HW2

Selahattin Cem Öztürk 21802856 CS342-2

Q1

Existing			
A	B	C	D
15	6	9	10

current state:

Alloc	A	B	C	D
P_0	2	0	2	1
P_1	0	1	1	1
P_2	4	1	0	2
P_3	1	0	0	1
P_4	1	1	0	0
P_5	1	0	1	1

Max	A	B	C	D
P_0	9	5	5	5
P_1	2	2	3	3
P_2	7	5	4	4
P_3	3	3	3	2
P_4	5	2	2	1
P_5	4	4	4	4

Available	A	B	C	D
	6	3	5	4

Need	A	B	C	D	
P_0	7	5	3	4	-
P_1	2	1	2	2	-
P_2	3	4	4	2	-
P_3	2	3	3	1	-
P_4	4	1	2	1	-
P_5	3	4	3	3	

with available (6, 3, 5, 4), we can give the needed (2, 3, 3, 1) to P_3. After P_3 finishes our available becomes (7, 3, 5, 5). with that (2, 1, 2, 2) can be allocated for P_1 and after P_1 finishes our available becomes (7, 4, 6, 6). Then needed resources (3, 4, can be given to P_2 after P_2 finishes available becomes (11, 5, 6, 8) with this P_0 can be given (7, 5, 3, 4) & available becomes (13, 5, 8, 9) then P_4 can run & finish & available becomes (14, 6, 8, 9) finally P_5 can run finish and available becomes (15, 6, 9, 10) so state is safe

$P_3 \rightarrow P_1 \rightarrow P_2 \rightarrow P_0 \rightarrow P_4 \rightarrow P_5$

$(6,3,5,4) \rightarrow (7,3,5,5) \rightarrow (7,4,6,6) \rightarrow (11,5,6,8) \rightarrow (13,5,8,9) \rightarrow$
$\rightarrow (14,6,8,9) \rightarrow (15,6,9,10)$

if P_5 requests (3, 2, 3, 3) assume that we allowed it according to Banker's algorithm. Available & Need matrix become

Available			
A	B	C	D
3	1	2	1

	Need			
	A	B	C	D
P_0	7	5	3	4
P_1	2	1	2	2
P_2	3	4	4	2
P_3	2	3	3	1
P_4	4	1	2	1
P_5	0	2	0	0

in this state none of the processes can suffice their needs completely in order to finish & release the holded resources. So if P_5 given (3, 2, 3, 3), deadlock occurs. thus according to Banker's Algorithm this request should not be granted in order to avoid deadlock.

Q2

a) if there is no TLB, our EAT formula is

$$EAT = 2 * (150_{ns}) = 300_{ns}$$

since without TLB, we have to access for page table & data/instruction. Thus 2 memory access

b) with TLB our EAT is

$$EAT = (1 * 150_{ns} + 10_{ns}) \cdot \frac{85}{100} + (2 * 150_{ns} + 10_{ns}) \cdot \frac{15}{100}$$

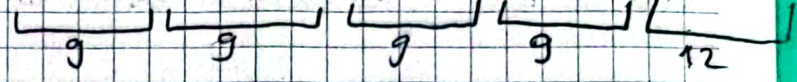
$$= \frac{160 \cdot 85}{100_2}{}^{17}\,ns + \frac{310 \cdot 15}{100_2}{}^{3}\,ns = (136 + 46.5)ns = 182.5_{ns}$$

c) if 2 lvl page used without TLB we would need 3 memory accesses for each (1 for each lvl & 1 for data) so;

c-a) $EAT = 3 \cdot 150_{ns} = 450_{ns}$

c-b) $EAT = \underbrace{(1 \cdot 150_{ns} + 10_{ns}) \cdot \frac{85}{100}}_{136_{ns}} + \underbrace{(3 \cdot 150_{ns} + 10_{ns}) \cdot \frac{15}{100}}_{69_{ns}} = 205_{ns}$

Q3

a) virtual address: 

physical address is also 48 bits. We know that the offset of the virtual & physical addresses are the same. thus 48 - 12 = 36 bits are used as a frame number in a PTE

b)

page sizes are $2^{12} = 4\,KB = 4096$ bytes

and 4th level table contains $2^9 = 512$ entries so 4th level table can map $2^9 \cdot 2^{12} = 2^{21} = 2\,MB$

since each entry is 8 bytes every table is $2^9 \cdot 2^3 = 4\,KiB$

also 1 entry maps 4 KB · from 2^{12} → offset.

So 2 MB = 0x200000 from starting at 0x000 ... 0

for every 0x200000, we need another 4th lvl table

our 16MB of virtual memory starts at 0x300000

so 1 MB mapped with 256 entries in one 4th lvl table [0x300000 - 0x400000]

and 1 MB mapped with 256 entries in one 4th lvl table [0x800000 - 0x900000) ← half of the entries used

between [0x400000 - 0x800000), 2 · 4th lvl used with both containing 512 used entries. In total 4 table for 4th lvl

For 3rd lvl :

0x300000 → 0's 0011 0000 0000

0x400000 0's 0100 — —

0x600000 0's 0111 —

0x800000 0's 1000 —

} 4 entries used inside the 1 3th lvl page for the 2nd & 1st lvl's

1st 2nd 3rd 4th 1 page used so

1 + 1 + 1 + 4 = 7 pagetable in total

7 x 4KB = 28KB needed

c) In contrast to part b, all the starting addresses starts at the 2MB boundary. By keeping that in mind;

- 128KB is much less than 2MB so 1 4th lvl page table is enough. Also, only 128KB/4KB = 32 entry is used.
- for data segment we need $2^{31}/2^{21} = 2^{10}$ 4th lvl page since $2^{10} > 2^9$, 1 3rd lvl is not enough we need 2.

data → 0x 8 0000 0000 (offset)

= 0's 1000 0000 0000 0000 0000 0000 (offset)

1st | 2nd | 3rd | 4th | offset

code → 0x 1000 000 (offset)

= 0's 0001 0000 0000 0000 (offset)

1st | 2nd | 3rd | 4th | offset

stack → 0x F00000 00000 (offset)

0's 1111 0000 0000 0000 0000 0000 0000 (offset)

1st | 2nd | 3rd | 4th | offset

as seen for the stack, we need different 2nd lvl page. Since it is 4MB. it needs $2^{22}/2^{21} = 2$ 4th lvl page in summary,

code: 1 first, 1 second, 1 third, 1 fourth

data: same ↓, same ↓, 2 third, 2^{10} fourth

stack: same ↓, 1 second, 1 third, 1 fourth

$1 + 2 + 4 + (2^{10} + 3) = (10 + 2^{10})$ page tables

$(10 + 2^{10})$ 4KB needed ≈ 4MB

Q4

underlined one is the victim pointer

				Fault
2:	2(1)			F
4:	2(1)	4(1)		F
1:	2(1)	4(1)	1(1)	F
3:	3(1)	4(0)	1(0)	F
4:	3(1)	4(1)	1(0)	
6:	3(0)	6(1)	1(0)	F
3:	3(1)	6(1)	1(0)	
6:	3(1)	6(1)	1(0)	
1:	3(1)	6(1)	1(1)	
2:	3(0)	6(0)	2(1)	F
1:	1(1)	6(0)	2(0)	F
5:	1(1)	5(1)	2(0)	F
2:	1(1)	5(1)	2(1)	
5:	1(1)	5(1)	2(1)	
4:	1(0)	5(0)	4(1)	F
1:	1(1)	5(0)	4(0)	
2:	1(0)	2(1)	4(0)	F
4:	1(0)	2(1)	4(1)	
3:	3(1)	2(1)	4(0)	F
				+ F
				11 F

optimal:

Reference string:

2	4	1	3	4	6	3	6	1	2	1	5	2	5	4	1	2	4	3
2	2	2	3	3	3	3	3	3	3	3	5	5	5	4	4	4	4	
	4	4	4	4	6	6	6	6	2	2	2	2	2	2	2	2	2	
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
F	F	F	F		F			F	F		F							F

9 Fault.

Q5

FCFS:

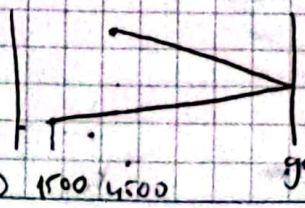
5200 - 4500 = 700
5200 - 2000 = 3200
9500 - 2000 = 7500
9500 - 1300 = 5200
4300 - 1500 = 2800
5100 - 1500 = 3600
9600 - 5100 = 4500
9600 - 4000 = 5600
4600 - 4000 = 600

$\Sigma = 33700$

SCAN: Ordered (ascending)

1500, 2000, 4000, 4300, [4500 >>], 4600, 5100, 5200, 9500, 9600

it will;



go to 9999 while serving requests then it will move to other side again while serving. It stops at 1500 so

(9999 - 4500) + (9999 - 1500) = 13 998

SSTF: It chooses the closest one.

initially it is at 4500>>

it will choose: 4300, 4000, 4600, 7100, 5200, 2000, 1500, 9500, 9600
in this order:

so the in total: 200 + 300 + 600 + 500 + 100 + 3200 + 500 + 8000 + 100

= 13500

Q6

a)

4(KB)/30(MB/s) = 0.130ms. This is the transfer time for 4KB. Avg rotational latency is: ((1/(3600/60))s)/2 = 0.083 s = 83ms. Then total time to transfer 4KB is 83 + 4 + 0.130 = 87.13 ms = T. Number of disk I/Os per second = 1/T = 11. Throughput: 0.042 MB/s

b)

512 x 4KB / 30 = 0.04 seconds = 40ms. Time for one I/O = 83 + 4 + 40 = 127ms = T. Number of disk I/Os per second = 1/T = 7 Throughput = 7 x 4 x 512/1024 = 14 MB/s

Q7

a) In an inode we can store 8KB/8 = 512 pointers. Total file size is: $10 + 512 + 512^2 + 512^3 \approx 2^{27}$ blocks. That makes: $2^{27} \cdot 2^{12} = 2^{39}$ Bytes. This max file size supported is ≈ 512GB

b) A leaf index block can map $2^9 \cdot 2^{12} = 2^{21} = 2$MB of file data for 8GB we need $2^{33}/2^{21} = 2^{12} = 4096$ leaf index blocks. Thus we have to use one single lvl leaf index block and for the remaining second level leaf index block. 4096 - 1 = 4095 second lvl index block.

c)

(i) $2^{15}/2^{12} = 8$. It is in the file block 8. Hence the respective block is pointed directly by the inode. Thus we need 1 access

(ii) $2^{23} = 8$MB. since a single index block can map $2MB = 2^9 \cdot 2^{12}$

8MB/2MB = 4 So we need to access the first-level index block of the two-lvl index, then a second-lvl index block and then the data block so we need 3 disk accesses

(iii) $2^{32} = 4$GB, since three-lvl index structure can map $2^{12}(2^9)^3 = 2^{27} \cdot 2^{12} = 2^{39}$ we have to access first lvl, second-lvl, third-lvl & data so:

4 access