[redacted]

Lecture[redacted] Virtual Memory Management

[redacted]

[redacted]

Review Questions:

1. Explain the difference between internal and external fragmentation. What is the most internal fragmentation if address spaces > 4kB, 4kB <= page size <= 8kB.
   ➔ Internal: When page wastage occurs inside a page due to functions such as malloc() and free(). External: Between memory units (pages) assigned to different processes.
   Larger the page size, more is the internal fragmentation.

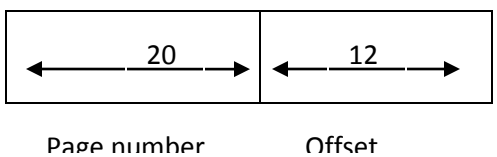
2. Consider a paging system with the page table stored in memory
   If a memory ref. takes 200 nsec, how long does a paged memory ref. take?
   If we add TLBs, and TLB hit ratio is 90%, what is the effective mem. ref. time? Assume TLB access takes 0 time
   ➔ Paged mem reference takes twice the time (one for page table access and another for the memory access itself). So total of 400 nsec.
   If TLB hit ratio is 90 % then
   Eff mem ref time = 0.9(200) + 0.1(400) = 220nsec

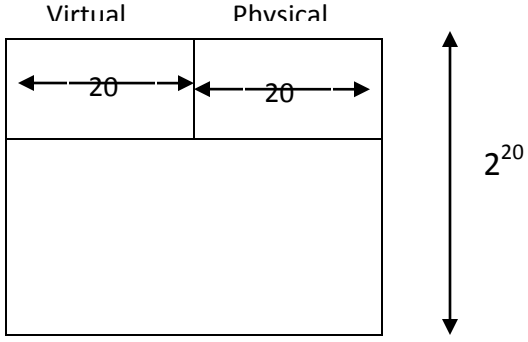
   Paging is useful but downsides are: translations have to be kept in main memory. Page tables shd :
   i) Occupy as little main memory (spatial constraint)
   ii) Accesses should be fast (temporal)

   There is one page table for each process. It is not like PCB because the size of page table for each process can vary.

3. How much memory is needed to store an entire page table for 32-bit address space if
   Page size = 4kB, single-level page table
   Page size = 4kB, 3-level page table, p1=10, p2=10, d=12 (Ignore this for now)

   For addressing 4Kb, we need 12 bits of address space.





4kB = 2^{12}

Hence number of pages, = $2^{32}/2^{12}=2^{20}$

Size of page table = 2^{20} x 40/8 Bytes = 5MB

Although VMM needs extra memory, why is it still feasible?

➔ Because most processes donot use full address spaces. Hence not all the frames in physical memory are used. So page table size is limited.

Page table resides in the OS portion of the address space. Address allotted for the OS is limited. Typically linux gives 1GB for OS and 3GB address space for processes. So there is a restriction on the page table size.

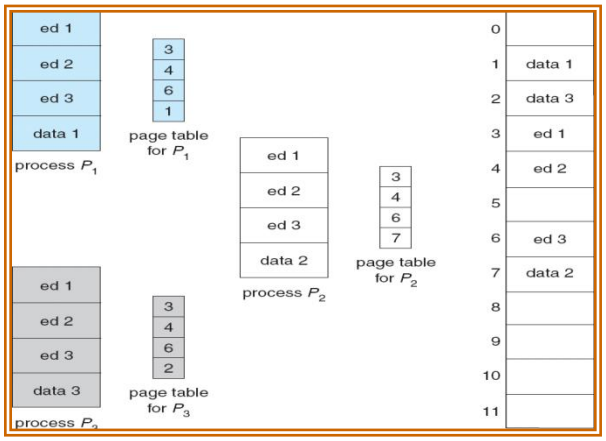
Additional Information Stored in Page Tables

1. Protection Information: Valid Invalid Bit. This bit indicated whether the entry in the page table has been actually assigned a frame in the physical address. Note that processes do not use their address spaces completely, so some entries might be invalid.
   TLB should also have the valid/invalid bit

   How should the TLB and page list search algorithm change?
   If invalid address is fed in MMU then it will look up in TLB, if it finds translation then check the valid/invalid bit… If invalid then MMU should generate a trap resulting in segmentation fault.

2. Certain Permissions: Read/write permissions for pages just like permissions for files. When do we need to declare the pages as read only?

Consider a scenario:



Sometimes same application is shared by multiple processes. So the code pages are the same only the private data pages are different. In above figure, the editor code for all the three processes share the same three frames in physical addresses. So protection is needed so that one process does not modify the code

3. Address space identifiers (ASID):
   Recall how TLB works:

   Consider a scenario where two processes are sharing CPU. The zeroth page of P1 mapped to frame number 0. The seroth page of P2 is mapped to frame 6 in physical address.

   1. Consider P1 is running.. In the TLB one of the entries maps zeroth page to zeroth frame. CPU generates address with virtual address in page 0.
   2. MMU checks if valid or not checking first in the TLB .
   3. Suppose mapping not in TLB make a long trip to the main memory.
   4. TLB now populated with (0,0)
   5. Suppose if process is context switched out.
   6. Now P2 issues load/store instruction in address space with virtual page number 0.
   7. MMU looks into TLB and finds a match (0,0)

INCORRECT ADDRESS TRANSLATION !!!!!!!!!

What can be done?

1. Flushing TLB with every context switch.
   Pros: This is simple approach.
   Cons: As each new process is context switched in the TLB is empty. So performance of system is slow in the beginning, but picks up as TLB is slowly populated

2. TLB can be made to store addresses of both the processes. TLBs are then indexed by processes.

Address Space Id →

Process	Virtual	Physical
P1	0	0
P2	0	6

Address Space ID known as ASID

Pros: Process need not be slowed down

Cons: Makes TLB more complex.

Leads to software managed TLBs. Where TLBs can be configured via software.

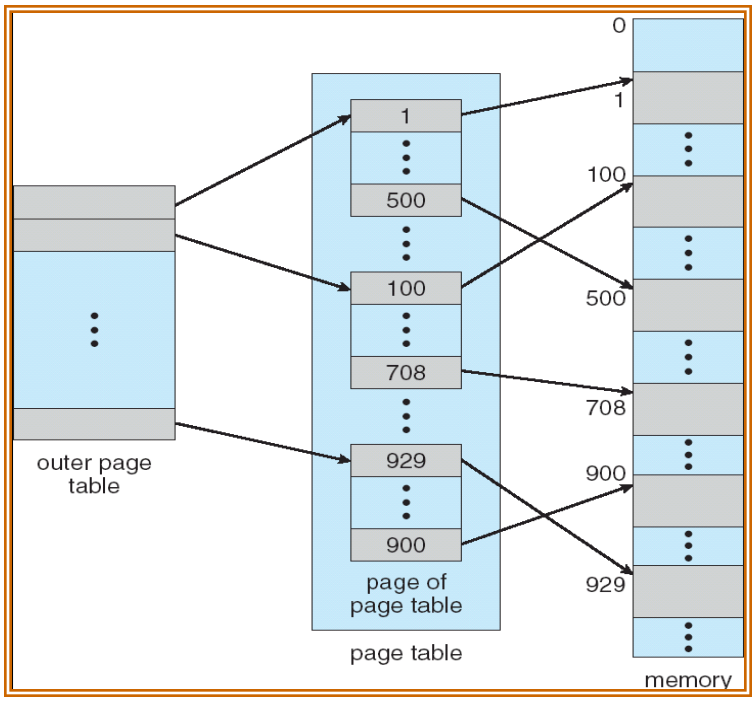
Page Table Implementation

Hierarchical Page Table:

Consider a page table where the prefix are 4 bit in size

Page No	Prefix in Virtual Address	Physical Page Nos
0	0000	15
1	0001	22
2	0010	8
3	0011	5
4	0100	12
5	0101	11
6	0110	3
7	0111	1

Note that we can exploit the fact that for first 4 addresses, in the address space the first two bits are "00". Similarly for the next four addresses and so on.. So we can have a two level hierarchy where, the first hierarchy is indexed by the last two bits and contains the physical page nos and the first level of hierarchy is indexed by first two bits contains the address to the table of next hierarchy.


0
1
100
500
708
900
929
1
500
100
708
929
900
outer page table
page of page table
page table
memory