1

1 1. A system uses simple base and bounds mechanism to virtualize the address space. You need to fill the base and bound values using the information provided in the below table.

Virtual Address (VA)	Physical Address (PA)
0	1000
100	1100
1999	2999
2000	fault

Base 1000 Ans.

↳ since 0_{VA} maps to 1000_{PA}.

Bound ~~2000~~ 2000 Ans

↳ since $(0+1999)_{VA}$ maps to $(1000+1999)_{PA}$

but, $(0+2000)_{VA}$ gives page fault

(Assume PA < VA + bound is the condition, with strict inequality)

2 2. (a) Assume the page size remains the same. If the physical memory size (in bytes) is doubled, how does the number of bits in each entry of the page table change?

Solution: ~~P~~ Let page table entry size be |pte| (= |pfn| + |vpn|). |x| = size(x) in bits

→ page frame number, virtual page number

If physical memory size is doubled, then 1 extra bit is required, to ~~make~~ keep full physical memory addressable. ⇒ (i.e., page frame number has 1 more bit)

Hence, number of bits in each entry of the page table INCREASES BY 1.

(b) If the physical memory size (in bytes) is doubled, how does the number of entries in the page table change?

Solution: Assuming that page table is not inverted, and that processes did not change their VA mapping during the doubling of physical memory size, there will be NO CHANGE in the number of entries of the (per-process) page table.

This is because new mappings did not happen during the doubling.

(c) If the page size (in bytes) is doubled, how does the number of entries in the page table change?

Solution: If page size is doubled, and virtual address space of process did not change, then number of pages will be halved. ~~So, the~~ Also, note that, each entry in page table corresponds to 1 page (each). So, the number of entries in page table will, hence, be HALVED.

Note the Assumption : Byte-addressable machine !!

(d) Suppose that you have a system with 8-bit virtual addresses, 8 pages of virtual memory, and 4 pages of physical memory. How large is each page? Assume memory is byte addressed.

Solution: Given, virtual memory per-process is of 2^8 B, there are 2^3 pages of virtual memory, ~~2~~ 2^2 pages of physical memory.

$\Rightarrow$ size(page) $= \frac{2^8}{2^3} = 2^5 = \boxed{32B}$ (i.e., 32 Bytes)
Ans.

$\boxed{2}$ 3. Assume main memory has space to hold a maximum of 3 pages. The page access pattern is as follows A B C D A B D C B A.

A	M	A	M	B	H
B	M	B	M	A	H
C	M	D	H		
D	M	C	M		

Hit/Miss Table (H=Hit M=Miss)

↳ 7 misses.

(a) How many page faults will you get with FIFO? $\boxed{\cancel{A}\,\cancel{B}\,\cancel{C}\,\cancel{D}ABE}$

Solution: There are $\boxed{7}$ page faults with FIFO.

(Memory) Queue: $\perp \xrightarrow[(M)]{A} A \xrightarrow[(M)]{B} AB \xrightarrow[(M)]{C} ABC \xrightarrow[(M)]{D} BCD \xrightarrow[(M)]{A} CDA \xrightarrow[(M)]{B} DAB \xrightarrow[(H)]{D} \xrightarrow[(M)]{C} ABC \xrightarrow[(H)]{B} \xrightarrow[(H)]{A} \boxed{ABC}$ final state (queue)
(blank)

Note: underlined ↓ element at risk of eviction, if miss occurs.

(b) How many page faults will you get with LRU? (Least Recently Used)

A	M	A	M	B	H
B	M	B	M	A	M
C	M	D	H		
D	M	C	M		

Solution: There are $\boxed{8}$ page faults with LRU.

(Memory): $\perp \xrightarrow[(M)]{A} \underline{A} \xrightarrow[(M)]{B} \underline{A}B \xrightarrow[(M)]{C} \underline{A}BC \xrightarrow[(M)]{D} \underline{B}CD \xrightarrow[(M)]{A} \underline{C}DA \xrightarrow[(M)]{B} \underline{D}AB \xrightarrow[(H)]{D} \underline{A}BD \xrightarrow[(M)]{C} \underline{B}DC \xrightarrow[(H)]{B} \underline{D}CB \xrightarrow[(M)]{A} \boxed{\underline{C}BA}$ final state
(blank)

2

(c) How many page faults will you get with OPT ? ↳ optimal algorithm

A	M	A	H	B	H
B	M	B	H	A	H
C	M	D	H		
D	M	C	M		

Here we Evict the latest accessed element in future

Solution: There are $\boxed{5}$ page faults with OPT.

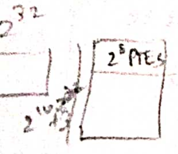
(Memory): $\perp \xrightarrow[(M)]{A} A \xrightarrow[(M)]{B} AB \xrightarrow[(M)]{C} ABC \xrightarrow[(M)]{D} AB\underline{D} \xrightarrow[(H)]{A} \xrightarrow[(H)]{B} \xrightarrow[(H)]{D} AB\underline{D} \xrightarrow[(M)]{C} AB\underline{C} \xrightarrow[(H)]{B} \xrightarrow[(H)]{A} \boxed{ABC}$ final state
(blank)

(d) LRU is an approximation of OPT, which is provably optimal. Does that mean LRU will always perform better than FIFO ? why or why not ? explain your answer.
↳ because of counterexample

Solution:

$\boxed{NO}$ LRU being an "approximation of a provably optimal algorithm does not mean that it will perform better than FIFO. A counterexample is present in $\boxed{Q3(a) \text{ and } Q3(b)}$ above. LRU had evicted element 'A' since it was the least recently accessed, but this was moments before A was accessed again. We could argue that $\boxed{\text{on average}}$, LRU performs better than FIFO, but by virtue of the counterexample, this is $\boxed{\text{NOT ALWAYS}}$ true.

$\boxed{4}$ 4. (a) Assuming a two-level page table (32-bit virtual addresses, 4KB pages, 4-byte page table entry size),what is the minimum number of pages needed to store the two-level page table (including the page directory) when there are 1029 contiguous valid pages in the virtual address space?
↳ $2^{10}+2^2+1 = 2^{10}+3$



—0—

Solution: Size of per-process virtual memory $= 2^{32}$B , size of pages $= 2^{12}$B
Number of pages per process $= \frac{2^{32}}{2^{12}} = 2^{20}$ pages ,(PTE size = 4B)
$\Rightarrow$ Size of page table $= \boxed{2^{22}B}$ ~~this~~, ~~Number of pages for page table~~ $\frac{2^{22}}{2^{12}} = 2^{10}$ pages

Number of PTEs per page $= \frac{2^{12}}{2^2} = 2^8$ PTEs.
Number of valid PTEs $= 1029 = \boxed{2^{10}+3}$ $\Rightarrow$ we would need $\left\lceil \frac{2^{10}+3}{2^8} \right\rceil = \boxed{5}$ pages for inner page table
$+ \boxed{1}$ page for outer page table, to address these 5 ~~for these 5~~

Hence, minimum no. of pages is (for 2 level pg table) $\Rightarrow \boxed{6 \text{ PAGES}}$ Ans

(b) Assume the following: a 32-bit address space with 1-KB pages. Assume each page table entry (PTE) is 4 bytes. Assume there are 100 processes in the system. If each process uses only one virtual page, what is the worst-case total size of all of these page tables?

page size

Solution: Size of virtual address space ⇒ 2^{32} B, per process · page size = 2^{10} B.

⇒ Number of pages per process = $\frac{2^{32}}{2^{10}} = 2^{22}$ pages

~~Num~~ Size of PTE = 4B ⇒ Size of page table per process = $2^{22} \times 4 = \boxed{2^{24}\text{B}}$

For 100 processes, worst case total Size of all page tables = $2^{24} \times 100$B = $\boxed{1600\text{MB}}$ (roughly 1.6GB) Ans.

−1

(c) A system has 8 page frames and each process has a page table of four entries in which a "X" indicates the corresponding page is not in physical memory. Suppose a snapshot of the page tables at certain moment is shown below. You need to prepare the inverted page table using the information provided in the page table of the three processes.

VPN	Process 0 (pid)
0	6
1	X
2	0
3	4

VPN	Process 1 pid
0	X
1	3
2	X
3	2

VPN	Process 2
0	1
1	7
2	X
3	5

(Given: per-process page tables.
Goal: per-system/memory inverted page table)

Solution:
Required Inverted page table:

~~Here, same as pid~~

→ virtual page number within process

Page Frame Number	Process Identifier	VPN
0	0	2
1	2	~~3~~ 0
2	1	~~3~~ 3
3	1	1
4	0	3
5	2	3
6	0	0
7	2	1

The 'x' entries are not considered in construction of the inverted page table

−2

Note: In an inverted page table, number of page table entries is equal to number of physical frames. It is not a per-process construct, but a per-memory construct. (system-level, essentially)

1 5. Effective memory access time on a TLB miss with a 4-level page table compared with a 2-level page table is ____________________(faster/slower and by how much ?)

Solution: $\boxed{\text{Slower}}$. It takes $\boxed{\text{double}}$ the time to navigate/walk the page table to retrieve the address mapping. So, cost of TLB miss is doubled. (Need to go 4 levels in, rather than 2 levels)

0.5