Paging improvements in xv6

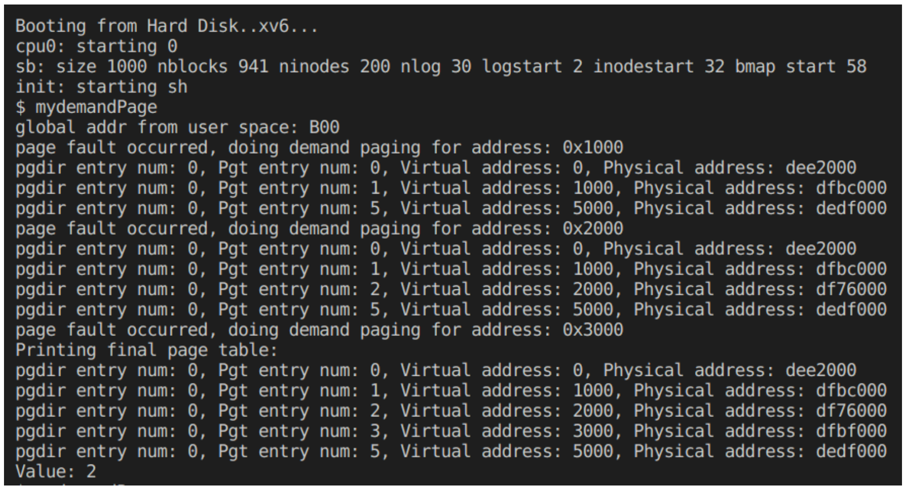
Aaryan, CO21BTECH11001

Part 1:

The tar file of this part is named as `xv6_part1.tar.gz`
In the first part of the assignment, we implemented `mydemandPage` system call.
Some important changes made to implement the same are:

1. In `trap.c`, switch case for `T_PGFLT` (page fault) is added at line 81.
2. In `exec.c`, `sz += ph.memsz - ph.filesz;` is added to prevent allocating memory space for dynamic variables and allocate for the read-only code/data.
3. `mydemandPage.c` file is made to test the system call.

Here is a result of the same using `N = 3000`.



```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ mydemandPage
global addr from user space: B00
page fault occurred, doing demand paging for address: 0x1000
pgdir entry num: 0, Pgt entry num: 0, Virtual address: 0, Physical address: dee2000
pgdir entry num: 0, Pgt entry num: 1, Virtual address: 1000, Physical address: dfbc000
pgdir entry num: 0, Pgt entry num: 5, Virtual address: 5000, Physical address: dedf000
page fault occurred, doing demand paging for address: 0x2000
pgdir entry num: 0, Pgt entry num: 0, Virtual address: 0, Physical address: dee2000
pgdir entry num: 0, Pgt entry num: 1, Virtual address: 1000, Physical address: dfbc000
pgdir entry num: 0, Pgt entry num: 2, Virtual address: 2000, Physical address: df76000
pgdir entry num: 0, Pgt entry num: 5, Virtual address: 5000, Physical address: dedf000
page fault occurred, doing demand paging for address: 0x3000
Printing final page table:
pgdir entry num: 0, Pgt entry num: 0, Virtual address: 0, Physical address: dee2000
pgdir entry num: 0, Pgt entry num: 1, Virtual address: 1000, Physical address: dfbc000
pgdir entry num: 0, Pgt entry num: 2, Virtual address: 2000, Physical address: df76000
pgdir entry num: 0, Pgt entry num: 3, Virtual address: 3000, Physical address: dfbf000
pgdir entry num: 0, Pgt entry num: 5, Virtual address: 5000, Physical address: dedf000
Value: 2
```

Observations:

1. When the size of the array is increased, the number of page table entries increases.
2. Here is data of size of array, number of page faults and final number of pages

Size of array	Number of page faults	Final number of pages
3000	3	5
5000	5	7
10000	10	11

3. This data verifies that the page size in xv6 is 4KB.

Part 2:

The tar file of this part is named as `xv6_part2.tar.gz`
In the second part of the assignment, we implemented copy-on-write (COW) in xv6.
Some important changes made to implement the same are:

1. In `proc.c`, a function named `copyuvm` is called during `fork`. It is replaced with the function `copyuvm_COW`, which will be implemented in `vm.c`
2. The function `copyuvm` takes two parameters: `pgdir` is the parent's page directory, `sz` is the size of the user-mode memory (`proc->sz`). `copyuvm` first maps the kernel to the child process (`setupkvm`). The return value of `setupkvm` is the page directory for the child process. For each user page, `copyuvm` first reads the parent's page table entry (`walkpgdir`), which consists of the address of the physical frame and the protection bits. It then allocates a new page (`kalloc`) and copies the parent's memory to this page (`memmove`).
   The new function named `copyuvm_COW` which doesn't allocate a new page to the child process. Instead, it creates a new page table for the child process and maps the parent's physical frame read-only in both parent and child. At last, it also flushes the TLB.
3. A function named `read_cr2` is implemented in `trapasm.S`, which is used to read the control register `cr2`.
4. A function named `flush_all_tlb` is implemented in `trapasm.S`, which is used to flush the TLB.
5. A page fault handler `handle_pgflt` is implemented in `vm.c`
6. `getReferenceCount`, `decrementReferenceCount`, `incrementReferenceCount` functions are implemented in `kalloc.c`, which takes a physical address as argument.
7. `getReferenceCount` is used to get the number of pages that map their virtual address to physical address.
8. `decrementReferenceCount` is used to decrement the number of pages that map their virtual address to physical address. This function is called when a page fault occurs and a copy is allocated to some process.
9. `incrementReferenceCount` is used to increment the number of pages that map their virtual address to physical address. This function is called when a new child process is created.
10. A file named `myCOW.c` is made to test the functionality of COW.
11. `pgtPrint` system call is changed to print the write bit as well.

Observations:
1. Number of page table entries increases with the increasing number of elements in the array.
2. When there is no child of a parent, the write bit is 1, but after `fork`, write bit becomes 0.
3. When either child or parent asks for write permission, it is handled by the page fault handler and the write bit for the process who asked for write permission is changed to 1.