Cow

Design

> 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.

What is cow fork()?

> 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.

Implementation

1. maintain a counter for each 4kb physical page:

   To manage the lifecycle of the physical memory in cow mode, we need to know how many pagetable mapping to a physical page:

   > if no pagetable mapping, this page is free
   >
   > if only 1 pagetalbe mapping to this page, this page can be write and read by this process
   >
   > if there are more than 1 page table mapping to a physical page, is page is only readable

   ```
   //maintain the mapping counter for each page
   struct {
     struct spinlock lock ;
     int  pageMapNum[PHYSTOP>>12];
   } pgMap;
   void            addMap(uint64 pa);
   void            subMap(uint64 pa);
   int             getMap(uint64 pa);
   ```

2. modify the memory copy related functions

   in `uvmcopy()` when a process what to copy other's page table, just add a new mapping to the old physical page , and add up map counter and set the new only readable and add a cow flag

```

      // disable write and add cow flag
        flags = flags | PTE_COW;
        flags = flags  & (~PTE_W);

      *pte = PA2PTE(pa) | flags;

      // do not alloc new memory,
      addMap((uint64)pa);
      if(mappages(new, i, PGSIZE, pa, flags) != 0){
        goto err;
      }
```

3. handle write panic

   As we set the cow memory read-only, so when a process wanna write some bits to a that new page, it will active the trap handler. In this place, I will mapping a new page for this page and set it writable

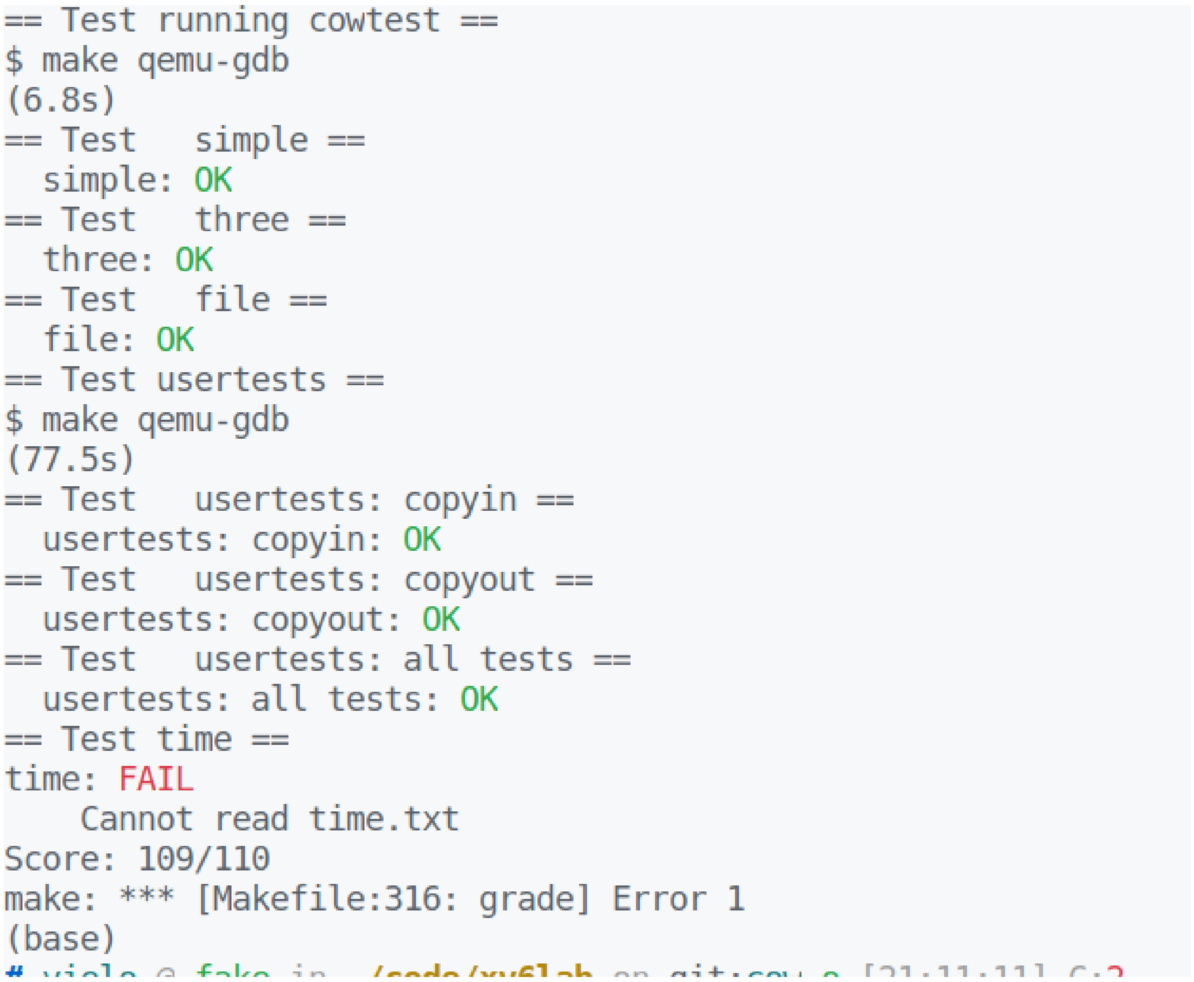
```
// handler for cow, mapp va to a new page
int handle_cow(pagetable_t pt,uint64 va){
  pte_t *pte = walk(pt,va,0);
   if (*pte & PTE_COW){

     uint64 flags = PTE_FLAGS(*pte);
     uint64 pa = PTE2PA(*pte);
     char *mem;
     if ((mem=kalloc())==0){
       //printf("no memory for kalloc");
       return -1;
     }
     memmove(mem,(char*)pa,PGSIZE);
     flags = (flags | PTE_W) & (~PTE_COW);
     kfree((void*)pa);
     *pte = PA2PTE((uint64)mem) | flags;
   }
   return 0;

```

Results

pass all test



```
== Test running cowtest ==
$ make qemu-gdb
(6.8s)
== Test   simple ==
  simple: OK
== Test   three ==
  three: OK
== Test   file ==
  file: OK
== Test usertests ==
$ make qemu-gdb
(77.5s)
== Test   usertests: copyin ==
  usertests: copyin: OK
== Test   usertests: copyout ==
  usertests: copyout: OK
== Test   usertests: all tests ==
  usertests: all tests: OK
== Test time ==
time: FAIL
    Cannot read time.txt
Score: 109/110
make: *** [Makefile:316: grade] Error 1
(base)
# viola @ fake in ~/code/xv6lab on git:cow o [21:11:11] C:2
```