Lab 6 - Copy On Write

Motivation

When a process forks a child, all the code, data and stack variables are copied to the child processes address space and the child continues its execution. However, most of the pages remain unaltered (For example, the pages belonging to the code section). Hence, it is better to keep a single copy of pages in physical memory and make both parent and child processes' virtual addresses point to the same memory. A new copy of the page is deferred until there is a write to the shared page.

Idea

- During fork(), just copy the page tables entries from parent to child.
    - the entries are made to point to the same physical pages and marked as **non-writeable** in both parent and child pagetables.
- When there is a write by any of the involved processes to the shared page, it causes a pagefault. In the pagefault handler, a new physical page is allocated to the faulting process and the contents are copied from the original page. The new page is set as writable.
    - To identify if the page fault is due to COW, it may be useful to have a way to record, for each PTE, whether it is a COW mapping. You can use the RSW (reserved for software) bits in the RISC-V PTE for this. Let's call this PTE_C

Implementation Details

1. Modify uvmcopy() (invoked from fork in proc.c) to map the parent's physical pages into the child. Instead of allocating new pages, clear PTE_W in the PTEs of both child and parent and set the PTE_C bit appropriately. (10 marks)
2. Modify usertrap() to recognize page faults. When a page-fault occurs on a COW page, allocate a new page with kalloc(), copy the old page to the new page, and install the new page in the PTE with PTE_W set. Set the PTE_C bit appropriately. Requires that the old mapping for the page is unmapped. You can achieve this with uvmunmap.
(20 marks)
3. Next, ensure that each physical page is freed when the last PTE reference to it goes away, perhaps by implementing reference counts in kalloc.c. (10 marks)
4. Finally, modify copyout() to use the same scheme as page faults when it encounters a COW page. (10 marks)

Things to be taken care of

1. Multiple processes can use the same physical page. Hence, before freeing a page, it needs to be ensured that no other process references the same physical page.
2. When there is a single owner of a COW page, then its cow and non-writeable flags can be reset and it need not obtain a copy of the page.

   Consider three processes P1, P2, P3 and P1 > P2 > P3, where Pi > Pj denotes Pi is parent of Pj.

   Say P3 performed write on a cow page. It gets a new copy.
   Then P1 performs write on the cow page. It gets a new copy.
   Now P2 being the single owner, need not make a copy of the page upon write.

Submission details:

1. Implement your solutions in a fresh download of the xv6 source code. Write your answers in a Latex/Word document, convert it to PDF and name it as YOUR ROLL NO.pdf. This will serve as a report for the assignment.

2. Put your entire solution xv6 folder, and the YOUR ROLL NO.pdf in a common folder named YOUR ROLL NO LAB6.

3. Compress the folder YOUR ROLL NO LAB5 into YOUR ROLL NO LAB5.tar.gz and sub-mit the compressed folder.

4. NOTE: Make sure to run make clean, delete any additional manual and the .git

folder from the xv6 folder before submitting.