CS143A Spring 2020 Name: _____
Homework 4

Please do not modify the structure of this document. Due: ***6/5 at 11:59 pm.***

---

1) Segmentation (30 points)

It is nearing the end of the quarter, and Alice has been messing around a bit too much with memory management. With one misclick, Alice suddenly sees that half of her notes for OS class have disappeared. Panicked, Alice decides to dig through her notes on memory management for anything. Fortunately for her, she finds out that each of her notes are still in memory, but scattered in memory; each of her notes are conveniently in a different segment. In addition, she finds each of the segment's start and end addresses, as shown below:

Segment	**Base**	**End**	**Length (delete when releasing HW)**
0	255	580	325
1	1924	1984	60
2	3313	3750	437
3	1150	1620	470
4	2890	3005	115
5	4871	4969	98

a) [12] This is about the best that Alice can manage, so she then calls you for help. However, before asking you for help, Alice decides to try some logical addresses which are formatted as **<segment, offset>**. Help Alice by writing the corresponding physical address, marking an **X** for whether the physical address is legal or not, and briefly explain.

<segment, offset>	**Physical Address**	**Legal**	**Not Legal**	**Explanation**
<0,230>	485	X		255 + 230 = 485
<1, 40>	1964	X		1924 + 40 = 1964
<2, 600>	3913		X	3750 < 3913
<3, 400>	1550	X		1150 + 400 = 1550
<4, 112>	3002	X		2890 + 112 = 3002
<5, 1>	4872	X		4871 + 1 = 4872

b) [12] It's just not Alice's day, and after trying out the above logical addresses, Alice makes another misclick and starts memory compaction in the system. Fortunately, Alice has stopped all other processes from running, so the above segments can be assumed to be the only segments in the system. Help Alice by filling in the base address table below with each segment's new base address.

Segment	Base
0	0
1	795
2	970
3	325
4	855
5	1407

2) Replacement and fault management (40 points)

Given all the bad luck that Alice has been having around memory management, she reluctantly decides to drop messing around with Bob's toaster to get revenge, and instead decides to focus hard on the OS class. However, karma works in funny ways, and while Bob was preparing for another prank on Alice, he accidentally enables a bug which completely ruins his system's page replacement algorithms and reduces the maximum number of frames available to 4.

Suppose that Bob was planning to reference the following pages, as below.

e, c, b, e, a, g, d, c, e, g, d, a

a) Given that Bob only has 4 frames, all initially empty (which will cause the first unique page to cost one fault each), how many page faults can Bob expect to happen in the best case scenario (i.e., using the Optimal algorithm)? Fill in the table accordingly, writing "Y" for page fault and "N" for no page fault, and count the number of page faults that occur.

Time	1	2	3	4	5	6	7	8	9	10	11	12
Ref String	e	c	b	e	a	g	d	c	e	g	d	a
Frame 0	e	e	e	e	e	e	e	e	e	e	e	e
Frame 1		c	c	c	c	c	c	c	c	c	c	c
Frame 2			b	b	b	g	g	g	g	g	g	g
Frame 3					a	a	d	d	d	d	d	a
Page fault?	Y	Y	Y	N	Y	Y	Y	N	N	N	N	Y

Total # page faults: 7

NOTE: time 12 can have many different choices for replacement since the 'next used page' is not defined at the end of the reference string

b) Bob needs to consider the options he has. What happens if Bob uses the LRU algorithm?

Time	1	2	3	4	5	6	7	8	9	10	11	12
Ref String	e	c	b	e	a	g	d	c	e	g	d	a
Frame 0	e	e	e	e	e	e	e	c	c	c	c	a
Frame 1		c	c	c	c	g	g	g	g	g	g	g
Frame 2			b	b	b	b	d	d	d	d	d	d
Frame 3					a	a	a	a	e	e	e	e
Page fault?	Y	Y	Y	N	Y	Y	Y	Y	Y	N	N	Y

Total # page faults: 9

c) What about if Bob uses LRU approximation algorithm: Second Chance?

Time	1	2	3	4	5	6	7	8	9	10	11	12
RS	e	c	b	e	a	g	d	c	e	g	d	a
F0	e(0)	e(0)	e(0)	e(1)	e(1)	e(0)	e(0)	e(0)	e(1)	e(1)	e(1)	e(0)
F1		c(0)	c(0)	c(0)	c(0)	g(0)	g(0)	g(0)	g(0)	g(1)	g(1)	g(0)
F2			b(0)	b(0)	b(0)	b(0)	d(0)	d(0)	d(0)	d(0)	d(1)	d(0)
F3					a(0)	a(0)	a(0)	c(0)	c(0)	c(0)	c(0)	a(0)
Page fault	Y	Y	Y	N	Y	Y	Y	Y	N	N	N	Y

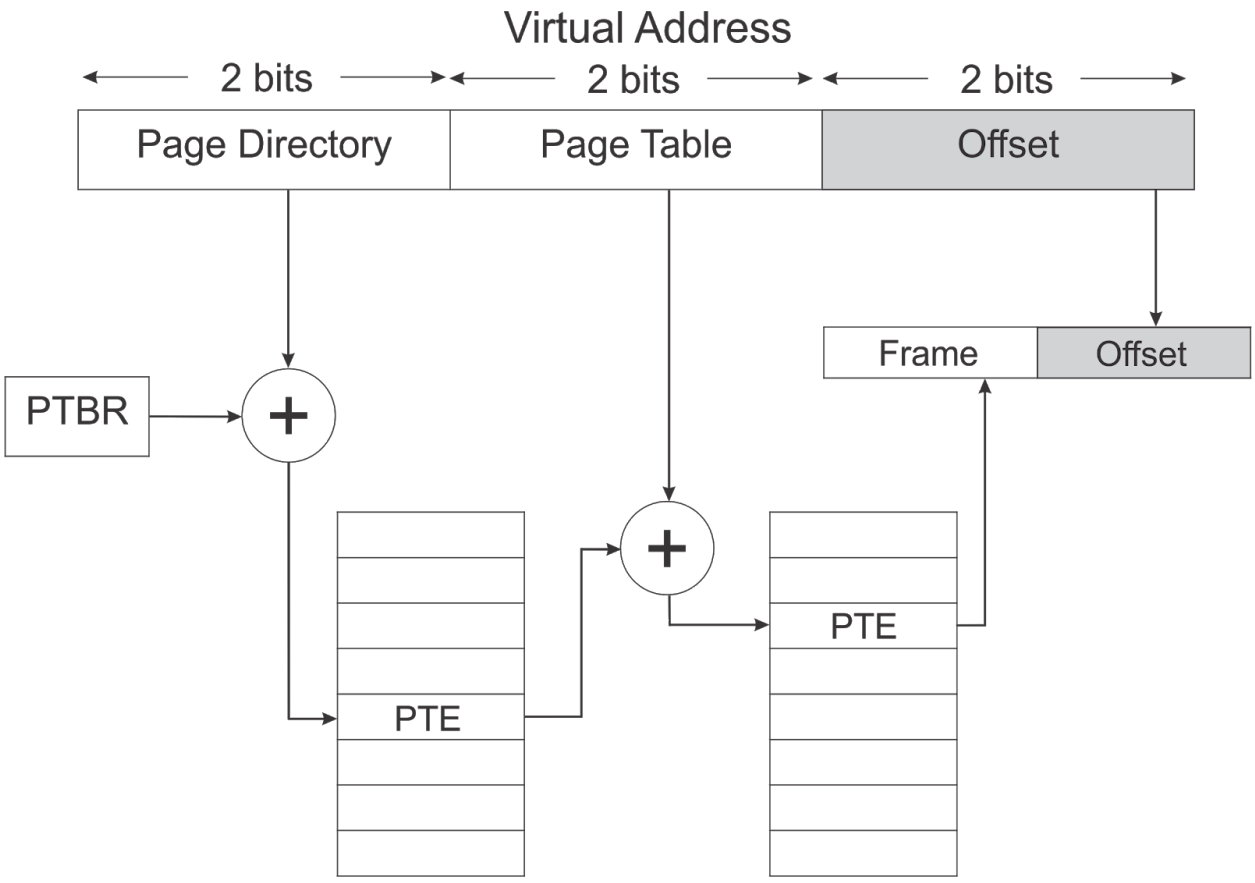
Total # page faults: 8

d) Bob wishes to know how well each of the algorithms performed. Given the reference string that Bob used, rank the algorithms from 1=best, 3=worst.

Algorithm	Rank (1=best, 3=worst)
Optimal	1
LRU	3
LRU: Second chance	2

3) Address Translation (30 points)

Bob's Problems continue: fixing one bug leads to another. As Bob is trying to fix the issue with the page replacement algorithms, his system gets a segmentation fault and creates a core dump file. However, Bob gets lucky because he had previously modified his system to use a simplified version of address translation using a two-level level paging scheme. In addition, he remembers that virtual addresses are 6-bit binary numbers which can be "chopped" into three fields, each of which are 2-bits wide. The three fields define the page directory, page table and offset indices.



Note: The following are used for brevity:

PD=Page Directory
PT=Page Table
PTE=Page Table Entry
F=Frame

Bob’s core dump (the contents of memory) looks as follows:

PTBR	11110

Memory Address	Contents	Note
11110	100010	PD
11111	100100	
100000	100110	
100001	101000	
100010	101010	PT0
100011	101100	
100100	101110	PT1
100101	110000	
100110	110010	PT2
100111	110100	
101000	110110	PT3
101001	111000	
101010	10000111	F0
101011	10001010	
101100	10010010	F1
101101	10010011	
101110	11001011	F2
101111	100000	
110000	10110101	F3
110001	10110101	
110010	10111100	F4
110011	10111000	
110100	10101100	F5
110101	11011	
110110	11111100	F6
110111	10110100	
111000	110010	F7
111001	1101001	

Bob can only handle looking at a few memory addresses at a time, but he knows that with a 6-bit addressing scheme, there are 64 memory locations. However, he decides to ignore the rest of the memory addresses for now. To debug, Bob knows that he must try translating virtual addresses to physical addresses, but he is clueless on how to start. After hours struggling, Bob gives up and asks Alice for help, to which Alice eventually agrees on the condition that Bob doesn't continue pranking her. Bob agrees, and the two start working together. Fill in the table below to calculate the physical address corresponding to each virtual address.

Virtual Address	Page Directory	Page Directory offset	Page Table (Level 2)	Page Table Offset	Page Address	Offset	Translated value
010101	11110	01	100100	01	110000	01	10110101
110001	11110	11	101000	00	110110	01	10110100
100100	11110	10	100110	01	110100	00	10101100
000000	11110	00	100010	00	101010	00	10000111

4) File System Operation (Optional)

The end of the quarter is in sight, and since then Alice and Bob have gotten along better together. The final is soon, and they want to do the best that they can. Therefore, they decide that they should pool both of their notes together into one big file so that they can learn from each other. Suppose that this joint file currently consists of 200 blocks in memory, and that the file control block (and the index block, in the case of indexed allocation) is already in memory. As Alice and Bob edit the file by adding or removing blocks, suppose that the system must respond with some number of disk I/O operations. Of course, since there is a lot of material covered in the OS class, they don't want to overwhelm the system. However, they are unsure of whether the system uses a contiguous allocation strategy, a linked allocation strategy, or an indexed (single-level) allocation strategy, so they decide to calculate the number of disk I/Os required for each of them. Assume that for the contiguous allocation strategy, there is no space to grow in the beginning, but there is room at the end. Additionally, assume that the block information to be added is already stored in memory.

	Contiguous	Linked	Indexed
The block is added at the beginning			
The block is added in the middle.			
The block is added at the end.			
The block is removed from the beginning.			
The block is removed from the middle.			
The block is removed from the end.			