Week 8 - Friday

CS222

Last time

- What did we talk about last time?
- Converting strings to numbers
- Converting numbers to strings
- Users and passwords

Questions?

Project 4

Quotes

Measuring programming progress by lines of code is like measuring aircraft building progress by weight.

Bill Gates

Passwords

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 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
```

Time

Time

- In the systems programming world, there are two different kinds of time that are useful
- Real time
  - This is also known as wall-clock time or calendar time
  - It's the human notion of time that we're familiar with
- Process time
  - Process time is the amount of time your process has spent on the CPU
  - There is often no obvious correlation between process time and real time (except that process time is never more than real time elapsed)

Calendar time

- For many programs it is useful to know what time it is relative to some meaningful starting point
- Internally, real world system time is stored as the number of seconds since midnight January 1, 1970
  - Also known as the Unix Epoch
  - Possible values for a 32-bit value range from December 13, 1901 to January 19, 2038
  - Systems and programs that use a 32-bit signed **`int`** to store this value may have strange behavior in 2038

time()

- The **time()** function gives back the seconds since the Unix Epoch
- Its signature is:

```
time_t time(time_t* timePointer);
```

- **time_t** is a signed 32-bit or 64-bit integer
- You can pass in a pointer to a **time_t** variable or save the return value (both have the same result)
- Typically we pass in **NULL** and save the return value
- Include **time.h** to use **time()**

```
time_t seconds = time(NULL);
printf("%d seconds have passed since 1970",
      seconds);
```

Time structures

- Many time functions need different structs that can hold things
- One such struct is defined as follows:

```
struct timeval
{
        time_t tv_sec;              // Seconds since Epoch
        suseconds_t tv_usec;        // Extra microseconds
};
```

gettimeofday()

- The **gettimeofday()** function offers a way to get higher precision timing data
- Its signature is:

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

- From the previous slide, **timeval** has a **tv_secs** member which is the same as the return value from **time()**
- It also has a **tv_usec** member which gives microseconds (millionths of a second)
- The **timezone** pointer **tz** is obsolete and should have **NULL** passed into it
- Include **sys/time.h** (**not** the same as **time.h**) to use this function

Timing with `gettimeofday()`

- **`gettimeofday()`** is a reliable way to see how long something takes
- Get the start time, the end time, and subtract them

```
double start;
double end;
struct timeval tv;
gettimeofday(&tv, NULL);
start = tv.tv_sec + tv.tv_usec/1000000.0;
someLongRunningFunction();
gettimeofday(&tv, NULL);
end = tv.tv_sec + tv.tv_usec/1000000.0;
printf("Your function took %.3f seconds",
      end - start);
```

ctime()

- What about printing out a human-readable version of the time?
- **`ctime()`** takes a **`time_t`** value and returns a string giving the day and time

```
printf(ctime(time(NULL)));
//prints Fri Mar 15 14:22:34 2013
```

- Alternatively, **`strftime()`** has a set of specifiers (similar to **`printf()`**) that allow for complex ways to format the date and time

Broken down time structure

```
struct tm
{
    int tm_sec; // Seconds (0-60)
    int tm_min; // Minutes (0-59)
    int tm_hour; // Hours (0-23)
    int tm_mday; // Day of the month (1-31)
    int tm_mon; // Month (0-11)
    int tm_year; // Year since 1900
    int tm_wday; // Day of the week (Sunday = 0)
    int tm_yday; // Day in the year (0-365; 1 Jan = 0)
    int tm_isdst; /* Daylight saving time flag
    > 0: DST is in effect;
    = 0: DST is not effect;
    < 0: DST information not available */
};
```

gmtime(), localtime(), and mktime()

- **gmtime()** and **localtime()** convert a **time_t** value to a struct that contains "broken down" time
  - **gmtime()** gives UTC time (used to be called Greenwich Mean Time)
  - **localtime()** gives the local time, assuming it is set up correctly

```
time_t seconds = time(NULL);
struct tm* brokenDownTime = NULL;
brokenDownTime = localtime(&seconds);
if( (*brokenDownTime).tm_wday == 1 )
      printf("It's just another manic Monday.\n");
```

- **mktime()** can convert from a broken down time back into **time_t**

Jiffies

- How accurate is the microsecond part of **`gettimeofday()`**?
- It depends on the accuracy of the software clock in your system
- This clock measures time in units called **jiffies**
- A jiffy used to be 10 milliseconds (100 Hz)
- They raised the accuracy to 1 millisecond (1000 Hz)
- Now, it can be configured for your system to 10, 4 (the default), 3.3333, and 1 milliseconds

Process time

- For optimization purposes, it can be useful to know how much time a process spends running on the CPU
- This time is often broken down into
  - **User time:** the amount of time your program spends executing its own code
  - **System time:** the amount of time spent in kernel mode executing code for your program (memory allocation, page faults, file opening)

The `time` command

- You can time a program's complete execution by running it with the time command
  - It will give the real time taken, user time, and system time
- Let's say you've got a program called **`timewaster`**
  - Run it like this:

```
time ./timewaster
```

  - Output might be:

```
real 0m4.84s
user 0m1.030s
sys 0m3.43s
```

Lab 8

Upcoming

Next time…

- Structs

Reminders

- Keep working on Project 4
- Read K&R Chapter 6
- Read LPI Chapter 10
- **Employer Meet and Greet**
  - Monday, March 19, 2018 from 11 a.m. – 1 p.m.
  - Masters Center (Mineral Gallery)
  - Employers attending:
    - Clark Associates
    - Donegal Insurance Group
    - Gross Investments
    - Hershey Entertainment and Resorts
    - Members 1st Credit Union
    - Northwestern Mutual
    - WebpageFX