OS2022 MP2 Report

4.2.1 Print a page table

1.Explain how pte, pa values are obtained in detail, and the calculation of va in the vmprint() output. Literally how they are done in your code.

Ans: 透過`riscv.h`可知道，`pageable_t`是pte的指標。且一個page size為4096Byte而page table entry的大小為8Bytes，因此總共有512個entry在每個table上。因此我們可以透過iterate每一個entry來取得PTE，而pa可以透過`PTE2PA`這個macro得到。而va的部分，因為我們知道每個entry都會向下指到一個大小為4KB的table，而且最後指到的frame size會等於page size，可知在最後一層page table上，每走一個entry，所對應到的virtual address會多4096，而第二層page table所對應之的virtual address每走一個entry會多$4096 \cdot 512$，第一層page table所對應之的virtual address每走一個entry會多$4096 \cdot 512 \cdot 512$，且一開始的virtual address是從0開始，因此可以透過這樣的規律計算va。

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

Ans:
entry: 0-0-0是對應到text and data，當我們在`mp2_1.c`中建立一個global variable並且把它的位址印出時即可看出他的virtual address在這個範圍內。
entry: 0-0-1是對應到guard page，因為他的`PTE_U` bit沒有set。
entry: 0-0-2是stack，可以透過在`mp2_1.c`中建立一個local variable並把它的位址印出來即可出他的virtual address在這個範圍內。
entry: 255-511-510是trapframe，因為它的`PTE_X` bit沒有set，而且`mp2_1.c`沒有`sbrk()`因此也不會有heap的virtual address，所以可知道是trapframe。
entry: 255-511-511是trampoline，因為他是所有的address最大的。

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

Ans:
(a) Memory space usage:
如果一個process在physical memory有N個frame，inverted page table就只需要N個entry。Multilevel page table雖然也是有N個entry指到physical memory，但它需要前幾個level的table才能指到最後一個實際指到physical frames的table，因此它的space usage較多。
(b) Lookup time / efficiency of the implementation.
因為inverted page table需要一一做linear search，因此會花比較久的時間才能找到virtual address對應到的physical address。而multilevel page table可以在constant time查出對應的frame，但它要花的memory access time較多。如果frame數量很多，由上述可知multilevel page table將會較efficiency。

4.3 Demand Paging and Swapping

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

Page Table會在第五步更改。在更動以前，可以透過address找到一個disk block，且address的`PTE_S`會on，但`PTE_V`會off；而在更動後，可以透過address找到一個physical memory的frame且`PTE_S`會off，但`PTE_V`會on。

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: kernel會去查page table。
Step2: 如果發現`PTE_V` off，就會發生page fault。在`usertrap`這個function透過`r_scause`去看錯誤的原因，如果是因為讀寫不存在的page，就透過`handle_pgfault`來處理。
Step3: 在`handle_pgfault`透過`r_stval`取得發生錯誤的是哪個va，並且透過`walk`取得pte，並查看其`PTE_S`是否on。
Step4: 如果`PTE_S`有on，透過`read_page_from_disk`在secondary storage去讀入data。
Step5: 改動address，以pa取代block number，並且將`PTE_V` on且將`PTE_S` off。
Step6: 如果`handle_pgfault`成功，就會回傳`0`，否則回傳負值。若成功之後會重新做一次本來在做的instruction。