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

For doing this I followed the below steps.

1. In the uvmcopy() function which is in vm.c in the kernel, there is allocation of new pages for new process and the data of the old process is copied into these new pages and these new pages will be mapped into the page table of new process.

    - I have modified this uvmcopy function, so that the new pages are not allocated to the new process and old pages are directly mapped to new process pagetable.
    - We should make this shared page as not writable. This is implemented using ***pte &= PTE_W** and, we need to mark this shared page as COW page. to do this I have added another flag in riscv.h and I named it as PTE_C. So, this PTE_C bit will also be set here. This will be implemented by making ***pte | = PTE_C**.

2. We need to modify the usertrap() function in trap.c to handle the write pagefaults that are due to this Copy On Write. Whenever 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 had allocated a new page and copied the data from old page to this new page and made this new page writable and non COW page.

    - If the write interrupt is not due to Copy On Write page , then user trap will be printed. Otherwise new allocation of page is done and the old page is first unmapped and this new page is mapped to faulting address.

3. The old physical page will be freed when the last Page Table Entry references to it goes away. This can be implemented by maintaining the reference count array in kalloc.c.

    - I have declared an array named refcountpg[PHYSTOP>>PGSHIFT], 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.
    - Reference count will be decremented in kfree() and the page is freed only the refcountpg of that page reaches zero. Using this refcountpg array , I made the decision to free a page when its last reference goes away.

4. In the final step, I have used same scheme that is used in usertrap in copyout() in vm.c to handle page faults due to COW pages.