MP2 Report

資工二 B10705005 陳思如

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

   pte: the address of the page table entry, so it will be the pagetable address plus i entries for the ith pte.

   $$ith\ pte = pagetable + i$$

   pa: the physical memory address, which means where the page table entry points to.

   $$pa = PTE2PA(pte)$$

   va: the virtual memory address, which is where the aligned pages located at. Since one page table has 512 entry, one page has 4096 bytes, and there are three levels in the multilevel pagetable. So we need to know which level it is at and which entry is it in to get the va.

   if it is in level 1and is the ith entry,

   $$va = i \cdot 512^{(3-1)} \cdot PGSIZE$$

   if it is in level 2 and is the jth entry, the upper level 1 is the ith entry

   $$va = (i \cdot 512^{(3-1)} + j \cdot 512^{2-1}) \cdot PGSIZE$$

   if it is in level 3 and is the kth entry, the upper level 1 is the ith entry, the upper level 2 is the jth entry

   $$va = (i \cdot 512^{(3-1)} + j \cdot 512^{(2-1)} + k \cdot 512^{(1-1)}) \cdot PGSIZE$$

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.

```
$ mp2_1
page table 0x0000000087f57000
+-- 0: pte=0x0000000087f57000 va=0x0000000000000000 pa=0x0000000087f53000 V
|   +-- 0: pte=0x0000000087f53000 va=0x0000000000000000 pa=0x0000000087f52000 V
|       +-- 0: pte=0x0000000087f52000 va=0x0000000000000000 pa=0x0000000087f54000 V R W X U
|       +-- 1: pte=0x0000000087f52008 va=0x0000000000001000 pa=0x0000000087f51000 V R W X
|       +-- 2: pte=0x0000000087f52010 va=0x0000000000002000 pa=0x0000000087f50000 V R W X U
+-- 255: pte=0x0000000087f577f8 va=0x0000003fc0000000 pa=0x0000000087f56000 V
    +-- 511: pte=0x0000000087f56ff8 va=0x0000003fffe00000 pa=0x0000000087f55000 V
        +-- 510: pte=0x0000000087f55ff0 va=0x0000003fffffe000 pa=0x0000000087f65000 V R W
        +-- 511: pte=0x0000000087f55ff8 va=0x0000003ffffff000 pa=0x0000000080007000 V R X
$ QEMU: Terminated
```

From the above result and the knowledge in memory layout, we know that text, data, stack and heap will have the U bit on and others don't. And if the sections have different bits on then it will be seperate in different pages. Also, the Figure 1 shows that 0 is at the bottom and MAXVA at top. Then we can conclude that:

`0-0-0` : data and text, because it's virtual address is lower and it has the U bit on.

`0-0-1` : guard page, because it's at relative low virtual address and it doesn't have the U bit on. That is because guard page is a place used to protect stack overflow.

`0-0-2` : stack, because it's in higher virtual address and has the U bit on.

`255-511-510` : trapframe, because it is in higher virtual address but it's not the highest.

`255-510-511` : trampoline, because it is in the highest virtual address.

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: there is one for each process but limited size.

   Inverted page table: a correspond page table for each occupied frame, every frame must have one.

   When there isn't many stuff in the frame working, the multilevel page table still alloc all the spaces it may need later and keep it empty until pages are mapped. However, the inverted page table only grows pages that matches the frame. So if there is only few frames then the inverted page table has small size. That makes inverted page better because it can **control the size** of the page table based on the frame which minimize the wastage.

   On the other hand, if there is lots of stuff need to be used, the multilevel page table may not be enough. That's because we restricted the size and the levels of the page table. But the inverted page table always have to match with the frame, so it can grow more and **have the flexibility** to match the memory and data.

   (b) Lookup time / efficiency of the implementation

   Multilevel page table: **go through virtual memory address** and pointers to reach the physical memory address.

   Inverted page table: the entry is marked as its **pid and physical frame number**.

   That is, we can implement the inverted page table by **hash map** . Then we can get the physical memory address in O(1). We don't need to go through the whole multilevel page table to find where the physical address is match with the given virtual address.

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

   In step 3, Because of the pagefault, the pagetable is directed to a disk memory by the `PTE2BLOCKNO` .

   In step 4, the data on the disk will be read to a new physical memory space which `kalloc()` made out by `read_page_from_disk()` .

   In step 5, we use `copyout()` to match the physical address with the virtual address. Then we off the PTE_S and on the PTE_V since we now have a physical memory fot this pagetable entry.

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

   Step1: `walk(pagetable, va, alloc)`

   use the virtual address to find the data store in the physical address

   Step2: `usertrap()` `handle_pgfault()` `*pte & PTE_S`

   usertrap happens since the page table entry didn't map to any physical address but the PTE_S is on.

   Step3: `PTE2BLOCKNO(*pte)`

   get the blockno number on the disc to know where to get the data.

   Step4: `read_page_from_disk(ROOTDEV, pa, blockno)` `kalloc()`

   move out the data on the disc into the physical memory address which is pointed by the page table

   Step5: `*pte &= ~PTE_S` `*pte |= PTE_V` `pa2pte()`

   since we have moved the swapped pages back to the physical memory. we have to turn off the PTE_S and turn on the PTE_V and match the physical memory with the virtual address.

   Step6:

   return from the usertrap condition and return the physical address that it wants.