MP2 Report

t11902210 張一凡

Part 1: Please answer the following questions:

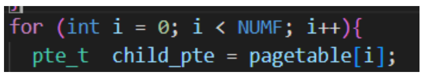
1. Explain how pte, pa and va values are obtained in detail. Write down the calculation formula for va.

Answer:

PTE, PA, and VA are obtained through a process called virtual memory management, which allows the computer to use more memory than it physically has by temporarily transferring data from the RAM to the hard disk. Next, I will explain three addresses from universal obtain and obtain in this topic

(1)PTE (Page Table Entry)

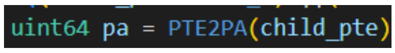
The PTE value is obtained by multiplying the virtual address by the size of the page table entry (8 bytes on 64-bit systems), and then adding the base address of the page table. In MP2, I used pagetable[i] to directly find the value of pte and print it out, where NUMF is 512=64 * 8, because pagetable is 64 bytes, and each byte is 8 bits



```
for (int i = 0; i < NUMF; i++){
  pte_t  child_pte = pagetable[i];
```

(2) PA (Physical Address)

The physical address is obtained by taking the page frame number from the PTE and multiplying it by the size of a page (usually 4KB). In MP2, I directly use the tool function PTE2PA in the xv6 system to calculate the storage address of pa.



```
uint64 pa = PTE2PA(child_pte)
```

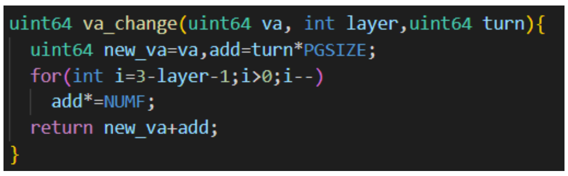
(3) VA (Virtual Address)

The VA value is obtained by adding the additional address to the page number. The additional address is the part of the virtual address that identifies the byte within the page, while the page number identifies the page in memory. In MP2, because the va address is determined, specific formulas and calculation methods are required, as follows.

$$VA = (Page\ Number \times Page\ Size) + Addition$$

The Page Number is the page number for the virtual address, and it stands for NUMF or NUMF*NUMF in mp2. The Page Size is the size of a page in bytes. (It is concrete as 4096 in mp2). And the Addition is within the page in bytes, it is the fixed address or determined by the former level.

What I call in vmprint is a recursive function to print, so the calculation function of va is relative to the data returned from the previous layer of printing, so it is based on the number of layers to determine the address to be added.

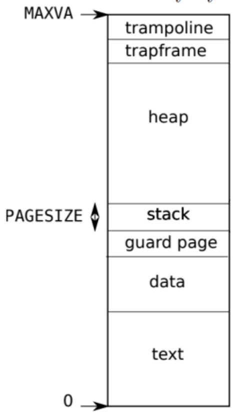


```
uint64 va_change(uint64 va, int layer,uint64 turn){
  uint64 new_va=va,add=turn*PGSIZE;
  for(int i=3-layer-1;i>0;i--)
    add*=NUMF;
  return new_va+add;
}
```

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.**

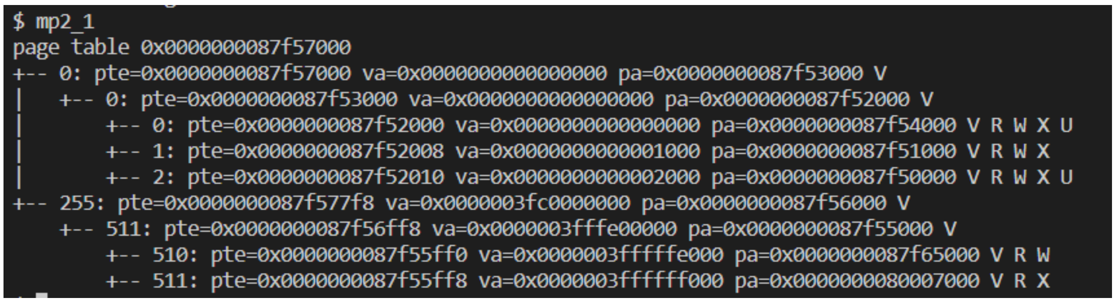
Answer:

Figure 1: The virtual memory layout for xv6



In Figure 1, each memory section represents a corresponding and specific range of physical memory addresses. The correspondence between page table entries and memory parts is established based on the page frame numbers contained in the page table entries. The page frame number represents the physical location of the page in memory, which can be mapped to a specific memory part in Figure 1.

Trampoline and test correspond to the maximum and minimum memory addresses, respectively. Then, according to my output example, the corresponding names can be obtained according to the layout order of each space. At the same time, the specific names can also be determined based on U X V W S, etc



```
$ 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
```

va	Memory name	reason
0x0000000000000000	text	the lowest address X bit is on
0x000000000000100	guard page	U bit is not on
0x0000000000002000	stack	
0x0000003fffffe000	trapframe	va matches TRAPFRAME
0x0000003ffffff000	trampoline	the highest address va matches TRAMPOLINE

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

(a) Memory space usage

(b) Lookup time / efficiency of the implementation.

Answer:

(a) The inverted page table requires less memory than the multilevel page table.

Inverted page tables use a table to map virtual addresses to physical addresses, which requires much less memory than traditional page tables because it does not need to store information about each virtual page. A multi-layer page table uses a hierarchical structure of tables to map virtual addresses to physical addresses, which requires more memory than an inverted page table because it needs to store information about each virtual page and requires additional tables and pointers to implement the hierarchical structure.

(b) The inverted page table provides faster lookup times than the multilevel page table, but the multilevel page table can be more efficient for certain operations.

The inverted page table provides fast lookup time because it only needs to search one table to find the mapping between virtual and physical addresses. However, this method requires searching the entire table, so its efficiency may be lower. A multi-level page table provides slower lookup time than an inverted page table, as it requires searching multiple tables to find the mapping between virtual and physical addresses. However, this method can use a hierarchical structure to quickly locate relevant page tables, thereby improving efficiency.

Part 2:

The worlflow of demand paging. Silberschatz, A., Galvin, P. B., & Gagne, G. Operating system concepts.

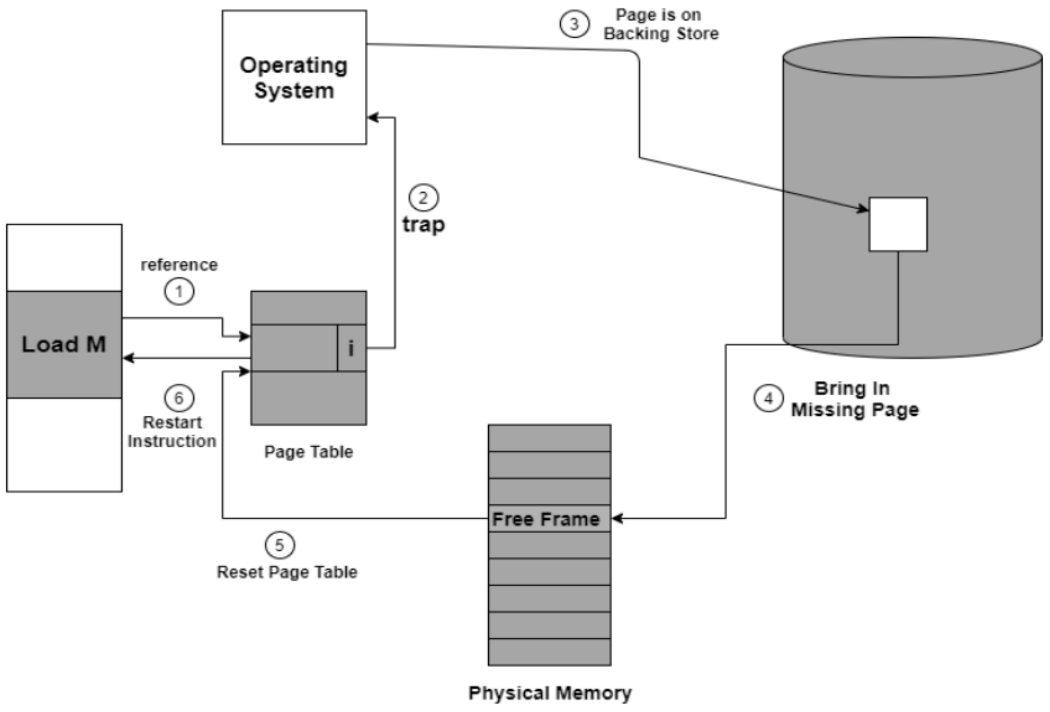


Figure 2 shows the workflow of demand paging in several steps. Please answer the following questions:

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

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

Answer:

(a) Change in Step 5

After the memory is brought back to the missing page according to the program instructions, the address previously transferred from the pagetable cannot be directly used. Instead, the pagetable data area and the physical memory space of the PA must be reallocated. (If there is not enough space and it is necessary to reapply, use the relevant solution for page fault.) At the same time, it is necessary to correct the V and S data tags by using the methods prompted in the appendix and the program, including set and unset, and finally modify P through the | operator again.

(b) See the table below for detailed explanations of each step.

Step	Name	Explain & Function
1	Reference	Determine the corresponding obtain data location of the PA address memory by traversing the available data space in the page table (if there is no valid location, provide instructions to the operating system). Need function: madvise()
2	Trap	Due to the invalidation of the entry calculated in the reference step, it is necessary to submit the operating system for problem modification and call related functions to allocate physical space or find swapping backing store （disk） space. Need function: handle_pgfault()
3	Page is on backing store	The page is called in the previous step. The handle_pgfault() function found the correct backing store （disk） space available for obtainment. Need function: walk()
4	Bring in missing page	Some space on physical memory is allocated, and the page data on the backing store (disk) is read to that space. Need function:begin_op(),read_page_from_disk(), bfree_page() , end_op()
5	Reset page table	Due to the backflow of backing store (disk) data, the physical space allocation has changed, so the page table has changed, creating a new page table for recording the address storage of virtual memory. Need function: madvise()
6	Restart instruction	Returning to the original process is equivalent to solving the data exchange problem between backing store（disk） and physical space through the completed step process. Need function: madvise()