**Paging**

**Submission:**

* Deadline: Wednesday, November 15, 2023, 8:00 pm HKT.
* Answer ALL questions. Add additional pages if necessary.
* Submit this answer sheet via Canvas->Assignments->Tutorials->Tutorial 6.

**Questions**

1. Before doing any translations, let’s use the simulator paging-linear-translate.py to study how linear page tables change size given different parameters. Compute the size of linear page tables as different parameters change. Some suggested inputs are below; by using the -v flag, you can see how many page-table entries are filled. First, to understand how linear page table size changes as the address space grows, run with these flags:

-P 1k -a 1m -p 512m -v -n 0

-P 1k -a 2m -p 512m -v -n 0

-P 1k -a 4m -p 512m -v -n 0

Then, to understand how linear page table size changes as page size grows:

-P 1k -a 1m -p 512m -v -n 0

-P 2k -a 1m -p 512m -v -n 0

-P 4k -a 1m -p 512m -v -n 0

Before running any of these, try to think about the expected trends. How should page-table size change as the address space grows? As the page size grows? Why not use big pages in general?

**Answer:**

**The size of a linear page table is directly influenced by the size of the address space and the size of each page. As the address space grows, the size of the page table increases because the address space is divided into fixed-size pages, so more pages equate to a larger page table. For example, if the address space is 1MB and the page size is 1KB, there would be 1024 entries in the page table. If the address space increases to 2MB, the page table would double in size to 2048 entries. Conversely, as the page size grows, the size of the page table decreases because larger pages mean fewer pages are needed to cover the same address space. For example, if the address space is 1MB and the page size is 1KB, there would be 1024 entries in the page table. If the page size increases to 2KB, the page table would halve in size to 512 entries. However, we don’t always use large pages because they can lead to internal fragmentation, where the memory being allocated is somewhat larger than the requested memory, leading to wasted space. Large pages can also be less flexible for handling small amounts of data. Therefore, the goal is to balance the size of the page table, which impacts memory overhead, with the size of the pages, which impacts fragmentation and flexibility, based on the specific needs of the system.**

1. Now let’s do some translations. Start with some small examples and change the number of pages that are allocated to the address space with the -u flag. For example:

-P 1k -a 16k -p 32k -v -u 0

-P 1k -a 16k -p 32k -v -u 25

-P 1k -a 16k -p 32k -v -u 50

-P 1k -a 16k -p 32k -v -u 75

-P 1k -a 16k -p 32k -v -u 100

What happens as you increase the percentage of pages that are allocated in each address space?

**Answer:**

**As the percentage of pages allocated in each address space is increased (indicated by the -u flag), an increase is observed in the number of entries in the page table that are actually being used. The control over what percentage of the possible pages are valid (i.e., have been allocated) is provided by the -u flag. For instance, with -u 0, the page table is essentially empty as none of the pages are allocated. As the value of -u is increased, more entries in the page table are used because more pages become valid. By the time -u 100 is reached, every entry in the page table is used as all possible pages are valid. This shows how the size of the page table can be influenced not just by the size of the address space and the page size, but also by the extent to which the address space is actually being used**

1. Now let’s try some different random seeds, and some different (and sometimes quite crazy) address-space parameters, for variety:

-P 8 -a 32 -p 1024 -v -s 1

-P 8k -a 32k -p 1m -v -s 2

-P 1m -a 256m -p 512m -v -s 3

Which of these parameter combinations are unrealistic? Why?

**Answer:**

**-P 8 -a 32 -p 1024 -v -s 1: This combination is unrealistic because it specifies a page size of 8 bytes, which is extremely small for a modern system. Most systems today use page sizes of 4KB or larger.**

**-P 8k -a 32k -p 1m -v -s 2: This combination is more realistic. It specifies a page size of 8KB, an address space size of 32KB, and a physical memory size of 1MB. These are small values, but they could be applicable for a system with limited resources or for certain embedded systems.**

**-P 1m -a 256m -p 512m -v -s 3: This combination is unrealistic because it specifies a page size of 1MB, which is quite large. While large page sizes can be used in certain scenarios to reduce the overhead of managing the page table, a 1MB page size would not be suitable for general-purpose use due to the increased potential for internal fragmentation.**

1. Use the simulator paging-multilevel-translate.py to perform translations. Run:

$ ./paging-multilevel-translate.py -s 3103

You are given the value of the PDBR, a complete dump of each page of memory, and a list of virtual addresses to translate. Solutions to the first two addresses are given below for reference:

* Virtual Address 4a14: Translates to what Physical Address (and fetches what value)? Or Fault?

Virtual Address 4a14:

--> pde index:0x12 [decimal 18] pde contents:0x9a (valid 1, pfn 0x1a [decimal 26])

--> pte index:0x10 [decimal 16] pte contents:0xd8 (valid 1, pfn 0x58 [decimal 88])

--> **Translates to Physical Address 0xb14 --> Value: 17**

* Virtual Address 685e: Translates to what Physical Address (and fetches what value)? Or Fault?

Virtual Address 685e:

--> pde index:0x1a [decimal 26] pde contents:0xbf (valid 1, pfn 0x3f [decimal 63])

--> pte index:0x2 [decimal 2] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])

--> **Fault (page table entry not valid)**

For each of the following virtual addresses, write down the physical address it translates to or write down that it is a fault (e.g., page directory entry not valid, page table entry not valid).

* Virtual Address 5a23
* Virtual Address 14ab
* Virtual Address 257e
* Virtual Address 7988
* Virtual Address 75cf
* Virtual Address 3350
* Virtual Address 0a70
* Virtual Address 55f9

**Answer:**

**5a23:**

**--> pde index: 0x16 [decimal 22] pde content: 0xb5 (valid 1, pfn 0x35 [decimal 53])**

**--> pte index: 0x11 [decimal 17] pte content: 0x7f (valid 0, pfn 0x7f [decimal 127])**

**--> Fault (page table entry not valid)**

**14ab:**

**--> pde index: 0x05 [decimal 5] pde content: 0x7f (valid 0, pfn 0x7f[decimal 127])**

**--> Fault (page table entry not valid)**

**257e:**

**--> pde index: 0x09 [decimal 9] pde content: 0xc4 (valid 1, pfn 0x44 [decimal 68])**

**--> pte index: 0xb [decimal 11] pte content: 0x7f (valid 0, pfn 0x7f [decimal 127])**

**--> Fault (page table entry not valid)**

**7988:**

**--> pde index: 0x1e [decimal 30] pde content: 0xbb (valid 1, pfn 0x3b [decimal 59])**

**-> pte index: 0xc [decimal 12] pte content: 0x9d (valid 1, pfn 0x1d [decimal 29])**

**-> Translates to Physical Address 0x3a8 --> Value: 19**

**75cf:**

**--> pde index: 0x1d [decimal 29] pde content: 0xc6 (valid 1, pfn 0x46 [decimal 70])**

**--> pte index: 0xe [decimal 14] pte content: 0x81 (valid 1, pfn 0x1 [decimal 1])**

**---> Translates to Physical Address 0x2f --> Value: 07**

**3350:**

**--> pde index: 0xc [decimal 12] pde content: 0xd4 (valid 1, pfn 0x54 [decimal 84])**

**--> pte index: 0x1a [decimal 26] pte content: 0xc0 (valid 1, pfn 0x40 [decimal 64])**

**-->** **Translates to Physical Address 0x810 --> Value: 04**

**0a70:**

**--> pde index: 0x2 [decimal 2] pde content: 0xba (valid 1, pfn 0x3a [decimal 58])**

**--> pte index: 0x13 [decimal 19] pde content: 0x87 (valid 1, pfn 0x7 [decimal 7])**

**--> Translates to Physical Address 0xf0 --> Value: 1e**

**55f9:**

**--> pde index: 0x15 [decimal 21] pde content: 0xd5 (valid 1, pfn 0x55 [decimal 85])**

**--> pte index: 0xf [decimal 15] pte content: 0x86 (valid 1, pfn 0x6 [decimal 6])**

**--> Translates to Physical Address 0xd9 --> Value: 1d**