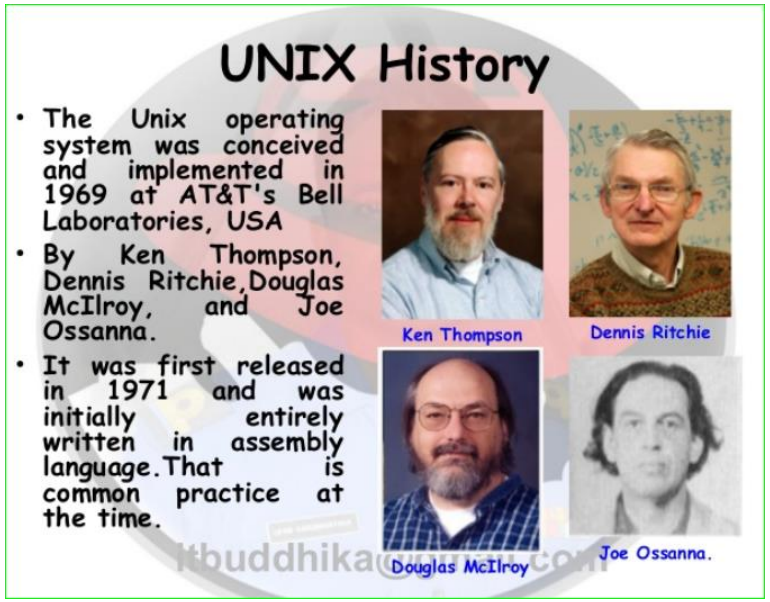
Operating Systems Practice

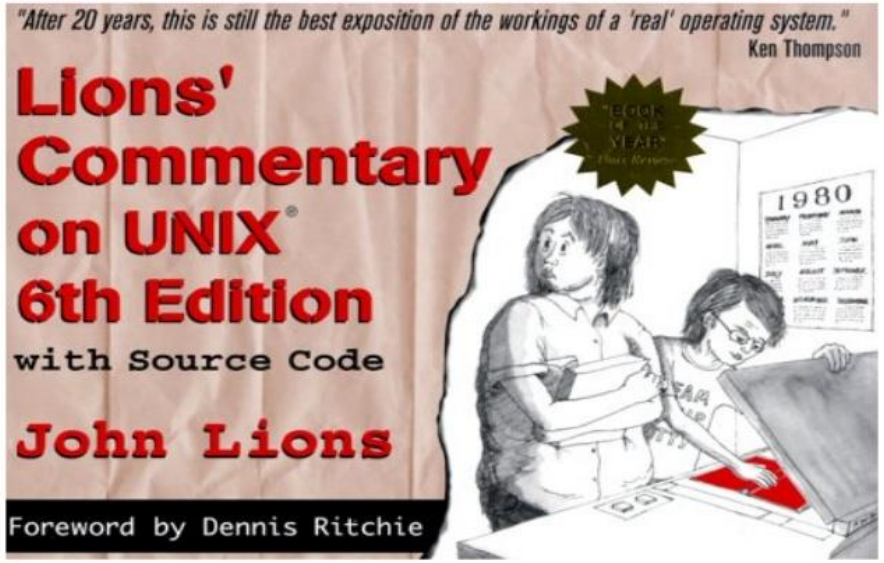
1: xv6 installation

Kilho Lee

Xv6

- Unix-like teaching operating system developed by MIT
- Reimplementation of v6 for a modern x86−based multiprocessor using ANSI C.
- Provide basic interface introduced by Ken Thompson and Dennis Ritchie's Unix operating system, as well as mimicking Unix's internal design





- Latest xv6 source
  - git://github.com/mit-pdos/xv6-public.git
- Source booklet
  - https://pdos.csail.mit.edu/6.828/2018/xv6/xv6-rev10.pdf
- Commentary book
  - https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev10.pdf
  - The line numbers in this commentary book refer to the source booklet

References

- Xv6-books
  - https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev10.pdf
- Contents
  - Ch.0: Operating system interface
  - Ch.1: Operating system organization
  - Ch.2: Page tables
  - Ch.3: Traps, interrupts, and drivers
  - Ch.4: Locking
  - Ch.5: Scheduling
  - Ch.6: File system
  - Ch.7: Summary
  - Appendix A: PC hardware
  - Appendix B: The boot loader

References

- Read the manual.

Code: creating the first process

Now we'll look at how the kernel creates user-level processes and ensures that they are strongly isolated.

After `main` (1217) initializes several devices and subsystems, it creates the first process by calling `userinit` (2520). `Userinit`'s first action is to call `allocproc`. The job of `allocproc` (2473) is to allocate a slot (a `struct proc`) in the process table and to initialize the parts of the process's state required for its kernel thread to execute. `Allocproc` is called for each new process, while `userinit` is called only for the very first process. `Allocproc` scans the `proc` table for a slot with state `UNUSED` (2480-2482). When

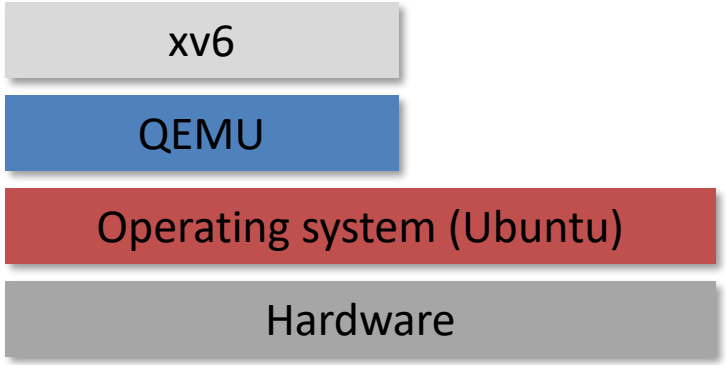
Xv6 System calls

System call	Description
fork()	Create a process
exit()	Terminate the current process
wait()	Wait for a child process to exit
kill(pid)	Terminate process pid
getpid()	Return the current process's pid
sleep(n)	Sleep for n clock ticks
exec(filename, *argv)	Load a file and execute it
sbrk(n)	Grow process's memory by n bytes
open(filename, flags)	Open a file; the flags indicate read/write
read(fd, buf, n)	Read n bytes from an open file into buf
write(fd, buf, n)	Write n bytes to an open file
close(fd)	Release open file fd
dup(fd)	Duplicate fd
pipe(p)	Create a pipe and return fd's in p
chdir(dirname)	Change the current directory
mkdir(dirname)	Create a new directory
mknod(name, major, minor)	Create a device file
fstat(fd)	Return info about an open file
link(f1, f2)	Create another name (f2) for the file f1
unlink(filename)	Remove a file

Figure 0-2. Xv6 system calls

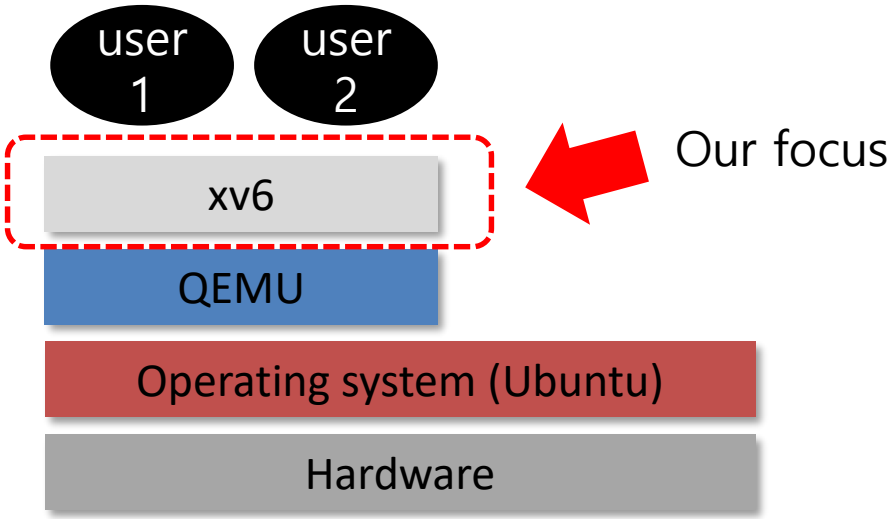
xv6 operating system (Cont'd)

- Xv6 does boot on real hardware, but typically we run it using the QEMU emulator
  - QEMU
    - Open source machine emulator and virtualizer



xv6 operating system (Cont'd)

- Xv6 does boot on real hardware, but typically we run it using the QEMU emulator
  - QEMU
    - Open source machine emulator and virtualizer



break;

Docker

- If needed, you can run xv6 on top of the container
  - If your host OS is not **Ubuntu 18.04 LTS**.
  - If you don't want to run xv6 on top of virtual machines.

- Prerequisites
  - Docker installation
    - https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository
  - The Dockerfile
    - (you can find this in `https://github.com/KilhoLee/xv6-ssu.git`)

Docker installation

- Setup Docker's apt repo

```
$ sudo apt-get update
$ sudo apt-get install ca-certificates curl
$ sudo install -m 0755 -d /etc/apt/keyrings
$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
$ sudo chmod a+r /etc/apt/keyrings/docker.asc
$ echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt-get update
```

- Install Docker packages

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

- Test

```
$ sudo docker run hello-world
```

  - This command downloads a test image and runs it in a container. When the container runs, it prints a confirmation message and exits.

Docker

- How to setup

  1) Build a docker image from the Dockerfile

  # sudo docker build –t xv6:0.0 [PATH_TO_DOCKERFILE]

  (sudo docker build –t xv6:0.0 ./)

  2) Run a shell in the container

  # docker run –it –v /home/[USERNAME]/xv6-ssu:/home/[USERNAME]/xv6-ssu xv6:0.0

  3) Do your job

  root@7513cfb39180:~# .....

  - Restart the closed container

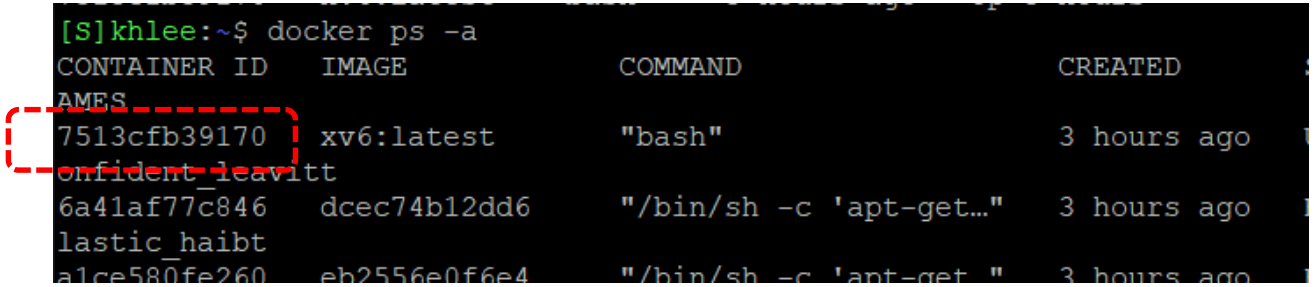
  # docker ps -a

  Find the closed container ID

  # docker restart 7513cf

  - Run another shell in the container

  # docker exec –it 7513cf bash



Xv6 Installation

1. Setting up environments
   1) Install Ubuntu **18.04.4 LTS** (errors may happen in other versions)
   2) sudo apt-get update
   3) sudo apt-get install build-essential gcc-multilib git qemu gdb

2. Install xv6
   1) cd ~
   2) git clone https://github.com/KilhoLee/xv6-ssu.git
   3) cd xv6-ssu/templates
   4) tar xvzf xv6_ssu_init.tar.gz
   5) cd xv6_ssu_init
   6) make qemu-nox

Xv6 Installation

```
[LT]khlee:(main)~/xv6-ssu/xv6_ssu_master$
[LT]khlee:(main)~/xv6-ssu/xv6_ssu_master$ make qemu-nox
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O1 -Wall -MD -ggdb -m32 -Werror -fno-o
 -I. -fno-pie -no-pie   -c -o user/init.o user/init.c
ld -m    elf_i386 -N -e main -Ttext 0 -o user/_init user/init.o ulib.o usys.o printf.o umalloc
objdump -S user/_init > user/init.asm
objdump -t user/_init | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/init.sym
./mkfs fs.img README user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user
r/_sh user/_stressfs user/_usertests user/_wc user/_zombie
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
balloc: first 565 blocks have been allocated
balloc: write bitmap block at sector 58
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.i
2 -m 512
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```

Stop xv6

1) Ctrl ‘A’ + ‘X’

break;

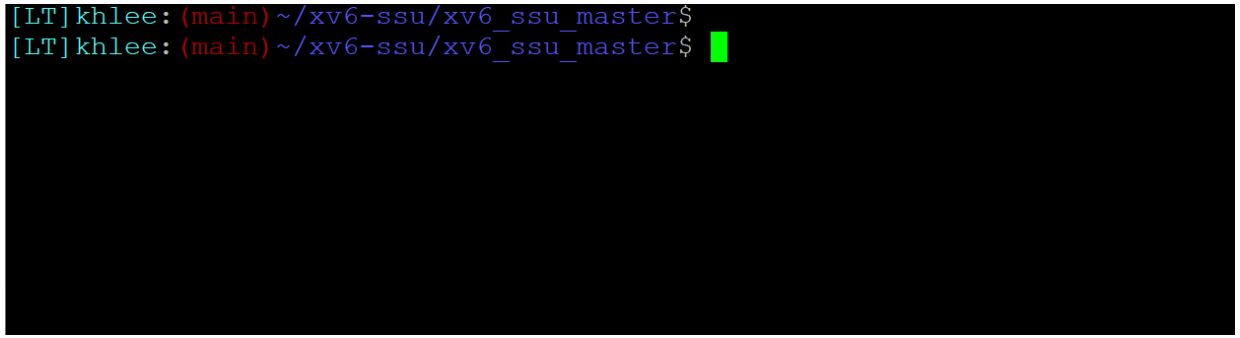
Useful tools: ctags, grep, cscope

- ctags
  - ctags –R *
  - ~/.vimrc 수정
    - Set tags=MY-PATH/tags
  - in vim,
    - ctrl +] : 해당 심볼의 정의로 이동
    - Ctrl + t : 이전 위치로 복귀
- grep
  - grep –rn “bread” * // bread 라는 문자열이 쓰인 곳을 찾아줌
- cscope
  - Database
  - Search symbol / definition

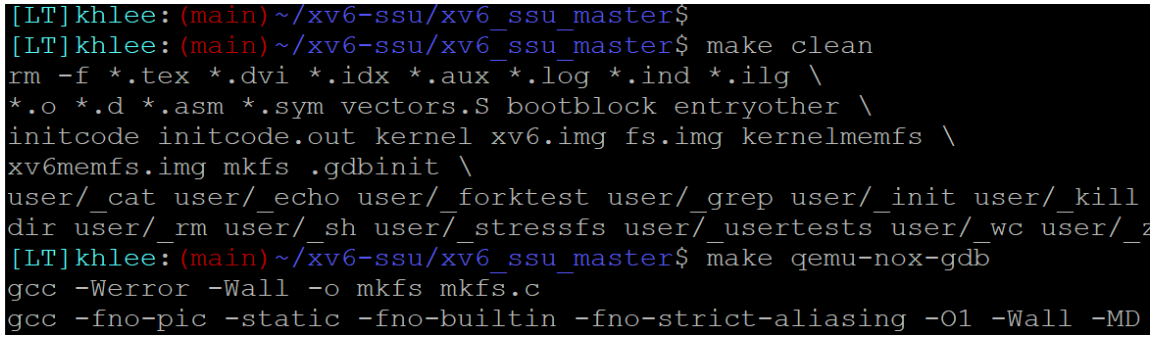
Xv6 + GDB

- Debugging xv6 with GDB

Terminal 1: Run gdb to send a command



Terminal 2: Run xv6 with gdb



Xv6 + GDB

- Debugging xv6 with GDB

Terminal 1: Run gdb to send a command

```
khlee@khlee-x1: ~/xv6-ssu/xv6_ssu_master
[LT]khlee:(main)~/xv6-ssu/xv6_ssu_master$ gdb kernel
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel...done.
warning: File "/home/khlee/xv6-ssu/xv6_ssu_master/.gdbinit" auto-loading has beer
our `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
        add-auto-load-safe-path /home/khlee/xv6-ssu/xv6_ssu_master/.gdbinit
line to your configuration file "/home/khlee/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/
```

Terminal 2: Run xv6 with gdb

```
[LT]khlee:(main)~/xv6-ssu/xv6_ssu_master$
[LT]khlee:(main)~/xv6-ssu/xv6_ssu_master$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill
dir user/_rm user/_sh user/_stressfs user/_usertests user/_wc user/_
[LT]khlee:(main)~/xv6-ssu/xv6_ssu_master$ make qemu-nox-gdb
gcc -Werror -Wall -o mkfs mkfs.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O1 -Wall -MD
```

~/.gdbinit 수정, 아래 문구 추가
add-auto-load-safe-path [Your_xv6_path]/.gdbinit

```
khlee@khlee-x1: ~/xv6-ssu/xv6_ssu_master
  1 add-auto-load-safe-path /home/khlee/xv6-ssu/xv6_ssu_master/.gdbinit
~
~
```

Xv6 + GDB

Terminal 1: Run gdb to send a command

```
[LT]khlee:(main)~/xv6-ssu/xv6_ssu_master$ gdb kernel
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/license
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel...done.
+ target remote localhost:26000
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
```

Connection succeeds!

Terminal 2: Run xv6 with gdb

```
sed "s/localhost:1234/localhost:26000/" < .gdbinit.tmpl > .gdbinit
*** Now run 'gdb'.
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.i
g,index=0,media=disk,format=raw -smp 2 -m 512  -S -gdb tcp::26000
```

Xv6 + GDB

- Break in the precedure “exec”

Terminal 1: type commands

```
(gdb) b exec
Breakpoint 1 at 0x80100912: file exec.c, line 12.
(gdb) c
Continuing.
[Switching to Thread 2]
The target architecture is assumed to be i386
=> 0x80100912 <exec>:   push   %ebp

Thread 2 hit Breakpoint 1, exec (path=0x1c "/init", argv=0x8c
12      {
(gdb) n
=> 0x8010091e <exec+12>:        call   0x8010332e <myproc>
20        struct proc *curproc = myproc();
(gdb) n

Thread 2 received signal SIGTRAP, Trace/breakpoint trap.
=> 0x801032ad <mycpu+17>:       mov    0x80111d00,%esi
mycpu () at proc.c:48
48        for (i = 0; i < ncpu; ++i) {
(gdb) d i
warning: bad breakpoint number at or near '0'
(gdb) display ncpu
1: ncpu = 2
(gdb)
```

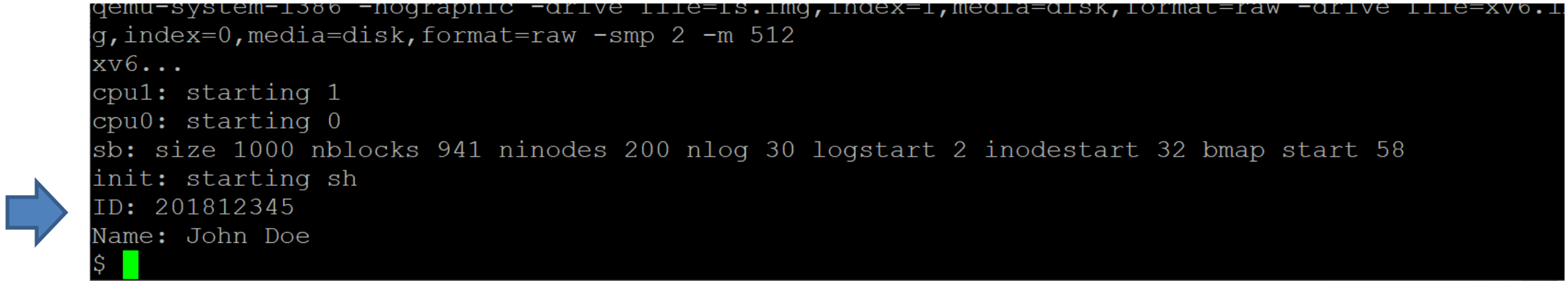
Terminal 2: xv6 stops running in the breakpoint

```
img,index=0,media=disk,format=raw -smp 2 -m 512  -S -gdb tcp::26000

xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
```

P0-1. Print Your Name!

- Print your ID and Name in the xv6 boot message
  - Find the initial program code and insert your code into it.



- Hand-in procedure
  - P0-1_201812345.jpg
    - Result Screenshot
    - Your code Screenshot
  - Upload your file into LMS
  - Hint: find the "main()" function

P0-2. Get the Process size with GDB

- Breakpoint at `fork()`
- Display `np->size` at line number 199

- Hand-in procedure
  - P0-2_201812345.jpg
    - Screenshot gdb
  - Upload your file into LMS

```
176
177 // Create a new process copying p as the parent.
178 // Sets up stack to return as if from system call.
179 // Caller must set state of returned proc to RUNNABLE.
180 int
181 fork(void)
182 {
183   int i, pid;
184   struct proc *np;
185   struct proc *curproc = myproc();
186
187   // Allocate process.
188   if((np = allocproc()) == 0){
189     return -1;
190   }
191
192   // Copy process state from proc.
193   if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
194     kfree(np->kstack);
195     np->kstack = 0;
196     np->state = UNUSED;
197     return -1;
198   }
199   np->sz = curproc->sz;
200   np->parent = curproc;
201   *np->tf = *curproc->tf;
202
203   // Clear %eax so that fork returns 0 in the child.
204   np->tf->eax = 0;
205
206   for(i = 0; i < NOFILE; i++)
207     if(curproc->ofile[i])
208       np->ofile[i] = filedup(curproc->ofile[i]);
209   np->cwd = idup(curproc->cwd);
210
211   safestrcpy(np->name, curproc->name, sizeof(curproc->name));
212
213   pid = np->pid;
214
215   acquire(&ptable.lock);
216
217   np->state = RUNNABLE;
218
219   release(&ptable.lock);
220
221   return pid;
222 }
```