# Question 1

Who owns the pages in physical memory? How would you denote ownership?

The process owns that physical memory until it is taken away by the system or when the user uses the computer in such a way that the system needs to take away that section of memory from that process for the user demand. I think everything is ran by a process. That process uses that physical space in memory to do its work. So, I would have to say the process owns it to some degree. The system can write out those areas at anytime depending on the algorithm. The user does not directly access the main memory, but it does do things with in. for example using a word processor or playing a game does use main memory.

With that said, I think ownership is once again owned by the process, but if you want to categorize the process by if it’s a system process or a user process then we can easily just make a bit in the page table, TLB, or Physical memory dedicated to signify who owns the process.

1 for system and 0 for user.

# Question 2

When you replace an entry in the TLB, do you have to write it back to physical memory? Why or why not?

No, The frame should be in physical memory already. Even if its dirty, you’re still not doing any writing. The only time I believe anything gets written is when the frame and page number are getting inserted into the TLB, but that should not affect physical memory or the page table.

# Question 3

Does the size of the TLB make a difference in your statistics? Experiment.

It does make a difference. I experimented with Ram3.cpp by just changing the size of the TLB array. As you can see from my statistics (last page), When the TLB only had 8 records, the TLB hit rate was just under 2%. When the size was 32 records, it was over 7%. When the size is the same as page size (256), only gets to 55% (also have to take into account the physical size was only 128). So, there is diminishing returns. Also, when the TLB is bigger it seems that the Page fault goes down.

# Extra cred

Can this virtual memory manager have more levels of memory?

Of course it can. You would have to do it by

* 16\*16\*256
* 16\*16\*16\*16
* 4 \* 4 \* 4 \* 4 \* 4 \* 4 \* 4
* 2 \* 2 \* 2 \* 2 \* 2 \* 2 \*2 \* 2 \* 2 \* 2 \* 2 \* 2 \* 2 \* 2 \* 2 \* 2

In theory you need to split it up by 65536. The book says you shows that you cut the higher bits in half making the last 4 bits page table 1, the next 4 the second page, and the first byte the offset. With this scheme. We would have to create a two-page tables. Then make each row point to a secondary page table. Then finally the frame area. But We just need to cut up 16 bits in a way that meets the 65536. So, 16^4 , 4^8, 2 ^16 are perfect candidates for making a page table. Also, we can mix and match any combination. This means we can have 16\*16\*256 or the mirror of that do 16 \* 4 \* 4 \* 4 \* 4 \* 256. Just keep in mind, whatever is the last page in the lineup that’s how many bytes you need to bring in. for example If I made a paging system that is 256 \* 16 \* 16, I need to read and write 16 bytes of memory when I page in or out.

## Statistics

### Ram1.cpp stats (original)

Page Fault Rate - 19.6%

TLB Hit Rate - 4.9%

### Ram2.cpp (original +modification)

Page Fault Rate - 34.14%

TLB Hit Rate - 3.947%

### Ram3\_128 stats (original + modification + R/W)

Page Fault Rate - 34.14%

TLB Hit Rate - 3.947%

faults that led to writes to bin - 123

### Ram3\_128\_32 stats (original + modification + R/W + bigger TLB)

Page Fault Rate - 33.27%

TLB Hit Rate - 7.133%

faults that led to writes to bin - 117

### Ram3\_128\_8 stats (original + modification + R/W + smaller TLB)

Page Fault Rate - 34.53%

TLB Hit Rate - 1.962%

faults that led to writes to bin - 125

### Ram3\_256 stats (original + modification + R/W + bigger main memory)

Page Fault Rate - 19.6%

TLB Hit Rate - 4.9%

faults that led to writes to bin – 0

### Ram3\_128\_256 (original + modification + R/W + bigger TLB)

Page Fault Rate - 22.04%

TLB Hit RAte - 54.98%

faults that led to writes to bin - 48