ECE/CS 333
Pentium III Memory System
Virtual Memory Handout Solutions

See class handout for the rest of the base information and the diagrams

1. How many entries can the level-1 page table hold?

   Solution: 2^{10} entries. From examining the level-1 page table in the diagram, we see that **10 bits** of the virtual page number are being used to index into the level-1 PDE (page directory entry table).

2. Assume there is **a single task** running on the system

   - The task's **heap** area is allocated in the ***virtual*** address range **0x660000 – 0x666600**
   - The task's **stack** area is allocated in the ***virtual*** address range **0x7999400 – 0x8000000**
   - The task's **data** area is allocated in the ***virtual*** address range **0x1000 – 0x1400**
   - No other sections

   a. How many valid **page table entries** are there?

Solution: **1648 pages**

To solve this problem we need to calculate the number of virtual pages that each segment (heap, stack, data) use.

From examining the diagram, we see that the virtual page number portion of the virtual address is 20 bits. Since the **virtual address is 32-bits** (see diagram) and the addresses are given in hexadecimal, we can figure out the page number by ignoring the lower 3 hexadecimal digits (aka the lower 12 bits).

General method: We can calculate the number of pages for each segment by taking the **difference** of the virtual page numbers. After calculating the number of pages for each segment, they can be totaled to figure out the total number of active/valid page table entries.

(NOTE: remember to round in the correct direction. You need to round because using even a small part of a page requires that a whole page needs to be allocated.).

i. heap– virtual page number range 0x660 to 0x666. Notice that 0x666600 (the end of the address range) is greater than 0x000, **AND** heap addresses grow **upwards**. So, the ending virtual page number needs to be rounded up to the next virtual page number since part of the next virtual page is being used.

So, **0x667 minus 0x660 = 7_{10} pages.**

ii. stack – virtual page number range 0x7999 – 0x8000. The stack is a special case of memory which grows **downwards by common convention (addresses start high and addresses get smaller as new items are allocated)**, so although the starting end of the address range is 0x7999400, to count the fact that it's using part of a page, we need to round **down**.
So, **0x8000 – 0x7999 = 0x666 pages (remember, we're working with hexadecimal)= 1638_{10} pages**

iii. Text – virtual page number range: 0x1 – 0x2. Again, we round up since 0x1400 has lower 12 bits greater than 0x000.

So, **0x2 minus 0x1 = 1_{10} pages.**

b. How many valid **page directory entries** are there? 5 page directory entries are used

Solution Method:

To figure this out, you need to examine the diagram again to find the basic information you need to solve the problem.

The leftmost 10 bits of the virtual address (labeled VP1 in the diagram) are used to index the PDE (page directory entry) table. So, examine how many unique 10-bit values are specified for each of the heap, stack, text. Again, you can do this by calculating the difference.

I'll let you check the conversion of the hexadecimal to binary and only show the binary below for the VP1 fields:

heap – start: 0000000001_2 to end: 0000000001_2 So, the heap uses 1 PDE entry
stack - start: 0000100000_2 (30_{10}) to end: 0000011110_2 (or, 32_{10}) So, the stack uses 3 PDE entries
text – start: 0000000000 to 0000000000 So, the text area uses 1 PDE entry

c. How much memory is in use by the **page directory** and the **page tables**?

Solution: There is one page directory table and one page table, each with 2^{10} entries each. So

2 tables * 2^{10} entries * 4 bytes per entry = 2^{13} bytes = 8 KB of memory…. much smaller than the 4GB needed for a single-level (aka flat) page table.

3. If the Pentium III used a flat page table (single-level), how much space would that page table take up? (NOTE: in the handout, the width of the PDE and PTE entries is given to be 32 bits, so 4 bytes.)

2^{20} * 4 bytes per entry = 4 GB