OS LAB ASSIGNMENT -3

NAME:-ASHWIN KR
ROLL:- CE21BTECH11008

VMPRINT() Code:-

```
void vmprintRecursive(pagetable_t pagetable, int level) {
 for(int i = 0; i < 512; i++){
   pte_t pte = pagetable[i];

//if both pte and valid of page table entry is non zero then go
   if(pte & PTE_V) {
     uint64 child = PTE2PA(pte);
     for(int i = 0; i <= level; i++) {
       printf("..");  //denotes how deep tree is
       if(i != level) printf(" ");
     }

     printf("%d: pte %p pa %p\n", i, pte, child);
     if((pte & (PTE_R | PTE_W | PTE_X)) == 0)
         vmprintRecursive((pagetable_t)child, level + 1);
       //again calling cmprintRecursive function with increasing the label
   }
 }
}

void
vmprint(pagetable_t pagetable) {
 printf("page table %p\n", pagetable);
 vmprintRecursive(pagetable, 0);
}
```

Q1. Print the page table of init process

```
xv6 kernel is booting

hart 2 starting
hart 1 starting
page table 0x0000000087f6c000
..0: pte 0x0000000021fda001 pa 0x0000000087f68000
.. ..0: pte 0x0000000021fd9c01 pa 0x0000000087f67000
.. .. ..0: pte 0x0000000021fda41b pa 0x0000000087f69000
.. .. ..1: pte 0x0000000021fd9817 pa 0x0000000087f66000
.. .. ..2: pte 0x0000000021fd9407 pa 0x0000000087f65000
.. .. ..3: pte 0x0000000021fd9017 pa 0x0000000087f64000
..255: pte 0x0000000021fdac01 pa 0x0000000087f6b000
.. ..511: pte 0x0000000021fda801 pa 0x0000000087f6a000
.. .. ..510: pte 0x0000000021fdd007 pa 0x0000000087f74000
.. .. ..511: pte 0x0000000020001c0b pa 0x0000000080007000
```

Q2. Print the page table of sh process

```
init: starting sh
page table 0x0000000087f5f000
..0: pte 0x0000000021fd6c01 pa 0x0000000087f5b000
.. ..0: pte 0x0000000021fd6801 pa 0x0000000087f5a000
.. .. ..0: pte 0x0000000021fd701b pa 0x0000000087f5c000
.. .. ..1: pte 0x0000000021fd641b pa 0x0000000087f59000
.. .. ..2: pte 0x0000000021fd6017 pa 0x0000000087f58000
.. .. ..3: pte 0x0000000021fd5c07 pa 0x0000000087f57000
.. .. ..4: pte 0x0000000021fd5817 pa 0x0000000087f56000
..255: pte 0x0000000021fd7801 pa 0x0000000087f5e000
.. ..511: pte 0x0000000021fd7401 pa 0x0000000087f5d000
.. .. ..510: pte 0x0000000021fdb407 pa 0x0000000087f6d000
.. .. ..511: pte 0x0000000020001c0b pa 0x0000000080007000
```

Q3. Print the page table of user-level sleep process (refer to Lab Assignment 2)

```
$ sleep-CE21BTECH11008 10
page table 0x0000000087f42000
..0: pte 0x0000000021fcf801 pa 0x0000000087f3e000
.. ..0: pte 0x0000000021fcf401 pa 0x0000000087f3d000
.. .. ..0: pte 0x0000000021fcfc1b pa 0x0000000087f3f000
.. .. ..1: pte 0x0000000021fcf017 pa 0x0000000087f3c000
.. .. ..2: pte 0x0000000021fcec07 pa 0x0000000087f3b000
.. .. ..3: pte 0x0000000021fce817 pa 0x0000000087f3a000
..255: pte 0x0000000021fd0401 pa 0x0000000087f41000
.. ..511: pte 0x0000000021fd0001 pa 0x0000000087f40000
.. .. ..510: pte 0x0000000021fd8007 pa 0x0000000087f60000
.. .. ..511: pte 0x0000000020001c0b pa 0x0000000080007000
Time for sleep is 10 $
```

Q4. What is the size of the page frame, number of entries in a page table and address bits of virtual address space and physical address space in xv6?

Ans-Xv6 runs on Sv39 RISC-V thus-

Size of page frame- 4KB

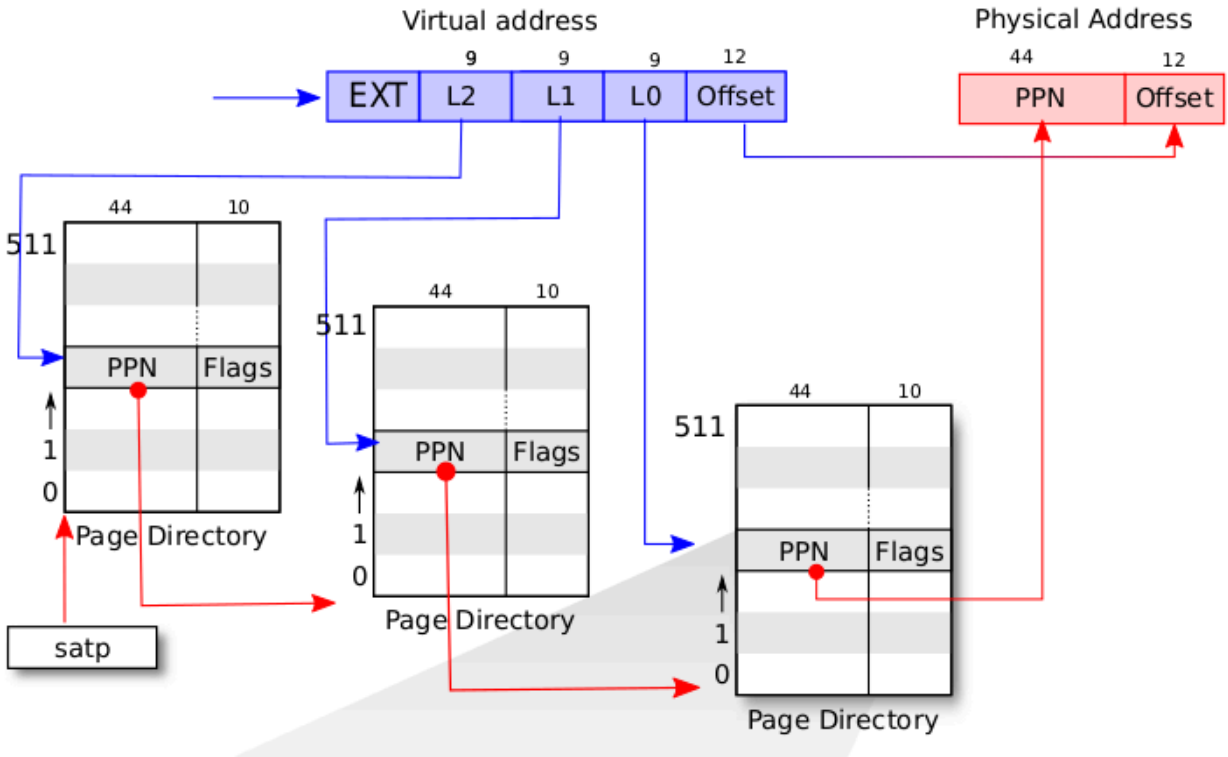
Number of entries in a page table = 512 (three level caching -in each page else total 2^27)

Address bits of virtual address space = 64 bits (out of which 39 bits are used)

Address bits of physical address space = 56 bits (44 from page table entry + 12 bits from offset)

Q5. How does xv6 allocate bits of virtual address space for multi-level paging and offsetting?

sol-There are 64 bits virtual address space out of which only 39 are used for multi-level paging. rest 25 bits are not used. It uses three level paging.After dedicating 12 bits for offset because size of each page table entry is 4KB leading to 9 bits and thus 27 bits are left. This is divided into three part of 9 bits each.First 9 bits is used for indexing in first point table,second 9 bits for intermediate page table and rest 9 bits for lowest level page table that keeps the information of part of actual address.Although a CPU walks the three-level structure in hardware as part of executing a load or store instruction, a potential downside of three levels is that the CPU must load three PTEs from memory to perform the translation of the virtual address in the load/store instruction to a physical address.



Q6. Given a pte, how do you determine whether it's valid (present) or not and how do you determine page frame number in pte is of next level page table or user process'? List out some valid pte entries from one of the Q1/Q2/Q3 which fall under above two categories.

Sol- Given page table entry is 54 bits out of 10 used in flags and the rest 44 pointing to the next page table or memory address. Out of 10 bits least significant bit i.e. rightmost bits represent valid or invalid bits. If the last bit is 1 then it is valid (present) and can be accessed while if the valid bit is 0 then it is not present and thus has to load.

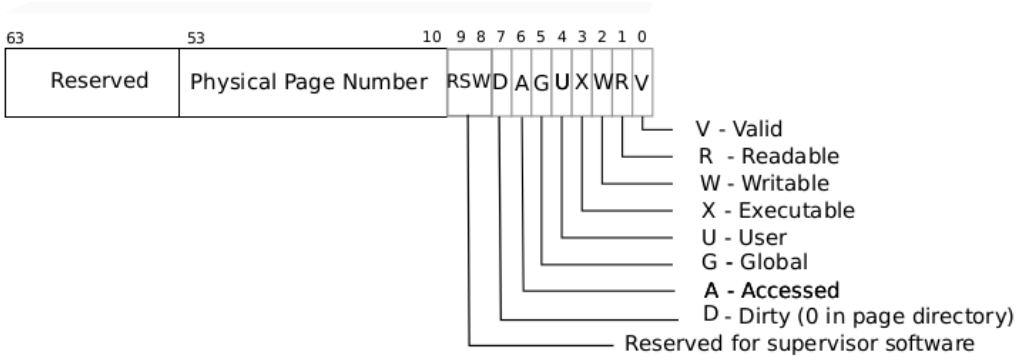


Figure 3.2: RISC-V address translation details.

This is the page table entry from sleep process and entry after pte written is page table entry .
$ sleep-CE21BTECH11008
page table 0x0000000087f43000
..0: pte 0x0000000021fcfc0**1** pa 0x0000000087f3f000
.. ..0: pte 0x0000000021fcf80**1** pa 0x0000000087f3e000
.. .. ..0: pte 0x0000000021fd001**b** pa 0x0000000087f40000
.. .. ..1: pte 0x0000000021fcf41**7** pa 0x0000000087f3d000
.. .. ..2: pte 0x0000000021fcf00**7** pa 0x0000000087f3c000
.. .. ..3: pte 0x0000000021fcec1**7** pa 0x0000000087f3b000
..255: pte 0x0000000021fd080**1** pa 0x0000000087f42000
.. ..511: pte 0x0000000021fd040**1** pa 0x0000000087f41000
.. .. ..510: pte 0x0000000021fdcc0**7** pa 0x0000000087f73000
.. .. ..511: pte 0x0000000020001c0**b** pa 0x0000000080007000

Here last 4 bits of the page table entry is marked bold like 1,b,7
1=0001
7=0111
b=1011 all have LSB as 1 means it is the valid bit and clearly seen above there page address is printed over as there entry is a valid entry.

The above page table is showing only those entries which are valid while invalid entries will go and bring and finally will make the LSB valid.Don't print PTEs that are not valid. Thus other page entries than this will be invalid.

Q7. By referring to the pagetables of init/sh/sleep processes, fill out the following table. Please explain your response as remarks, in brief.

	init process	sh process	sleep process	Remarks
No. of page frames consumed by the page table				
Internal fragmentation in the page table(s) in bytes				
No. of page frames allocated for the process				
No. of page frames allocated for TEXT segment of the process and their physical addresses				
No. of page frames allocated for Data/Stack/Heap segments of the process				
and their physical addresses				
Any dirty pages?				
Any kernel mode (controlled) page frames?				

In this we have to find number of pages assigned ot text segment and pages assigned to data/stack/heap. This can be done through the flags associated with virtual memory. Both have their specific properties like

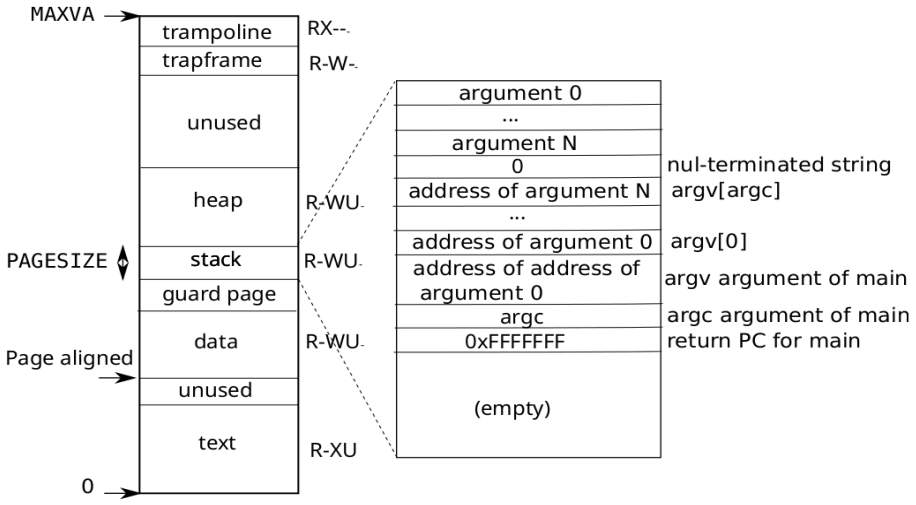


Figure 3.4: A process's user address space, with its initial stack.

Like it can be seen that Text have permission of RXU -have permission of read,execute and user but no write permission thus it ensures that it cant modify the code segment itself . while data/stack/heap have permission RWX - have read write and execute.
For the dirty bit we can check the 8th bit from LSB(left most) .
See figure 3.2 above for clear distinction of flag.

No of page frame allocated for the proces:- 1 page table entry = 512 frames

No. of page frames allocated for TEXT segment of the process and their physical addresses = RXU bit will be 1 while W bit =0
DAGU XWRV == DAG1 1011
PAGES ENDING WITH 1b is necessarily text segment=marked in red

No. of page frames allocated for Data/Stack/Heap segments of the process
addresses = RWX bit will be 1 while W bit =0
DAGU XWRV == DAG1 1111
PAGES ENDING WITH 17 is necessarily Data/stack/heap segment=marked in BLUE

Any dirty pages?
8th bit (from left side) with 1 will represent dirty bit. Here no dirty bit.

Any kernel mode (controlled) page frames?
5 th bit denote U if 1 then in user mode else if U=0 then in kernel mode. Marked with **Black.**

//this is for init process
page table 0x0000000087f6c000
..0: pte 0x0000000021fda001 pa 0x0000000087f68000
.. ..0: pte 0x0000000021fd9c01 pa 0x0000000087f67000
.. .. ..0: pte 0x0000000021fda41b pa 0x0000000087f69000
.. .. ..1: pte 0x0000000021fd9817 pa 0x0000000087f66000
.. .. ..2: pte 0x0000000021fd9407 pa 0x0000000087f65000
.. .. ..3: pte 0x0000000021fd9017 pa 0x0000000087f64000
..255: pte 0x0000000021fdac01 pa 0x0000000087f6b000
.. ..511: pte 0x0000000021fda801 pa 0x0000000087f6a000
.. .. ..510: pte 0x0000000021fdd007 pa 0x0000000087f74000
.. .. ..511: pte 0x0000000020001c0b pa 0x0000000080007000
No of page frame consumed = 6 i.e. 0,1,2,3,510,511
Internal fragmentation=out of 512 entry only 6 are valid in last page table
=(512-6)*each page table entry* = *506**44 bits =2783 Bytes

//this is for sh
page table 0x0000000087f5f000
..0: pte 0x0000000021fd6c01 pa 0x0000000087f5b000
.. ..0: pte 0x0000000021fd6801 pa 0x0000000087f5a000
.. .. ..0: pte 0x0000000021fd701b pa 0x0000000087f5c000
.. .. ..1: pte 0x0000000021fd641b pa 0x0000000087f59000
.. .. ..2: pte 0x0000000021fd6017 pa 0x0000000087f58000
.. .. ..3: pte 0x0000000021fd5c07 pa 0x0000000087f57000
.. .. ..4: pte 0x0000000021fd5817 pa 0x0000000087f56000
..255: pte 0x0000000021fd7801 pa 0x0000000087f5e000
.. ..511: pte 0x0000000021fd7401 pa 0x0000000087f5d000
.. .. ..510: pte 0x0000000021fdb407 pa 0x0000000087f6d000
.. .. ..511: pte 0x0000000020001c0b pa 0x0000000080007000

No of page frame consumed = 7 i.e. 0,1,2,3,4,510,511
Internal fragmentation=out of 512 entry only 6 are valid in last page table
=(512-7)*each page table entry* = *506**44 bits =2777.5 Bytes

Number of page frame allocated for the process

//this is for sleep process
$ sleep-CE21BTECH11008 10
page table 0x0000000087f42000
..0: pte 0x0000000021fcf801 pa 0x0000000087f3e000
.. ..0: pte 0x0000000021fcf401 pa 0x0000000087f3d000
.. .. ..0: pte 0x0000000021fcfc1b pa 0x0000000087f3f000
.. .. ..1: pte 0x0000000021fcf017 pa 0x0000000087f3c000
.. .. ..2: pte 0x0000000021fcec07 pa 0x0000000087f3b000
.. .. ..3: pte 0x0000000021fce817 pa 0x0000000087f3a000
..255: pte 0x0000000021fd0401 pa 0x0000000087f41000
.. ..511: pte 0x0000000021fd0001 pa 0x0000000087f40000
.. .. ..510: pte 0x0000000021fd8007 pa 0x0000000087f60000
.. .. ..511: pte 0x0000000020001c0b pa 0x0000000080007000

No of page frame consumed = 6 i.e. 0,1,2,3,510,511
Internal fragmentation=out of 512 entry only 6 are valid in last page table
=(512-6)*each page table entry* = *506**44 bits =2783 Bytes