MP2 report

1.1

要獲得 pte 的話，我是直接把 pagetable 當作一個 512 個大小的 array 去使用，pagetable[i]就是 pte 了。當然標準作法是用 walk，但 walk 比較適合用在 va 對應的 pa 在很深的 level 時用的。因為在 vmprint 中，每一層都要去搜索，每個 pte 都要查詢，所以把 pagetable 當陣列去搜索其實就可以了。

Pa 的話…就 PTE2PA 啊。有 PTE 就有 Pa 了。

Va 的話比較麻煩。根據 xv6 的結構，是把 v 分成 25 個無用 bits，3*9bits 的 index，還有 12bits 的 offsets。我在 vmprint 中是看當下是在哪一層 pagetable 去決定 va 怎麼算的。要找 index 的話，就記錄說這個 pte 是 512 個中的第 i 個位置，這個就是 index 了。但因為層數在後面的 pte 在算 va 時要用到 parent 的 pte 的 index，所以我就直接把 index 當成函式的參數傳下去了。最後再根據 xv6 的結構和有的 index 去轉換位數就好。第一層是 index1<<30，第二層是 index1<<30+ index2<<21，第三層是 index1<<30+ index2<<21+ index3<<12。

1.2

可以看到有 data 是在 0x0000003fc0000000 之後的。看圖和參考

MAXVA 可知這是 trampoline 和 trapframe 的 data。至於 va=0 的 data 就是 stack 裡的東西，因為後面得 testcase 可以知道在 main 裡修改的資料會影響到這個部分。

1.3

在記憶體使用的部分，inverted page table 一定大很多，因為是給所有 process 的。每個 process 都有各自的 stack、trampoline、trapframe 的 data 要存。而且只有 1 層的話，就會有 27bits 的 index，需要 2^27 大小的 array。這非常大。而 multilevel page table 的話，雖然最多也是約 2^27 大小，但因為可以沒用的 pte 就不用 allocate 下層的 pagetable，所以事實上會少很多。像 mp2_1 中，地 1 個 pagetable 有 512 個，只有 2 個有效，再下一層也只有 2 個有效，所以隨便估都不會超過 10 萬個 pte，明顯比 2^27 少很多。

搜索時間的話，inverted page table 是 O(n)，multilevel page table 類似 divide 是 O(logn)，multilevel page table 比較少。實作上 inverted page table 比較簡單，畢竟只有一層。multilevel page table 還要處理要不要 allocate 和第幾層、指到哪裡之類的問題，比較複雜。

3.1

第 5 步。都說是 reset page table 了。spec 有說，flag 的話，V、

R、W、X、U 要 on 起來。Address 的話，會把 disk 中 pa 的值存回來，把 blockno 覆蓋掉，flag 去掉 S 加上 V。

首先第一步修改資料時，process 會檢查 page table，然後如果沒有，就會第二步 trap system，進入 sys_sbrk，進入 usertrap，檢查是 page fault，傳到 page fault handler，然後處理。找到 va，kalloc 去 allocate，memset 為 0。page fault handler 中間有 mappages，那個就包含第 3、4 步的 disk I/O 的操作了。確認 page 在 disk，把 page 拿出來。這裡的 free frame list 實作上應該是在 myproc()->sz 允許的範圍內，PTE_V 還沒 on 起來的 frame。然後修改，PTE_V on 起來，就變成 page table 的一部份了。

Swapping 的話，一樣找到 ca，kalloc 去 allocate，memset 為 0，然後用 balloc_page 宣告一個 blockno，write page to disk 函式直接用，kfree 原本的 pa，修改 pte。如果是 read 就 read page from disk，bfree_page 原本的 blockno。