Project OS

Satyam Kumar,Nayan Punia,Ajay Kumar
2021CS50605,2021CS50127,2021CS11211

April 24, 2024

1 Enable Memory Page Sharing

For this part, we have created a struct called rev_map which basically keep count of number of processes which are pointing to a physical page. In the copyuvm function when a child process is generating from parent process, we mark all physical pages of parent process as not writable and increases the reference count of copied page in the above declared struct.

2 Handle Writes on Shared Pages and Optimization

Now when any process want to write in shared page, we would get a T_PGFLT trap signal. In the page fault handler function, we get virtual address of that page using rcr2 registers. Then we check in rev_map entry corresponding to the above physical page and got the page reference count. If the page reference count is equal to 1, we directly set the write permission in page table entry. In the else case we get a free page with write permission using kalloc and copied content from shared page to new page, update the page table entry of the process trying to write shared page by pointing to the new page. And we also decrement the reference count of the shared page and setting the reference count of 1 to the newly allocated page. In the deallocuvm function, we are reducing the page reference count corresponding to the freed page. While in the allocuvm, we are increasing the page reference count of the allocated page.

3 COW with Swapping

Now we update rev_map struct by also storing the processes that are pointing to any page along with reference count. We have keep array of page table entries of size NPROC that are pointing to a physical page. We have also put a array to know which index is free to store the page table entry. Then we enable swap space. In swap blocks we have array of page_permissions of each page table entry pointing to a physical page, array of pte_t*, and array called fill to know

which entry are empty or not. Now when kalloc is not able to provide a free page, it called swap_space function, which provides a free page to it by swapping a page to disk. Swapping policy is similar to lab4. Now we have a victim page and it might be the case that this page is referenced by many processes. So we get put the victim page content on the disk and store the page permission flags in sblocks struct. We have also updated the page table entry of such process by storing in them slot id and enabling swapped flags. Then we make page reference count of this page zero and also decreases the rss of processes pointing to the victim page.

Now when any process try to access swapped page, we would get a page fault. So we update handle page fault function by also handling the swapped page case. In the swapped page case, we allocate a new page and restore permission and update all the PTEs. Then we update the rev_map by increasing the reference count and storing the page table entries. We also free the swap slots and increases the rss.