2. Null pointer dereference

In xv6, the user program text are loaded into the very first part of address space with virtual address 0. Thus when dereference a null pointer which is point to the 0 virtual address, we will get a return value which ia actually the program text. To fix this problem, what I have done is to load program code in "next" virtual page and copy the text to the "next" page. Finally we need to set the program entry at the "next" page.
I have modified the following place:

* exec.c: load an empty page at address 0 and leave it there, copy our code and data begin at the second page.
* proc.c: copyuvm() should copy begin at the second page.
* syscall.c: argptr() the pointer address should not be 0.
* Makefile: change the user program' entry to 0x1000 not 0.

3. Share memory

Share memory is used for different processes to communicate with each other. Each process can read or write in the share memory. The value in the share memory can be reaches by all process.
The

I define and implement a function `shmeminit()` which is used to allocate 4 page memory for sharing. Define a gloable array to save these four memory pages' virtual address and the number process using this share memory.Then call this `shmeminit()` during booting in `main.c`. Then define and implement function `shmem_access()` and `shmem_count` function. Then modify the `fork()` for copy share memory address from parent process to son process. Then modify `freevm()` to keep share memory not be free. Modify the `allocuvm()` to check boundary not get into the share memory address.

The details I change is showing below:

* vm.c: implement functions `shmeminit()` , `shmem_access()`, `shmem_count()`. Modify `freevm()` not to free share memory page.
* proc.c: Modify `fork()` to copy the share memory address. Modify `wait()` function to count the process number of using share memory. Modify `allocuvm()` to check the boundary not get into the share memory address.
* syscall.c, syscall.h, defs.h, user.h, sysproc.c: add system call sys_shmem_access and sys_shmem_count
* main.c: add the `shmeminit()` function in the `main()` to initial the share memory.

The implementation of `shmem_access(int index)` function

* if the share memory address of this index have already in the process page table, then return it directly.
* Start from top of virtual address in user space(0x80000000 which is KERNBASE)
* The first virtual address of share memory should be starting at KERNBASE - PGSIZE.
* Walk through the page table and map virtual addresses to their physical addresses.
* Physical can be obtain form our global array which is used to track the share memory virtual address.

The implementation of `shmem_count(int index)` function

* Initialize the count of process using share memory as 0 in `shmeminit()`
* When map this share memory into the process page table. The count should add 1.
* When fork() is being called, the share memory used by father process should also used by son process, so each should add 1.
* When wait() is being called, the son process has finished, so the share memory is not being used by this process, it should minus 1.

Test case
I write a test case using twice fork(). The process 1 read the value in share memory. The process 2 change the value in share memory, the process 3 set the value in share memory. Then using `wait()` to let process runing order as 3->2->1.
So the result of process 3 should be the value it set. The result of process 2 should be the value it changed. The result of process 1 should be the same one of process 2.
Call the `shmem_count()` in each process, and because of the `wait()`, all of the number should be 1.

To run the share memory test case, use commond: `testmemory`