CSC209 Week 9 Notes

Hyungmo Gu

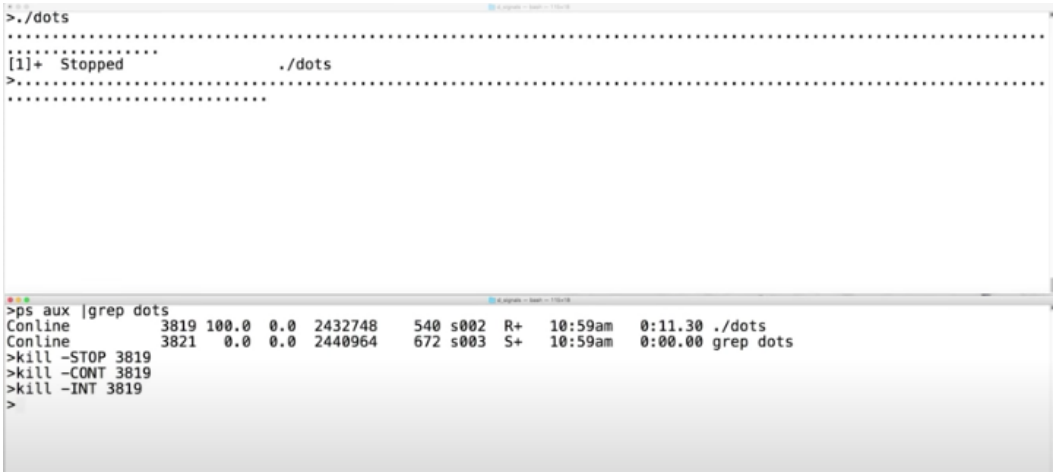
August 31, 2020

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 occured

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




```
>>>./signals_example_1.out # <- This is done in separate terminal
>>> ps aux | grep ./signals_example_1.out
>>> kill -STOP <PID>
>>> kill -CONT <PID>
>>> kill -INT <PID>

```

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.

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

void handler(int);

int main () {
    struct sigaction newact;
    newact.sa_handler = handler; // <- like catch statement in python
    newact.sa_flags = 0;
    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 guarenteed 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	1
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:

```
    0   1   1   1   //<- this is 7
    0   1   0   0   //<- this is 4
    --------------
    0   1   1   1   //<- this is 7

    so, 7 | 4 = 4

```

* ~ : NOT

a	~ a
0	1
1	0

Example:

```
    0   1   1   1   //<- this is 7
    --------------
    1   0   0   0   //<- this is 8

    so, ~ 7 = 8

```

* ^ : XOR

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

Example:

```
    0   1   1   1   //<- this is 7
    0   1   0   0   //<- this is 4
    --------------
    0   0   1   1   //<- this is 3

    so, 7 ^ 4 = 3

```

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

    Example:

```
i = 7
j = i << 1

0   1   1   1   //<- this is 7
--------------
1   1   1   0   //<- this is 14

so, 7 << 1 = 14
```

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

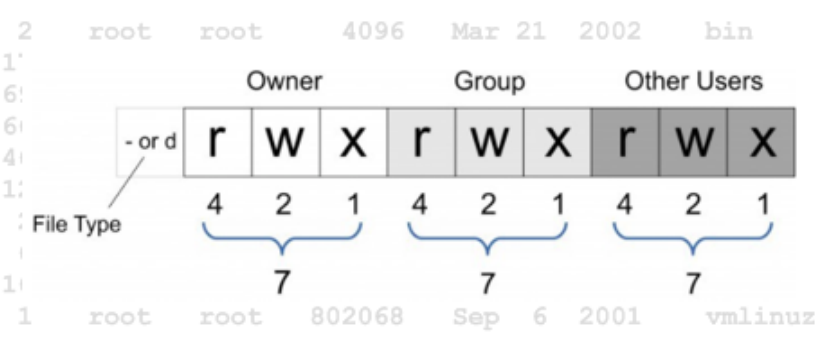
    Example:

```
i = 7
j = i >> 1

0   1   1   1   //<- this is 7
--------------
0   0   1   1   //<- this is 3

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

```
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
dr-xr-xr-x
drwxr-x---
drwxr-xr-x
drwxrwxrwx
drwxr-xr-x
-rw-r--r--
```



#	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

```
r---------
100000000 // <- Binary
[4]   [0]   [0]    // <- Octal

chmod 400 <file>

```

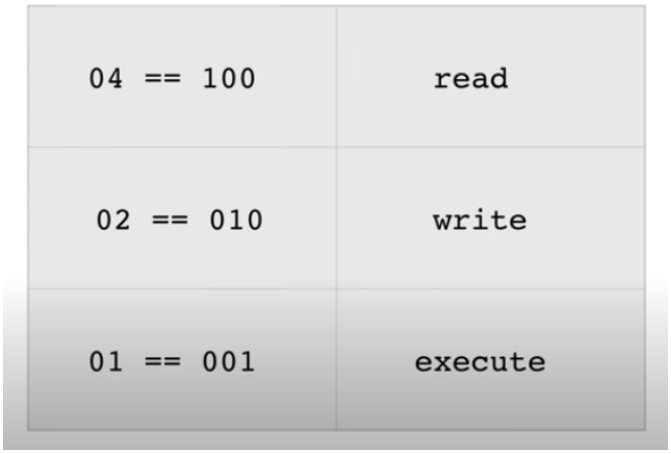
2. Setting read and write only permissions to a file

```
    rw-rw-rw-
    110110110 // <- Binary
    [4+2=6] [4+2=6] [4+2=6]  // <- Octal

    chmod 666 <file>

```

3. Setting all permissions to a file

```
    rwxrwxrwx
    110110110 // <- Binary
    [4+2+1=7] [4+2+1=7] [4+2+1=7]  // <- Octal

    chmod 777 <file>

```

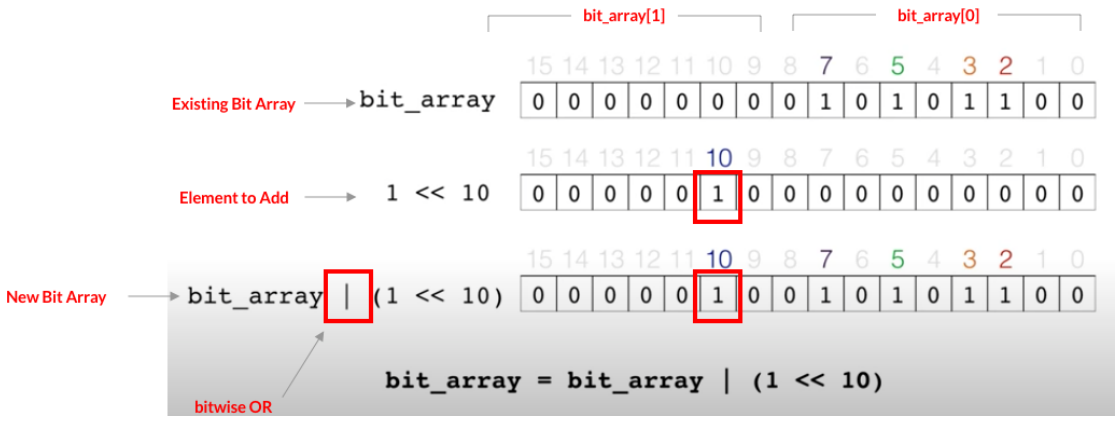
- Octal Bit Flags in C
  - Are written with preecding 0



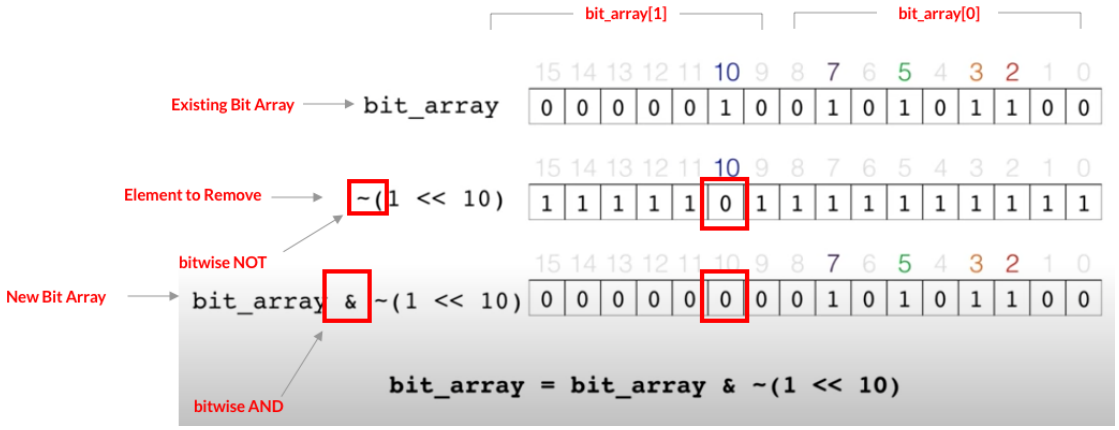

```c
#include <stdio.h>
#include <unistd.h>
#include <stdio.h>

#define S_IRUSR 0000400 //R for owner
#define S_IRGRP 0000400 //R for group
#define S_IROTH 0000400 //R for others

int main () {
    mode_t mode = S_IRUSR | S_IRGRP | S_IROTH; // Example 1
    mode_t mode = 0400 | 040 | 004; // Example 2 (<- Notice
   bitwise or is used)
    mode_t mode = 0444; // Example 3
    return(0);
}
```

Bit Manipulation 4 of 4

- Bit array
  - Reduces usage of space
    - Traditional array uses 32 bits per slot
    - Bit array uses 1 bits per slot
- Bit Masking
  - is a technique to add/remove specific element in bit array
  - **Adding element** → Bitwise OR



  - **Removing element** → Bitwise AND



  - General Rule
    1. Create Bit array

```c
#define N 4

unsigned bit_array[N];

```

2. Determine target position k of element in unsigned array

```
    int i = k/N;

```

3. Determine which bit in element to modify

```
    int pos = k%N;

```

4. Complete by performing Bitwise OR/AND

```
    flag = 1 << pos; // <- ~(1 << pos) if performing Bitwise
   AND
    bit_array[i] = bit_array[i] | flag;

```

5. More details: http://www.mathcs.emory.edu/ cheung/Courses/255/Syllabus/1-C-intro/bit-array.html

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

#define INTSIZE 32
#define N 4

typedef struct bits {
    unsigned int field[N];
} Bitarray;

int setzero(Bitarray *b) {
    return (memset(b, 0, sizeof(Bitarray)) == NULL);
}

void set(unsigned int value, Bitarray *b) {
    int index = value / INTSIZE;
    b->field[index] |= 1 << (value % INTSIZE);
}


void unset(unsigned int value, Bitarray *b) {
    int index = value / INTSIZE;
    b->field[index] &= ~(1 << (value % INTSIZE));
}

int ifset(unsigned int value, Bitarray *b) {
    int index = value / INTSIZE;
    return (1 << (value % INTSIZE) & b->field[index]);
}


int main () {
```

```
37          Bitarray a1;
38          setzero(&a1);
39
40          // Add 1, 16, 32, 65 to the set
41          set(1, &a1);
42          set(16, &a1);
43          set(32, &a1);
44          set(65, &a1);
45
46          // expecting: [0x00010002, 0x00000001, 0x000000010, 0]
47          printf("%x %x %x %x\n",
48              a1.field[0], a1.field[1], a1.field[2], a1.field[3]);
49      }
50
51
```

Listing 1: bit_manipulation_example_4.c