Operating Systems-2 CS3523

Programming Assignment-6: Implementing demand paging and copy-on-write.

- B. Asli Nitej Reddy

- CS21BTECH11011

Implementation:

- As a part of assignment, I implemented the demand paging which is not implemented in xv6.
- Changes done are modification in exec.c and trap.c
- In trap.c I added the code in the default case after (checking if there is any mistake in kernal if yes, it's our mistake so using the panic("trap"))
- After the above thing I added code for the special case what we created i.e. if there is error because of pagefault and if there is error we go into the if clause like below figure

```
if (tf->trapno == T_PGFLT)
{

    uint a = PGROUNDDOWN(rcr2());
    cprintf("page fault occurred, doing demand paging for address: %x\n", a);

    char *memory = kalloc();
    pde_t *pagedir = myproc()->pgdir;

    memset(memory, 0, PGSIZE);

    if (mappages(pagedir, (char *)a, PGSIZE, V2P(memory), PTE_W | PTE_U) < 0)
    {
        cprintf("allocuvm out of memory\n");
        kfree(memory);
        break;
    }
    break;
}
```

- First we are storing the the fault address from the cr2 register using the rcr2() and then to the nearest page table entry using the PGROUNDDOWN()
- Next we allocate a physical page frame using the kalloc() for that fault page we got
- The page table entry for that will be created using the mappages() function, which maps the allocated physical page to the virtual address in the page table
- The PTE_W and PTE_U flags indicate that the mapping should be writable and user-accessible inside the mappages()
- If the mappages() function fails to allocate a page table entry then an error message is printed and the allocated physical page is freed using kfree().
- Now we will see what we did in exec.c
- Here what we did is to only allocate the page to the read-only code/data by changing the parameters of allocuvm() inside the forloop
- Here we also changed the size of sz again in the end of for loop to make sure we do not change the size of it to new rather we changed it to the old value inside loop.

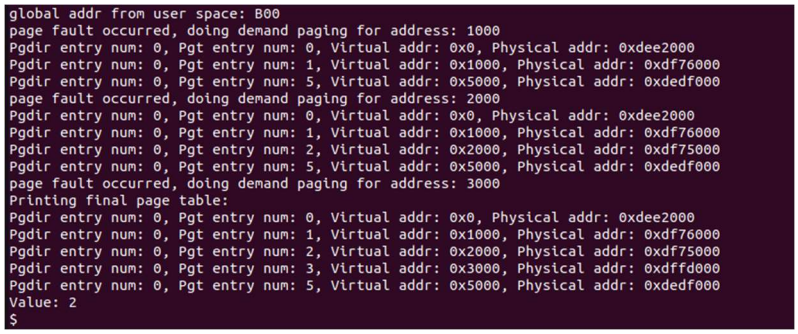
```
// Load program into memory.
sz = 0;
for (i = 0, off = elf.phoff; i < elf.phnum; i++, off += sizeof(ph))
{
    if (readi(ip, (char *)&ph, off, sizeof(ph)) != sizeof(ph))
    goto bad;
    if (ph.type != ELF_PROG_LOAD)
    continue;
    if (ph.memsz < ph.filesz)
    goto bad;
    if (ph.vaddr + ph.memsz < ph.vaddr)
    goto bad;
    if ((sz = allocuvm(pgdir, sz, ph.vaddr + ph.filesz)) == 0)
    goto bad;
    if (ph.vaddr % PGSIZE != 0)
    goto bad;
    if (loaduvm(pgdir, (char *)ph.vaddr, ip, ph.off, ph.filesz) < 0)
    goto bad;

    sz = sz - ph.filesz + ph.memsz;
}
```

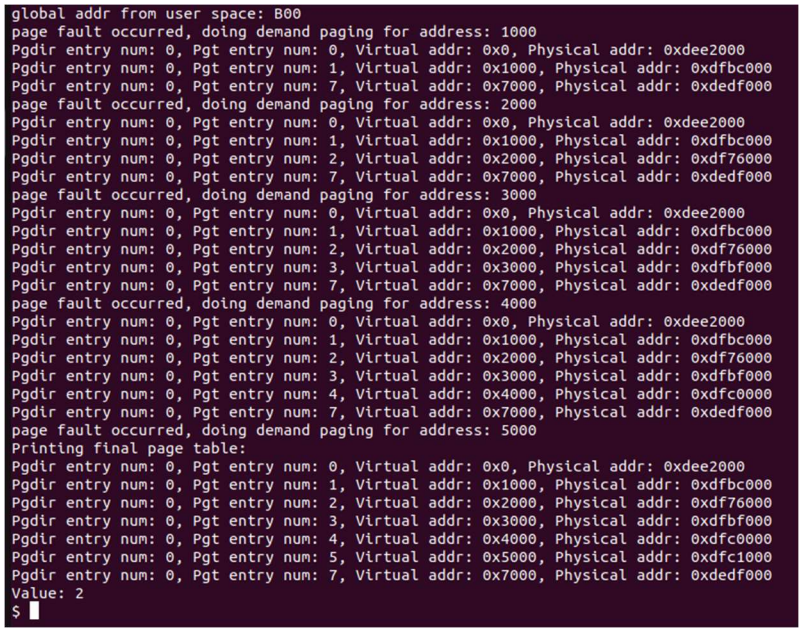
- Like that I implemented the demand paging in the low level inside the xv6 system

Observations:

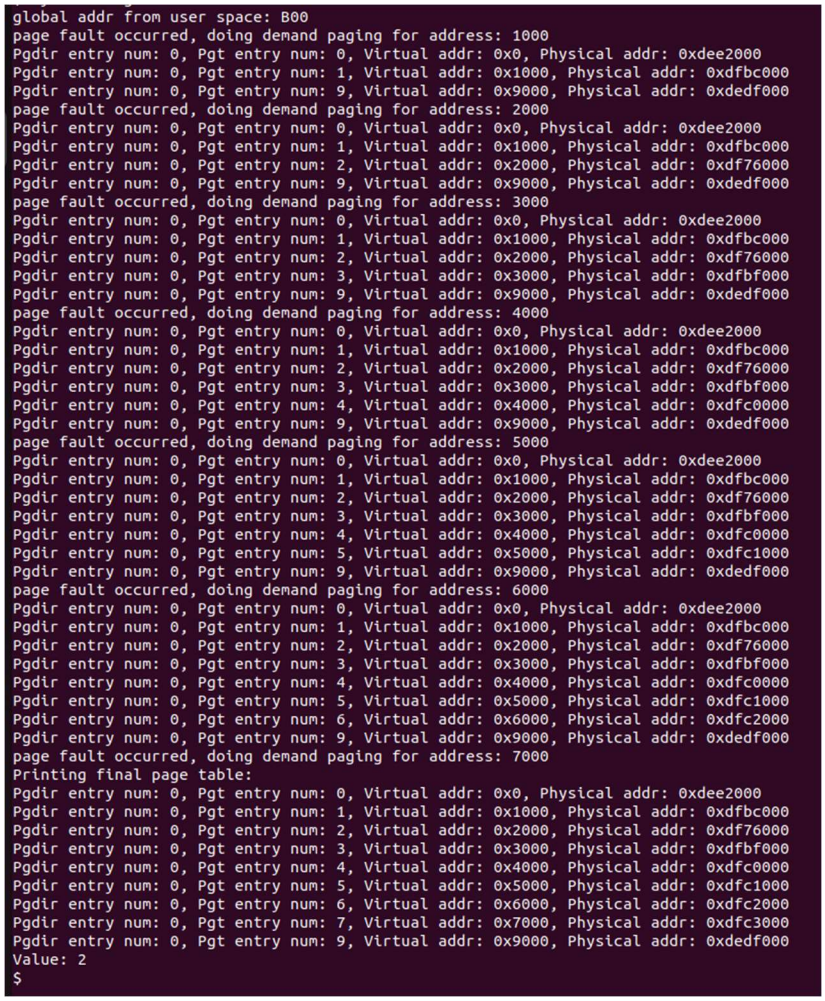
- What I observed is that there are already two pages which are loaded inside it and
- In this experiment we are increasing the size of global array and seeing the changes
- And after increasing the size of N we can clearly see that no of final page entries increases and the number of entries also increases.
- The total number of page faults increased.
- Clearly, we can see these observations from below figures



For N = 3000



For N = 5000



For N = 7000

What I learned:

- What I learned is to implement the demand paging in xv6,
- How the mappages() works,
- How to get the page fault address from rcr2(),
- How growproc() systemcall works,
- Implementing a program call like mydemandPage as we not did for a program call in 1st assignment in that assignment it's a system call