Segmentation and Paging (Topic 6)

홍 성 수

서울대학교 공과대학 전기 공학부
Real-Time Operating Systems Laboratory

Segmentation (1)

- Problem with base and bound relocation:
  - Only one segment: How can two processes share code while keeping private data areas (e.g., shared editor)?
- Multiple segments.
  - Permit process to be split between several areas of memory.
  - Use a separate base and bound for each segment, and also add two protection bits (read and write).

Segmentation (2)

- Each memory reference indicates a segment and offset in one or more of three ways:
  - Top bits of address select segment, low bits the offset.
  - Segment is selected implicitly by the instruction
    - Examples: Code vs. data, stack vs. data, or 8086 prefixes.
- Segment table holds the bases and bounds for all the segments of a process.
- Memory mapping procedure consists of table lookup + add + compare.
  - Example: PDP–10 with high and low segments selected by high-order address bit.

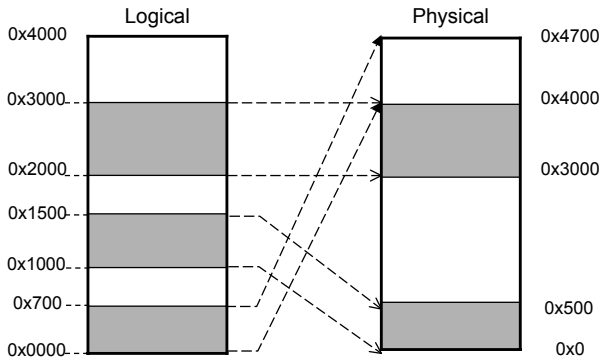
Segmentation (3)

- Segmentation example: 2-bit segment number, 12-bit offset:

Segment	Base	Bounds	RW
0	0x4000	0x6FF	10
1	0x0	0x4FF	11
2	0x3000	0xFFF	11
3			00

Segmentation (4)

- Address space mapping:



Segmentation (5)

- Where is 0x240, 0x1180, 0x265c? How about 0x3002, 0x1600?
- Managing segments:
  - Keep copy of segment table in process control block.
  - When creating process, allocate space for segment, fill in PCB bases and bounds.
  - When process dies, return segments to free pool.

Segmentation (6)

- When there's no space to allocate a new segment:

  Compact memory (move all segments, update bases) to get all free space together.

  Or, swap one or more segments to disks to make space and then bring segments back in before letting process run.

  - Must then check during context switching and bring segments back in before letting process run.

Segmentation (7): Pros and Cons

- To enlarge segment:
  - See if space above segment is free. If so, just update the bound and use that space.
  - Or, move the segment above this one to disk, in order to make the memory free.
  - Or, move this segment to disk and bring it back into a larger hole (or, maybe just copy it to a larger hole).

Segmentation (8): Pros and Cons

- Advantage of segmentation:
  - Segments can be swapped and assigned to storage independently.
- Problems:
  - External fragmentation:
    - Segments of many different sizes, have to be allocated contiguously.

    This problem also applies to base and bound schemes.

Paging (1)

- Paging: Goal is to make allocation and swapping easier, and to reduce memory fragmentation.
  - Make all chunks of memory the same size, call them pages.

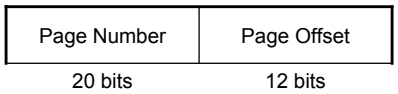
    Typical sizes range from 512-16K bytes.
  - For each process, a page table defines:

    The base address of each of that process' pages.

    Protection (read-only) and existence bits.

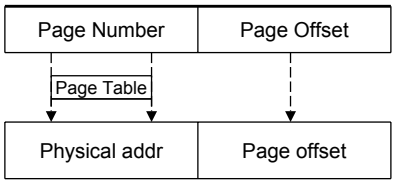
Paging (2)

- Translation process: Page number always comes directly from the address. Example (SPARCstation):



Paging (3)

- Since page size is a power of two, no comparison or addition is necessary. Just do table lookup and bit substitution.



Paging (4): Pros and Cons

- Easy to allocate:
  - Keep a free list of available pages and grab the first one.
  - Easy to swap since everything is the same size, which is usually the same size as disk blocks to and from which pages are swapped.

Paging (5): Pros and Cons

- Problems:
  - Efficiency of access:
    - Even small page tables are generally too large to get loaded into fast memory in the MMU.
    - Page table are kept in main memory and the MMU has only the page table's base address.
    - It thus takes one overhead reference for every real memory reference.

Paging (6): Pros and Cons

- Table space:
  - If pages are small, the table space could be substantial.
    - Example: 32-bit address space with 1k pages.
    - Page table size = 16 Megabytes
    - What if the whole table has to be present at once ?
      - Partial solution: keep base and bounds for page table, so only large processes have to have large tables.
- Internal fragmentation: Page size doesn't match up with information size. The larger the page, the worse this is.

Paged Segmentation (1)

- Paging and segmentation combined: Use two levels of mapping to make tables manageable.
  - Each segment contains one or more pages.
  - Segments correspond to logical units: Code, data, stack.
    - Segments vary in size and are often large.
  - Pages are for the use of the OS; they are fixed-size to make it easy to manage memory.

Paged Segmentation (2)

– Going from paging to P+S is like going from single segment to multiple segments, except at a higher level.

Instead of having a single page table, have many page tables with a base and bound for each. Call the stuff associated with each page table a segment.

Paged Segmentation (3)

- System 370 example: 24-bit virtual address space; 4 bits of segment number, 8 bits of page number, and 12bits of offsets.

Seg # (4bits)	Page #(8 bits)	Page offset (12bits)

– Segment table contains real address of page table along with the length of the page table (a sort of bounds register for the segment). Page table entries are only 12bits long, while real addresses are 24 bits long.

Paged Segmentation (4)

- Example of S/370 paging:

Segment table

Base	Bound	Prot
0x2000	0x14	R
0x0000	0x00	
0x1000	0x0D	RW

Memory

0x001F	
0x0011	0x2020
...	
0x0003	
0x002A	
0x0013	0x2000
0x000C	
0x0007	0x1020
...	
0x0004	
0x000B	
0x0006	0x1000

Paged Segmentation (5)

- Map the following addresses from virtual to physical
  - 0x002070 read:
    Seg 0, page 2; PTE @ 0x002004; addr = 0x003070
  - 0x202016 read:
    Seg 2, page2; PTE @ 0x001004; addr = 0x004016
  - 0x104C84 read:
    Seg 1, page4; PTE @ (Protecton Error).
  - 0x011424 read:
    Seg 0, page 17; PTE @ 0x002022; addr = 0x01f424
  - 0x210014 write:
    Seg 2, page 16; PTE @ (Bounds violation)

Paged Segmentation (6)

- If a segment isn't used, then there's no need to even have a page table for it.
- Can share at two levels:
  - Single page.
  - Single segment (whole page table).
- Pages eliminate external fragmentation, and make it possible for segments to grow without any reshuffling.

Paged Segmentation (7)

- If page size is small compared to most segments, then internal fragmentation is not too bad.
- The user is not given access to the paging tables.
- If translation tables are kept in main memory, overhead could be very high:
  - 1 or 2 overhead references for every reverence.

Paged Segmentation (8)

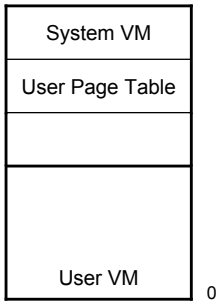
- Another example: VAX.
  - Address is 32 bits, top two select segment.
    Four base-bound pairs define page tables:
    - System, P0, P1, unused.
    - One segment contains operating system stuff, two contain stuff of current user process.
  - Read-write protection information is contained in the page table entries, not in the segment table.
  - Pages are 512 bytes long.

Paged Segmentation (9)

- Solution is to use the system page table to map the user page tables so the user page tables can be scattered:
  - System base-bounds pairs are physical addresses, system tables must be contiguous.
  - User base-bounds pairs are virtual addresses in the system space. This allows the user page tables to be scattered in non-contiguous pages of physical memory.

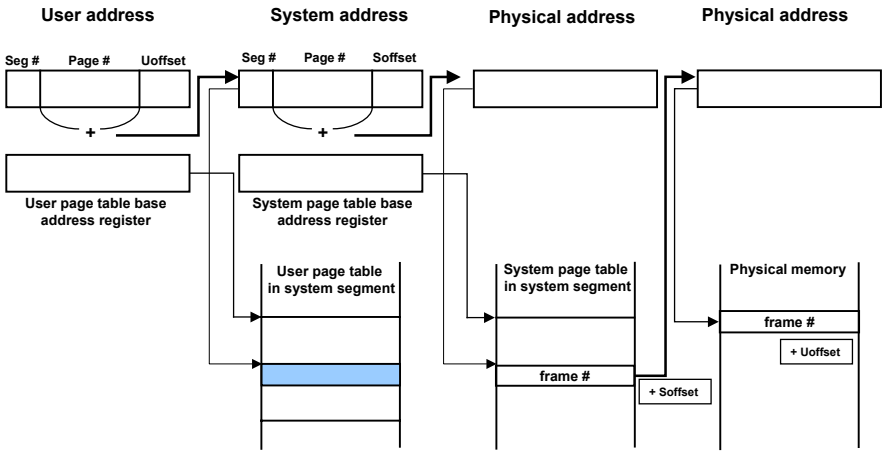
Paged Segmentation (10)

– The result is a two level scheme.



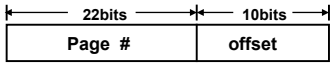
1) User generates address.
2) Lookup in User Page Table.
3) Lookup in System Page Table.
4) Access physical address.

Paged Segmentation (11)

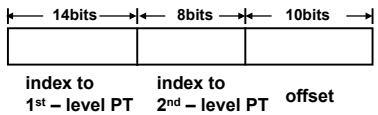


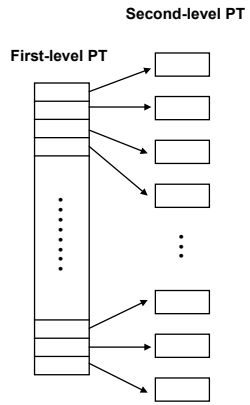
Paged Segmentation (12)

- Multi-level page table



– Size of PTE: 4byte
– Size of PT: 2^{22} X 4byte = 16MB
– No of PTEs in a page: 256 entries
– No of pages in PT: 16K
– Size of first-level PT: 16K X 4byte = 64KB
–





Paged Segmentation (13)

- Problem with segmentation and paging: Extra memory references to access translation tables can slow programs down by a factor of two or three.
  – Too many entries in translation tables to keep them all loaded in fast processor memory.
- Remember notion of locality: At any given time a process is only using a few pages or segments.

Paged Segmentation (14)

- Solution: Translation Look-aside Buffer (TLB).
  - A translation buffer is used to store a few of the translation table entries. It's very fast, but only remembers a small number of entries.
  - On each memory reference:

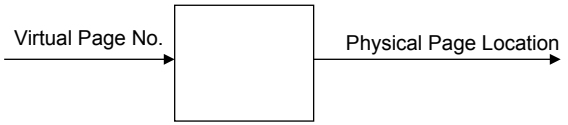
    First ask TLB if it knows about the page.

    If so, the reference proceeds fast.

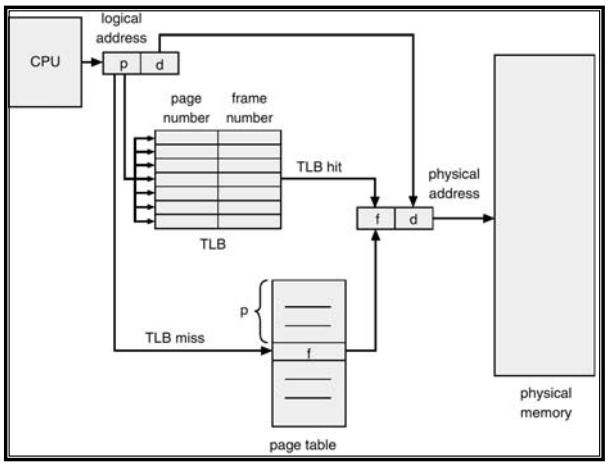
Paged Segmentation (15)

- If TLB has no info for page, MMU must go through page and segment tables to get info. Reference takes a long time, but give the info for this page to TLB so it will know it for next reference. (TLB must forget one of its current entries in order to record new one).

  Also called a Translation Buffer (TB)



Paging Hardware With TLB



Effective Access Time

- Associative Lookup = ε time unit
- Assume memory cycle time is 1 microsecond
- 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)

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

Paged Segmentation (16)

- TLB is just a memory with some comparators.
  - Typical sizes of memory: 64 to 2K entries.
  - Each entry holds a virtual page number and the corresponding physical page number.
- How can memory be organized to find an entry quickly?
  - One possibility: Search whole table from start on every reference.

Paged Segmentation (17)

  - Another possibility: Restrict the info for any given virtual page to fall in exactly one location in the memory. Then only need to check that one location. E.g. use the low-order bits of the virtual page numbers as the index into the memory. Like an array.
  - Another possibility: Check all entries in parallel.
    - Expensive but fast.

Paged Segmentation (18)

- TLBs are a lot like hash tables except simpler (must be implemented in hardware). Some hash functions are better than others.
  - Is it better to use low page number bits than high ones ?
    - Low ones are best: If a large contiguous chunk of memory is being used, all pages will fall in different slots.
  - Is there any way to improve on the TLB hashing functions?
    - Use bits of the segment in the hash.

Paged Segmentation (19)

- Another approach: let any given virtual page use either of two slots in the TLB. Make memory wider, use two comparators to check both slots at once.
  - This is about as fast as the simple scheme, but a bit more expensive (two comparators instead of one, also have to decide which old entry to replace when bringing in a new entry).

Paged Segmentation (20)

- Advantage: Less likely that there will be conflicts that degrade performance (takes three pages falling in the same place, instead of two).
- Explain names:

  (1) Direct mapped.

  (2) Set associative.

  (3) Fully associative.

Paged Segmentation (21)

- TLB can be mostly hidden from the operating system.
  - Possible exception: Context switches.
- Must either:
  - Flush TLB during each context switch.
  - Or, store a Process ID (PID) in the TLB entry.
- Example of a TLB: MIPS R2000/R3000
  - CPU used in DecStations, and SGI machines.
  - Addresses are 32 bits: 12 bit page offset. (i.e. 4K pages)
  - TLB entry format: (64 bits)
  - 64 TLB entries.

Paged Segmentation (22)

20	6	6	20	1	1	1	1	8
VPN	PID	0	PFN	N	D	V	G	0

G — Global, valid for any PID.
V — Entry is valid.
D — Dirty bit, page has been modified.
N — Don't cache the page.
PFN — Physical address of the page.
PID — Process ID for entry (or called ASID).
VPN — Virtual page number.

Paged Segmentation (23)

- In practice, TLB's have been extremely successful:

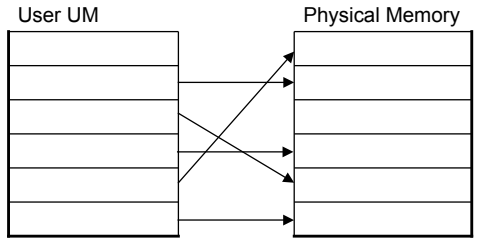
  98% ratio is typical for 128 entries.

Paged Segmentation (24)

- Problems: How does the operating system get information from user memory ?
  - Example: I/O buffers, parameter blocks.
  - Note that the user passes the OS virtual addresses.

Paged Segmentation (25)

- Note addresses that are contiguous in the virtual address space may not be contiguous physically. I/O operations may have to be split up into multiple blocks.



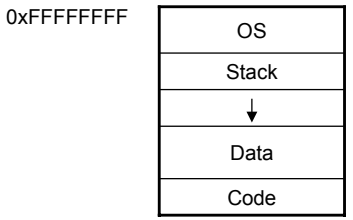
Paged Segmentation (26)

- Possible solutions:

  (1) Run the OS unmapped.
  - OS read the page tables and translate user addresses in software.
  - I/O operations may have to be split up into multiple blocks.

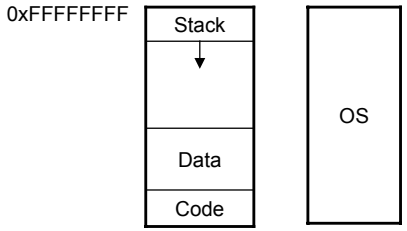
Paged Segmentation (27)

(2) Translate both OS and I/O addresses thru the TLB.
- (Ex) SunOS on a SPARC Station (SPARC V7)
- Both system and user information visible at once (but can't touch system stuff unless running with special protection bit set
- IO devices DMA into virtual addresses.



Paged Segmentation (28)

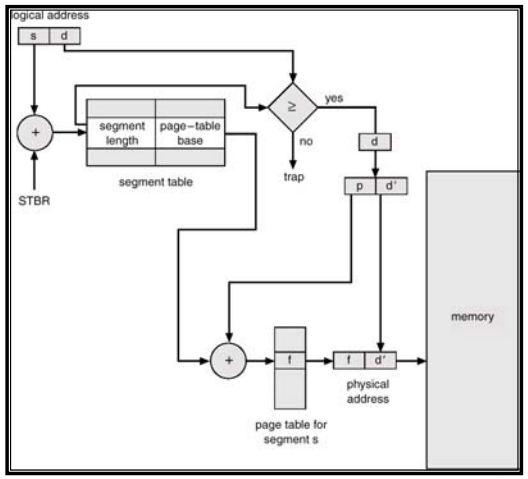
- Suppose the operating system also runs mapped in a different address space than the user. Then it must generate a page table entry for the user area. Some machines provide special instructions to get the user stuff. Note that under no circumstances should users be given access to mapping tables. (sun4u SPARC V9)



Case Study: Multics (1)

- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.
- Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a *page table* for this segment.

Case Study: Multics (2)



Case Study: Intel 386 (1)

- As shown in the following diagram, the Intel 386 uses segmentation with paging for memory management with a two-level paging scheme.

Case Study: Intel 386 (2)