MP2 Report Student ID：B09902046 Name：何翊豪

I. Report: 4.2 Print a Page Table (5%)

1. I got the value of pte, pa and va by the following methods：

pte：the current pagetable+i ,i is the ith PTE of the level(0 <= i < 512), which means printf("%p",pagetable+i); .

pa：First, get the current PTE by "pte_t current_pte = pagetable[i] (0 <= i < 512)". Then we can get pa by "uint64 pa = PTE2PA(current_pte)".

calculation of va：At each level, the new va will be calculated by adding some values from the old va(last level, the initial value of old va = 0). The rules are：

```
if (level == 0) nextva += 4096*(512*512*(uint64)i);
else if (level == 1) nextva += 4096*(512*(uint64)i);
else if (level == 2) nextva += 4096*((uint64)i);
```

Also, i is the ith PTE of the level(0 <= i < 512).

2. From Figure 1, we can know trampoline has the largest virtual address (below MAXVA), and guard page or text has a relatively smaller virtual address. Also, we have some clues about the permissions, such as 0x0000000000001000 doesn't set the U bit. Hence, by the relationship of the value (large or small) of virtual address and the flags, we can know：

0x0000000000000000：text (smallest)

0x0000000000001000：guard page (no U bit on)

0x0000000000002000：stack

0x0000003fffffe000：trapframe

0x0000003ffffff000：trampoline (largest)

3.

Compare	Inverted page table	Multilevel page table
(a) Memory space usage	Require **less memory** to store page table.	Require **more memory** to store page table.
(b) Lookup time /efficiency	Need **more time** to search the table, **bad efficiency**.	Need **less time** to search the table, **better efficiency**.

II. Report: 4.3 Demand Paging and Swapping (10%)

1. In step 5 (Reset Page Table), the page table is changed. By the procedure of demand paging and architecture of xv6, after we bring the missing page back into the physical memory, then we can start to reset the page table.

The physical address (pa) of the page table entry is now indicating the physical address of the frame in physical memory where we bring back our missing page, instead of a page in the disk represented by a previously used block number.

Also, we erase the PTE_S bit, and set the PTE_V bit of the page table entry.

2.

Step 1：Checking the corresponding PTE (page table entry), since PTE_S bit (invalid bit) is on, go to step 2.

Step 2：Trapping in OS, by function “usertrap” in trap.c, due to r_scause() equals to 13 or 15, it calls the function “handle_pgfault”. In “handle_pgfault”, we discover that our demanding page is on back store (disk) by having PTE_S bit, and go to step 3.

Step 3：From step 3 to step 5, the function I called in my implementation is “handle_pgfault”. Step 3 will choose the correct page (block number) we want in the disk, by using PTE2BLOCKNO.

Step 4：Still in function “handle_pgfault”, we will pick a free frame on physical memory (“kalloc()” on xv6), and then we perform “read_page_from_disk” to bring the missing page back in the physical memory. Also, use “bfree_page” to deallocate the page we don’t need any more in the disk.

Step 5：At step 5, in function “handle_pgfault”, we erase PTE_S bit and set PTE_V bit to reset the page table.

Step 6：At last, we can restart the instruction which was interrupted by the whole procedure of demand paging. (i.e., trapping in OS)