Operating System
Week 1 - introduction

Korawit Orkphol, D.Eng

1

Operating System

- 03603332 Operating System (First semester of 2022)
- Credits : 3 (3 hrs. In class-0 hr in Labs-6 hrs self-study)
- Prerequisites : 03603223 Computer Architecture and Organization
- Course description : Concepts of operating systems. Processes and concurrency. Process management and scheduling. Input/output management. Memory management. File systems. Computer systems security.
- Duration : 15 weeks
- Location & date time : 17301 Mon. 16.30-19.30
- MSTEAM code : tml9h8o (This team is the channel for communication between instruction and students, assignments, relevant files, and online meeting.
- Scoring : in-class activities 10%, assignments 20%, project 20%, midterm exam 20%, final exam 20%, class attend 10%
- Grading : A: > 80, B+ to - D: based on group of students, F : <40
- Contact : 23617 or korawit@eng.src.ku.ac.th or facebook
- Material: mostly based on Operating Systems: Three Easy Pieces (wisc.edu)

2

Content for 15 weeks (3 topics at a week)

Intro	Virtualization		Concurrency	Persistence	Security
Preface	3 *Dialogue*	12 *Dialogue*	25 *Dialogue*	33 *Dialogue*	52 *Dialogue*
TOC	4 Processes	13 Address Spaces code	26 Concurrency and Threads code	36 I/O Devices	53 *Intro Security*
1 *Dialogue*	5 Process API code	14 Memory API	27 Thread API code	37 Hard Disk Drives	54 *Authentication*
2 Introduction code	6 Direct Execution	15 Address Translation	28 Locks code	38 Redundant Disk Arrays (RAID)	55 *Access Control*
	7 CPU Scheduling	16 Segmentation	29 Lock-based Data Structures	39 Files and Directories	56 *Cryptography*
	8 Multi-level Feedback	17 Free Space Management	30 Condition Variables code	40 File System Implementation	57 *Distributed*
	9 Lottery Scheduling code	18 Introduction to Paging	31 Semaphores code	41 Fast File System (FFS)	
	10 Multi-CPU Scheduling	19 Translation Lookaside Buffers	32 Concurrency Bugs	42 FSCK, and Journaling	**Appendices**
	11 *Summary*	20 Advanced Page Tables	33 Event-based Concurrency	43 Log-structured File System (LFS)	*Dialogue*
		21 Swapping: Mechanisms	34 *Summary*	44 Flash-based SSDs	Virtual Machines
		22 Swapping: Policies		45 Data Integrity and Protection	*Dialogue*
		23 Complete VM Systems		46 *Summary*	Monitors
		24 *Summary*		47 *Dialogue*	*Dialogue*
				48 Distributed Systems	Lab Tutorial
				49 Network File System (NFS)	Systems Labs
				50 Andrew File System (AFS)	xv6 Labs
				51 *Summary*	

Operating Systems: Three Easy Pieces (wisc.edu)

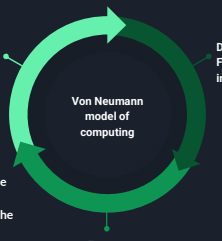
3

What OS do you know?

https://www.mentimeter.com/app/presentation/875b0f60af24de5c78dd812074fb8f54/bd75375ca296

4

What computer does when running a program?



After it is done with this instruction, the processor moves on to the next instruction, and so on and so on, until the program finally completes

John von Neumann - Wikipedia

5

Primary goal of OS (1)

- Sound Simple, right? But in this class we will be learning that while a program runs, a lot of other wild things are going on with the primary goal of making the system **easy to use.**
- There is a body of software that responsible for making it easy to run programs (allowing you to seemingly run many at same time), allowing programs to share memory, enabling programs to interact with devices, and others.
- The body of software is called **operating system (OS),** it is in charge of making sure that system operates correctly and efficiently in an easy-to-use manner.

6

Primary goal of OS (2)

- OS does through a general technique called **virtualization**, OS takes a physical resource (processor, or memory, or a disk) and transforms it into a more general, powerful, and easy-to-use **virtual** form of itself. Thus we sometimes refer OS as a virtual machine.
- In order to allow users to tell OS what to do and make use of the features of the virtual machine (running a program, or allocation memory, or accessing file), OS provides some interfaces (APIs) you can call. A few hundred **system calls**, are available to applications, we also sometimes called **standard library** to applications.

7

Primary goal of OS (3)

- Finally because Virtualization allows
  - Many programs to run (CPU sharing)
  - Many programs to concurrently access their own instruction and data (memory sharing)
  - Many programs to access devices (disk sharing and so forth)
- OS sometimes known as a resource manager. CPU, memory, disk is a resource of the system, OS's role to manage those resources efficiently or fairly or indeed with many other possible goals in mind.

8

Virtualizing CPU (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
#include "common.h"

int
main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];
    while (1) {
        Spin(1);
        printf("%s\n", str);
    }
    return 0;
}
```

Figure 2.1: **Simple Example: Code That Loops And Prints (cpu.c)**

```
prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
A
^C
prompt>
```

spin(), a function that repeatedly checks the time and return once it has run for a second

ostep-code/intro at master · remzi-arpacidusseau/ostep-code · GitHub

9

Virtualizing CPU (2)

```
prompt> ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 7353
[2] 7354
[3] 7355
[4] 7356
A
B
D
C
A
B
D
C
A
...
```

Figure 2.2: **Running Many Programs At Once**

Let's run many different instances of this same program. Even though we have only one processor, somehow all four of these programs seem to be running at the same time! How does this magic happen?

10

Virtualization CPU (3)

- In turns out that OS and some help from hardware, is in charge of this **illusion**, system has a very large number of virtual CPUs.
- Turning a single CPU into a seemingly infinite number of CPUs and thus allowing many programs to seemingly run at once in what we call **virtualizing the CPU**, the focus of the first major part of this course.
- Run programs, stop them, otherwise tell the OS which programs to run, there need some interfaces (APIs) that you can use to communicate your desires to OS.
- You might notice the ability to run multiple programs at once raises all sorts of new questions.
  - If two programs want to run at a particular time, which should run? Policy of the OS
  - We will study them as we learn about the basic mechanisms that OS implement (a resource manager)

11

Virtualizing Memory (1)

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int
main(int argc, char *argv[])
{
    int *p = malloc(sizeof(int));                   // a1
    assert(p != NULL);
    printf("(%d) address pointed to by p: %p\n",
           getpid(), p);                            // a2
    *p = 0;                                         // a3
    while (1) {
        Spin(1);
        *p = *p + 1;
        printf("(%d) p: %d\n", getpid(), *p);       // a4
    }
    return 0;
}
```

Figure 2.3: **A Program That Accesses Memory (mem.c)**

```
prompt> ./mem
(2134) address pointed to by p: 0x200000
(2134) p: 1
(2134) p: 2
(2134) p: 3
(2134) p: 4
(2134) p: 5
^C
```

Physical memory is just an array of bytes, to read memory, one must specify an address to access the data stored there, to write memory, one must also specify the data to be written to the given address. malloc() is a function to allocate memory.

12

Virtualizing Memory (2)

```
prompt> ./mem &; ./mem &
[1] 24113
[2] 24114
(24113) address pointed to by p: 0x200000
(24114) address pointed to by p: 0x200000
(24113) p: 1
(24114) p: 1
(24114) p: 2
(24113) p: 2
(24113) p: 3
(24114) p: 3
(24113) p: 4
(24114) p: 4
...
```

Figure 2.4: **Running The Memory Program Multiple Times**

The result shows that it allocated memory at the same address(0x200000) but each seems to be updating the value independently. Each running program has its own private memory, instead of sharing the same physical memory with other running programs. OS is virtualizing memory (Virtual address space) OS somehow maps onto the physical memory of the machine. Why ? who know, I have a reward for you

13

Concurrency(1)

```
#include <stdio.h>
#include <stdlib.h>
#include "common.h"
#include "common_threads.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        counter++;
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: threads <value>\n");
        exit(1);
    }
    loops = atoi(argv[1]);
    pthread_t p1, p2;
    printf("Initial value : %d\n", counter);

    Pthread_create(&p1, NULL, worker, NULL);
    Pthread_create(&p2, NULL, worker, NULL);
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("Final value   : %d\n", counter);
    return 0;
}
```

Figure 2.5: **A Multi-threaded Program (threads.c)**

14

Concurrency(2)

```
prompt> gcc -o thread thread.c -Wall -pthread
prompt> ./thread 1000
Initial value : 0
Final value   : 2000
```

```
prompt> ./thread 100000
Initial value : 0
Final value   : 143012    // huh??
prompt> ./thread 100000
Initial value : 0
Final value   : 137298    // what the??
```

15

Concurrency (3)

- As it turns out, the reason for these odd and unusual outcomes relate to how instructions are executed, which is one at a time. Unfortunately, a key part of the program above, where the shared counter is incremented, takes three instructions:
- one to load the value of the counter from memory into a register, one to increment it, and one to store it back into memory.
- Because these three instructions do not execute atomically (all at once), strange things can happen. It is this problem of concurrency that we will address in great detail in the second part of this course

16

Persistence(1)

- In system memory, data can be easily lost, as devices such as DRAM store values in a volatile manner; when power goes away or the system crashes, any data in memory is lost. Thus, we need hardware and software to be able to store data persistently; such storage is thus critical to any system as users care a great deal about their data.
- The hardware comes in the form of some kind of input/output or I/O device; in modern systems, a hard drive is a common repository for long lived information, although solid-state drives (SSDs) are making headway in this arena as well.
- The software in the operating system that usually manages the disk is called the **file system**; it is thus responsible for storing any files the user creates in a reliable and efficient manner on the disks of the system.

17

Persistence(2)

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <assert.h>
4  #include <fcntl.h>
5  #include <sys/types.h>
6
7  int main(int argc, char *argv[]) {
8      int fd = open("/tmp/file", O_WRONLY|O_CREAT|O_TRUNC,
9                    S_IRWXU);
10     assert(fd > -1);
11     int rc = write(fd, "hello world\n", 13);
12     assert(rc == 13);
13     close(fd);
14     return 0;
15 }
```

Figure 2.6: **A Program That Does I/O (`io.c`)**

18

Persistence(3)

- To understand this better, let's look at some code. Figure 2.6 presents code to create a file (/tmp/file) that contains the string "hello world".
- To accomplish this task, the program makes three calls into the operating system.
  - The first, a call to open(), opens the file and creates it;
  - the second, write(), writes some data to the file;
  - the third, close(), simply closes the file thus indicating the program won't be writing any more data to it.
- These system calls are routed to the part of the operating system called the file system, which then handles the requests and returns some kind of error code to the user.

19

Persistence(4)

- The file system has to do a fair bit of work: first figuring out where on disk this new data will reside, and then keeping track of it in various structures the file system maintains.
- Doing so requires issuing I/O requests to the underlying storage device, to either read existing structures or update (write) them.
- As anyone who has written a device driver knows, getting a device to do something on your behalf is an intricate and detailed process.
- It requires a deep knowledge of the low-level device interface and its exact semantics. Fortunately, the OS provides a standard and simple way to access devices through its system calls. Thus, the OS is sometimes seen as **a standard library**.

20

Design Goal(1)

- basic goals is to build up some abstractions in order to make the system convenient and easy to use.
- Abstractions are fundamental to everything we do in computer science. Abstraction makes it possible to write a large program by dividing it into small and understandable pieces, to write such a program in a high-level language like C without thinking about assembly, to write code in assembly without thinking about logic gates, and to build a processor out of gates without thinking too much about transistors.
- Abstraction is so fundamental that sometimes we forget its importance, but we won't here; thus, in each section, we'll discuss some of the major abstractions that have developed over time, giving you a way to think about pieces of the OS.

21

Design Goal(2)

- One goal in designing and implementing an operating system is **to provide high performance**;
- another way to say this is our goal is to minimize the overheads of the OS. Virtualization and making the system easy to use are well worth it, but not at any cost; thus, we must strive to provide virtualization and other OS features without excessive overhead.
- These overheads arise in a number of forms: extra time (more instructions) and extra space (in memory or on disk).
- We'll seek solutions that minimize one or the other or both, if possible. Perfection, however, is not always attainable, something we will learn to notice and (where appropriate) tolerate.

22

Design Goal(3)

- Another goal will be to provide **protection** between applications, as well as between the OS and applications.
- Because we wish to allow many programs to run at the same time, we want to make sure that the malicious or accidental bad behavior of one does not harm others; we certainly don't want an application to be able to harm the OS itself (as that would affect all programs running on the system).
- Protection is at the heart of one of the main principles underlying an operating system, which is that of isolation; isolating processes from one another is the key to protection and thus underlies much of what an OS must do

23

Design Goal(4)

- The operating system must also run non-stop; when it fails, all applications running on the system fail as well. Because of this dependence, operating systems often strive to provide a high degree of **reliability**.
- As operating systems grow evermore complex (sometimes containing millions of lines of code), building a reliable operating system is quite a challenge — and indeed, much of the on-going research in the field

24

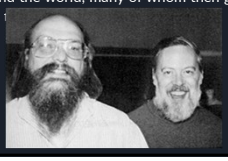
Design Goal(5)

- Other goals make sense: **energy-efficiency** is important in our increasingly green world;
- **security** (an extension of protection, really) against malicious applications is critical, especially in these highly-networked times; mobility is increasingly important as OSes are run on smaller and smaller devices.
- Depending on how the system is used, the OS will have different goals and thus likely be implemented in at least slightly different ways. However, as we will see, many of the principles we will present on how to build an OS are useful on a range of different devices.

25

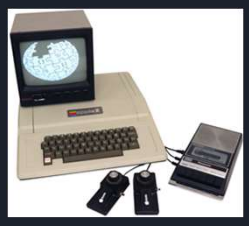
History of OS

One of the major practical advances of the time was the introduction of the UNIX operating system, primarily thanks to **Ken Thompson** (and Dennis Ritchie) at **Bell Labs** (yes, the phone company). UNIX took many good ideas from different operating systems (particularly from Multics, and some from systems like TENEX and the Berkeley TimeSharing System), but made them simpler and easier to use. Soon this team was shipping tapes containing UNIX source code to people around the world, many of whom then got involved and added to the system



26

Modern Era

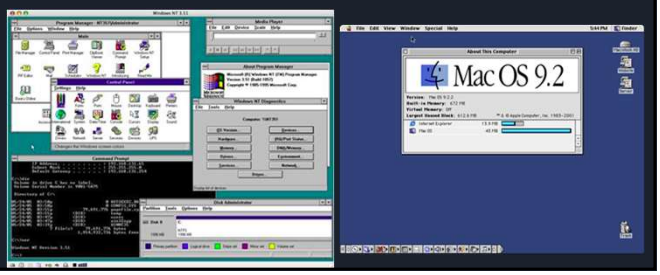


27

Microsoft DOS

28

Windows NT & Mac OS 9.2



29

Linus Torvalds the Linux Inventor

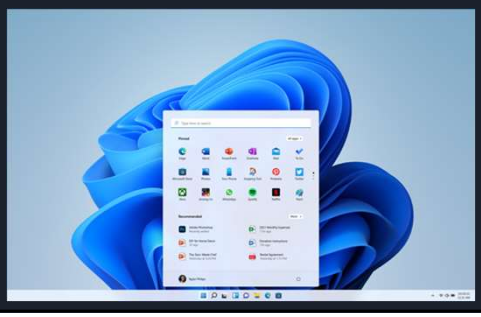
Linux Distributions Timeline (wikimedia.org)

30

Linux : Ubuntu

31

Windows 11

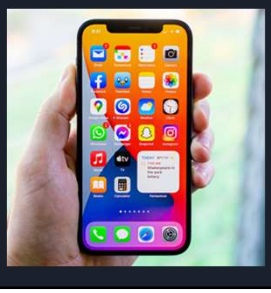


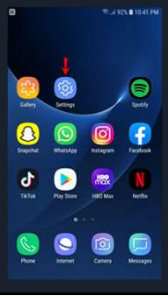
32

macOS Ventura

33

OS for mobile





34

Apple watch os

35

WSL & install Ubuntu on Window 11

Install Ubuntu on WSL2 on Windows 10 | Ubuntu

36

Lab notes: install git & git clone

```
korawit@DESKTOP-8OIT9FH:~$ git clone https://github.com/remzi-arpacidusseau/ostep-code.git
Cloning into 'ostep-code'...
remote: Enumerating objects: 312, done.
remote: Counting objects: 100% (72/72), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 312 (delta 57), reused 51 (delta 51), pack-reused 240
Receiving objects: 100% (312/312), 54.28 KiB | 1.70 MiB/s, done.
Resolving deltas: 100% (143/143), done.
korawit@DESKTOP-8OIT9FH:~$ dir
ostep-code
korawit@DESKTOP-8OIT9FH:~$ cd ostep-code/
korawit@DESKTOP-8OIT9FH:~/ostep-code$ cd intro
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ ls
Makefile  README.md  common.h  common_threads.h  cpu.c  io.c  mem.c  threads.c
```

Install Git | Atlassian Git Tutorial

37

Install make (1)

```
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ make

Command 'make' not found, but can be installed with:

sudo apt install make        # version 4.2.1-1.2, or
sudo apt install make-guile  # version 4.2.1-1.2
```

38

Install make (2)

```
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ sudo apt install make
[sudo] password for korawit:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  make-doc
The following NEW packages will be installed:
  make
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 162 kB of archives.
After this operation, 393 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/main amd64 make amd64 4.2.1-1.2 [162 kB]
Fetched 162 kB in 1s (123 kB/s)
Selecting previously unselected package make.
(Reading database ... 31836 files and directories currently installed.)
Preparing to unpack .../make_4.2.1-1.2_amd64.deb ...
Unpacking make (4.2.1-1.2) ...
Setting up make (4.2.1-1.2) ...
Processing triggers for man-db (2.9.1-1) ...
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$
```

39

Install gcc

```
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ sudo apt-get install gcc
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu cpp cpp-9 gcc-9 gcc-9-base libasan5 libatomic1 libbinutils
  libc-dev-bin libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0 libctf0 libgcc-9-dev libgomp1 libisl22 libitm1 liblsan0
  libmpc3 libquadmath0 libtsan0 libubsan1 linux-libc-dev manpages-dev
Suggested packages:
  binutils-doc cpp-doc gcc-9-locales gcc-multilib autoconf automake libtool flex bison gdb gcc-doc gcc-9-multilib
  gcc-9-doc glibc-doc
The following NEW packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu cpp cpp-9 gcc gcc-9 gcc-9-base libasan5 libatomic1 libbinutils
  libc-dev-bin libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0 libctf0 libgcc-9-dev libgomp1 libisl22 libitm1 liblsan0
  libmpc3 libquadmath0 libtsan0 libubsan1 linux-libc-dev manpages-dev
0 upgraded, 28 newly installed, 0 to remove and 0 not upgraded.
Need to get 28.5 MB of archives.
After this operation, 123 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu focal/main amd64 binutils-common amd64 2.34-6ubuntu1 [207 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal/main amd64 libbinutils amd64 2.34-6ubuntu1 [474 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal/main amd64 libctf-nobfd0 amd64 2.34-6ubuntu1 [47.0 kB]
```

40

make

```
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ make
gcc -o cpu cpu.c -Wall
gcc -o mem mem.c -Wall
gcc -o threads threads.c -Wall -pthread
gcc -o io io.c -Wall
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ dir
Makefile  README.md  common.h  common_threads.h  cpu  cpu.c  io  io.c  mem  mem.c  threads  threads.c
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$
```

41

cpu.c

```
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ ./cpu "A"
A
A
A
A
A
A
^C
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$
```

42

Run multiple cpu.c

```
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 5109
[2] 5110
[3] 5111
[4] 5112
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ A
B
C
D
A
B
C
D
B
A
C
D
B
A
C
```

43

How to kill processes

```
korawit@DESKTOP-8OIT9FH:~$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 08:10 ?        00:00:00 /init
root         143       1  0 08:10 ?        00:00:00 /init
root         144     143  0 08:10 ?        00:00:00 /init
root         145     144  0 08:10 pts/0    00:00:01 /mnt/wsl/do
root         160     143  0 08:10 ?        00:00:00 /init
korawit      161     160  0 08:10 pts/1    00:00:01 docker serv
root        5086       1  0 15:16 ?        00:00:00 /init
root        5087    5086  0 15:16 ?        00:00:00 /init
korawit     5088    5087  0 15:16 pts/3    00:00:00 -bash
korawit     5109    5088 99 15:17 pts/3    00:00:26 ./cpu A
korawit     5110    5088 99 15:17 pts/3    00:00:26 ./cpu B
korawit     5111    5088 99 15:17 pts/3    00:00:26 ./cpu C
korawit     5112    5088 99 15:17 pts/3    00:00:26 ./cpu D
root        5113       1  0 15:17 ?        00:00:00 /init
root        5114    5113  0 15:17 ?        00:00:00 /init
korawit     5115    5114  0 15:17 pts/2    00:00:00 -bash
korawit     5128    5115  0 15:17 pts/2    00:00:00 ps -ef
korawit@DESKTOP-8OIT9FH:~$ kill -9 5109 5110 5111 5112
korawit@DESKTOP-8OIT9FH:~$
```

```
A
C
D
B
A
^C
[1]   Killed                  ./cpu A
[2]   Killed                  ./cpu B
[3]-  Killed                  ./cpu C
[4]+  Killed                  ./cpu D
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$
```

44

mem.c

```
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ ./mem 0
(5132) addr pointed to by p: 0x557b5231f2a0
(5132) value of p: 1
(5132) value of p: 2
(5132) value of p: 3
(5132) value of p: 4
(5132) value of p: 5
(5132) value of p: 6
(5132) value of p: 7
^C
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$
```

45

Run multiple mem.c

```
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ ./mem 0 & ./mem 0 &
[1] 5133
[2] 5134
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ (5133) addr pointed to by p: 0x55675e5752a0
(5134) addr pointed to by p: 0x55f6388002a0
(5134) value of p: 1
(5133) value of p: 1
(5134) value of p: 2
(5133) value of p: 2
(5134) value of p: 3
(5133) value of p: 3
(5133) value of p: 4
(5134) value of p: 4
(5134) value of p: 5
(5133) value of p: 5
(5133) value of p: 6
(5134) value of p: 6
(5134) value of p: 7
(5133) value of p: 7
(5134) value of p: 8
(5133) value of p: 8
^C
```

```
setarch $(uname --machine) --addr-no-randomize /bin/bash
```

46

threads.c

```
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ ./threads 1000
Initial value : 0
Final value   : 2000
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ ./threads 100000
Initial value : 0
Final value   : 143092
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ ./threads 100000
Initial value : 0
Final value   : 135903
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$
```

47

io.c

```
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ ./io
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$ cat /tmp/file
hello world
korawit@DESKTOP-8OIT9FH:~/ostep-code/intro$
```

48