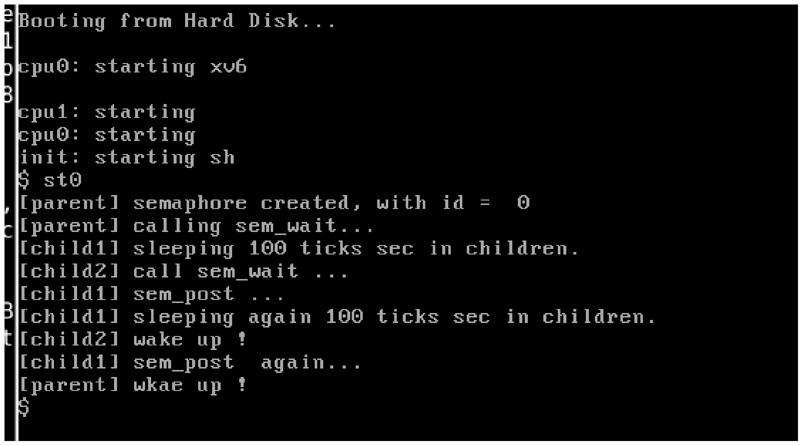
LAB 06

E02

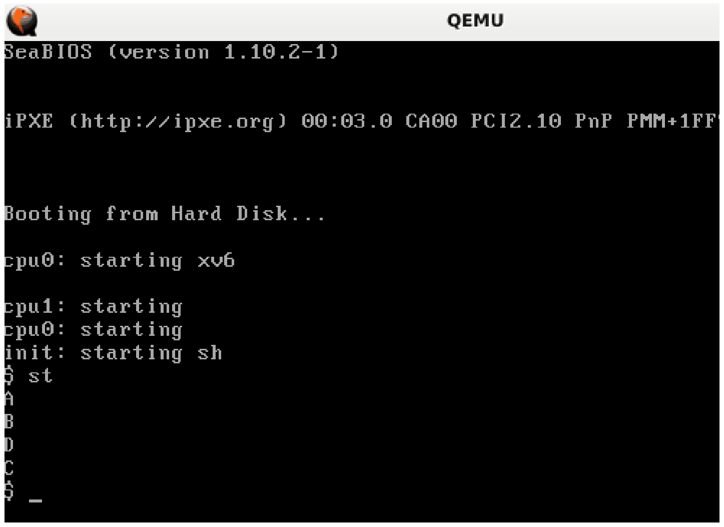
For this exercise I've first take a lot of time to read the xv6 code, and understanding (or trying to understand) the relationships. I've firstly implements sample new system calls to be sure of what I was doing. Then, i've implement my custom semaphores system calls. The modified C files are in the current folder, and you can test the result easily by unzipping the archive and running make qemu; there are two test files, st (st.c) and st0 (st0.c), the last one if my personal tests (sample, the goal is only to check if the counter works fine). Note that in the function sleep(int x), x isn't the number of seconds but the number of ticks (and there are a lot of ticks per seconds following the environment).



```
Booting from Hard Disk...

cpu0: starting xv6

cpu1: starting
cpu0: starting
init: starting sh
$ st0
[parent] semaphore created, with id =  0
[parent] calling sem_wait...
[child1] sleeping 100 ticks sec in children.
[child2] call sem_wait ...
[child1] sem_post ...
[child1] sleeping again 100 ticks sec in children.
[child2] wake up !
[child1] sem_post  again...
[parent] wkae up !
$
```



```
SeaBIOS (version 1.10.2-1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF


Booting from Hard Disk...

cpu0: starting xv6

cpu1: starting
cpu0: starting
init: starting sh
$ st
A
B
D
C
$ _
```

Summary of changes

The system calls in syscall.c use functions defined in file.c (like suggested). This file.c contains a structure "semapore", with fields "state" (to know is the semaphore is allocated, initiated or active), "counter" and "lock" (the lock is of type spinlock). Then, an array of size SEM_NMAX (defined in param.h) is allocated and its semaphores initiated to state = -1 (meaning the are unallocated).

For the rest of the code It's not especially hard to understand and I've commented every difficult part, so I will not copy the code here (the most interesting part is at the begin of file.c). I think that the obtained results and file.c are sufficiently convincing.

About the choice of user mode / kernel mode, the kernel mode was mandatory for sem_wait and sem_post, since in user mode we haven't access to the functions sleep/wakeup (the kernel function sleep).