OS LAB

Navneet Agarwal - 140100090
Ritwick Chaudhry – 14D070063

1. Data Structures:

a. An array pgRefCount to store the reference count for each physical frame. Therefore the size of the array is PHYSTOP/PGSIZE.
b. Spinlock pgCountLock is used to lock the pgRefCount while updating it
c. Spinlock pgFaultLock is used so that only 1 pagefault is handled at one time due to the multi-core architecture
d. A numFreePages count is maintained in the kmem structure

2. Changes to Code:

a. ***proc.c***
   i. Replaced the call of copyuvm with the modified version cowuvm
b. ***sysproc.c***
   i. Added the syscall function for getNumFreePages which returns the number of free pages.
c. ***usys.S***
   i. **SYSCALL(getNumFreePages)** added to file to define syscall
d. ***user.h***
   i. Added the definition of getNumFreePages as int getNumFreePages(void)
e. ***syscall.h***
   i. Define the system call number for the new system call
f. ***syscall.c***
   i. Extern functions for the new system call is declared
   ii. The system call is added in the static array containing all the system calls.
g. ***mmu.h***
   i. A new MACRO (PGCOUNTADDR(a)) is defined which takes the physical address and returns the index of the frame for the pgRefCount array.
h. ***kalloc.c***
   i. The integer numFreePages is added to the kmem struct
   ii. A function getNumFreePages is added which returns the currently available number of free pages.
   iii. Also the number of free pages is incremented in the kfree function
   iv. The number of free pages is deducted in the kalloc function
i. ***defs.h***
   i. Added the definition for retNumFreePages which is called by getNumFreePages

ii. Added the definition for cowuvm which is the modified version of copyuvm
iii. Added the definition for managePageFault which is called on a page fault

j. ***trap.c***
   i. The entry for the page fault trap is added in the switch case which calls the function managePageFault()

k. ***vm.c***
   i. Both the spinlocks (pgCountLock and pgFaultLock) are defined and initiated in the kvmalloc() function
   ii. The pgRefCount for the frames are updated when the mappages is called. This is done in the inituvm, allocuvm, cowuvm and copyuvm functions.
   iii. The cowuvm function is defined which performs the following functions:
      1. It maps the pages of the child process with the physical frames of the parent with the flags set as READ ONLY (rather than making a copy for each physical frame of the parent (like in copyuvm)).
      2. Also it increments the pgRefCount for each frame of the parent's physical memory
      3. If the entire mapping is successful, it changes the permissions of all the frames of the parent as READ ONLY
      4. Then lcr3(v2p(pgdir)) is called to flush the TLB cache entries
   iv. managePageFault() function which manages page traps.

3. Functioning

a. We start with maintaining the reference count to each physical frame(for user pages). And update it whenever new pages are mapped or the ptes are deleted.
b. Also a system call is made to get the number of free pages that are currently available in the freelist pages list
c. Fork calls cowuvm in place of copyuvm.
d. The function cowuvm in place of allocating a new memory image to the child, makes a new page directory pointing to the same physical frames as that of the parent. Also the permissions for both the child and the parent are set to READ ONLY. Also the TLB cache entries are flushed
e. Now whenever the parent or child tries writing to the memory image (which is now READ ONLY) a page fault is generated
f. A function to manage page faults is made. This function checks if the trap is generated by an actual invalid memory access or due to a fork.
g. If it's not due to a fork, a message is printed and the process is killed.
h. Else, if the reference count of the frame corresponding to the virtual trap address was greater than 1, a copy of the entire memory images is made and a new pgdir is made. The page directory is switched and then the old page directory is freed.

i. If the reference count was 1, this means that it is the last process pointing to that memory image. Hence just the page permissions are changed back to WRITABLE

4. Testcases

Testcase testcow has been given . It simply first prints out the number of free pages in the parent then, forks a child and then prints out the number of free pages in the child. After the child is reaped the parent again prints out the number of free pages which eventually comes out to be the same as the initial value.

5. Instructions for running:

i. Extract 14D070063_140100090.tar.gz into the xv6 folder.
ii. Copy the contents of the untarred folder generated (namely PatchFiles/ directory and script.sh) into the xv6 source code folder
iii. Run “bash script.sh” which will modify the source code to our code
iv. Run “make”
v. Run “make qemu”
vi. To execute TestCase, execute “testcow”