COL633: Project Report

Copy-On-Write in XV6

Aakash Chaudhary (2023MCS2483)

Bhuvnesh Kumar (2023MCS2011)

Ashish Modi (2023MCS2012)

Copy-On-Write in XV6

Problem Statement:

During the fork system call, xv6 creates a new page directory and copies the memory contents of the parent process into the child process. If the process *P1* allocates an array of 1MB and subsequently invokes the fork system call, xv6 assigns an additional 1MB of memory to the child process *P2* while copying the contents of *P1* into *P2*. If neither *P1* nor *P2* modify the 1MB array, then we unnecessarily created a copy. In order to improve memory utilization, we would like to extend xv6 with the copy-on-write (COW) mechanism.

Methodology:

1. **Enable Memory Page Sharing:**

   When copyuvm gets called, instead of creating separate copy of pages, it maps the Page table of child process to the pages of parent process' and mark them Read-only.
   A "rmap" is maintained to keep track of pages->process mapping, which stores the reference count of pages and 64-bit map for "pid" in which each bit represent the process id of the process referring the corresponding page.

   After mapping same page with parent and child processes, rmap reference count is increased, and corresponding pid bits are set for the page.
   Further, we used lcr3() for loading the page directory address in TLB.

Some functions are used to update/access the "rmap":

getRmapRef(): return the reference count of given page.
setRmapRef(): set the reference count of given page with given value.
incRmapRef(): increase the reference count of the input page.
decRmapRef():decrease the reference count of the input page.
setRmapPagePid(): set the pid index of process which refers the input page.
unsetRmapPagePid(): un-set the pid index of process which refers the input page.
getRmapPagePid(): return pids of referring processes corresponding to the input page.
setAllRmapPagePid(): set/unset all the pid index of processes with given 64-bit value.

2. **Handle Writes on Shared Pages:**

A page fault will occur, when a process tries to write to a page which has read-only permission and present in physical memory, then cowalloc() will be called which checks if the page is shared by 1 or more processes, if it is shared by multiple processes then a new pages is allocated in physical memory with same content and mapped to the page table of the current process. “rmap” is updated accordingly and TLB is cached with pagetable’s address.
If page is mapped with only process then, the page permissions are changed to “writable” and TLB is cached.

Another case of page fault is, when the page is not present in the physical memory, it means that page is present in swap area in disk. To handle this, new memory space is allocated in physical memory and page is swapped-in from swap area to this memory space. Also the rss value is increased by page size.

3. **Swapping-out the pages:**

When physical memory gets full, then some pages needs to be swapped-out to swap area. To select page which needs to be swapped out, we selected victim process which have highest “rss” value and lowest “pid” as tie breaker, then come to finding the victim page of this process, we pick the page which is not accessed, is user page, and present in physical memory. If we failed to find such page then 10% of total pages present in memory will by made non-accessed. Then again same procedure will be followed to find the victim page.

Now, we have the victim page which needs to be swapped-out, page contents are moved to swap area with all its permissions and reference information, and marked as not present in the memory.

As we moved page to swap area, we also have to update the page table entry of all the processes which map to the victim page with swap block index, so that in future it can be easily swapped-in.

Some other updates:

- When kfree() is called, reference count of page is decremented in “rmap”, if it becomes 0, it means there is no process referring to the page, so we unset all the pid map bits in “rmap” then freed it.
- When exec(), allocuvm(), deallocuvm() and growproc() gets called rss value of process is getting updated with available number of pages available(in bytes) in physical memory.
- When process exit, all the swap slots are marked as free in corresponding swap slot index.