The following files have been modified/edited:

User.h:
- Added procdump system call function prototype declarations

Syscall.h:
- Added lines to connect procdump to its respective ordering numbers

Syscall.c:
- Added procdump function to system call declarations and to array of pointers

Usys.S:
- Added SYSCALL(procdump) so that it can be recognized as system calls

Sysproc.c:
- Added and implemented the procdump system call function utilizing my implemented function that was added within proc.c

Defs.h:
- Added declarations for our cow() and procdump() functions that were implemented in proc.c and vm.c

README.md:
- Wrote in email and description of how to compile and run the three test files.

Makefile:
- Added the the testcow.c file to the compile list in UPROGS and EXTRA

Trap.c:
- Added in a case within the switch statements that would catch page faults. It first checks if it is in kernel mode, and if so calls the pagefault() function.

Mmu.h:
- Inserted another flag that will keep track if process memory is shared or not.

Proc.c:
- Changed the copyuvm() to a cow() call within fork()
- Added onto procdump() to correctly output the right information when testcow is called by looping through the page table.

Testcow.c:
- This file shows the specific data that pertains to page tables before and after processes get forked, therefore creating child processes.

Vm.c:
- At the beginning of the file, I create a structure for our counter that hold both the array of type uint, but also a spin lock as to make sure that when we increment an index in said array, it is an atomic operation.
- In inituvm(), I initialize the counter to 0 at the end of the method
  - The same is done within allocuvm() as a means to zero out the counter for each virtual page

- In deallocuvm(), I decrement the counter at the specific index as a result of deallocation. Once this is done, we check if the counter is zero, and if so, update the restive flags.
- I implemented cow(), a function that utilizes the copy-on-write technique. As opposed to the copyuvm() function, cow() maps the child to the parent's page, updates the flags of the page, and increments the counter.
- Finally, I implemented the pagefault() method. This specific function allocates a new space of memory that the process looks at. After receiving the address, the function checks that the table is shared and present. Once it does that, it gets the physical address and checks if the table is shared (If not, it updates the shared and writable flags). If it is shared, that's where new memory is allocated and the data is copied. The flags are then set and the counter is decreased.