

CS179F: Projects in Operating System

Week 2

Emiliano De Cristofaro and Lian Gao

Team

- Instructor: Emiliano De Cristofaro (emilianodc@cs.ucr.edu)
 - I am a Professor in CSE working on security, privacy, and cybersafety
 - **Office hours: <https://calendly.com/emilianodc/cs179f> (Wed-Thu 11-12)**
- TA: Gao Lian
 - PhD student in cybersecurity
 - Office hours in lab on Wednesdays (more details later)

Projects, with deadlines

- 5 projects in xv6-riscv, one every 2 weeks, each 20% of the final grade
 1. Unix Utilities: sleep, find, xargs
 2. Memory Allocation
 3. Copy-On-Write
 4. File System: large files and symbolic links
 5. mmap

Project “Rules”

- Each project should be finished **individually**, unless the class size increases unexpectedly
 - Discussions are fine and encouraged
 - TA and I are there for help, try Piazza first before email. Lab office hours before anything else
 - Other “ways” to get coding done? E.g., Github Copilot?
- Late policy
 - 20% penalty if within 48 hours
 - 0% beyond 48 hours (exceptions granted with evidence)

Class Material

<https://github.com/emidec/cs179f-fall23>

Resources

- Operating Systems: Three Easy Pieces, Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
- XV6: A Simple UNIX-like Teaching Operating System, Russ Cox, Frans Kaashoek, and Robert Morris
- Lions' Commentary on UNIX' 6th Edition, John Lions, Peer to Peer Communications. ISBN: 1-57398-013-7. 1st edition (June 14, 2000)
- A good guidance: <https://pdos.csail.mit.edu/6.828/2023/labs/guidance.html>

Communication

- Piazza (<https://piazza.com/ucr/fall2023/cs179f>) as the main communication channel
 - Announcements, slides, projects, polls, etc.
 - Discussion and Q&A
- Canvas (<https://elearn.ucr.edu/courses/110956>)
 - For assignments and grades

Week 1

- Tue Oct 3: Lecture
- Wed Oct 4: EDC + Lian lab

Week 2

- Tue Oct 10: Lecture
- Wed Oct 11: Lian lab

Week 3

- Wed Oct 18: EDC + Lian lab

**Lab 1 due:
Oct 20th, 1:59pm**

Week 4

- Tue Oct 24: Lecture
- Wed Oct 25: Lian lab

Week 5

- Wed Nov 1: EDC + Lian lab

**Lab 2 due:
Nov 3rd, 1:59pm**

Week 6

- Tue Nov 7: Lecture
- Wed Nov 8: Lian lab

Week 7

- Wed Nov 15: EDC + Lian lab

**Lab 3 due:
Nov 17th, 1:59pm**

Week 8

- Tue Nov 21: Lecture
- Wed Nov 22: Lian lab

Week 9

- Wed Nov 29: EDC + Lian lab

**Lab 4 due:
Dec 1st, 1:59pm**

Week 10

- Tue Dec 5: Lecture
- Wed Dec 6: Lian lab

**Lab 5 due:
Dec 15th, 1:59pm**

Environment — xv6

- We will use the XV6 operating system as a base for our projects
 - A re-implementation of Unix Version 6 for a modern RISC-V multiprocessor using ANSI C
- Familiarize yourself with XV6 on how it is organized and implemented:
 - Take a look at the online version of the Lions commentary
 - Look at the source code, etc.

Tools

- **xv6-riscv** (see previous slide/class GitHub)
- **qemu** (open source machine emulator and virtualizer)
- **labs code** (on class repo)

See README.md on the class repo (<https://github.com/emidec/cs179f-fall23>) for more info on how to set everything up

Note: currently having trouble with Mac (use Linux VM) and new versions of qemu (use v4 or v5, not v8)

Lab 1

- Implement the UNIX program **sleep** for xv6
- Write a simple version of the UNIX **find** program: find all the files in a directory tree with a specific name
- Write a simple version of the UNIX **xargs** program: read lines from the standard input and run a command for each line, supplying the line as arguments to the command

See class git repo / <https://github.com/emidec/cs179f-fall23/blob/xv6-riscv-fall23/doc/lab1.md>

Lab 1 — Util

- Quick reference:
 - `$ make qemu // compile and run xv6`
 - `$ make grade // test your solution with the grading program`
 - `$./grade-lab-util sleep`
 - `$ Make GRADEFLAGS=sleep grade`
 - To quit qemu type: `ctrl+a x`
- To compile your program:
 - Add your program under `/xv6-riscv/user` named as `<prog>.c`
 - Modify `UPROGS` in Makefile accordingly

Dynamic Memory Allocation

- Allocator maintains a **heap** as collection of variable sized *blocks*, which are either *allocated* or *free*
- Types of allocators
 - Explicit allocator: application allocates and frees space (e.g., `malloc` and `free` in C)
 - Implicit allocator: application allocates, but does not free space (e.g., garbage collection in Java, ML, and Lisp)
- Will use explicit memory allocation

Lab 1 Start — Boot xv6

- Fetch the xv6 source for the lab and check out the util branch:

```
$ git clone git@github.com:emidec/cs179f-fall23
Cloning into 'xv6-riscv'...
...
$ cd xv6-riscv
$ git checkout util
Branch 'util' set up to track remote branch 'util' from 'origin'.
Switched to a new branch 'util'
```

Git

- As per previous slide, you switched to a branch (git checkout util) containing a version of xv6 tailored to this lab.
- To learn more about Git, take a look at the [git user's manual](#), or, you may find this [overview of git](#) useful. Git allows you to keep track of the changes you make to the code.
- For example, if you are finished with one of the exercises, and want to checkpoint your progress, you can commit your changes by running:

```
$ git commit -am 'my solution for util lab exercise 1'
Created commit 60d2135: my solution for util lab exercise 1
1 files changed, 1 insertions(+), 0 deletions(-)
$
```

- You can keep track of your changes using `git diff` command
 - Running `git diff` will display the changes to your code since your last commit, and `git diff origin/util` will display the changes relative to the initial code

Build and run xv6

```
$ make qemu
  riscv64-unknown-elf-gcc      -c -o kernel/entry.o kernel/entry.S
  riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -DSOL\ _UTIL -MD -mcmodel=medany -ffreestanding
-fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o kernel/start.o kernel/start.c
  ...
  riscv64-unknown-elf-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/\_zombie user/zombie.o user/ulib.o user/
usys.o user/printf.o user/umalloc.o
  riscv64-unknown-elf-objdump -S user/\_zombie > user/zombie.asm
  riscv64-unknown-elf-objdump -t user/\_zombie | sed '1,/SYMBOL TABLE/d; s/ .\* / /; /^$/d' > user/zombie.sym
  mkfs/mkfs fs.img README user/xargstest.sh user/\_cat user/\_echo user/\_forktest user/\_grep user/\_init user/\_kill
user/\_ln user/\_ls user/\_mkdir user/\_rm user/\_sh user/\_stressfs user/\_usertests user/\_grind user/\_wc user/
\_zombie
  nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 954 total 1000
  balloc: first 591 blocks have been allocated
  balloc: write bitmap block at sector 45
  qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -drive
file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$
```


Build and run xv6 (continued)

- If you type `ls` at the prompt, you should see output similar to the following:

```
$ ls
.          1 1 1024
..         1 1 1024
README    2 2 2059
xargstest.sh 2 3 93
cat        2 4 24256
echo       2 5 23080
forktest   2 6 13272
grep       2 7 27560
init       2 8 23816
kill       2 9 23024
ln         2 10 22880
ls         2 11 26448
mkdir      2 12 23176
rm         2 13 23160
sh         2 14 41976
stressfs   2 15 24016
usertests  2 16 148456
grind      2 17 38144
wc         2 18 25344
zombie     2 19 22408
console    3 20 0
```

These are the files that `mkfs` includes in the initial file system; most are programs you can run. You just ran one of them: `ls`.

`xv6` has no `ps` command, but, if you type `Ctrl-p`, the kernel will print information about each process. If you try it now, you'll see two lines: one for `init`, and one for `sh`.

To quit `qemu` type: `Ctrl-a x`.

Lab 1 — sleep

- Implement the UNIX program sleep for xv6; your sleep should pause for a user-specified number of ticks.
- A tick is a notion of time defined by the xv6 kernel, namely the time between two interrupts from the timer chip.
- Your solution should be in the file user/sleep.c.

Hints 1/2

- Before you start coding, read Chapter 1 of the xv6 book
- Look at some of the other programs in user/ (e.g., user/echo.c, user/grep.c, and user/rm.c) to see how you can obtain the command-line arguments passed to a program
- If the user forgets to pass an argument, sleep should print an error message
- The command-line argument is passed as a string; you can convert it to an integer using atoi (see user/ulib.c)
- Use the system call sleep

Hints 2/2

- See `kernel/sysproc.c` for the xv6 kernel code that implements the sleep system call (look for `sys_sleep`), `user/user.h` for the C definition of sleep callable from a user program, and `user/usys.S` for the assembler code that jumps from user code into the kernel for sleep.
- Make sure `main` calls `exit()` in order to exit your program
- Add your sleep program to `UPROGS` in `Makefile`; once you've done that, `make qemu` will compile your program and you'll be able to run it from the xv6 shell
- Look at Kernighan and Ritchie's book *The C programming language* (second edition) (K&R) to learn about C

Sleep Success

```
$ make qemu
...
init: starting sh
$ sleep 10
(nothing happens for a little while)
$
```

- Your solution is correct if your program pauses when run as shown above. Run `make grade` to see if you indeed pass the sleep tests.
- Note that `make grade` runs all tests, including the ones for the assignments below. If you want to run the grade tests for one assignment, type:

```
$ ./grade-lab-util sleep      OR      make GRADEFLAGS=sleep grade
```

Find

- Write a simple version of the UNIX find program: find all the files in a directory tree with a specific name. Your solution should be in the file user/find.c.
- Some hints:
 - Look at user/lis.c to see how to read directories
 - Use recursion to allow find to descend into sub-directories
 - Don't recurse into "." and ".."
 - Changes to the file system persist across runs of qemu; to get a clean file system run make clean and then make qemu
 - You'll need to use C strings. Have a look at K&R (the C book), for example Section 5.5
 - Note that == does not compare strings like in Python. Use strcmp() instead
 - Add the program to UPROGS in Makefile

Find — Correct

```
$ make qemu
...
init: starting sh
$ echo > b
$ mkdir a
$ echo > a/b
$ find . b
./b
./a/b
$
```

xargs

- Write a simple version of the UNIX xargs program: read lines from the standard input and run a command for each line, supplying the line as arguments to the command. Your solution should be in the file user/xargs.c.
- The following example illustrates xarg's behavior:

```
$ echo hello too | xargs echo bye  
bye hello too  
$
```

- Note that the command here is "echo bye" and the additional arguments are "hello too", making the command "echo bye hello too", which outputs "bye hello too"

xargs

- Please note that xargs on UNIX makes an optimization where it will feed more than one argument to the command at a time.
- We don't expect you to make this optimization. To make xargs on UNIX behave the way we want it to for this lab, please run it with the -n option set to 1. For instance:

```
$ echo "1\\n2" | xargs -n 1 echo line  
    line 1  
    line 2  
$
```

Hints

- Use fork and exec to invoke the command on each line of input. Use wait in the parent to wait for the child to complete the command
- To read individual lines of input, read a character at a time until a newline ('\n') appears
- kernel/param.h declares MAXARG, which may be useful if you need to declare an argv array
- Add the program to UPROGS in Makefile
- Changes to the file system persist across runs of qemu; to get a clean file system run make clean and then make qemu

Testing

- To test your solution for xargs, run the shell script xargstest.sh. Your solution is correct if it produces the following output:

```
$ make qemu
...
init: starting sh
$ sh < xargstest.sh
$ $ $ $ $ $ hello
hello
hello
$ $
```

Submission

- Submit lab report and the code diff file on canvas
 - Lab report: pdf format; including screen shots and explanations of the key changes you made, and the screenshots of your output messages.
 - Code diff file: the generated diff.txt after your run “git diff > diff.txt”
- Demo with TA during the lab sessions after the ddl (a signup sheet for the demo will be released after the ddl).
- Grade specification:
 - Boot xv6: 2 credits
 - Sleep: 4 credits
 - Find: 7 credits
 - Xargs: 7 credits

Additional Credit

- Individual or group project (max 3 people)
- Tl;dr: identify, understand, re-produce, and discuss an academic paper on (operating) systems security
- Step 1: express interest via email or office hours (group or individual)
- Step 2: identify 1-2 papers from a list
- Step 3: book office hours every ~2 weeks to discuss
- Step 4: demo + project report