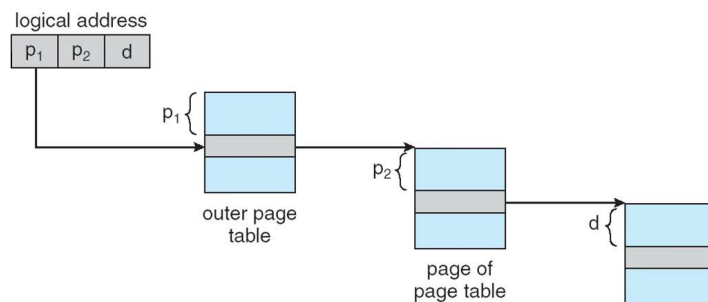
9

Two-Level Page Table Example



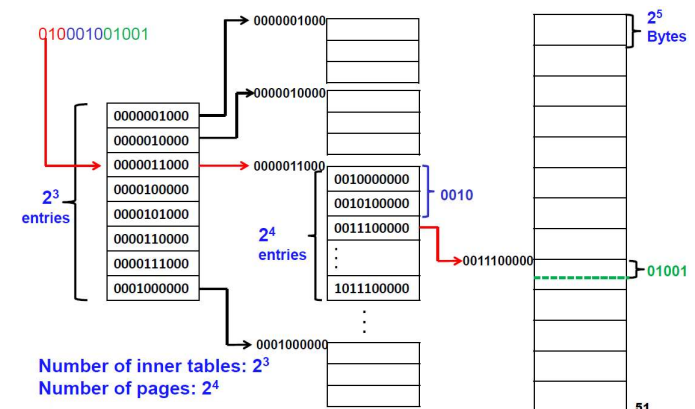
10

Two-Level Address Translation



11

Two-Level Address Translation Example



12

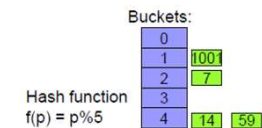
64-bit Address

- How about 64-bit address? (assume each entry needs 4 Bytes)
 - $42(p1) + 10(p2) + 12(\text{offset})$
 - outer table requires $2^{42} \times 4B = 16TB$ contiguous memory
 - $12(p1) + 10(p2) + 10(p3) + 10(p4) + 10(p5) + 12(\text{offset})$
 - outer table requires $2^{12} \times 4B = 16KB$ contiguous memory
 - 6 memory accesses !!!
- Examples
 - SPARC (32-bit) and Linux use 3-level paging
 - Motorola 68030 (32-bit) uses 4-level paging

13

Hashed Page Table

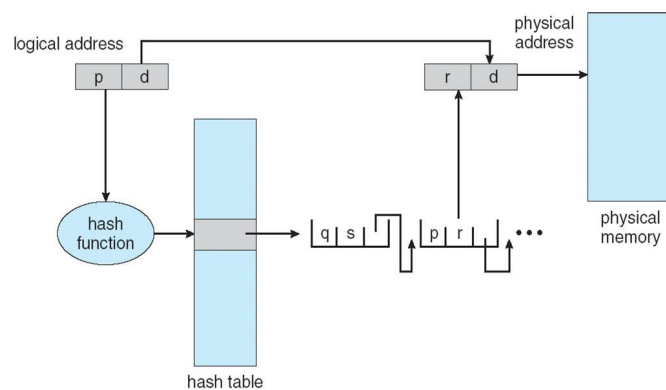
- Commonly-used for address > 32 bits
- Virtual page number is hashed into a hash table
- The size of the hash table varies
 - Larger hash table → smaller chains in each entry
- Each entry in the hashed table contains
 - (Virtual Page Number, Frame Number, Next Pointer)
 - Pointers waste memory
 - Traverse linked list waste time and cause additional memory references



14

14

Hash Page Table Address Translation



15

15

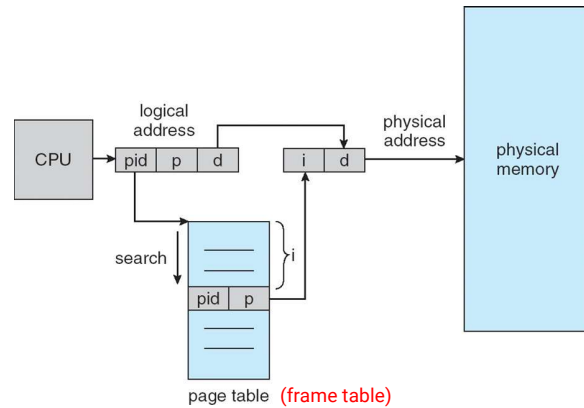
Inverted Page Table (Frame Table)

- Maintains **no** page table for each process
- Maintains a **frame table** for the whole memory
 - One entry for each real frame of memory
- Each entry in the hashed table contains
 - (PID, Page Number)
- Eliminate the memory needed for page tables but **increase memory access time**
 - Each access needs to search the whole frame table
 - Solution: use hashing for the frame table
- Hard to support shared page / memory

16

16

Inverted Page Table Address Translation



17

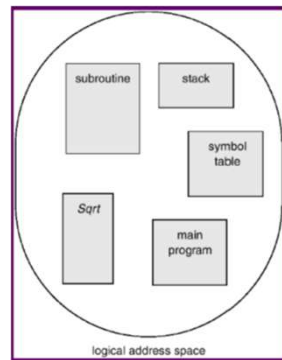
Segmentation

18

18

Segmentation

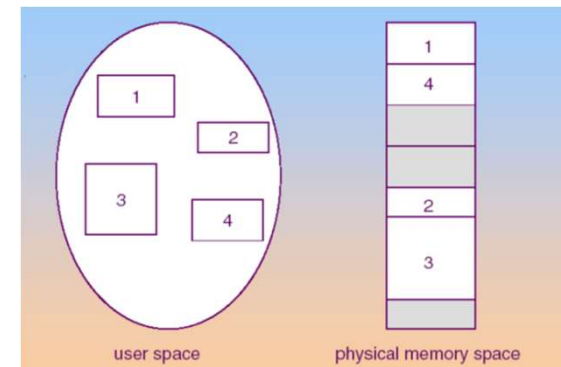
- Memory-management scheme that supports **user view of memory**
- A program is a collection of segments
- A segment is a logical unit such as
 - Main program
 - Function
 - Object
 - Local/global variables
 - Stack
 - Symbol table
 - Arrays



19

19

Logical View of Segmentation



20

20

Segmentation Table

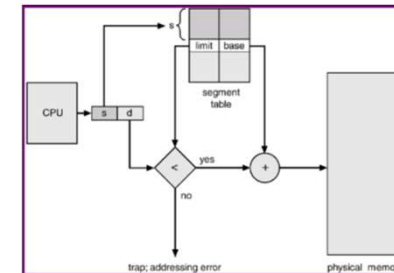
- Logical address: (seg#, offset)
 - Offset is only limited by physical address
- **Segmentation table**
 - Maps two-dimensional physical addresses
 - Each table entry has
 - **Base (4 bytes)**: the start physical address
 - **Limit (4 bytes)**: the length of the segment
- **Segment-table base register (STBR)**
 - The physical address of the segmentation table
- **Segment-table length register (STLR)**
 - The number of segments

21

21

Segmentation Hardware

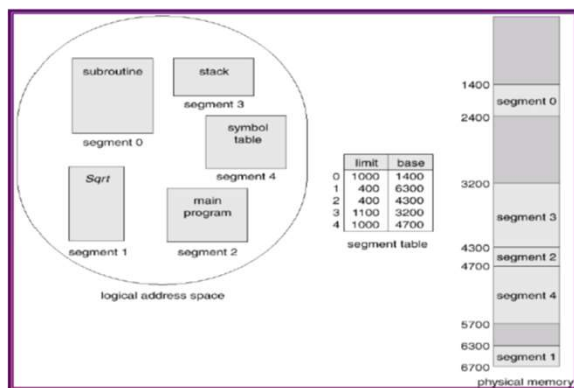
- Limit register is used to check offset length
- MMU allocate memory by assigning an appropriate **base address for each segment**
 - Physical address cannot overlap between segments



22

22

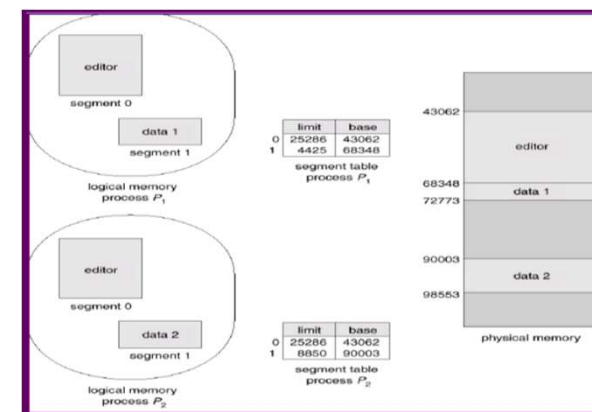
Example of Segmentation



23

23

Sharing of Segments



24

24

Protection and Sharing

- Protection bits associated with segments
 - Read-only segment (code)
 - Read-write segments (data, heap, stack)
- Code sharing occurs at **segment level**
 - Shared memory communication
 - Shared library
- Share segment by having same base in two segment tables

25

25

Segmentation v.s. Paging

	Paging	segmentation
Length	Fixed	Varied
Fragmentation	Internal	External
Table entry	Page number → frame number	Seg ID → (base addr, limit length)
View	Physical memory	User program

26

26

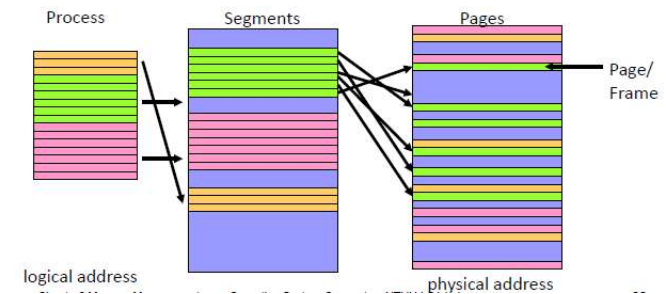
Segmentation with Paging

27

27

Basic Concept

- Apply **segmentation** in **logical** address space
- Apply **paging** in **physical** address space

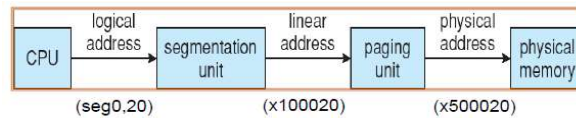


28

28

Address Translation

- CPU generates logical address
 - Given to **segmentation unit**
 - Produces **linear addresses**
 - Linear address given to paging unit
 - Generates **physical address** in main memory
- Segmentation and paging units form equivalent of MMU



29

29

Objective Review

- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques

30

30