Limitations of segmentation

- External Fragmentation
- Segment has to be contiguous in memory
  - Growing/Shrinking: Could result in relocation
- Segment larger than memory size cannot be supported efficiently
  - Segments could be broken into multiple segments
  - But the number of segments is limited by the number of segment registers

Paging

- Address space is divided into units called pages (similar to how address space was divided into segments)
- Page size is always defined (Though there is a maximum limit, segments size could vary)
- Physical memory is also split into page units
- Pages in address space is called as virtual pages and that of physical memory is called as physical pages

Address Translation

- Like all the methods, paging also needs a way to translate address from VA to PA
- Given a VA, split them into two parts: (a) Offset: Low order bits (b) VPN (Virtual page Number): High order bits. (Think of the similarity with respect to addressing in segmentation)
- Number of bits for offset is determined by the page size. (1K = 10 bits, 4K = 12 bits)
- We need a mapping from the VPN to PFN (Page frame number - index number of physical pages)
- The mapping table is called page table

Page Table

- Each process has its own page table
- Since page table could be large, it cannot be stored in MMU. So, page table is stored in memory and the starting address of the page table is given to MMU. This gets stored in PTBR (Page table base register) in MMU.
- Simple form of page table is linear page table where mapping between VPN to PFN exists and every such entry is called PTE (Page Table Entry)

Protection through Page Table

- Segmentation had the bound check and protection bits mechanism
- Every PTE contains the following information
  - PFN: Page frame number
  - Valid bit: If the translation between VPN to PFN is valid
  - Access bit: Type of access allowed - Read or write
  - Dirty bit: If the page is modified by the process
  - Present bit: If the page is present in memory or not (else in swap device)

Linear Page Table Efficiency
• Space
  o Size of page table can be huge (assume the linear page table structure)
  o Assume 32-bit address bits, 4 GB of address space is possible
  o Given size of each page is 4K, then there could $2^{20}$ pages.
  o Assume every PTE entry to be 4 bytes, then size of page table
    ▪ Maximum number of pages supported * size of PTE
    ▪ $2^{20} \times 4 = 4$ MB
  o If there are N processes in the system, then page table for all processes consume ($N \times 4$ MB).

• Access Latency
  o Page table is used during address translation
  o But since page table is stored in memory, every translation would incur an additional memory access (access to the page table)

Design of Hardware mechanism

• All translation is done in hardware for the translation to be really fast
  o Split VPN, Offset
  o Use VPN to fetch PTE (involves memory reference since page table is in memory)
  o Get PFN from PTE
  o Make appropriate access checks
  o $PA = PFN + Offset$
• However, accessing memory is going to slow down the process in case of linear page table
• We need to look at better page table structures to make this fast

Questions

• How does page size have an impact on fragmentation?
• Under what assumptions, segmentation is similar to paging?
• What is the equivalent of bound check in paging?
• How do you prevent user mode programs from accessing pages of OS?