CSE303: Concepts and Tools for Software Development

UW course catalog: Introduction to key concepts and tools in the development of software not introduced in the introductory programming courses. Includes programming with explicit memory management and layout (e.g. C or C++), techniques for group software development, modern design, implementation, and testing patterns and strategies, and societal impact.

David Notkin ● Autumn 2009

Groups ~2-3 people, ~2 minutes

- Jot down some of the "key concepts and tools in the development of software" that **were** introduced "in the introductory programming courses"

Concepts and tools in 303

- Unix/Linux operating system
- Basic C and C++ programming such as pointers, arrays, memory and resource management, preprocessor
- Programming tools such as debuggers, profilers, compilation managers, and version control
- Software engineering practices related to specification, partner programming, reuse, and testing
- Societal/ethical issues in and implications of computing

Why another OS? Language? Tool? Practice? Why ethics?

Concepts	Abilities	Skills
Procedures as an abstraction	Defining and invoking procedures	Defining and invoking procedures in {Java | C}
Looping	Defining loops Convert loops to recursion	Loops in {Java | C }
Program failure	Debugging	Debugging programs in language C or with a specific debugger
Dynamic memory use: creating objects, reusing space	Allocation and garbage collection vs. allocation and explicit deletion	Managing space with C and C++
…		

Not perfectly defined groups – it's the idea, not the details

And if you think education is the same as training…

Why ethics?

- As computer users, you have legal and policy obligations – Appropriate Use of UW Resources (protecting infrastructure, privacy, copyright policies, public records, …)
- As computer professionals, you have even deeper obligations – SE Code of Ethics and Professional Practice
  - Act consistently with the public interest; act in the best interests of clients and employers, consistent with public interest; maintain integrity and independence in professional judgment, …
- The fundamentals may not differ from other fields, but specifics due to computers are significant; also, computers and software are now a part of almost every complex system
  - *As society becomes more dependent on computing, we have greater societal and ethical and legal obligations*

OS's & languages & compilers, oh my!

- …& run-time systems & editors & environments & debuggers & …?

"Toto, we're not in 142/143 anymore"

- Other programs are needed to run our program
- We need multiple programs to be able to share a single computer and associated resources (disks, network, etc.)
- Programs can help us develop programs

examples?

Programming languages & compilers

- Develop programs that tend to be simpler, easier to port, easier to change, easier to get right, easier to understand, etc.
- Different languages tend to help under different constraints

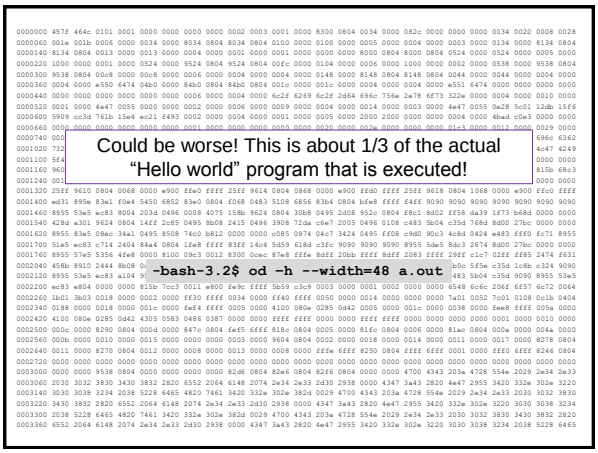
```
/* Hello World program */
#include <stdio.h>
main()
{
  printf("Hello World\n");
}
```

compiled using
-bash-3.2$ cc world.c

```
.file   "hello.c"
        .section        .rodata
.LC0:
        .string "Hello World"
        .text
.globl main
        .type   main, @function
main:
        leal    4
        andl    $
        pushl   -
        pushl   %
        movl
        pushl   %
        subl    $
        movl    $
        call    puts
        addl    $20, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
Hat 4.3.0 0/
        .section        .note.GNU-
stack,"",@progbits
```

Compilers translate programs in a high-level language into (more) executable programs

-bash-3.2$ cc -S world.c



Could be worse! This is about 1/3 of the actual "Hello world" program that is executed!

-bash-3.2$ od -h --width=48 a.out

Operating systems

examples?

- Software that manages activities and resources of a computer – the *kernel* is an OS core
- Abstractions that programs can use – and that application programmers use to write programs
- An easier-to-use interface to the hardware, reducing need to handle nitty-gritty details including
  - executing programs (and multi-tasking)
  - memory management (and virtual memory)
  - file systems, disk and network access
  - an interface to communicate with hardware
  - a user interface (often graphical)

Run-time systems

examples?

- A layer of software that fits between the compiler and the operating system
- Its primary objective is to allow programs written in a specific language (or class of languages) to do things during execution that are difficult to do during compilation
  - Examples may include memory management, object-oriented dispatch (which method gets called in a class hierarchy), input/output, etc.

Groups ~2-3 people, ~2 minutes

- List programs that help you develop programs

Programs to help develop programs

- Program editors
- Programming environments
- Debuggers
- Performance analysis tools
- Version control and configuration management tools
- …
- …more from you?

Unix: our OS of focus

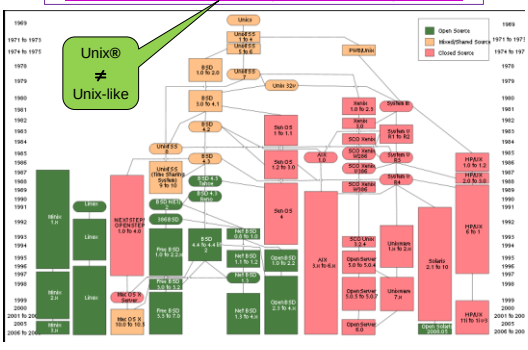
- Multics: time-sharing OS from MIT, GE and Bell Labs starting around 1964s – quite complex, with innovations in access control, memory/file management, and much more
  - Best known names associated with Multics include Fernando Corbató, Jerry Saltzer, Jack Dennis, Peter Denning, …
- Unix: time-sharing OS developed at Bell Labs (roughly hand-in-hand with the C programming language) around 1969 – highly simplified view of data, programs, etc.
  - Best known names associated with Unix include Ken Thompson, Dennis Ritchie, Brian Kernighan, Doug McIlroy

I am to 1964 as you are to about 1999: first web cast (Victoria's Secret), first GPU (NVIDIA), 600 MHz Pentium III, 802.11b, …

Unix key ideas

- Written in a high-level language (C)
- Virtual memory
- Hierarchical file system; "everything" is a file
- Lots of small programs that work together to solve larger problems
- Security, users, access, and groups
- Human-readable documentation included

Way more than one "Unix-like system"! 100's, maybe 1000's



http://en.wikipedia.org/wiki/File:Unix_history-simple.en.svg

Linux

- Linux: A kernel for a Unix-like operating system
  - commonly seen/used today in servers, mobile/embedded devices, ...
    - source can be downloaded, free to use, constantly being improved/updated by the community
- GNU: A "free software" implementation of many useful Unix-like tools.
  - many GNU tools are distributed with the Linux kernel
- Distribution ("distro"): A pre-packaged set of Linux software – examples are Ubuntu, Fedora, …

The Shell

- The *shell* is an interactive program that uses user input to manage the execution of other programs
  - There are many different shells, most of which are quite similar: sh, csh, tcsh, bash, ksh, …
  - **`bash`**: the default shell program on most Unix/Unix-like systems (and our focus)
- Why should I learn to use a shell when graphical user interfaces (GUIs) exist?
  - Faster, work remotely, programmable, customizable, repeatable
  - Many (old-timers?) prefer using a command line for many tasks

Input, output and errors

- **`bash`**, and essentially all shells, have a standard place to get input (**`stdin`**), to produce output (**`stdout`**), and to produce errors (**`stderr`**)
- All of these "streams" default to the shell console – a program reading from **`stdin`** will read from the console and writing to **`stdout`** and **`stderr`** will write to the (same) console
- We'll see simple ways to "redirect" these standard streams to take input from files, to write to files or printers, etc.

File and directories

- "Everything" is a file – a stream of bytes – and the most important kind of files to start are
  - Regular files, which contain data such as programs, data, html, etc.
  - Directories, which contain sets of files
- Every file has a name and is found in a directory
- The shell tracks your current working directory: this allows you to name files relative to that directory (so you can run them, use them for input or output, etc.)
- The shell also names your home directory – the "base" directory for your programs and data

Basic commands

command	description
`ls`	lists files in a directory
`pwd`	outputs the current working directory
`cd`	changes the working directory
`chsh`	changes your shell (persistently)
`man`	provides info about a command

Now Notkin runs a SSH Tectia terminal client for a brief demo

X Windows (aka X11 or X)

- For many years, Unix was entirely command-line based – that is, there was a single "window" that ran a shell
- Around 1984, as part of Project Athena, X Windows was developed at MIT and is the basis for most graphical user interfaces for Unix/Unix-like systems
  - X does not constrain the user interface, so desktop environments (such as GNOME and KDE) may look quite different from one another
- X runs over a network and the application being run can be on a different machine than the user's machine – that is, the window you see on your machine may be connected to a program on another machine

Administrivia: read web page

- *Read the web page*
- **Read the web page**
- Grading
  - 50% weekly homework assignments
    - Homework #0: available on Catalyst from 9/30/09 @ 3:30PM through 10/2/09 @ 12 Noon
    - Homework #1: due Thursday October 8, 11:30PM
  - 5% written work on societal impact of computing
  - 20% midterm (11/2/09)
  - 25% final exam (12/15/09)
- TAs – Collin and Hao
- Readings – start now!

Questions?

- I am not good at reading minds … so make sure I know what's on your mind!
- In class, out of class, …
- NOW!