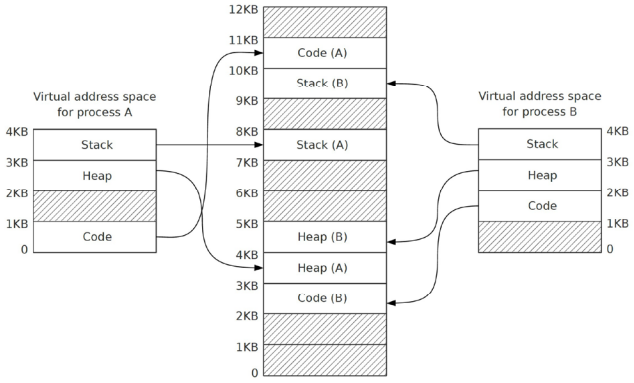
Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE7376: Operating Systems: Interface and Implementation

Homework 3

Problem 1 (25 Points)

In a memory segmentation operating system, consider a machine with a 12KB physical memory space running an OS that uses a 4KB virtual memory space for its processes, where a 1KB means 1024 bytes. The machine is executing two processes (***A*** and ***B***), where each process has 3 allocated segments with the following mappings:



a) Show the state of a basic segment table containing the information represented in the diagram above. The table should include the following columns (write all numbers in decimal):
   - *Segment*. Each of the above segments are representing by an index from 0 to 5 where 0 is for ***A***'s Code segment and 5 for ***B***'s Stack segment (i.e., indices are sorted by process (first ***A***, then ***B***), and then by base virtual address (lower virtual address first)).
   - *Process*. Process identifier (***A*** or ***B***).
   - *Virtual address*. Base address in virtual address space.
   - *Size*. The segment size in KB.
   - *Physical address*. Base address in physical address space.

Segment Index	Segment	Process	Virtual address	Size (KB)	Physical address
0	Code (A)	A	0	1	10KB
1	Heap (A)	A	2KB	1	3KB
2	Stack (A)	A	3KB	1	7KB
3	Code (B)	B	1KB	1	2KB
4	Heap (B)	B	2KB	1	4KB
5	Stack (B)	B	3KB	1	9KB

b) Assume the following virtual memory access addresses in decimal:

i. Process A, address 2148
ii. Process A, address 1324
iii. Process A, address 4095
iv. Process B, address 512
v. Process B, address 2048
vi. Process B, address 3272

Which of the above are valid virtual memory access addresses? and for those valid accesses, find the following:

- The corresponding Segment Index (from the table you formed above).
- The *Offset* within the segment in decimal.
- The *Physical* address in decimal.

Index	Process	Virtual address	VirtBase + Offset	isValid VirtAddr?	Segment Index	Offset	Physical address
i	A	2148	2KB + 100	Yes	1	100	3172
ii	A	1324	1KB + 300	No	-	-	-
iii	A	4095	3KB + 1023	Yes	2	1023	8191
iv	B	512	0KB + 512	No	-	-	-
v	B	2048	2KB + 0	Yes	4	0	4096
vi	B	3272	3KB + 200	Yes	5	200	9416

Problem 2 (25 Points)

Assume the following simple memory segmentation operating system:

- The OS supports two segments for a process:
  - Segment ***A*** that starts from the beginning of the address space and is positive growing for code and a heap, and
  - Segment ***B*** that starts from the end of the address space and is negative growing for a stack.
- The Virtual Address space size is 128 bytes.
- Physical memory size is 512 bytes.
- Segment register information:
  - Segment ***A*** base (grows positive) = 0
  - Segment ***A*** limit = 20
  - Segment ***B*** base (grows negative) = 511
  - Segment ***B*** limit = 20

Which of the following are valid virtual memory access addresses? and for those valid accesses, what are the corresponding physical addresses? Explain your answers.

a) 29
b) 123
c) 16
d) 90
e) 10

Valid VirtAddr Range: SegA[0-20] + SegB[107-127]

Index	Virtual address	isValid VirtAddr?	Loc @ Seg	Physical address
a	29	No	-	-
b	123	Yes	B	507
c	16	Yes	A	16
d	90	No	-	-
e	10	Yes	A	10

Problem 3 (25 Points)

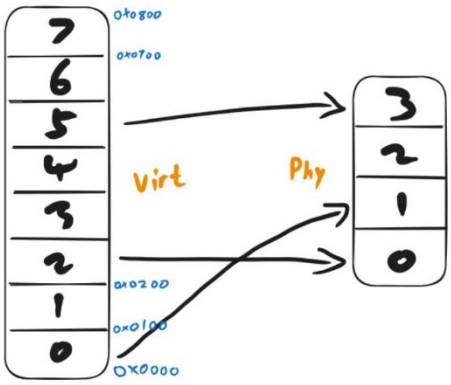
Consider a system using paging as the memory virtualization technique, with 256-byte pages, virtual memory spaces formed of 8 pages, a physical memory space formed of 4 frames, and the following content for the page table of process ***A***:

Page	Frame
0	1
2	0
5	3

a) Draw a diagram representing the above page table for process ***A***. Draw the virtual address space on the left, and the physical on the right. Assign numbers to both virtual pages and physical frames and draw an arrow connecting each page with its associated frame.
b) How many bits are used to represent the Virtual Address? How many for a Physical Address? How many for the page offset? Explain your answer.
c) Translate the following decimal numbers, which represent virtual addresses, into physical addresses (if such addresses exist): 418, 0, 581, 460. Show all the translation steps.

256-byte / page → 256 instances → 8-bit offset Addr
8 pages / VirtSp → 3-bit PTE input width (VPN?)

Therefore,
11 bits can represent this Virtual Memspace.



Virtual address	VirtAddr (Hex)	VirtBase + Offset	isValid VirtAddr?	PhyBase + Offset	Physical address
418	0x1A2	P1 + 162	No	-	-
0	0x000	P0 + 0	Yes	F1(0x1) + 0	256 (0x100)
581	0x245	P2 + 69	Yes	F0(0x0) + 69	69 (0x045)
460	0x1CC	P1 + 204	No	-	-

Problem 4 (25 Points)

A page-based virtual memory is using **a linear page table** where its:

- Virtual address space size = 128 bytes
- Physical memory size = 1024 bytes
- Page size = 16 bytes

The leftmost bit in each PTE is the VALID bit. If this bit is 1, the rest of the entry is the PFN. If the bit is 0, the page is not valid. Assume the shown contents of the page table (in **hexadecimal**).

VPN	PTE
0	0x80000034
1	0x00000000
2	0x00000000
3	0x00000000
4	0x8000001e
5	0x80000017
6	0x80000011
7	0x8000002e

a) In front of each of the following virtual addresses, write its corresponding physical address as a **3-digit hexadecimal** number in the format 0xabc. For an invalid virtual address, write FAULT for its physical address.

VirtSp 128 Bytes → 7-bit VirtAddr
Page size 16 bytes → 4-bit Offset Address
7-4 → 3-bit VPN

Virtual address	VPN	Offset	isValid PTE?	Physical Address
0x34	3	0x4	No	-
0x44	4	0x4	Yes	0x1E4
0x57	5	0x7	Yes	0x177
0x18	1	0x8	No	-
0x06	0	0x6	Yes	0x346

b) Explain how a **multi-level page table** (by utilizing a page directory) helps in reducing the memory needed compared to a **linear page table**.

Space Size: In a linear page table, all possible virtual addresses need to be represented, leading to a potentially huge table that consumes a significant amount of memory. This linear structure becomes increasingly impractical as the address space grows larger. A multi-level page table, on the other hand, breaks down the address space into smaller, more manageable chunks using multiple levels of tables. At the top level, there's a page directory, which is essentially an array of pointers to lower-level page tables. It can easily scale to accommodate larger address spaces by adding more levels or increasing the size of the page tables at each level.

Space Density: In a linear page table, most likely address spaces are sparse, meaning that not all possible virtual addresses are actually in use. With a multi-level page table, only the portions of the address space that are actively being used need to be represented in memory. Unused portions of the address space don't require memory allocation, thus saving space.