Operating Systems-2 CS3523

Programming Assignment-5: Syscall Implementation Report

- B. Asli Nitej Reddy

- CS21BTECH11011

a)

System call working:

- See what I understood is system calls is a way to interact with OS and access privileged (not user) instructions.
- So, they provide a connection between user and kernel.
- When a system call is made, kernel will switch to kernel mode from user mode.
- And also a number will be copied into eax register corresponding to the system call
- After a system call completed its execution the mode changes to user mode again
- First init.c will run and also set up various resources needed.
- Then it will go to user.h(Prototype for systemcall)
- Then usys.S(assembly code that sets arguments and calls syscall() in syscall.c)
- In syscall.c we will have syscall() which on invocation will go to appropriate kernel function
- Now we know the the kernel function will be in sysproc.c which will contain appropriate function.

- Now to print variable things we will use cprintf()

What I learned:

- how to implement a system call in xv6
- based on above I understood how control flow happen at aa basic idea when we invoke a system call
- use of already defined functions to create a new system call
- how the paging looks in the x86 thing like directory and in page (2 level paging)
- printing virtual and physical address

b)

Observations:

At start without adding any variables locally no of valid pages in my case is 2

1. Here in the 1st experiment is to add arrGlobal[10000] array globally and to check if the no of valid page entries increases or not based on the observation for the given sized array I got 12 this is because when we declare an array globally the space allocated to it will be in data section and it is limited to process virtual space compared to it's stack which is allocated at other place so no of valid pages is high here are the small observations based on it

   Size vs no of valid page entries

   10000 = 12
   20000 = 22
   30000 = 31

40000 = 41

But see total no of pages is same in the above as all are there in same directory it seems but for the size

10000000 = 9768

But in this total no of pages also increases which is saying is it accessing many directories compared to before

2. Same like above but in 2nd experiment we are declaring it in locally with array name as arrLocal[10000] here we can observe that even though size of array increased on even if not declared whatever the case no of valid page entries remains constant because when variables are declared locally we allocate them into stack but stack is allocated at a different place than in page table no we are not seeing the change in this case

   Size vs no of valid pages entries

   10000 = 2
   20000 = 2
   30000 = 2
   40000 = 2

3. Here in the 3rd experiment we are doing the same thing over and over and checking the changes in the no of valid entries, virtual address, physical address.

   What I observed is even though I am running multiple times no of entries, virtual entries are not changing but the physical address are changing this is because we are allocating the physical frames while we are executing the program so pages may be allocated to different frames

Extra:

For part 1 question I am not printing for system call named write because after each letter a new line is printing so reading the output makes hard and I think that will be hard