CS 35L Software Construction Laboratory (Lab1-A)

Mon, Sept 26, 2011

Course Information:

Course Web: http://cs.ucla.edu/classes/fall11/cs35L/
Piazza: http://piazza.com/class#fall2011/cs35l/
Instructor in charge: Paul Eggert, Boelter 4532J.
Office hours: Mondays 11:55–12:55 and Tuesdays 10:30–11:30.
Instructor: Jiwen Cai (jwcai@cs.ucla.edu)
Tentative Office hours: Tuesday and Thursday 16:30-17:30 in Boelter 2432
Prerequisite: CS 31

What is CS35?

This course leads you to train yourself as a professional and efficient software engineer.

In this course, we will cover:

Basic linux commands, vim, emacs, man, shell scripting, python, make, automake, version control (diff, git, svn), gdb, valgrind, and etc.

Grading:

50% homeworks and in-class presentations
10 homesworks, each homeworks contains two parts: 1) Lab 2) Assignment
We will go through the lab in class and leave assignments to you.
The deadline of the 1st homework is Sept 30, 2011 (this Friday!!!).
Lateness penlty: 1 day late: 1%, 2 days late: 2%, 3 days late: 4%, ...
Hour code: you may discuss in form of study group, but no copying from others

50% final exam
Final exam: December 08, 2011 3:00 PM - 6:00 PM, no makeup.

Today's plan:

Introduction to several basic linux commands.
Learning Vim
Walking through part of Lab 1

Wednesday's plan:

Introduction to file attributes
More Linux commands
Learning Emacs
Finishing Lab 1

Unix and Linux:

Read wiki by yourslef:
http://en.wikipedia.org/wiki/Unix
http://en.wikipedia.org/wiki/Linux

CLI (Command Line Interface) vs GUI (Graphic User Interface)

Steep learning curve
Speed with commands
Low resources usgae
Power of scripting
Convenient remote access

Linux File System

Starts from root

Tree structured hierarchy

Command 1: ls -- list directory contents

Command 2: cd -- change directory

Command 3: pwd -- print name of current/working directory

Helper Commands:

Command 4: man -- an interface to the on-line reference manuals

Command 5: which -- locate a command

Command 6: whereis -- locate the binary, source, and manual page files for a command

Sometimes, just try [command] --help or [command] -h

Command is CLI programs with arguments:

man, ls, cd, and etc are actually programs, i.e. executable binary files

$PATH environment variable tells the system where to find such programs

echo $PATH -- view the current path

Vim Level 1: Survival

Command mode and Insert Mode

i	switch to insert mode before the current position
a	switch to insert mode after the current position (append)
I	jump to the first non-blank character in the current line and switch to the insert mode
A	jump to the last character of the current line and switch to the insert mode
x	delete one character
:wq	save and exit (:w save, :q exit)
:q!	exit without saving
hjkl	move cursor

Vim Level 2: Feeling good

o	insert new line below the cursor
O	insert new line above the cursor
0	move to the begin of the line
^	move to the first non-blank character in the line
$	move to the end of the line
g_	move to the last non-blacnk character in the line
/	search for pattern
dd	delete current line
p	paste

Vim Level 3 and more:

Please read: http://blog.interlinked.org/tutorials/vim_tutorial.html

```
CS 35L Software Construction Laboratory (Lab1-B)
Wed, Sept 28, 2011

FAQ:
   How can I install Linux into my own laptop?
      Two opitions: 1) Ask Linux working group for help
                    2) Use VirtualBox (http://piazza.com/class#fall2011/cs35l/4)

   How should I prepare lab1.log?
       "record each action in a file lab1.log"
       Such actions include: linux command you typed in CLI
                             commands in man pages
       You can also literally descripe what you did to figure out the answer,
       as long as someone else who reads your log file can reproduce the answer.

Linux file ownership:
    Files and directories are owned by a user
    Files and directories are assigned a group

Linux file attributes (10 bits):
    First bit: file type
        -   normal file
        d   directory
        l   denotes a symbolic link
    the rest 9 bits: "Three groups of three"
        first   what the owner can do
        second  what the group members can do
        third   what other users can do
    The triplet:
        first   r: readable.
        second  w: writable.
        third   x: executable.
    9 bits can be translated into a group of thre digitals
        rwx rwx rwx -> 111 111 111 -> 777
        rw- r-- r-- -> 110 100 100 -> 644
    command: chmod -- change file mode bits
        chmod 644 "filename"
        chmod [''references''][''operator''][''modes''] ''filename''
        references: u (owner), g (group), o (other), a (all)
        operator:   +(add), -(remove), = (set)
        modes:      r, w, x, s

Useful Linux commands:
    mkdir -- make directories
    rmdir -- remove empty directories
    cp -- copy files and directories
    rm -- remove files or directories
    mv -- move (rename) files
    apropos -- search the whatis database for string
    readlink -- print value of a symbolic link or canonical file name
    find -- search for files in a directory hierarchy
    locate -- find files by name
    top -- display Linux tasks
    ps -- report a snapshot of the current processes
    kill -- send a signal to a process (The  default signal for kill is TERM)

Learning Emacs:
    "mode" in Emacs is different than "mode" in Vim
    "Emacs is a good operating system, but it lacks a good editor"
    Combination of keys:
        Ctrl-x --> C-x
        Meta-x --> M-x (Meta key: second ctrl)

    C-h k    helper for key sequence
    C-h F    helper for command
    C-g      cancel command

    C-x C-c  exit Emacs
    C-x C-f  open file

    Move in Emacs
    C-n      next line
    C-p      previous line
    C-a      begin of line
    C-e      end of line
    C-f      move forward for one character
    C-b      move backward for one character
    M-f      move forward for one word
    M-b      move backward for one word
    NOTE: You can also move cursor in ternimal using such keys.
          (Maybe you need set meta-key in ternimal first)
    C-v      page down
    M-v      page up

    Search in Emacs
    C-s      incremental search

    Undo and Redo in Emacs
    C-x u    undo and redo, use C-g to change direction

    Cut, Copy, and Paste
    C-d      delete/cut/kill one character
    M-d      delete/cut/kill one word
    C-k      delete/cut/kill line
    M-k      delete/cut/kill sentence
    C-w      delete/cut/kill region
    M-w      copy region
    C-SPC    set mark to select region, like visual mode in Vim
    C-y      paste/yank the last stretch of killed text
    M-y      Replace just-yanked stretch of killed text with a different stretch
```

CS 35L Software Construction Laboratory (Lab2-A)
Mon, Oct 3, 2011, Ver 1.1

How to search package in apt?

```
    apt-cache search <package-name>
```

Linux file attributes: r w x s

s: The set-user-ID-on-execution and set-group-ID-on-execution bits.
This causes any persons or processes that run the file to have access to system resources as though they are the owner of the file

```
    -rws------ 1 root root 14024 Sep 9 1999 secret
    -rwxr-xr-x 1 root users 12072 Sep 9 1999 test
```

Any user in mail group can access file "secret" via test program, but no one rather the root can open secret directly.

Copy files from remote server

```
    command scp [[user@]host1:]file1 ... [[user@]host2:]file2
      eg: scp your_seas_username@lnxsrv.seas.ucla.edu:/usr/share/dict/words .
```

Command Redirection

```
    >: write stdout to a file (NOTE: this will overwrite an existing file)
    >>: append stdout to a file
    <: use contents of a file as stdin
    NOTE: stdout: standard output, (eg) printf("hello world\n");
          stdin:  standard input
          http://en.wikipedia.org/wiki/Standard_streams
```

Command Pipeline

```
    command_1 | command_2 | command_3
    NOTE: redirect the output of the first tool to the input of the following one
          eg: ls | less, ls -l | grep Oct
```

Basic Regular Expression

A regular expression, often called a pattern, is an expression that specifies a set of strings.
http://en.wikipedia.org/wiki/Regular_expression

```
    .    Matches any single character
    [ ]  Matches a single character that is contained within the brackets
         eg. [abc] [a-z] [a-zA-Z]
    *    Matches the preceding element zero or more times
         eg. ba* matches "b", "ba", "baa", etc
    ?    Matches the preceding element zero or one time
         eg. ba? matches "b" or "ba".
    +    Matches the preceding element one or more times
         eg. ba+ matches "ba", "baa", "baaa", and so on.
```

Shell Scripting

The first line starting with #! (shebang line or hashbang line)
http://en.wikipedia.org/wiki/Shebang_(Unix)
Tells the system which interpreter to interpret and execute the script.
Makes shell scripts more like real excutable programs.

```
      eg. #! /bin/sh
          #! /usr/bin/python
```

Variables

In command line: export A=23
In scripts "export" can be omitted.
Refer a variable: $A
Variables hold string values.
Quotation mark's function is to link two words as one.
Output: "echo" or "printf"

Sample code 1:

```
# /bin/bash
sum=0
i=0
while (( $i <= 10 ))
do
    let sum=$sum+$i
    let i=$i+1
    echo $sum
done
```

Sample code 2:

```
#! /bin/bash
VALID_PASSWORD=abcd1234
echo "Please enter the password:"
read PASSWORD
if [ $PASSWORD == $VALID_PASSWORD ]; then
        echo "You have access!"
else
        echo "ACCESS DENIED!"
fi
```

More Linux Commands to learn:

```
  tr   -- transliterate files with a pattern
  sort -- sort lines of text files
  head -- display first lines of a file
  tail -- display the last part of a file
  comm -- select or reject lines common to two files
  cmp  -- compare two files byte by byte
  ln   -- make links
  grep -- print lines matching a pattern
```

CS 35L Software Construction Laboratory (Lab2-B)
Wed, Oct 5, 2011, Ver 1.0

Shell Scripting (Continued)

How to execute a shell script?

Two options:

1) Make the file executable by adding "+x" attribute (chmod +x file_name)
2) Call the bash interpreter directly (bash file_name)

Running in the background

& at the end of the command/line of code

Shell doesn't wait for the command to finish if the program is running in the background.

Shell parameters

The first parameter to the shell is known as $1, the second as $2, etc.
The collection of ALL parameters is known as $*.
Sample code 3:

```
#! /bin/bash
printf "the first parameter is: %s\n" $1
printf "the second parameter is: %s\n" $2
printf "echo the collection of ALL parameters is: %s\n" $*
```

Note:
There is something *WRONG* within the piece of code shown above.

More Linux Commands

grep: [g]local [r]egular [e]xpression [p]print

-- print lines matching a pattern
http://www.panix.com/~elflord/unix/grep.html

```
eg 1. cat file_1.txt | grep set
          print out lines with sring "set" in file_1.txt
eg 2. ls -l | grep 'o'
          print out files or directories whose name contains character o
eg 3. ps ax | grep chrome
          print out processes whose name contains the string "chrome"
```

sed -- Read and modify the input line by line

http://en.wikipedia.org/wiki/Sed
http://www.grymoire.com/Unix/Sed.html
Search and replace using sed

option: -n, --quiet, --silent
suppress automatic printing of pattern space

Pick out line using line number

```
eg 1. cat sample.txt | sed -n 1p
          print out the same line
eg 2. cat sample.txt | sed -n 1~2p
                               (first~step)
          print all the odd-numbered lines in the input stream
```

Search and replace

NOTE: The input could be a file or standard input (stdin)

```
eg 1. sed s/bad/good/ < sample.txt (the content of a file as stdin)
eg 2. sed s/bad/good/ sample.txt (the file name as a parameter)
```

```
         eg 3. cat sample.txt | sed s/bad/good/ (using pipeline)
         NOTE: By doing this, it only replace the first occurence
         NOTE: Global replacement
               sed s/bad/good/g  -- make changes to every occurence
     NOTE: sed in Mac OS's behaviour is really different

cmp  -- Compare two files byte by byte
     option: -s  --quiet  --silent
             Output nothing; yield exit status only.
             Exit status is 0 if inputs are the same,
                            1 if different,
                            2 if trouble.
             NOTE: exit code can be accessed via two approaches
                   1) echo $?
                   2) in shell script, use 'if clause'
```

CS 35L Software Construction Laboratory (Lab3-A)
Mon, Oct 10, 2011, Ver 1.0

Tarball

```
  What is a tarball? -- commonly used to refer to a file which contains other
                        files. Tar program itself does not compress the files.
                        Actually, tar works with a compression program like gzip
                        to compress the file
  An example:
  ftp://ftp.gnu.org/gnu/coreutils/coreutils-7.6.tar.gz
  .tar extension is for the actual tarball
  .gz extension suggests that this tarball is compressed by gzip
  .tar.gz is equivalent to .tgz
  A quick extract command:
  tar vxfz coreutils-7.6.tar.gz
    v -- produce verbose output. (optional)
    x -- extract files from an archive.
    f -- read the archive from or write the archive to the specified file.
    z -- compress the resulting archive with gzip.
         in extract or list modes, this option is ignored.
  A quick compress command:
  tar vcfz coreutils-7.6.tar.gz coreutils-7.6
    Note: coreutils-7.6.tar.gz is the tarball file which will be created
          coreutils-7.6 is the directory which will be packed and compressed
  More reading:
  http://maketecheasier.com/install-software-from-a-tarball-in-linux/2009/06/25
```

Basic gcc/g++

```
  gcc is used for c and g++ is used for c++

    +-------------+  compile   +--------------+  link   +-----------------+
    | source code | ---------> | object files | ------> | target program  |
    +-------------+            +--------------+         +-----------------+

  -- one (.c) source file will generate one object file
     g++ -o kernel.o -c kernel.cc
     g++ -o gui.o -c gui.cc
  -- link (combine) multiple object files into one target program
     g++ -o program kernel.o gui.o
```

configure and make

```
    When you get source code from others, you can try the following set of commands
  to "install" that program:
    ./configure
    make
    sudo make install

    Note: -- "configure" file is a executable script which automatically generates
             Makefile
          -- "Makefile" contains a set of rules which specify how to derive the
             target file
```

-- When you run command "make", try "make -j2". With "-j2", the make program will create two parallel threads to speed up the compilation

```
    +--------------+                    +------+
    | ./ configure | ---> Makefile ---> | make |
    +--------------+                    +------+
```

A sample make file:

```
 +-----------------------------------------------+
  CC     = gcc
  CFLAGS = -g

  all: helloworld

  helloworld: helloworld.o
          # Commands start with TAB not spaces
          $(CC) $(LDFLAGS) -o $@ $^

  helloworld.o: helloworld.c
          $(CC) $(CFLAGS) -c -o $@ $<

  clean:
          rm -f helloworld helloworld.o
 +-----------------------------------------------+
```

More reading:

-- Make (wiki page)
http://en.wikipedia.org/wiki/Make_(software)

-- a tutorial of using autoconf and automake with C++:
http://www.openismus.com/documents/linux/automake/automake.shtml

New commands that you need to know:
1. tar 2. gcc 3. g++ 4. make

Preview Python: http://docs.python.org/release/2.4.1/tut/tut.html

CS 35L Software Construction Laboratory (Lab3-B)
Wed, Oct 12, 2011, Ver 1.0

How to use a Makefile

1. name it as default (Makefile) and run command "make" in the same directory.
2. "make -f OtherMakefile" (not recommanded)

It is too complicated to write Makefile by ourselves

  Different system need different system
  autoconf -- generate the "configure" script
  automake -- generate Makefile
  -- a tutorial of using autoconf and automake with C++:
    http://www.openismus.com/documents/linux/automake/automake.shtml

Install program without sudo

1. Generate a Makefile that will install the program into another directory

```
./configure --prefix=/tmp/cu
```

2. Build the source code again (run "make")
3. Run "make install" without sudo (you don't need it now)

$PATH environment variable

    Check you $PATH value:

```
jwcai@eagles:/$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

    When you type a command into the shell, the system will search it in
  directories listed in the $PATH.
    Modify the $PATH
    Option 1. export PATH=/tmp/cu/bin:$PATH
    Option 2. modify ~/.bashrc and append the export command.
  Check with "which" command

Python Reading: http://docs.python.org/release/2.4.1/tut/tut.html
    Ch1 ~ Ch7: must read, read the rest if you want more powerful tools

Python in Interactive mode

  Simply type "python" in the command line
  -- It is a good calculator

```
     e.g. >>> import math
          >>> math.sqrt ((3 + 4.5) ** 2)
          7.5
```

  -- It is a good helper for pyhton

```
     e.g. >>> help(open)
```

Revisit: Run a (Python) script

  Option 1: python myscript.py
  Option 2: #! line and chomd +x

Indent is important in Python

```
# something is wrong with the following code
def test():
    x = 0
  for i in range(0, 10):
    x += i
     print x
  test()

Build-in Data Structure in Python
  -- Basic type: int, float, string
  -- Tuple: immutable, (generally) sequences of different kinds of stuff
  -- List: mutable, (generally) sequences of the same kind of stuff
     Ref: http://news.e-scribe.com/397
  -- Access tuple and list using subscriptions
     a[1], a[1:], c[2:4], a[-1], a[:-1]
  -- Dict: key, value pairs
     e.g. d = {}
          d["tom"] = 20
          d["jerry"] = 19
     d.values(), d.keys(), d.items()
```

CS 35L Software Construction Laboratory (Lab3-A)
Mon, Oct 17, 2011, Ver 1.0

In-class presentation

Starting from next Monday (10/24/11), until the last day of class (11/30/11)
Up to 3 students are allowed to present each day
5 ~ 10 min for each talk
The presentation is weighted 5% in the final grade

Linux/Unix login process

1. Hardware power on, boot load program
2. Loading system kernel
3. Starting init process (PID 1)
4. Wating for user name

```
> login:
```

5. User providing their password

```
> password:
```

Note: step 4 and 5 depends on the file "/etc/passwd" which looks like this:

```
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
                 (home dir)    (shell app)
```

6. Start the shell app

NOTE: check your default shell app (echo $SHELL)
More reading: http://www.ucblueash.edu/thomas/Intro_Unix_Text/Process.html

Command "diff" revisited

```
diff file_1 file_2
```

Note: without any option, diff will give you normal outut
a stands for added, d for deleted and c for changed

```
diff -u file_1 file_2
```

Note: with -u option, you will get unified format output
Programs using this format: patch, git, svn

Unified format output

a diff file starts with two lines describe two files

```
--- test        2011-10-17 08:51:11.200077000 -0700
+++ test2       2011-10-17 08:51:30.086132000 -0700
```

a diff file contains one or more change hunks starting with

```
@@ -l,s +l,s @@
```

The first pair of l,s is for the origin file, and the second pair for the new file. l indicates the start line number while s is the number of lines in the hunk.

More reading: http://en.wikipedia.org/wiki/Diff

Command "patch"

apply a diff file to an original
usage:

```
1/  patch [options] [originalfile [patchfile]]
2/  patch -p0 <patchfile
```

Note: Read Manpage to understand what's the difference between -p0 and -p1

Lab 4 Guidline

- 1/ You will download and untar the Coreutil 8.0 like what we did last week.
- 2/ Create 2 diff files by "copy and paste"
- 3/ Use "patch" command to apply these 2 diff files. Note: last week you edit source code manually, but this week you are not allowed to do so.
- 4/ There is some errors in the patch, try the patch and figure out the problem
- 5/ Edit the patch (i.e. the diff file) using vim or emacs
- 6/ Try applying the patch again
  (Hint: pay attention to line numbers)

Software Version Control

- Common features of Revision Control
  - 1/ roll-back feature. (a snapshot of the source code)
  - 2/ synchronization between team members
  - 3/ source code distribution (e.g. github)
  - 4/ quality assurance (QA) and code review

More reading: http://en.wikipedia.org/wiki/Revision_control

Two tools we will cover in Thursday: git and subversion (a.k.a. svn)
git http://en.wikipedia.org/wiki/Git_(software)
svn http://en.wikipedia.org/wiki/Apache_Subversion

```
CS 35L Software Construction Laboratory (Lab4-B)
Mon, Oct 19, 2011, Ver 1.0

Basic Idea of Version Control
  -- Centralization vs Decentralization

git
  -- distributed control
  -- be free to commit and let merge solve problems
  -- push and pull between users
  -- all patches (commits) are identified by its hash code
  -- in most cases people need a rendevouz point (e.g. github)
  -- offline commit

--Global setup
  set up git
  git config --global user.name "Jiwen Cai"
  git config --global user.email jwcai@cs.ucla.edu
--Clone a repository from remote
  git clone git://github.com/goodcjw/cs35l.git
--View change logs in a git repository
  git log
--View change logs in GUI (may not available for all systems)
  gitk
--View change logs for file with diffs
  git log -p $file
--View changes to tracked files
  git diff
  git diff $id1 $id2
--Find out who is responsible for a file (piece of code)
  git blame $file
--View uncommited chagnes
  git status
--Create a repository locally
  git init
  touch README
  git add README
  git commit -am'first commit'
--Return to the last committed state
  git reset --hard
  Note: you cannot undo a hard reset, all changes will be lost
--Move to a previous version
  git checkout $id
  Note: Like a "time machine"
  Note: You can look around, make experimental changes and commit them,
        and you can discard any commits you make in this state without
        impacting any branches by performing another checkout.
--Work with remote git server
  git push
  git pull
  Note: We need to get access to a git server and configure remote server
  Check this out:
  http://help.github.com/create-a-repo/
--More reading
```

```
  http://www-cs-students.stanford.edu/~blynn/gitmagic/
  Focus on: push and pull, branch

--------------------------------------------------------------------------------

SVN (aka subversion)
  -- centralized control
  -- one server, multiple users
  -- one chunk, multiple branches
  -- unique continous version (revision) number
  -- commit to the server and update from the server
  -- need internet access to commit

--Work with remote svn server
  svn checkout $url
  svn commit
  svn update
--More reading
  Command comparison between git and svn: http://git.or.cz/course/svn.html
```

CS 35L Software Construction Laboratory (Lab5-A)
Mon, Oct 24, 2011, Ver 1.0

C Language

Father of C and Unix: Dennis Ritchie and Ken Thompson
Invented in 1970's, still the second popular programming language

More reading:
http://en.wikipedia.org/wiki/Dennis_Ritchie
http://en.wikipedia.org/wiki/Ken_Thompson
http://www.cplusplus.com/

Top 5 Programming Language

Rank 2011	Rank 2010	Language	Rate	Delta
1	1	Java	17.913%	-0.25%
2	2	C	17.707%	+0.53%
3	3	C++	9.072%	-0.73%
4	4	PHP	6.818%	-1.51%
5	6	C#	6.723%	+1.76%

More reading:
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

C Language Review

Include head files:
C libs: #include <xxx.h>
Std C++ libs: #include <xxx>
e.g. "#include <cstring>" vs "#include <string.c>" vs "#include <string>"

#include <stdio.h>

http://www.cplusplus.com/reference/clibrary/cstdio

```
-- printf   write to stdout
   int printf ( const char * format, ... );
   %[flags][width][.precision][length]specifier
-- scanf    read from stdin
   int  scanf ( const char * format, ... );
   %[*][width][modifiers]type
-- getchar  get character from stdin
   int getchar ( void );
-- putchar  put character to stdout
   int int putchar ( int character );
```

Review for pipeline and redirection

```
  -- redirect output:  echo '12345' > test
  -- redirect input:   cat < test
  -- pipeline:         echo '12345' | cat
```

#include <string.h>

http://www.cplusplus.com/reference/clibrary/cstring/

```
-- strlen   get string length
   size_t strlen ( const char * str );
-- strcmp   compare two strings
   int strcmp ( const char * str1, const char * str2 );
   return 0: both strings are equal
```

```
  -- memcpy   copy block of memory
     void * memcpy ( void * destination, const void * source, size_t num );
  -- strcpy   copy string
     char * strcpy ( char * destination, const char * source );

#include <stdlib.h>
  http://www.cplusplus.com/reference/clibrary/cstdlib/
  -- malloc   allocate memory block (run time)
     void * malloc ( size_t size );    // size in bytes
              e.g. char* mem = malloc(memsize);
  -- free     deallocate space in memory
     void free ( void * ptr );
  -- atoi     convert string to integer
     int atoi ( const char * str );
  -- qsort    sort elements of array
     void qsort ( void * base, size_t num, size_t size,
                  int ( * comparator ) ( const void *, const void * ) );
     -- Function pointer
        int ( * comparator ) ( const void *, const void * )
        int compare (const void *a, const void *b)

A quick demo for next lecture
  http://irl.cs.ucla.edu/~jwcai/qsort.c

More reading for gdb
  http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html
```

CS 35L Software Construction Laboratory (Lab5-B)
Wed, Oct 26, 2011, Ver 1.0

A tutorial: http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html
A quick demo: http://irl.cs.ucla.edu/~jwcai/qsort.c

Preparing your source code for debugging

```
  -- Add g command line argument when compile the source code using gcc/g++
  -- With a -g option, the output file will be larger and runs slower
  e.g. gcc -g qsort.c -o qsort
```

When to use a debugger

```
  -- To figure out how a program is executed
  -- Reproduce some bugs
  -- Print program stack after crash
```

Loading a program

```
  -- Simplest situation: gdb name_of_executable
     e.g. gdb ls
  -- When the executable needs some parameters
     e.g. gdb gdb --args ls -lt
  -- In emacs: M-x gdb (require very reliable source code)
```

GDB command line

```
  break -- shortened as 'b'
    -- set breakpoint on file and line number
       b qsort.c:5
    -- set breakpoint on function
       b qsort
  info breakpoints
  -- display all breakpoints

  list  -- shortened as 'l'
    -- print out the lines of code around the line the program is stopped at

  e.g.
  (gdb) info breakpoints
  Num Type           Disp Enb Address            What
  1   breakpoint     keep y   0x0000000100000cde in main at qsort.c:6
          breakpoint already hit 1 time
  3   breakpoint     keep y   0x0000000100000db2 in qsort at qsort.c:19
          breakpoint already hit 3 times
  (gdb) disable 1       -- Note: disable a breakpoint
  (gdb) delete 3        -- Note: detelte a breakpoint

  r   -- short for 'run'
  n   -- short for 'next', move on to the next statement
  s   -- short for 'step', step into one function call
  n N -- make N movements
  s N -- continue N steps
  c   -- short for 'continue', execute until next breakpoint
  RET   -- i.e. hit enter button (repeat next, step, or continue)

  print var -- print the value of a variable (one time)
```

e.g.

```
(gdb) print pivot
$4 = 105 'i'
```

```
disp  var -- display the value of a variable
             (every time the variable is in current scope)
watch var -- a smart break point
             break the execution one the variable chagnes
```

e.g.

```
(gdb) watch e
Hardware watchpoint 5: e
(gdb) c
Hardware watchpoint 5: e
Old value = 3
New value = 7
qsort (str=0x7fff5fbff890 "adfhisst", l=6, r=7) at qsort.c:21
1: s = 6
```

backtrace

-- display a listing of function calls from the beginning of execution

-- extremely useful for after-crash debugging

e.g.

```
(gdb) backtrace
#0  qsort (str=0x7fff5fbff890 "acelr", l=3, r=4) at qsort.c:24
#1  0x100000ec0 in qsort (str=0x7fff5fbff890 "acelr", l=0, r=4) at qsort.c:42
#2  0x100000d52 in main () at qsort.c:12
```

CS 35L Software Construction Laboratory (Lab6-A)
Mon, Oct 31, 2011, Ver 1.0

A brief introduction to Operating System

1. Process

A process is an instance of a computer program that is being executed.
It contains the program code and its current activity (memory).

http://en.wikipedia.org/wiki/Process_(computing)
http://www.comptechdoc.org/os/linux/usersguide/linux_ugprocesses.html

2. Kernel Mode and User Mode

Ring 0 (Kernel)
Ring 3 (Application)
Kernel mode has more privileges:
Unristricted mode. Full instruction availability. I/O operation, area of memory accessed are unlimitied.

http://en.wikipedia.org/wiki/Ring_(computer_security)

3. System Call

User processes ask the kernel to do things for them by invoking system calls.
-- software interrupt, with a system call number
When the user process invokes a system call, the processor enters kernel mode, the processor and the kernel cooperate to save the user process's state, and the kernel executes appropriate code in order to carry out the system call. When it's done, it resumes the user process.

-- Unix/Linux System Call

The system call number will go in %eax
The arguments (up to five of them) will go in %edx, %ecx, %ebx, %edi, and %esi, respectively.
The kernel passes the return value back in %eax

```
#include <sys/syscall.h>
syscall(SYS_ID, ...);
```

Note: ... is the list of parameters

%eax	name	%ebx	%ecx	%edx	%esx	%edi
3	SYS_read	unsigned int	char *	size_t	-	-
4	SYS_write	unsigned int	const char *	size_t	-	-

http://bluemaster.iu.hio.no/edu/dark/lin-asm/syscalls.html

5. File Descriptor

A file descriptor is an abstract indicator for accessing a file
3 standard POSIX file descriptors

Integer value	Name
0	Standard Input (stdin)
1	Standard Output (stdout)
2	Standard Error (stderr)

http://en.wikipedia.org/wiki/File_descriptor

-- Input/Ouput redirection REVISITED:

```
cat file1 > file2
cat file1 1 > file2
cat file1 1 > file2 2>&1
```

6. Wrapped System Call

```
#include <unistd.h>
ssize_t      read(int fd, void * buf, size_t sz);
ssize_t      write(int fd, const void * buf, size_t sz);

#include <sys/stat.h>
int          fstat(int fd, struct stat *buf);
```

http://pubs.opengroup.org/onlinepubs/009695399/functions/fstat.html

7. Tools for Lab 6

Using strace to test the results.

```
$ strace -o catb_trace ./catb < testfile
```

Using time to measure the speed.

```
$ time ./catb < testfile
```

CS 35L Software Construction Laboratory (Lab7-A)
Mon, Nov 7, 2011, Ver 1.0

Process Revisited

Process: Program + Data:
Memory of a precess can be devided into three regions:

```
1) Text: programs and read-only data
         (loaded from text section of a executable file)
2) Data: stores static variables
         (loaded from bss section of a executable file)
3) Stack: stores dynamic varialbles, function frames
```

Function Call

```
+-----------------------------+
 void main() {
   function(1, 2, 3);
 }
+-----------------------------+
pushl $3         # push arguments from right to left
pushl $2         # when they are poped out the order is left to right
pushl $1
call function    # jump to the address of the callee function
                 # call will push the current IP register (instruction pointer)
                 # into the stack.
                 # This is the RET address for the callee function
pushl %ebp       # EBP: the base pointer of the current function's stack
                 #      i.e. the start boundary of the current function
movl %esp,%ebp   # ESP: the stack pointer of the current function's stack
subl $20,%esp    #      i.e. the cutting edge of the current function
```

Buffer: a contiguous block of computer memory that holds multiple instances of the same data type.

-- static buffer: allocated at load time on the data segment.
-- dynamic: allocated at run time on the stack.

Buffer Overflow

the result of stuffing more data into a buffer than it can handle.
sample code:

```
+-----------------------------+
 void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
 }

 void main() {
   char large_string[256];
   int i;

   for( i = 0; i < 255; i++)
     large_string[i] = 'A';

   function(large_string);
 }
+------------------------------+
bottom of                                                          top of
memory                                                             memory
               buffer            sfp   ret   *str
<------       [                ][    ][    ][    ]

top of                                                          bottom of
stack                                                               stack
```

IMPORTANT: buffer overflow allows us to change the return address of a function
More Reading:
http://insecure.org/stf/smashstack.html

Getting start with Lab 7

1. Grab the tarball, untar it
2. Apply the patch given from the course website
3. Run "./configure", "make" and "make install"
   -- "make install" is optional, if you wanna install it, you may face some problem related to unexisted user and group.
4. Fix other bugs if necessary
5. If you simply run "./thttpd" it will not work.
6. Read MANPAGE of thttpd carefully. Especially, pay attention to these options:
   -C, specifies a config-file for thttpd
   -p, set port
   -d, set root directory
7. If you are still confused, you may find the following link helpful:
   http://www.acme.com/software/thttpd/notes.html
   You do not need to go through everything in that note page, but i helps to have a feeling about how an http server should be configured.

CS 35L Software Construction Laboratory (Lab7-B)
Wed, Nov 9, 2011, Ver 1.0

Another Example on Buffer Overflow

```
   +----------------------------------------------+
    void function(int a, int b, int c) {
      char buffer1[5];
      char buffer2[10];
      int *ret;

      ret = buffer1 + 12;
      (*ret) += 8;
    }

    void main() {
      int x;

      x = 0;
      function(1,2,3);
      x = 1;
      printf("%d\n",x);
    }
  +----------------------------------------------+
```

```
bottom of                                                          top of
memory                                                             memory
           buffer2       buffer1   sfp   ret   a     b     c
<------   [            ][         ][    ][    ][    ][    ][    ]

top of                                                          bottom of
stack                                                               stack
```

  Note: memory can only be addressed in multiples of the word size.
        A word in this example is 4 bytes, or 32 bits.

Makefile and GCC Revisited

In Lab 7, you are required to use GCC with several options, like "-s", "-fno-stack-protector". You are not going to run gcc command directly.

Instead, you can achieve this by modifying the Makefile, especially the "$CFLAGS" variable.

```
   +----------------------------------------------+
    CC     = gcc
    CFLAGS = -g

    all: helloworld

    helloworld: helloworld.o
            # Commands start with TAB not spaces
            $(CC) $(LDFLAGS) -o $@ $^

    helloworld.o: helloworld.c
```

```
            $(CC) $(CFLAGS) -c -o $@ $<

    clean:
            rm -f helloworld helloworld.o
   +----------------------------------------------+
```

"-S" option of gcc

Stop after the stage of compilation proper; do not assemble.

The output is in the form of an assembler code file for each non-assembler input file specified.

GDB Revisited

-- Attach GDB to a process which is running:

```
$ gdb
$ attach PID
```

Setup Environment for mudflap

(Thanks to Jihyoung "Joseph" Kim for sharing his notes on mudflap)

for Ubuntu

```
$ sudo apt-get install gcc-opt
$ sudo apt-get install libgcc1
$ sudo apt-get install libgcc1-dbg
$ sudo apt-get install libmudflap0
$ sudo apt-get install libmudflap0-dbg
$ sudo apt-get install libmudflap0-4.5-dev
```

on SEAS lab lnxsrv

```
$ bash
$ export PATH='/usr/local_cs/linux/bin'
$ export LD_LIBRARY_PATH=/usr/local_cs/linux/lib
```

available port for section 2: 12200-12228

More about Lab 7

-- Make sure your thttpd is working:

1) Try visit http://localhost:8080
2) Find help with commands: "ps" and "grep"

CS 35L Software Construction Laboratory (Lab8-A)
Mon, Nov 14, 2011, Ver 1.0

Three Dimensions of Computer Security -- CIA

```
  -- C: Confidentiality
     eavesdropping
  -- I: Integrity
     modification, fabrication
  -- A: Availability
     denial of service (DoS)
```

more reading:
http://en.wikipedia.org/wiki/Information_security
http://www.cert.org/tech_tips/denial_of_service.html
http://en.wikipedia.org/wiki/Man-in-the-middle_attack

Cryptography

  -- Symmetric-key cryptography

```
                    +-----------+
        Message --> | Algorithm | --> Encrypted Data
                    +-----------+
                          ^
                          |
                         Key
                          |
                          V
                    +-----------+
 Encrypted Data --> | Algorithm | --> Message
                    +-----------+
```

  -- Public-key cryptography (a.k.a. Asymmetric-key cryptography)

```
                     Public Key
                          |
                          V
                    +-----------+
        Message --> | Algorithm | --> Encrypted Data
                    +-----------+

                     Private Key
                          |
                          V
                    +-----------+
 Encrypted Data --> | Algorithm | --> Message
                    +-----------+

                     Private Key
                          |
                          V
                    +-----------+
        Message --> | Algorithm | --> Signature
                    +-----------+
                     Public Key
                          |
                          V
                   +-----------+
      Signature --> | Algorithm | --> Message
                   +-----------+
```

more reading:
http://en.wikipedia.org/wiki/Cryptography
http://en.wikipedia.org/wiki/Public-key_cryptography

Getting Started with SSH

-- install openssh if you do not have it

```
  sudo apt-get install openssh-server
```

-- generate SSH key pairs (client side)

```
  ssh-keygen -t rsa
  Get two files:
    .ssh/id_rsa     (private key)
    .ssh/id_rsa.pub (public key)
```

-- authorizing access (server side)

.ssh/authorized_keys

This file stores authorized pubkeys from client machines. Append other's pubkey to this file can authorize access without typing passwords.

-- other ssh cammands

Remote host shell access

```
  ssh login@remote
```

Execute a single command on a remote host

```
  ssh login@remote 'command'
```

Secure Copy

```
  scp login@remote:/remote/path/to/file /local/path/to/file
```

more reading: http://kimmo.suominen.com/docs/ssh/

CS 35L Software Construction Laboratory (Lab9-A)
Mon, Nov 21, 2011, Ver 1.0

Thread (aka Light Weight Process)

A thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system.

Thread vs Process

Process:

-- an instance of a program in execution
-- independent entity, system resources (CPU time, memory, etc.) are allocated
-- separate address space. one process cannot access the variables and data structures of another process
-- no processes can directly access the memory of another process
-- must use IPC (inter-process communication, including: pipes, files, sockets, memory share) to communicate with other processes

Thread:

-- a thread is a particular execution path of a process
-- one process can have multiple threads
-- threads in one process use the same memory address space
-- each thread has its own registers and its own stack, but other threads can read and write the stack memory
-- overall, the overhead brought from thread switch is lower than precess switch

More reading:
http://en.wikipedia.org/wiki/Thread_(computer_science)

Thread's overhead

1) switching: store and load registers
2) synchronization

Synchronization

-- a problem

```
+-----------------------------------------------------------------------+
| static cnt = 0;                                                       |
| static a = 0;                                                         |
+-----------------------------------+-----------------------------------+
|             Thread A              |             Thread B              |
+-----------------------------------+-----------------------------------+
| void counter() {                  | void accumulate() {               |
|   cnt++;                          |   a += cnt;                       |
| }                                 |   cnt = 0;                        |
|                                   | }                                 |
+-----------------------------------+-----------------------------------+
```

-- solution: mutex (mutual exclusive)

POSIX Threads (aka libpthread, -lpthread)

-- API defined by POSIX standard
-- pthread API can be informally grouped into four major groups:

1) Thread Management   2) Mutexes
3) Condition variables   4) Synchronization

```
// creates a new thread and makes it executable
int pthread_create(pthread_t * thread,
                   const pthread_attr_t * attr,
                   void * (*start_routine)(void *),
                   void *arg);
// wait for termination of another thread
int pthread_join(pthread_t th, void **thread_return);
```

More reading:
http://en.wikipedia.org/wiki/POSIX_Threads
http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

```
  +----------------------------------------------------------------------------+
    #include <pthread.h>
    #include <stdio.h>
    #include <stdlib.h>

    #define NUM_THREADS     4

    void* basic_task(void* arg) {
      int tid;
      tid = *((int *) arg);
      printf("hello world! this is thread %d!\n", tid);
      return NULL;
    }

    int main(void) {
      pthread_t threads[NUM_THREADS];
      int thread_args[NUM_THREADS];
      int rc, i;

      /* create all threads */
      for (i = 0; i < NUM_THREADS; ++i) {
        thread_args[i] = i;
        printf("in main: creating thread %d\n", i);
        rc = pthread_create(&threads[i], NULL,
                            basic_task, (void*) &thread_args[i]);
        if (rc != 0) {
          printf("fail to create thread, abort...\n");
          exit(1);
        }
      }

     /* wait for all threads to complete */
     for (i = 0; i < NUM_THREADS; ++i) {
       rc = pthread_join(threads[i], NULL);
       if (rc != 0) {
         printf("fail to join thread, abort...\n");
         exit(1);
       }
     }
     return 0;
   }
  +----------------------------------------------------------------------------+
```