EE488: System Software Design
Fall 2022

Warm-up

Youjip Won

KAIST

Outline

- Build and run xv6
- add new programs to xv6.

Resources

- GitHub - mit-pdos/xv6-public: xv6 OS
- A booklet on xv6
  - Original source for xv6 booklet on x86 or on RISC-V.
  - How to download and compile the book.
- Differet platforms for xv6: Linux, MacOS, Windows
- Different ways of building xv6

```
%make qemu
```

  - Build xv6.

```
%make qemu-nox
```

  - Build xv6 without graphics display. This option will be used for building xv6 in WSL.

```
%make qemu-gdb
```

  - Build gdb attachable xv6.

```
%make qemu-nox-gdb
```

  - Build gdb attachable xv6 without graphics display. This option will be used for building xv6 in WSL.

xv6 - Makefile

```
qemu: fs.img xv6.img
	$(QEMU) -serial mon:stdio $(QEMUOPTS)

qemu-memfs: xv6memfs.img
	$(QEMU) -drive file=xv6memfs.img,index=0,media=disk,format=raw -smp $(CPUS) -m 256

qemu-nox: fs.img xv6.img
	$(QEMU) -nographic $(QEMUOPTS)

.gdbinit: .gdbinit.tmpl
	sed "s/localhost:1234/localhost:$(GDBPORT)/" < $^ > $@

qemu-gdb: fs.img xv6.img .gdbinit
	@echo "*** Now run 'gdb'." 1>&2
	$(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUGDB)

qemu-nox-gdb: fs.img xv6.img .gdbinit
	@echo "*** Now run 'gdb'." 1>&2
	$(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)
```

```
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -
drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
```

Xv6 on Linux

- Tools installation

```
$ sudo apt install -y wget git qemu build-essential gdb
```

- Source download

```
$ git clone https://github.com/mit-pdos/xv6-public.git
```

- Build and run xv6.

```
$ cd xv6-public && make qemu (or qemu-nox)
```

  - To quit, press `ctrl + a` then `c`.

- Run and run xv6 with GDB.

```
$ cd xv6-public && make qemu-gdb (or qemu-nox-gdb)
```

  - Open new terminal and switch to new terminal

```
$ cd xv6-public && gdb ./kernel (at another terminal)
```

After build and run xv6 on Linux

- Build and run xv6.

```
geese@quack: ~/Documents/xv6-public
File  Edit  View  Search  Terminal  Help
.o trap.o uart.o vectors.o vm.o  -b binary initcode entryother
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.0170636 s, 300 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 9.2974e-05 s, 5.5 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
326+1 records in
326+1 records out
167248 bytes (167 kB, 163 KiB) copied, 0.000531181 s, 315 MB/s
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -dr
ive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$
```

- Run xv6 with GDB.

```
geese@quack: ~/Documents/xv6-public
File  Edit  View  Search  Terminal  Help
geese@quack:~/Documents/xv6-public$ make qemu-gdb
sed "s/localhost:1234/localhost:26000/" < .gdbinit.tmpl > .gdbinit
*** Now run 'gdb'.
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media
=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw
-smp 2 -m 512  -S -gdb tcp::26000
```

(1) xv6 terminal.

```
geese@quack: ~/Documents/xv6-public
File  Edit  View  Search  Terminal  Help
<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/geese/Documents/xv6-public/.gdbinit" auto-loa
ding has been declined by your `auto-load safe-path' set to "$debu
gdir:$datadir/auto-load".
To enable execution of this file add
        add-auto-load-safe-path /home/geese/Documents/xv6-public/.
---Type <return> to continue, or q <return> to quit---
gdbinit
line to your configuration file "/home/geese/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/geese/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run fro
m the shell:
        info "(gdb)Auto-loading safe path"
(gdb)
```

(2) gdb terminal.

Xv6 on Linux [Troubleshooting]

- Incompatible compiler version
  - GCC 4.6.4 is recommended.
  - * Latest Ubuntu distribution may not have the GCC 4.6.4 in its repository.
  - Add the following Ubuntu source repository in `/etc/apt/sources.list`.
    - deb http://dk.archive.ubuntu.com/ubuntu/ trusty main universe
    - deb http://dk.archive.ubuntu.com/ubuntu/ trusty-updates main universe
  - Install GCC 4.6.4.

```
$ sudo apt update
$ sudo apt install -y gcc-4.6
```

Xv6 on Linux [Troubleshooting] (Cont.)

- Incompatible compiler version (Cont.)
  - Make executable binary linked to GCC 4.6.4.

```
$ sudo rm /usr/bin/gcc
$ sudo ln -s gcc-4.6 gcc
```

  - Check if the gcc points to the executable

```
$ ls -l | grep gcc
```

```
lrwxrwxrwx  1 root root       7 … gcc -> gcc-4.6*
```

```
$ gcc --version
```

```
gcc (Ubuntu/Linaro 4.6.4-6ubuntu2) 4.6.4 …
```

Xv6 on MacOS

- Install package manager for MacOS: homebrew or MacPorts
  - Package manager allows the user to download and to install various open source tools.
  - Widely used package manager is homebrew and macports.
- Install MacPorts
  - Download xcode from app store.
  - Install xcode command line tool

    ```
    $ xcode-select --install
    ```
  - Download package installer of MacPorts from https://www.macports.org/install.php. Install the correct MacPorts package for your MacOS version.
- Export path so that the shell can locate the installed tools.

  ```
  $ export PATH=/opt/local/bin:/opt/local/sbin:$PATH
  ```
- Install gcc with Macports

  ```
  $ sudo port install git wget && sudo port install qemu i386-elf-gcc
  ```
- Install gdb with homebrew (gdb 32bit is only available in brew but not with Macports)

  ```
  $ brew install i386-elf-gdb
  ```

  Download the xv6 source code.

  ```
  $ git clone https://github.com/mit-pdos/xv6-public.git
  ```

Xv6 on MacOS [build]

- Modify Makefile for xv6.

```
$ cd xv6-public && vi Makefile
```

  - Before the modification (line 32 of Makefile)

```
#TOOLPREFIX = i386-jos-elf
```

  - After the modification (line 32 of Makefile)

```
TOOLPREFIX = i386-elf-
```

- Build and run xv6

```
$ make qemu
```

- Build xv6 with GDB and run xv6. Make the xv6 attachable from GDB.

```
$ make qemu-gdb
```

  - Open another terminal. Run ‘gdb’ in the new terminal as follows. It will attach gdb to the xv6 kernel.

```
$ i386-elf-gdb ./kernel (at another terminal)
```

After build and run xv6 on MacOS

- Build and run xv6.

```
laurent01@Seungui-MacBookPro xv6-public % make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=
raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 15644
```

...

- Run xv6 with GDB.

```
laurent01@Seungui-MacBookPro xv6-public % make qemu-gdb
*** Now run 'gdb'.
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,me
dia=disk,format=raw -drive file=xv6.img,index=0,media=disk,forma
t=raw -smp 2 -m 512  -S -gdb tcp::25501
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodest
art 32 bmap start 58
init: starting sh
$
                          "Seungui-MacBookPro.lo" 15:08 24- 8-22
```

(1) xv6 terminal.

```
laurent01@Seungui-MacBookPro xv6-public % i386-elf-gdb ./[45/473]
GNU gdb (GDB) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licens
es/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 "--host=x86_64-apple-darwin21.3.0 --ta
rget=i386-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://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...
+ target remote localhost:25501
The target architecture is set to "i8086".
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
(gdb) disass
No function contains program counter for selected frame.
(gdb) c
Continuing.
^C
```

(2) gdb terminal.

Xv6 on Windows (Windows Subsystem for Linux)

- Install xv6 with windows subsystem for linux.
  - Run the PowerShell as administrator. (Link)
  - Install and run ubuntu under wsl.

```
$ wsl --install -d ubuntu
$ sudo apt-get update -y
$ sudo apt-get install git make gcc wget qemu qemu-kvm gdb -y
$ git clone https://github.com/mit-pdos/xv6-public.git
```

- Build and run xv6.

```
$ cd xv6-public && make qemu-nox
```

- Build and un xv6 with GDB.
  - Add "set auto-load safe-path /" line to configuration file "~/.gdbinit".

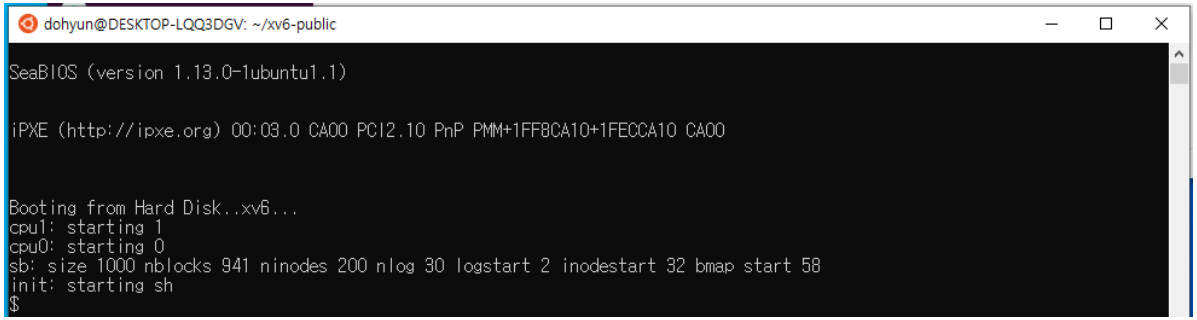
```
$ cd xv6-public
$ make qemu-nox-gdb
```

  - Open new ubuntu instance. You will get new terminal. Launch gdb with kernel binary for xv6.

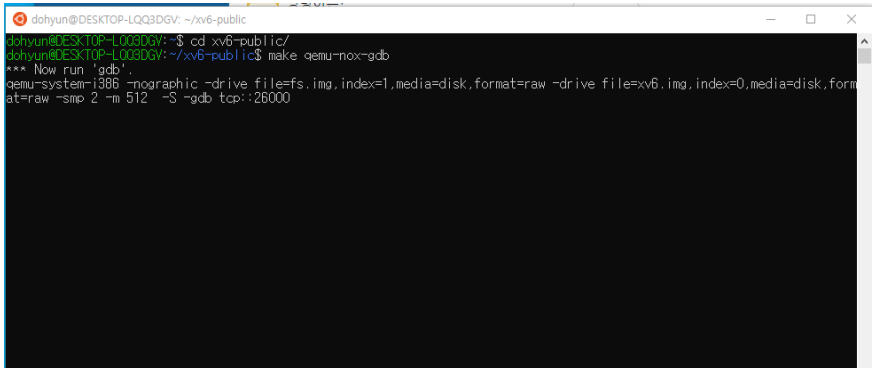
```
$ cd xv6-public
$ gdb ./kernel (at the new ubuntu instance)
```

After build and run xv6 on Windows

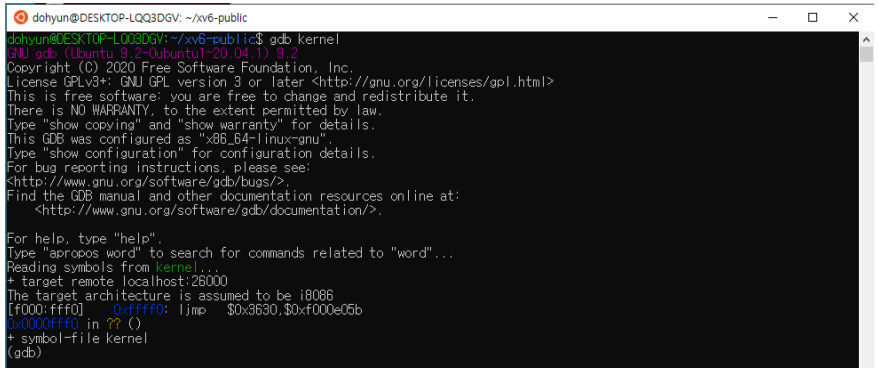
- Build and run xv6.



- Build and un xv6 with GDB.



(1) xv6 - ubuntu instance.



(2) gdb - ubuntu instance.

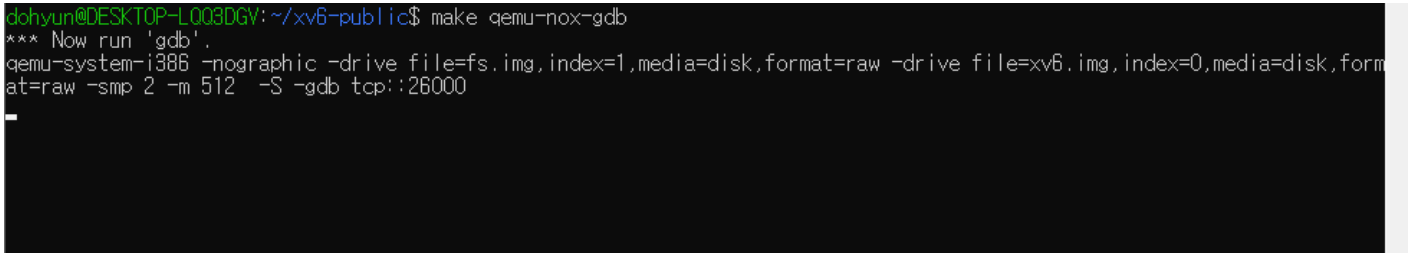
xv6 with GDB

- Each `make` command executes a following `qemu` command.
  - `$ make qemu`
    - `qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512`
  - `$ make qemu-gdb`
    - `qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512 -S -gdb tcp::26000`

Normal Vs. Debug mode booting

- Difference between two `qemu` commands is "-S -gdb tcp::26000"
  - `-S` : suspend the debug target just before the booting starts.
  - `-gdb tcp::[port]`
    - port number that is used to communicate with gdb.

- If "`make qemu-gdb`" or "`make qemu-nox-gdb`" is executed, QEMU stops until the gdb is connected.
- Xv6 cannot and boot up and operate.



Connect gdb to QEMU

- Open a new terminal.
- Go to the directory where kernel binary resides.
- Execute gdb with loading the target binary.

```
$ gdb [binary file to load]
```

```
$ gdb kernel                    // linux & WSL
$ i386-elf-gdb kernel           // macOS
```

```
dohyun@DESKTOP-LOQ3DGV:~/xv6-public$ gdb kernel
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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...
+ target remote localhost:26000
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
(gdb)
```

Output of WSL

Connect gdb to QEMU

① GDB loads the binary file.

② GDB is connected to QEMU with port number we set.

- It is done automatically, but if not, please type the following command.

```
$ (gdb) target remote localhost:26000
```

Output of WSL

Executing a program in xv6

- xv6 is not full fledged OS.
- We cannot compile a program within xv6.
- We need to compile xv6 and the program that we like to run in xv6 together.
  - Create a file in xv6 foler. ex: test.c

```
int
main(int argc, char *argv[]) {
    printf(1,"Hello world!\n");
    exit();
}
```

  - Add `_test` to to `UPROGS` as in the right box.

```
168 UPROGS=\
169   _cat\
170   _echo\
171   _forktest\
172   _grep\
173   _init\
174   _kill\
175   _ln\
176   _ls\
177   _mkdir\
178   _rm\
179   _sh\
180   _stressfs\
181   _usertests\
182   _wc\
183   _zombie\
184   _test\
185
```

- Build xv6 and run.
- run the new program.

xv6 source organization

- We need to use xv6 provided header file.
  - We cannot use `stdio.h`.
  - Use the following header files.

"`types.h`" : header file for variable types

"`user.h`" : header file for system calls

"`fcntl.h`" : header file for file IO

- Please add the following programnings to xv6 and run it.
  - practice1.c – practice8.c

practice1.c

```
#include "type.h"
#include "user.h"

int
main(int argc, char *argv[]) {
     int pid = fork();

     if(pid > 0) {
          printf(1,"parent: child=%d\n",pid);
          pid = wait();
          printf(1,"child %d is done\n",pid);
     } else if(pid == 0){
          printf(1,"child exiting\n");
          exit();
     } else {
          printf(1,"fork error\n");
     }

     exit();
}
```

practice2.c

```
#include "types.h"
#include "user.h"

int
main() {
     char *argv[3];
     argv[0] = "echo";
     argv[1] = "hello";
     argv[2] = 0;

     exec("./echo",argv);

     printf(1,"exec error\n");

     exit();
}
```

practice3.c

```
#include "types.h"
#include "user.h"

int
main(int argc, char *argv[]) {
     char buf[512];
     int n;
     for(;;){
          n = read(0,buf,sizeof buf);
          printf(1,"%d",n);
          if(n == 0)
               exit();
          if(n < 0) {
               printf(2,"read error!\n");
               exit();
          }
          if(write(1,buf,n) != n) {
               printf(2,"write error!\n");
               exit();
          }
     }
     exit();
}
```

practice4.c

```
#include "types.h"
#include "user.h"

int
main() {
    char *argv[2];
    argv[0] = "cat";
    argv[1] = 0;

    if(fork() == 0) {
        close(0);
        open("input.txt", 0); // 0 = O_RDONLY
        exec("cat", argv);
    } else {
        wait();
    }
    exit();
}
```

practice5.c

```
#include "types.h"
#include "user.h"

int
main() {
    if(fork() == 0) {
        write(1, "hello", 6);
        exit();
    } else {
        wait();
        write(1, "world\n", 6);
    }
    exit();
}
```

practice6.c

```
#include "types.h"
#include "user.h"

int
main() {
     int fd = dup(1);
     write(1, "hello ", 6);
     write(fd, "world\n", 6);
     exit();
}
```

practice7.c

```
#include "types.h"
#include "user.h"

int
main () {

     int p[2];
     char *argv[2];
     argv[0] = "wc";
     argv[1] = 0;

     pipe(p);
     if(fork() == 0) {
          close(0);
          dup(p[0]);
          close(p[0]);
          close(p[1]);
          exec("./wc",argv);
     } else {
          close(p[0]);
          write(p[1], "hello world\n",12);
          close(p[1]);
          wait();
     }
     exit();
}
```

practice8.c

```
#include "types.h"
#include "fcntl.h"
#include "user.h"

int
main (int argc, char *argv[]) {

    mkdir("./dir");
    int fd = open("./dir/file", O_CREATE|O_WRONLY);
    close(fd);

    mknod("/console",1,1);

    exit();
}
```