

# CSC209 Week 9 Notes

Hyungmo Gu

May 17, 2020

## Signals 1 of 2

- Introduction to Signals

- Signals

- \* are mechanisms that allow process or the os to interrupt currently running process and notify that an event has occurred

No	Name	Default Action	Description
1	SIGHUP	terminate process	terminal line hangup
2	SIGINT	terminate process	interrupt program
3	SIGQUIT	create core image	quit program
4	SIGILL	create core image	illegal instruction
5	SIGTRAP	create core image	trace trap
6	SIGABRT	create core image	abort program (formerly SIGIOT)
7	SIGEMT	create core image	emulate instruction executed
8	SIGFPE	create core image	floating-point exception
9	SIGKILL	terminate process	kill program
10	SIGBUS	create core image	bus error
11	SIGSEGV	create core image	segmentation violation
12	SIGSYS	create core image	non-existent system call invoked
13	SIGPIPE	terminate process	write on a pipe with no reader
14	SIGALRM	terminate process	real-time timer expired
15	SIGTERM	terminate process	software termination signal
16	SIGURG	discard signal	urgent condition present on socket
17	SIGSTOP	stop process	stop (cannot be caught or ignored)
18	SIGTSTP	stop process	stop signal generated from keyboard
19	SIGCONT	discard signal	continue after stop
20	SIGCHLD		

- How it Works

1. Using hotkey

- \* i.e. *CTRL + C* in terminal sends SIGINT
- \* i.e. *CTRL + Z* in terminal sends SIGSTOP

2. Using kill command

```

>./dots
.....
[1]+  Stopped                  ./dots
>
.....

>ps aux |grep dots
Conline 3819 100.0 0.0 2432748 540 s002 R+ 10:59am 0:11.30 ./dots
Conline 3821 0.0 0.0 2440964 672 s003 S+ 10:59am 0:00.00 grep dots
>kill -STOP 3819
>kill -CONT 3819
>kill -INT 3819
>

```

```

1  >>>./signals_example_1.out # <- This is done in separate
    terminal
2  >>> ps aux | grep ./signals_example_1.out
3  >>> kill -STOP <PID>
4  >>> kill -CONT <PID>
5  >>> kill -INT <PID>
6

```

## Signals 2 of 2

- Signals Handling

- sigaction

- \* **Syntax:** `int sigaction(int signum, const struct sigaction *act, NULL);`
- \* Is a part of *signal.h* library
- \* Is used to change the action taken by a process on receipt of a specific signal
- \* Works like try and catch in Python
- \* Don't worry about NULL :). Not knowing won't bite.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <signal.h>
4
5  void handler(int);
6
7  int main () {
8      struct sigaction newact;
9      newact.sa_handler = handler; // <- like catch statement in
    python
10     newact.sa_flags = 0;
11     sigemptyset(&newact.sa_mask);

```

```

12     return(0);
13 }
14
15 void handler(int code) {
16     fprintf(stderr, "Signal %d caught\n", code);
17 }
18

```

- \* Use *CTRL + Z* to terminate
- \* *kill -KILL <PID>* and *kill -QUIT <PID>* are two guaranteed ways to terminate a program.

## Bit Manipulation 1 of 4

- Introducing Bitwise Operations
  - When to use Bitwise Operations?
    - \* Lowlevel programming on embedded systems
  - Bitwise Operators in C
    - \* **&**: AND

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

### Example:

```

1      0   1   1   1   //<- this is 7
2      0   1   0   0   //<- this is 4
3      -----
4      0   1   0   0   //<- this is 4
5
6      so, 7 & 4 = 4
7

```

- \* **|**: OR

a	b	a   b
0	0	0
0	1	1
1	0	1
1	1	1

**Example:**

```

1      0   1   1   1   //<- this is 7
2      0   1   0   0   //<- this is 4
3      -----
4      0   1   1   1   //<- this is 7
5
6      so, 7 | 4 = 4
7

```

\* : NOT

a	$\sim a$
0	1
1	0

**Example:**

```

1      0   1   1   1   //<- this is 7
2      -----
3      1   0   0   0   //<- this is 8
4
5      so, ~ 7 = 8
6

```

\* ^ XOR

a	b	$a \wedge b$
0	0	0
0	1	1
1	0	1
1	1	0

**Example:**

```

1      0   1   1   1   //<- this is 7
2      0   1   0   0   //<- this is 4
3      -----
4      0   0   1   1   //<- this is 3
5
6      so, 7 ^ 4 = 3
7

```

## Bit Manipulation 2 of 4

- Hexadecimal Numbers

- Starts with '0x' at front
  - '0x' is for uncapitalized letters, i.e. '0xFFFF'
  - '0X' is for capitalized letters, i.e. 0xffff
- Uses 10 symbols '0, 1, 2, 3, 4, 5, 6, 7, 8, 9' and 6 extras ' $A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$ '.
  - i.e.  $0xFFFF = 15 \cdot 16^0 + 15 \cdot 16^1 + 15 \cdot 16^2 + 15 \cdot 16^3 + 15 \cdot 16^4 = 65535$

- The Shift Operators

- $\ll n$ : LEFT SHIFT
  - Shifts all bits to left by  $n$

**Example:**

```

1 i = 7
2 j = i << 1
3
4 0   1   1   1   //<- this is 7
5 -----
6 1   1   1   0   //<- this is 14
7
8 so, 7 << 1 = 14

```

- $\gg n$ : RIGHT SHIFT
  - Shifts all bits to right by  $n$

**Example:**

```

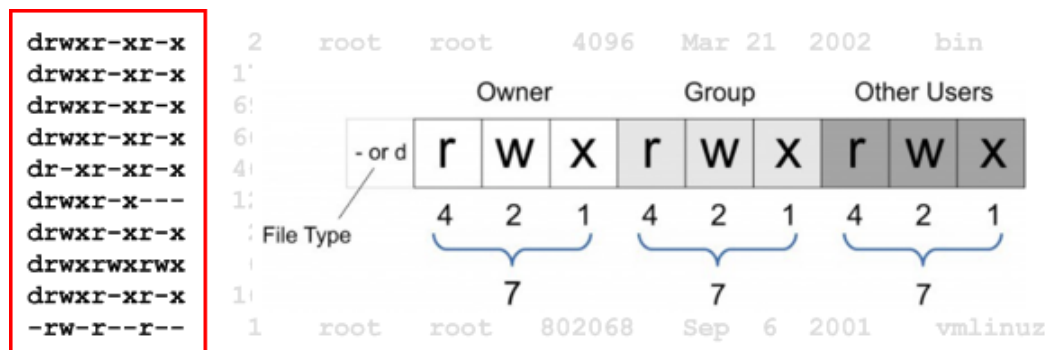
1 i = 7
2 j = i >> 1
3
4 0   1   1   1   //<- this is 7
5 -----
6 0   0   1   1   //<- this is 3
7
8 so, 7 >> 1 = 3

```

## Bit Manipulation 3 of 4

- Bit Flags

- Bit flags are boolean variables represented using 0 and 1
  - \* bool variables consume 1 byte (8 bits)
  - \* 0 or 1 consume 1 bit
- Use of Bit Flags
  - \* Embedded Software Programming
  - \* Graphics card
- Example
  1. chmod and unix file system



#	Permission	rwX	Binary
7	read, write and execute	rwX	111
6	read and write	rw-	110
5	read and execute	r-X	101
4	read only	r--	100
3	write and execute	-wX	011
2	write only	-w-	010
1	execute only	--X	001
0	none	---	000

1. Setting read-only permission to a file to owner

```

1  r-----
2  100000000 // <- Binary
3  [4] [0] [0] // <- Octal
4
5  chmod 400 <file>
6

```

2. Setting read and write only permissions to a file

```

1  rw-rw-rw-
2  110110110 // <- Binary
3  [4+2=6] [4+2=6] [4+2=6] // <- Octal
4
5  chmod 666 <file>
6

```

### 3. Setting all permissions to a file

```

1  rwxrwxrwx
2  110110110 // <- Binary
3  [4+2+1=7] [4+2+1=7] [4+2+1=7] // <- Octal
4
5  chmod 777 <file>
6

```

- Octal Bit Flags in C

- Are written with preceding 0

04 == 100	read
02 == 010	write
01 == 001	execute

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdio.h>
4
5  #define S_IRUSR 0000400 //R for owner
6  #define S_IRGRP 0000400 //R for group
7  #define S_IROTH 0000400 //R for others
8
9  int main () {
10     mode_t mode = S_IRUSR | S_IRGRP | S_IROTH; // Example 1
11     mode_t mode = 0400 | 040 | 004; // Example 2 (<- Notice
    bitwise or is used)
12     mode_t mode = 0444; // Example 3
13     return(0);
14 }

```

## Bit Manipulation 4 of 4

- Bit array

- Reduces usage of space

- \* Adding element  $\rightarrow$  Bitwise OR

- 1

- 2

- \* Removing element  $\rightarrow$  Bitwise AND

- Bit Masking

- is a technique to add/remove specific element in bit array

- Adding element to Bit Array

- Removing element to Bit Array