Operating System-2: CS3523
Programming Assignment 6: Paging

CS21BTECH11055 -SADINENI ABHINAY

April 20, 2023

1 Implementation of code:

1.1 Demand Paging

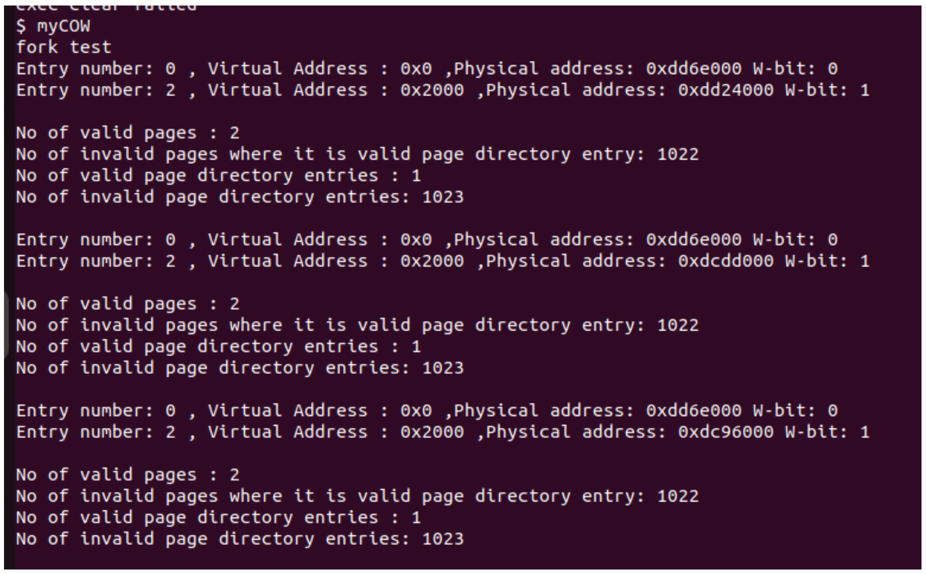
- **modify exec.c:** Originally exec.c is giving space for global variables without initialization of values now we modify to only allocate space for read only code segment ,etc .. (change the allocation value in sz=allocuvm(..))
- **Add a trap case for page fault:** Add T_PGFLT case in trap.c
- **invoke a page handler** in page fault trap case
- **fault handler** Create a function pgflthandler() in vm.c where we can easily accesses functions like walkpgdir which is used to get page table entry for a particular virtual address and mappages which virtual pages to physical frames
- **demand Pages:** for differentiating demand paging check for the present bit if not present means we have to allocate a page and map it this where we will use walkpgdir and mappages
- **create space**

1.2 Copy on write

- **fork:** In fork function it is using copyuvm(..) which creates a new pages for the child process and copyes the contents of parent processes.
- **modify copyuvm(..):**do not create new pages ,maps the child process entries to parents physical pages (just remove the new allocation code and change the parameter in mappages(..) to parent directory.) Do not forget to change the page entry to read only(change the write bit using bit wise operator (&))
- **change in kernel structure:** Add a new array called reference count for checking how many process have reference/accesed the pages

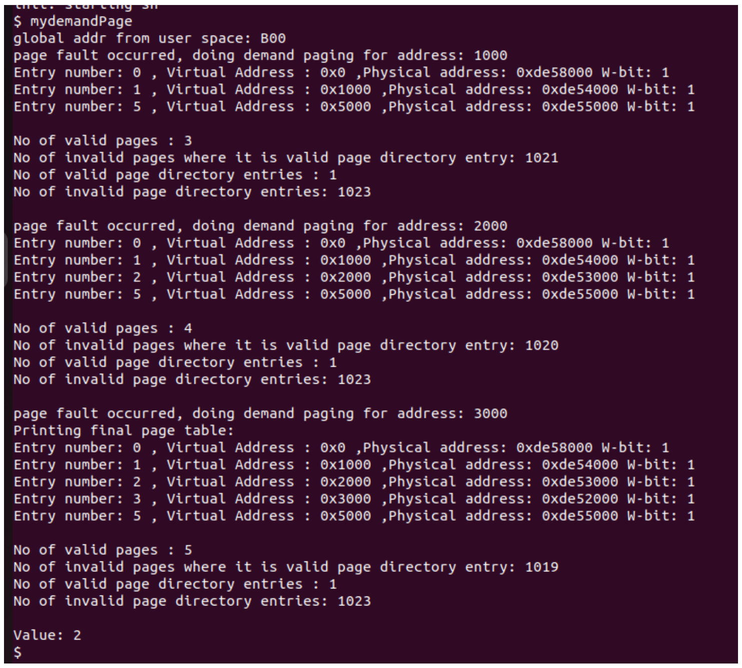
- **modify kalloc:** when kalloc is invoked it means new pages are created and we have to initialize their reference count to 1
- **modify k free:**when kfree is invoked we have decrease the reference count but not delete or free the frame as it might be used a child or parent of that process which invoke it if the reference count is zero then we can delete the frame
- **add new case in the fault handler:** In page fault error: if not demand paging then it must be copy on write , there are three cases here
  1. if the reference count is greater than one that means we have to create a new page and map with the free page because more than one process is using the page
  2. if the reference count is 1 then we make the page writable because means single process is writing .
  3. reference count is none of above cases then it means wrong address.

2 Output:



```
exec clear failed
$ myCOW
fork test
Entry number: 0 , Virtual Address : 0x0 ,Physical address: 0xdd6e000 W-bit: 0
Entry number: 2 , Virtual Address : 0x2000 ,Physical address: 0xdd24000 W-bit: 1

No of valid pages : 2
No of invalid pages where it is valid page directory entry: 1022
No of valid page directory entries : 1
No of invalid page directory entries: 1023

Entry number: 0 , Virtual Address : 0x0 ,Physical address: 0xdd6e000 W-bit: 0
Entry number: 2 , Virtual Address : 0x2000 ,Physical address: 0xdcdd000 W-bit: 1

No of valid pages : 2
No of invalid pages where it is valid page directory entry: 1022
No of valid page directory entries : 1
No of invalid page directory entries: 1023

Entry number: 0 , Virtual Address : 0x0 ,Physical address: 0xdd6e000 W-bit: 0
Entry number: 2 , Virtual Address : 0x2000 ,Physical address: 0xdc96000 W-bit: 1

No of valid pages : 2
No of invalid pages where it is valid page directory entry: 1022
No of valid page directory entries : 1
No of invalid page directory entries: 1023
```

Figure 1: myCOW: testing copy on write



```
$ mydemandPage
global addr from user space: B00
page fault occurred, doing demand paging for address: 1000
Entry number: 0 , Virtual Address : 0x0 ,Physical address: 0xde58000 W-bit: 1
Entry number: 1 , Virtual Address : 0x1000 ,Physical address: 0xde54000 W-bit: 1
Entry number: 5 , Virtual Address : 0x5000 ,Physical address: 0xde55000 W-bit: 1

No of valid pages : 3
No of invalid pages where it is valid page directory entry: 1021
No of valid page directory entries : 1
No of invalid page directory entries: 1023

page fault occurred, doing demand paging for address: 2000
Entry number: 0 , Virtual Address : 0x0 ,Physical address: 0xde58000 W-bit: 1
Entry number: 1 , Virtual Address : 0x1000 ,Physical address: 0xde54000 W-bit: 1
Entry number: 2 , Virtual Address : 0x2000 ,Physical address: 0xde53000 W-bit: 1
Entry number: 5 , Virtual Address : 0x5000 ,Physical address: 0xde55000 W-bit: 1

No of valid pages : 4
No of invalid pages where it is valid page directory entry: 1020
No of valid page directory entries : 1
No of invalid page directory entries: 1023

page fault occurred, doing demand paging for address: 3000
Printing final page table:
Entry number: 0 , Virtual Address : 0x0 ,Physical address: 0xde58000 W-bit: 1
Entry number: 1 , Virtual Address : 0x1000 ,Physical address: 0xde54000 W-bit: 1
Entry number: 2 , Virtual Address : 0x2000 ,Physical address: 0xde53000 W-bit: 1
Entry number: 3 , Virtual Address : 0x3000 ,Physical address: 0xde52000 W-bit: 1
Entry number: 5 , Virtual Address : 0x5000 ,Physical address: 0xde55000 W-bit: 1

No of valid pages : 5
No of invalid pages where it is valid page directory entry: 1019
No of valid page directory entries : 1
No of invalid page directory entries: 1023

Value: 2
$
```

Figure 2: mydemandPage: testing demand paging

3 observations:

3.1 mydemandPage:

1. for each 1000 iteration in mydemandPage the no of valid pages increased because as the loop is accessing the global variable the demand paging implemented is creating frames for them .
2. Write-bit is not changed.
3. if we change the size of global array still it is showing similar output that means demand paging is working when the page is not present in the memory
4. new pages are correctly mapped. As we can observe from the output

3.2 myCOW:

1. write-bit is changed because for different addresses because we are invoking fork which sets write bit to zero initially
2. when the child try to modify then the write bit is set 1 as it create new page for child

3. number of nested forks in the user program didn't seem to effect the Output as it because reference count takes of it
4. it best to use COW as it gives use flexibility on free frames.
5. it is nanoseconds overhead for creating new pages when modified by children or parent.
6. At start init will be created which acts a parent for all system process.

4 What i have learnt:

1. modifying kernel to implement various optimization like above two COW and demand paging
2. learnt about elf format and program headers
3. what is actual need of paging
4. also learned how to modify PCB structure and use it at will.