CS3523 OPERATING SYSTEMS :

REPORT:

IMPLEMENTATION :

PART-1:

- First of all we add a system call naming ‘pgtPrint’ to the existing system calls .Which was already done in Assignment -2 i.e. we modify the files usys.S , user.h , syscall.h , syscall.c , sysproc.c .
- The implementation of pgtPrint is in sysproc.c file . In that we print the entries who are valid & can be accessed in user mode .
- myproc->pgdir gives the pointer to the page directory & there are 1024 page directories (Macro NPDENTRIES) . Each of the page directory contains 1024 page tables (Macro NPTENTRIES). Page directory contains physical address of a pages & we can convert that to virtual by using P2V function & store that address in page .
- We check the every directory & table satisfying the conditions by computing the bit wise and with the bits PTE_U & PTE_P . We print the virtual address of an entry by using i, j i.e. page directory number , page table numnber .
- Then we call this system call pgtPrint from the user program mypgtPrint.c

PART-2:

- Here we do modifications in two files exec.c & trap.c .
- In exec.c file we change the memory allocation method by only allocating the read only code & allocating dynamic variables only when they are called .
- We do this by using the ELF program header (ph) .Intially sz is declared 0 when we are loading program into memory if we find any wrong conditions for the information stored by ph i.e. type , memsz , filesz we go to bad function which does the de - allocation .
- Here we allocate only read code by using function allocuvm & pgdir , sz , ph.vaddr + ph.filesz as arguments . Here ph.filesz indicates the read only code .
- We assign the integer returned by allocuvm to the variable sz . If sz==0 then we go to bad function for de-allocation .
- If sz is not equal to 0 . Then we re assign the value of sz to ph.vaddr+ph.memsz . Since we already allocated the read only code to the memory . But the sz should in should of ph.vaddr+ph.memsz . Since for the next iterations it should load from the total size of program i.e. ph.vaddr+ph.memsz .

- In trap.c we add a case for the trap error i.e. we add case T_PGFLT indicating page fault of a page . We then handle that page fault by assigning a free frame which can be obtained by kalloc( ). Then we assign the each value of that free fram to 0 by using mmset function . & Then we map the page that had given page fault to this free frame using mappages function . Here we pass rcr2( ) register as argument as it stores the virtual address of the page that caused page fault .
- Then we test this page fault handling by checking it on the user program mydemandPage.c

OBSERVATIONS & REASONING:

PART-1 :

- Intially before declaring the global array the number of entries that are valid & allowed to access in user mode were 2 .
- After declaring the global array ‘arrGlobal’ of size 10,000 the number of entries that are printed were 12 .
- As we are declaring a global array which consumes memory . So the number of pages in the page tables will be increased .Therfore there will be an increase in no of entries .
- After declaring a large size local array ‘arrLocal’ of size 10,000 , I am getting a error saying that “pid 3 mypgtPrint: trap 14 err 5 on cpu 0 eip 0x1f addr 0xbfd4--kill proc” . Which is due to unavailability of free memory in stack for that large size local array .
- So, by decreasing the size of local array to 10 or 100 We get again 12 entries of ouput on the terminal .
- Therfore the number of entries in the terminal before & after declaring the local array ‘arrLocal’ remains the same .
- After the repetition of execution of user program ‘mypgtPrint.c’ the number of entries remains the same i.e. ‘12’
- Also the physical address change for every execxution but the virual addresses remains the same for every execution.
- This is due to that for every execution , the physical address which was alloted before for a page may not be available now as it may be alloted to another process . So, the physical addresses vary for every execution .
- Where as the virtual addresses remains the same since virtual addresses of one process will be independent of another process & the virtual addresses will be alloted to each process by the CPU . After every execution of program the virtual allocation starts from ‘0’ and allocation goes in same manner for every execution . As those addresses won’t be accessed by another process . This says that virtual addresses will be same for every execution.

PART-2:

- Intially declaring the value of N i.e. size of global array as 300 in the user program 'mydemandPage.c' would give an output that contains 2 entries i.e. only 2 pages satisfy the conditions .
- After increasing the value of N to 3000 the no of entries when we call the system call pgtPrint are '3' and no of entries in the 2^{nd} time are '4' and no of entries we got when printing the final page table are '5' .These 5 pages will contain all the pages that appeared before on the ouput . Since we first print the entries for the first 1000 values in array& then the next 1000 values . So, the final entries will contain all the entries which appeared before . Also in this case we have got 3 page faults .
- When we increase the value of N to 5000 it had given 7 entries in final page table & 5 page faults are occured.When N is 10000 we get 12 entries & 10 page faults.
- Each page is of size 4kB i.e. 4* 1024 Bytes = 4096 Bytes . Each integer is of size 4 Bytes . So each page can have a maximum of 1024 integers in it . Intially no page will be in memory due to demand paging so page fault occurs after allocation of 1024 integers a page will be completely filled then a new page should be allocated which again gives a page fault . So, we get a page fault for every 1024 numbers of the array.

LEARNING FROM ASSIGNMENT :

- From this assignment I had learned how to access the pages that are valid & accessible from user mode . I have learned how to change physical address to logical address & vice versa i.e. usage of P2V & V2P functions
- I had also learned that repeating the execution doesn't alter the virtual addresses but alter the physical addresses.
- I had learned how to allocate a free frame & map a page which caused the page fault to that free frame . i.e. I have learned how to handle page faults .
- I have also learned how to change the memory allocation without loading the entire program into memory instead loading only some part.I have learned the difference between ph.memsz & ph.filesz
- I am not able to do this without knowing about ELF program headers . I have learned about the ELF program headers & it's contents

Submitted by

Kodavanti Rama Sravanth

CS20BTECH11027