Week 8 - Wednesday

# CS222

# Last time

- What did we talk about last time?
- Software engineering
- Testing
- GDB

# Questions?

# Project 4

# Quotes

*Good design adds value faster than it adds cost.*

Thomas C. Gale

# GDB

- GDB (the GNU Debugger) is a debugger available on Linux and Unix systems
- It is a command line utility, but it still has almost all the power that the Eclipse debugger does:
  - Setting breakpoints
  - Stepping through lines of code
  - Examining the values of variables at run time
- It supports C, C++, Objective-C, Java, and other languages

# Prerequisites

- C doesn't run in a virtual machine
- To use GDB, you have to compile your program in a way that adds special debugging information to the executable
- To do so, add the **`-ggdb`** flag to your compilation

```
gcc -ggdb program.c -o program
```

- Note: You will not need to do this on Friday's lab

# Starting GDB

- The easiest way to run GDB is to have it start up a program
- Assuming your executable is called **`program`**, you might do it like this:

```
gdb program
```

- It is also possible to attach GDB to a program that is running already, but you have to know its PID
- You  can also run GDB on a program that has died, using the core file (which is why they exist)

# Basic GDB commands

| Command | Shortcut | Description |
|---|---|---|
| `run` | `r` | Start the program running |
| `list 135` | `l` | List the code near line 135 |
| `list function` | `l` | List the code near the start of `function()` |
| `print variable` | `p` | Print the value of an expression |
| `backtrace` | `bt` | List a stack trace |
| `break 29` | `b` | Set a breakpoint on line 29 |
| `break function` | `b` | Set a breakpoint at the start of `function()` |
| `continue` | `c` | Start running again after stopping at a breakpoint |
| `next` | `n` | Execute next line of code, skipping over a function |
| `step` | `s` | Execute next line of code, stepping into a function |
| `quit` | `q` | Quit using GDB |

# Some String Issues

# A few final string issues

- What if you have a number and want a string version of it?
  - In Java:

```
int x = 3047;
String value = "" + x;                  //quick way
value = Integer.toString(x);            //fussy way
```

- What if you have a string that gives a numerical representation and you want the number it represents?
  - In Java:

```
String value = "3047";
int x = Integer.parseInt(value);
```

# String to integer

- In C, the standard way to convert a string to an **`int`** is the **`atoi()`** function
  - **`#include <stdlib.h>`** to use it

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

int main()
{
      char* value = "3047";
      int x = atoi(value);
      printf("%d\n", x);
      return 0;
}
```

# Implementing `atoi()`

- Now it's our turn to implement **`atoi()`**
  - Signature:

```
int atoi(char* number);
```

# Integer to string

- Oddly enough, this is a stranger situation
- Many systems have a non-standard function (also in **stdlib.h**) called **itoa()**
  - It takes the **int**, a buffer to hold the resulting string, and the base

```
char value[10];  //has to be big enough
int x = 3047;
itoa( x, value, 10 );
```

- The portable way to do this is to use **sprintf()**
  - It's like **printf()** except that it prints things to a string buffer instead of the screen

```
char value[10];  //has to be big enough
int x = 3047;
sprintf( value, "%d", x );
```

# Implementing `itoa()`

- Now it's our turn to implement **`itoa()`**
- We'll restrict ourselves to a base 10 version
  - Signature:

```
char* itoa(int number, char* string);
```

- It returns the pointer to the string in case this call is nested inside of another call (to **`strcat()`** or **`printf()`** or whatever)

# Users and Groups

# Users

- Recall that each user on a Linux system has a unique login name and a unique numerical identifier (the UID)
- Users can belong to one or more groups as well
- Where is this information stored?

# Password file

- The system has a password file stored in **`/etc/passwd`**
- Each line of this file corresponds to one user in the system and has seven fields separated by colons:
  - Login name
  - Encrypted password
  - UID
  - GID (group ID of the first group that the user is a member of)
  - Comment
  - Home directory (where you are when you log in)
  - Login shell (which shell you running when you log in)
- Example:

```
wittmanb:x:1000:100:Barry Wittman:/home/wittmanb:/bin/bash
```

# Catch-22

- Your computer needs to be able read the password file to check passwords
- But, even **`root`** shouldn't be able to read everyone's passwords
- Hash functions to the rescue!

# Cryptographic hash functions

- Take a long message and turn it into a short digest
- Different from hash functions used for hash tables
- Lots of interesting properties (lots more than these):

| | |
|---|---|
| Avalanching | • A small change in the message should make a big change in the digest |
| Preimage Resistance | • Given a digest, should be hard to find a message that would produce it |
| Collision Resistance | • Should be hard to find two messages that hash to the same digest (collision) |

# The Linux and Unix solution

- Instead of storing actual passwords, Linux machines store the hash of the passwords
- When someone logs on, the operating system hashes the password and compares it to the stored version
- No one gets to see your original password
  - Not even **`root`**!

# Back to the password file

- Inside the password file, we have encrypted passwords
- Everyone's password is safe after all

| Login Name | Password Hash |
|---|---|
| ahmad | IfW{6Soo |
| baili | 853aE90f |
| carmen | D390&063 |
| deepak | CWc^Q3Ge |
| erica | e[6s_N*X1 |

# Shadow password file

- Even though the password is disguised, it is unwise to let it be visible to everyone
  - Given a password digest (the hashed version) and lots of time, it is possible to figure out the password
- It's useful for the password file to be readable by everyone so that all users on a machine are known to all others
- A shadow password file stores the encrypted password and is readable only by privileged users
  - **`/etc/shadow`**

# Changing your password

- Amid all this discussion, it might be useful to know how to change your password
- I **don't** recommend that you do change your password
  - I'm honestly not sure how doing so will interact with your Active Directory (Windows) password
- The command is **`passwd`**

```
Changing password for wittmanb.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

# Changing the owner of a file

- You recall that we can change permissions for who can read, write, and execute a file using **chmod**
- But **chmod** depends on who the owner is
- What if you want someone else to be the owner of a file?
- The **chown** command can let you do that
- If I want my file **stuff.txt** to be owned by Dr. Leap, I would use the following command

```
chown leap stuff.txt
```

- On most systems, **chown** only works if you are **root**

# Groups

- Files are associated with a group as well as a user who is owner
- The groups are listed in the **`/etc/group`** file
- Each line of this file corresponds to a group and has four fields separated by colons:
  - Group name
  - Encrypted password
    - Often not used
  - Group ID (GID)
  - User list
    - Comma separated
- Example:

```
users:x:100:
jambit:x:106:claus,felli,frank,harti,markus,martin,mtk,paul
```

# Creating a group

- If you want to create a group, you have to be **root**
- If you're **root** (or using **sudo**), you can use the **groupadd** command
- To create the **awesome** group as **root**:

```
groupadd awesome
```

- Or using **sudo**:

```
sudo groupadd awesome
```

# Adding a user to a group

- Again, you have to be **root** to add a user to a group
- Use the **useradd** command
- To add user **wittmanb** to the **awesome** group as **root**:

```
useradd -g awesome wittmanb
```

- Or using **sudo**:

```
sudo useradd -g awesome wittmanb
```

# Changing the group for a file

- When you create a file, it is associated with some default group that you belong to
- You can use the **`chgrp`** command to change to another group that you belong to

```
chgrp awesome file.txt
```

- If you are root, you can use the chown command to change the group, using a colon

```
chown :awesome file.txt
```

# Quiz

# Upcoming

# Next time…

- Time from the Linux perspective
- Lab 8

# Reminders

- Keep working on Project 4
- Read LPI Chapter 10