cps104 recitation
Computer Organization and Programming
Info

About myself
- Alexandru Duțu
- alexdutu@cs.duke.edu
- http://www.cs.duke.edu/~alexdutu/

Slides are on course website

For questions visit piazza.com

For further questions email me or the professor

Even more questions attend office hours
How to start?

- Linux machines: teer[10-45].oit.duke.edu
- `scp -c blowfish <assignment>.tar <NetId>@teer[10-45].oit.duke.edu:~`
- `ssh -X -c blowfish <NetId>@teer[10-45].oit.duke.edu`
- `tar xvf <assignment>.tar`
UNIX/Linux

- ls, cd, man, cat, cp, mv, chmod, diff, df, du, env, find, gzip, lpq, lpr, mkdir, less, passwd, pwd, ps, rm, cal, users, who, whoami, date, grep, tar, ssh
- Editors (emacs, vi, vim, pico, nano)
- Compilers (gcc, icc, xl C/C++, IAR)
- Debuggers (gdb)
- Reverse engineering (objdump)
useful in Linux

- `~` - user’s home directory
- `.` - current directory
- `..` - one level up in the directory tree
- `./<program name>` - execute a program from the current directory that is not in $PATH environment variable
Editors

- emacs
- vi or vim
  - easy and intuitive shortcuts
  - configurable (lots of plugins)
  - in terminal and with GUI (gvim or macvim)
  - maintains Unix look and feel
  - http://www.cs.duke.edu/~alexdutu/others/vimrc
Using vim

- multiple modes
  - normal mode (command mode) - used for movement and for processing the text file
  - visual mode (v, V, ctrl-v) - like normal mode but the movement commands apply to extend highlight
  - insert mode (i, I, a, A) - used for text insertion
  - esc get you back to normal mode

- `<command>` - `:%s/<string>/*replacement*/`

- movement: h, l (left, right) j, k (down, up)

Useful plugins for vim

- NERDtree - gives a tree of the file system
- snipmate - provides snippets for an increased productivity
- taglist - displays “tags” within your current C source file
  - tags = function names, variables, type definitions etc.
  - it requires ctags to be installed (included in linux and unix distributions)
Making a program

- compilation (gcc)
  - assembly code

- assembling (as)
  - objects (.o or .obj, do not confuse with objects from OOP)
  - libraries (static or dynamic)

- linking (ld)
  - binaries
  - libraries

- gcc is directly producing the binary
compiling

- GNU compiler collection - gcc (http://gcc.gnu.org/)
- How to compile?
  - gcc -c example.c
- Is the binary ready?
- How to link?
  - gcc -o example example.o
Can we automate the process of binary creation?

`make` - GNU utility for managing multiple commands

`make files`
- rules - `<name>`: `<source files> <command>`
- macros - `<macro name> = <command>`
debugging

- `gdb <binary>` (gdb example)
  - but first produce debug symbols for that binary
  - use `-g` flag when you compile `gcc -g -c example.c`

- `next (n)`, `step (s)`

- `print (p) <variable name>`

- `breakpoints - break (b)`
  - `b <line number or function name>`
  - `b <file name>:<line number>`
  - `info b` - display all breakpoints

- `continue (c)` - continues execution until next breakpoint
reverse engineering

- fast disassembly - gdb disas
  - `gdb <binary name>`
  - `info line main` - returns the address of main function if the binary has a main function
  - `diasas <start address for main>`
  - you don’t need debug symbols for this

- objdump
  - `objdump -i`
  - `objdump -D <binary name> > binary.s`
  - complete disassembly: `objdump -Dslx <binary name> > binary.s`