

COMP3911 Suggested Exercises #1b - Solutions

1) Page table is shown here: (cells with arrows pointing at them get updated as part of the page table updates listed below and are unset prior to the actions noted here):

Page Table

	V	RO	D	U	Page Frame	Disk Addr
0x0		1				5
0x1		1				B
0x2	1	1			4	8
0x3	1	1			D	4
0x4	1				10	
0x5	1				5	
0x6	1				1	
0x7						11
0x8						
0x9						
0xA						
0xB						
0xC						
0xD						
0xE	1				11	6
0xF						C <-new

- Access Page 5, (in memory), set reference bit Physical memory address 0x2804.
- Access Page 4, (in memory), set reference and dirty bits. Physical memory address 0x8244.
- Access Page 3, (in memory), RO bit set, "segmentation" error, remove from run queue, deallocate pages. Physical memory address would have been 0x6911.
- Access Page 2, page fault, schedule I/O to read (assume into empty frame 0x2); assign frame to P1. Schedule new process (ie; not P1) from RUN queue. When paging I/O is complete, schedule P1 as "Runnable". When P1 runs again, restart instruction. Page now is in memory, set reference bit. Physical memory address 0x2403.
- Access Page 9, segmentation fault (no data in memory or disk for this page).
- Access Page A, segmentation fault (no data in memory or disk for this page).

2) Base performance is calculated by:

Time = 2 x RAM access time

Time = 2 x 40 cycles

Time = 80 cycles

TLB performances are:

a) Time = $(0.8)(2 + 40) + (0.2)(2 + 40 + 40)$

Time = $33.6 + 16.4 = 50$ cycles

Difference = $50 / 80 = 0.625$

b) Time = $(0.95)(2 + 40) + (0.05)(2 + 40 + 40)$

Time = $39.9 + 4.1 = 44$ cycles

Difference = $44 / 80 = 0.55$

c) Time = $(0.2)(2 + 40) + (0.8)(2 + 40 + 40)$

Time = $8.4 + 65.6 = 74$ cycles

Difference = $74 / 80 = 0.925$

d) Time = $(0.05)(2 + 40) + (0.95)(2 + 40 + 40)$

Time = $2.1 + 77.9 = 80$ cycles

Difference = $80 / 80 = 1$

3) To calculate this, you must simply solve the equation:

$$50 = (0.25)(x + 25) + (0.75)(x + (25 + 25))$$

for x. The result is

$$x \geq 6.25$$

4) For page allocation of 3, the tables are:

String i

i/FIFO = 11 faults

7	6	8
9	7	A
8	B	C
A	7	

i/LRU = 10 faults

7	6	8
B	9	C
A	A	
	8	
	7	

i/LFU = 10 faults

7	6	8
	9	
	A	
	B	
	C	
	A	
	C	
	A	

i/Belady's = 8 faults

7	6	8
B	9	7
C	A	

i/Use Bit/FIFO = 11 faults

7	6	8
9	7	C
A	B	7
8	A	

String ii

ii/FIFO = 22 faults

1	2	3
4	5	2
3	4	5
2	3	4
5	2	3
4	5	2
3	4	5

i/LRU = 22 faults

1	2	3
4	5	2
3	4	5
2	3	4
5	2	3
4	5	2
3	4	5

ii/LFU = 22 faults

1	2	3
	4	5
	2	3
	4	5
	2	3
	4	5
	2	3
	4	5
	2	3
	4	5

ii/Belady's = 10 faults

1	2	3
4	3	4
5	4	5
2		

ii/Use Bit/FIFO = 22 faults

1	2	3
4	5	2
3	4	5
2	3	4
5	2	3
4	5	2
3	4	5

String iii

iii/FIFO = 21 faults

1	2	3
4	5	6
1	2	3
4	5	6
8	9	2
1	2	3
4	5	6

iii/LRU = 21 faults

1	2	3
4	5	6
1	2	3
4	5	6
8	9	2
1	2	3
4	5	6

iii/LFU = 21 faults

1	2	3
4	5	6
1	2	3
4	5	6
8	9	2
1	2	3
4	5	6

iii/Belady's = 18 faults

1	2	3
4	3	4
1	4	5
4	5	6
5	9	8
6	2	3

iii/Use Bit/FIFO = 21 faults

1	2	3
4	5	6
1	2	3
4	5	6
8	9	2
1	2	3
4	5	6

5)

Small Page Size	Large Page Size
- less wasted memory	- more wasted memory
- larger page table	- smaller page table

There is also the effect on number of page faults. For the case where there is lots of memory on the machine, there will be fewer faults for a large page size, since a program of a given memory size takes up fewer pages and each gets faulted in sooner or later. For a memory constrained system, a smaller page size wastes less memory and therefore will potentially generate fewer page faults, since the working sets can all be resident for some cases where the large page size wastes too much memory for that to be possible.

6) If the machine is doing a lot of swapping, it is memory constrained, so adding memory will help a lot. A machine swaps when it can't fit the working set for all current processes into memory at the same time. This forces the machine to move entire process segments of memory out to disk, resulting in large delays when these processes need to be run. Adding memory avoids the need to swap and the associated large delays waiting for swap disk I/O.

7) a) Since P_0 , the proportion of time the queue is empty is equal to the proportion of time the CPU is idle for the CPU run queue, then for the three cases rho (the load factor) is:

$$\rho_1 = 1 - P_0 = 1 - 0.55 = 0.45$$

$$\rho_2 = 1 - P_0 = 1 - 0.40 = 0.60$$

$$\rho_3 = 1 - P_0 = 1 - 0.68 = 0.32$$

For an M/M/1 queuing model, $N = \rho/(1 - \rho)$, so:

$$N_1 = \rho_1/(1 - \rho_1) = 0.45/(1 - 0.45) = 0.82$$

$$N_2 = \rho_2/(1 - \rho_2) = 0.60/(1 - 0.60) = 1.5$$

$$N_3 = \rho_3/(1 - \rho_3) = 0.32/(1 - 0.32) = 0.47$$

Since the Load Average is the measured average CPU run queue length and N_{1-3} as estimated by the model is approximately equal the Load Average for each case, the model is reasonably good one for this situation.

b) Upgrading the CPU with one that is 20% faster should increase the average service rate by 20%.

Since: $\rho = (\text{average arrival rate})/(\text{average service rate})$

Therefore: $\rho' = (\text{average arrival rate})/(\text{average service rate} * 1.2)$

Taking the peak load for the three occasions:

$$\rho'_2 = \frac{\rho_2}{1.2} = \frac{0.6}{1.2} = 0.5$$

Unfortunately, a 100% increase in web traffic would double the load from 0.5 to 1.0 and this would saturate the queue. So, your recommendation would be that a 20% faster CPU will not be good enough and a CPU of nearly double the speed will be required.