Lab1.1: Copy-on-Write fork - Design Document

Lab 1: Copy-on-Write Fork for xv6

> Your task is to implement copy-on-write fork in the xv6 kernel. You are done if your modified kernel executes both the cowtest and usertests programs successfully.

To start, clone the private lab assignment repo from GitHub classroom. Refer to the **setup tutorial page** for help.

```
$ git fetch
$ git checkout cow
```

The problem

The fork() system call in xv6 copies all of the parent process's user-space memory into the child. If the parent is large, copying can take a long time. In addition, the copies often waste memory; in many cases neither the parent nor the child modifies a page, so that in principle they could share the same physical memory. The inefficiency is particularly clear if the child calls exec(), since exec() will throw away the copied pages, probably without using most of them. On the other hand, if both parent and child use a page, and one or both writes it, a copy is truly needed.

The solution

The goal of copy-on-write (COW) fork() is to defer allocating and copying physical memory pages for the child until the copies are actually needed, if ever.

COW fork() creates just a pagetable for the child, with PTEs for user memory pointing to the parent's physical pages. COW fork() marks all the user PTEs in both parent and child as not writable. When either process tries to write one of these COW pages, the CPU will force a page fault. The kernel page-fault handler detects this case, allocates a page of physical memory for the faulting process, copies the original page into the new page, and modifies the relevant PTE in the faulting process to refer to the new page, this time with the PTE marked writeable. When the page fault handler returns, the user process will be able to write its copy of the page.

COW fork() makes freeing of the physical pages that implement user memory a little trickier. A given physical page may be referred to by multiple processes' page tables, and should be freed only when the last reference disappears.

The cowtest program

To help you test your implementation, we've provided an xv6 program called cowtest (source in user/cowtest.c). cowtest runs various tests, but even the first will fail on unmodified xv6. Thus, initially, you will see:

```
$ cowtest
simple: fork() failed
$
```

Submission1: Design Document

In the first submission, you need to submit a design document, which maps the solution mentioned above to modifications needed at code level. The design document should content two essential sections:

1. Look at the source code of cowtest, look at the syscalls that are involved. Then look at the source code of involved syscalls and their dependencies. Finally, summarize (1) what kernel data structures and APIs are involved, and (2) what sequence of events or API calls are executed when fork() is invoked.
2. Based on your understanding in part1 above, come up with a plan to realize the solution mentioned above, including but not limited to (1) what data structure(s) would be added/updated, and (2) what would be the workflow.