Enabling Copy on Write in XV6

Xv6 do not use copy on write functionality during fork () system call. That is, it copies all the code, data, and stack to the child processes address space when there is a fork. But this can be done in xv6 by making both child and parents virtual addresses to point to same physical memory initially and a new page is allocated only if there is write from any of the processes to this shared page. This is what essentially COW feature is. Now, in this we enable copy on write feature in xv6.

Step 1: -

In the kernel/vm.c/uvmcopy() function, there is allocation of newpages for the new process and the data of the old process is copied into these newpages and these newpages are mapped into the pagetable of new process.

I have modified this uvmcopy, so that new pages are not allocated to the new process and old pages are directly mapped to new process pagetable.

```
int
uvmcopy(pagetable_t old, pagetable_t new, uint64 sz)
{
  pte_t *pte;
  uint64 pa, i;
  uint flags;
 // char *mem;

  for(i = 0; i < sz; i += PGSIZE){
    if((pte = walk(old, i, 0)) == 0)
      panic("uvmcopy: pte should exist");
    if((*pte & PTE_V) == 0)
      panic("uvmcopy: page not present");
    pa = PTE2PA(*pte);
     *pte &=~PTE_W;
      *pte |= PTE_C;
      flags = PTE_FLAGS(*pte);
  //   if((mem = kalloc()) == 0)
    //   goto err;
   // memmove(mem, (char*)pa, PGSIZE);
    int pagenumber = (uint64)pa/PGSIZE;
     refcountpg[pagenumber]+=1;
     if(mappages(new, i, PGSIZE, (uint64)pa, flags) != 0){
      // kfree(mem);
       goto err;
     }
  }
  return 0;

 err:
  uvmunmap(new, 0, i / PGSIZE, 1);
  return -1;
}
```

We must make this shared page not writable. This is implemented using ***pte &= ~PTE_W** and, we have to mark this shared page as COW page. For this I have added another flag in riscv.h and this is named as PTE_C. So, this PTE_C bit is also set here. This is implemented using ***pte |=PTE_C.**

Step 2: -

We have to modify the kernel/trap.c/usertrap() function to handle the write pagefaults that are due to this COW. When there is a write pagefault , I have checked whether the page at the faulting address is COW or not. If it is cow page , I have allocated a new page and copied the data to this new page from old page and made this new page writable and made this new page non COW page. This is done as shown below.

```
else if(r_scause() == 15||r_scause()==13){
   uint64 va = r_stval();
   va = PGROUNDDOWN(va);
   pagetable_t pagetable = p->pagetable;
   pte_t * pte = walk(pagetable,va,0);
   if((*pte & PTE_C)==0){
     printf("usertrap(): unexpected scause %p pid=%d\n", r_scause(), p->pid);
     printf("            sepc=%p stval=%p\n", r_sepc(), r_stval());
     p->killed = 1;
   }
   else{
       uint64 pa = PTE2PA(*pte);
       char *mem;
       mem=kalloc();
       memmove((void*)mem,(void*)pa,PGSIZE);
       uint flags = PTE_U|PTE_V|PTE_W|PTE_X|PTE_R;
       if(refcountpg[pa/PGSIZE]==1){
            uvmunmap(p->pagetable,va,1,1);
            mappages(p->pagetable,va,1,(uint64)mem,flags);
       }
       else{
           uvmunmap(p->pagetable,va,1,0);
           refcountpg[pa/PGSIZE]--;
           mappages(p->pagetable,va,1,(uint64)mem,flags);
       }
   }

}
```

If the write interrupt is not due to COW page , then usertrap is printed. Otherwise new allocation of page is done and old page is first unmapped and this new page is mapped to faulting address.

Step 3: -

The old physical page is freed when the last PTE reference to it goes away. This is implemented by maintaining reference count array in kalloc.c. I have declared an array named **refcountpg[PHYSTOP>>PGSHIFT].** I incremented the refcountpg of a physical page when it is allocated in kalloc() and also in uvmcopy when the new process is mapped to the same page. This reference count is decremented in kfree() and the page is freed only the refcountpg of that page reaches 0. Using this refcountpg array , I have made decision to free a page when its last reference goes away.

Step 4: -

Finally, I used same scheme that is used in usertrap in kernel/vm.c/copyout() to handle page faults due to COW pages.

```
// Return 0 on success, -1 on error.
int
copyout(pagetable_t pagetable, uint64 dstva, char *src, uint64 len)
{
  uint64 n, va0, pa0;

  while(len > 0){
    va0 = PGROUNDDOWN(dstva);
    pa0 = walkaddr(pagetable, va0);
     pte_t *pte = walk(pagetable,va0,0);
    if((*pte & PTE_C)!=0){
        // uint64 pa = PTE2PA(*pte);
         char *mem;
         mem=kalloc();
         memmove(mem,(char*)pa0,PGSIZE);
          uint flags = PTE_U|PTE_V|PTE_W|PTE_X|PTE_R;
         if(refcountpg[pa0/PGSIZE]==1){
              uvmunmap(pagetable,va0,1,1);
              mappages(pagetable,va0,1,(uint64)mem,flags);
         }
         else{
             uvmunmap(pagetable,va0,1,0);
             refcountpg[pa0/PGSIZE]--;
             mappages(pagetable,va0,1,(uint64)mem,flags);
         }

       // uvmunmap(pagetable,va0,1,1);
       // mappages(pagetable,va0,1,(uint64)mem,flags);
     }

    if(pa0 == 0)
      return -1;
    n = PGSIZE - (dstva - va0);
    if(n > len)
      n = len;
    memmove((void *)(pa0 + (dstva - va0)), src, n);

    len -= n;
    src += n;
    dstva = va0 + PGSIZE;
  }
  return 0;
```