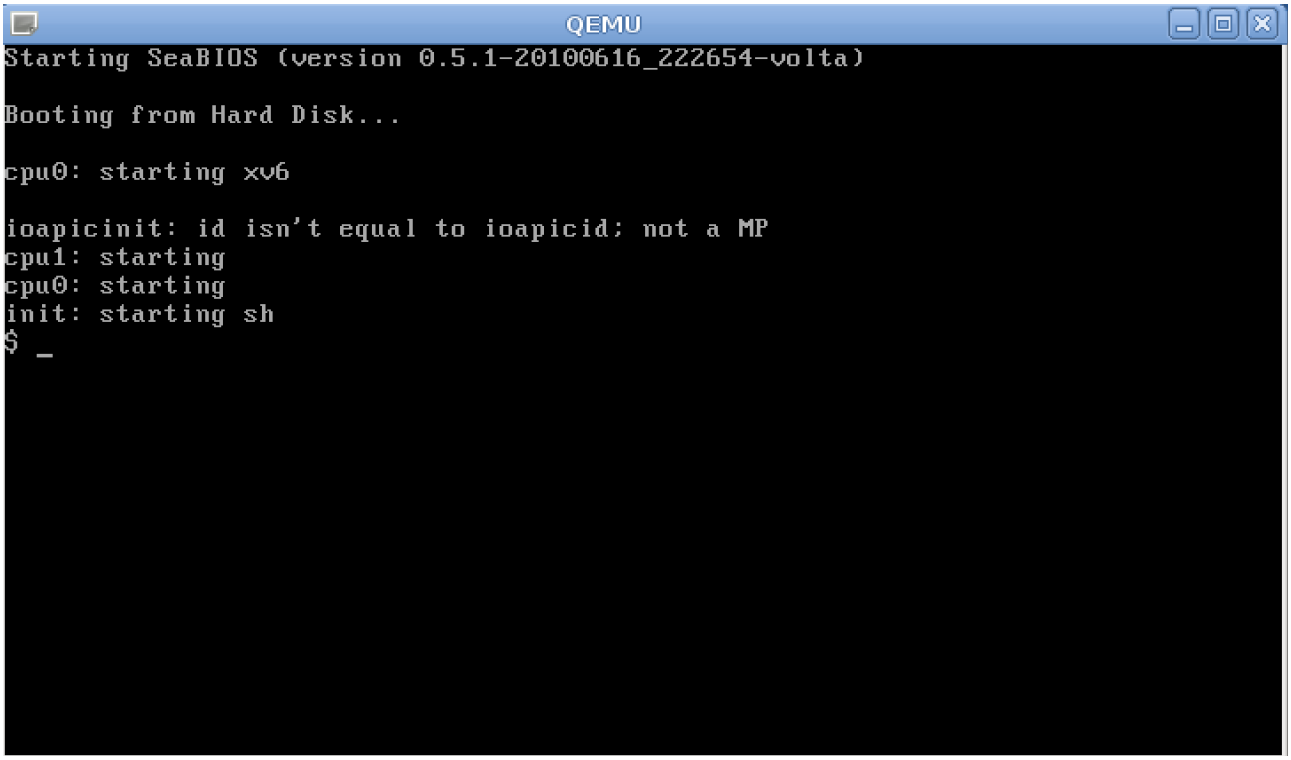
LAB 5.1
First of all i had to go to the directory “xv6” and compute the “make clean” command .
Once i do, i compiled the kernel with “make qemu” which compile and execute the xv6 kernel.
When the kernel runs, it show this window:



Through this command line it is possible to interact with the kernel by typing some system call implemented for the kernel.
After that i had to check if the script qemu.sh exists and in this case copying this string:

```
qemu -serial mon:stdio -hdb fs.img xv6.img -smp 2 -m 512  -S -gdb
tcp::26000
```

At this point has been possible to execute qemu by using this arguments passed through command line

```
qemu -serial mon:stdio -hdb fs.img xv6.img -smp 2 -m 512
```

Now by using cat command i created a file called “text.txt” in which there are the following string.

System and Device Programming

Now i executed the qemu.sh script and using another shell, i ran “ddd &” command which open the ddd debugger.
Now i executed this command:

```
wc < myname.txt | grep 1
```

At this point, in the debugger i had create a breakpoint in the “syscall” function in way that by printing the “num” variable and by watching in the sycall.h file i know which system call is computed to execute the command executed before in the command line

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14

#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21

#define SYS_sem_alloc       22
#define SYS_sem_init        23
#define SYS_sem_destroy     24
#define SYS_sem_wait        25
#define SYS_sem_post        26
```

```
void
syscall(void)
{
  int num;

  num = proc->tf->eax;
  if(num >= 0 && num < SYS_open && syscalls[num]) {
    proc->tf->eax = syscalls[num]();
  } else if (num >= SYS_open && num < NELEM(syscalls) && syscalls[num]) {
    proc->tf->eax = syscalls[num]();
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            proc->pid, proc->name, num);
    proc->tf->eax = -1;
  }
}
```

Made this, each time that the kernel computes a system call, i’m able to know which system call is by printing the variable num.
The list of the system call called by the kernel is the following:

SYS_read x 24
SYS_fork
SYS_wait
SYS_pipe
SYS_fork
SYS_fork
SYS_exec

SYS_read
SYS_read
SYS_write x 6
SYS_read
SYS_write x 2
SYS_read
SYS_exit
SYS_exit