OS LAB 06
REPORT

L NIKHIL KUMAR-140050037
UDAY KUSUPATI-140050048

Syscall getNumFreePages():

<u>Changes in code:</u>

syscall.h : added a new definitions “SYS_getNumFreePages”
syscall.c : added sys_getNumFreePagesinto the syscalls[] and declared the extern functions.
usys.S : added SYSCALL(getNumFreePages).
user.h : added int getNumFreePages(void).
defs.h: Added definition of getNumFreePages() function written in kalloc.c
    int getNumFreePages(void);
kalloc.c: implemented getNumFreePages() function which returns the count of total number of free pages available by scanning through the number of non NULL entries in the linked list kmem.freelist
sysproc.c : Implemented the function sys_getNumFreePages().
sys_getNumFreePages() just calls getNumFreePages() which is implemented in the kalloc.c .

Copy-on-Write:

<u>Changes in code:</u>

defs.h: Added definition of pageFaultHandler function written in kalloc.c
    void pageFaultHandler(void);
trap.c: In the switch case that handles traps, redirecting T_PGFLT to pageFaultHandler
vm.c:

Declarations of pageCount(char array) and countLock(spinlock) variables

inituvm:

On mapping of every new page to a frame, we increment the pageCount of the frame, with the lock held.

allocuvm:

Appropriately incremening the pageCount when a new page points to the frame, lock held appropriately

deallocuvm:

The pageCount is appropriately decreased when deallocated, and only when the pageCount==0, we free the frame, lock held appropriately

copyuvm:

Instead of allocating new frames for the child, we just map the child’s virtual addresses to the frames of the parent. And the pagetables of the child are copied from the parent’s for which we remove the PTE_W permissions before copying. After the mapping, the pageCounts is appropriately updated for all the pages that are doubly pointed by parent and child, with lock held appropriately. At the end we flush the TLB cache.

pageFaultHandler:

New function written to handle page fault. The faulting virtual address is obtained from the CR2 register through rcr2(). Various cases of error checking are handled like Fault at address inside kernel, faulting pte not in pgdir, non-reading permissions in pte and in case of no error we obtain the pte of the faulting address. Now we check if the pageCount of it is 1 or not. If it is 1, we give write permissions to that frame and TLB is flushed. Otherwise i.e if it is >1, we create the new frame using kalloc() and copy contents from old frame, decrement the pageCount and increment th pageCOunt of the new frame holding the lock. The new frame has all permissions. TLB cache is flushed.

TEST CASE:

In the parent, I am forking a child with a statement pid = fork();
There itself a pagefault occurs because we are writing into a variable.

Then in the parent, I am trying to read from the variable “a”, which now does not cause any page fault, and number of free pages does not decrease. So we are not allocating a new memory image while forking.

Now if I try to write into the variable “a” in the parent, it causes a pagefault here and copies the memory image and allots new pages for the page containing a(i.e the page which caused the page fault).

Then if I again try to write into “a”, it does not cause a pagefault because I have already created a new memory image for the page containing “a”. So in this step number of free pages does not decrease.