CSC209 Week 9 Notes

Hyungmo Gu

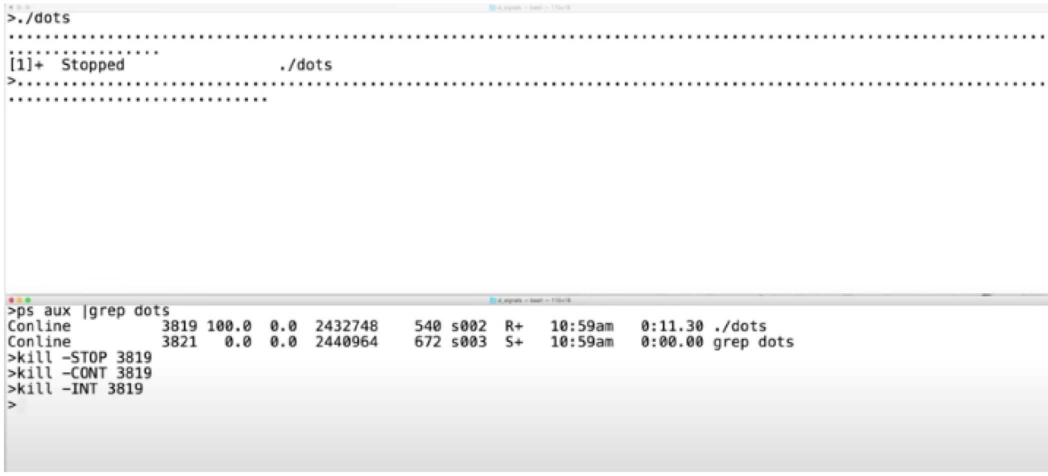
May 17, 2020

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 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.

```
#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);
```

```
        return(0);
    }

    void handler(int code) {
        fprintf(stderr, "Signal %d caught\n", code);
    }

```

- 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:

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

    so, 7 & 4 = 4

```

- **|:** 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 ˆ 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
    * Bit Flags are clean and fast :)
  - Use of Bit Flags
    * Embedded Software Programming
    * Graphics card
  - Example
    1. chmod and unix file system

```
drwxr-xr-x   2  root  root     4096  Mar 21  2002   bin
drwxr-xr-x  17  root  root    77824  Aug 11 14:40   dev
drwxr-xr-x  69  root  root     8192  Sep 25 18:15   etc
drwxr-xr-x  66  root  root     4096  Sep 25 18:15   home
dr-xr-xr-x  46  root  root        0  Aug 11 10:39   proc
drwxr-x---  12  root  root     4096  Aug  7  2002   root
drwxr-xr-x   2  root  root     8192  Mar 21  2002   sbin
drwxrwxrwx   6  root  root     4096  Sep 29 04:02   tmp
drwxr-xr-x  16  root  root     4096  Mar 21  2002   usr
-rw-r--r--   1  root  root   802068  Sep  6  2001   vmlinuz
```