Operation System MP2 Report

5.1.1 Print a Page Table

1. Explain how pte, pa and va values are obtained in detail. Write down the calculation formula for va.

PTE:

```
There are 3 levels of Page Table.
level0: PTE_0 = pagetable + level_0_index
level1: PTE_1 = PTE_0 + level_1_index
level2: PTE_2 = PTE_1 + level_2_index
```

PA:

```
uint64 pa = PTE2PA(*pte);
```

VA:

```
uint64 va = ((uint64)level_0_index << 30) + ((uint64)level_1_index <<
21) + ((uint64)level_2_index << 12);
```

2. Write down the correspondences from the page table entries printed by mp2_1 to the memory sections in Figure 1. Explain the rationale of the correspondence. Please take virtual addresses and flags into consideration.

VA	FLAGS	STRUCTURE	RATIONALE
0x0000000000000000	VRWXU	text+data	Start at 0x0000000000000000.
0x0000000000001000	VRWX	guard page	Users don't have permission on guard page.
0x0000000000002000	VRWXU	stack	The page after guard page.
0x0000003fffffe000	VRW	trapframe	Top two pages should be trapframe and trampoline. Trapframe is below trampoline and contains data so is readable and writeable.
0x0000003ffffff000	VRX	trampoline	Top two pages should be trapframe and trampoline. Trampoline is above trapframe contains code so is readable and executable.

3. Make a comparison between the inverted page table in textbook and multilevel page table in the following aspects:

(a) Memory space usage

multilevel page table > inverted page table
Since inverted page table means every proccess share the same page table, and multilevel page table means every process need individual page tables, we can know that memory space usage is **multilevel page table > inverted page table**.

(b) Lookup time / efficiency of the implementation.

inverted page table > multilevel page table
Lookup in a inverted page table need to traverse the whole table, while lookup in the multilevel page table is like lookup a value in an array with the index, much faster than traverse.

Demand Paging and Swapping

In which steps the page table is changed? How are the addresses and flag bits modified in the page table?

At step 5, the address would be the free frame's address and flag bits would be the same as the flag bits before the page being swapped into the backing store(without PTE_S).

Describe the procedure of each step in plain English in Figure 2. Also, include the functions to be called in your implementation if any.

I'm not quite sure

1. Try to access PA with VA, use `walk()`
2. Can't access PA -> traps -> tell OS by `if (r_scause() == 13 || r_scause() == 15)`
3. OS realize that the page we want to access is on the backing store.
4. We need to bring in the missing page from backing store to physical memory.

```
char *mem = kalloc();
memset(mem, 0, PGSIZE);
read_page_from_disk(ROOTDEV, mem, blk_no);
bfree_page(ROOTDEV, blk_no);
```

5. The entry is valid now, we need to update the page table with new flags and address.

```
*page |= PTE_V;
*page &= ~PTE_S;
*page = PA2PTE(mem) | PTE_FLAGS(*page);
```

6. Try everything from the begining again.