MP2 Report

b09902004 資工二 郭懷元

Print a Page Table

1.

- Suppose the variable for page table is `pt`, and we are at `idx`-th entry.
- `pte` is the physical address of the entry, so it's `&pt[idx]`.
- `pa` is the address that this entry points to, so it's `PTE2PA(pt[idx])`.
- Because my `vmprint()` is implemented recursively, calculation for `va` is like this:
  - At each level of page table, we have `va_st`, which is the virtual address the parent entry points to.
  - `va` of `idx`-th entry at level `l` with `va_st` is:

    ```
    va_st + ((uint64)idx << (12 + 9 * l))
    ```

2.

- `0: va=0x0000000000000000 V R W X U` `1: va=0x0000000000001000 V R W X` `2: va=0x0000000000002000 V R W X U`: text

  Because they are at the lowest address and the `X` bit is on.

- `510: va=0x0000003fffffe000 V R W`: trapframe

  Because `va` matches `TRAPFRAME` defined in `kernel/memlayout.h`

- `511: va=0x0000003ffffff000 V R X`: trampoline

  Because `va` matches `TRAMPOLINE` defined in `kernel/memlayout.h`

3.

- Memory space usage

    - Inverted page table uses less memory space because the OS can maintain just 1 table for all processes.

- Lookup time / efficiency of the implementation
    - Assuming both are implemented as an array (O(1) lookup time knowing index).
    - Assuming the size of virtual memory and physical memory are around the same order of magnitude.
    - Multilevel page table has better lookup time efficiency because the OS can go directly to the entry that `va` points to. With inverted page table, it would have to go through the entire table.

Demand Paging and Swapping

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

Page table is changed on step 5.

The address is changed from the block number on disk to physical memory address. `PTE_V` is flipped on, and `PTE_S` is flipped off. Other flags remain the same as stored in the page table.

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

1. Reference
    - The process looks up its page table to get physical address of the target virtual address.
2. Trap
    - Because the entry isn't valid, a page fault is triggered and the page fault handler starts to work.
    - Functions called: `handle_pgfault()`
3. Page is on backing store
    - The page fault handler finds that the target page is stored on backing store (likely a disk).
    - Functions called: `walk()`
4. Bring in missing page
    - Some space on physical memory is allocated, and the page data on backing store is read to that space.
    - Functions called: `begin_op()` , `read_page_from_disk()` , `bfree_page()` , `end_op()` .
5. Reset page table
    - The page table entry is updated with the physical address of the page. Valid bit and swap bit are also updated.
6. Restart instruction

    - Program counter and registers are restored, the original process continues.