EECS 22: Assignment 1

Prepared by: Che-Wei Chang, Prof. Rainer Dömer

Octorber, 2 2014

Due Tuesday 14 Oct 2014 at 11:00pm

1 Part1: Login to your Linux account

For this class, you will be doing your assignments by *logging on* to a shared machine (server) running the Linux operating system. Even though you may be using a personal computer or a workstation that is capable of computation locally, you will mainly be using them as *terminals* (clients), whose job is to pass keystrokes to the server and display outputs from the server.

To use a shared machine, first you need an *account* on the machine. EECS support has created an *account* for each student. To retrieve the username and password go to the following website:
`https://newport.eecs.uci.edu/account.py`.
The website asks for your UCInetID and the according password before giving you the account information of your new EECS account. Note that your browser may also ask you to accept a certificate to open the secure website. If you have a problem please contact your EECS 22 TA, (eecs22@eecs.uci.edu).

The names of the instructional servers are `zuma.eecs.uci.edu` and `crystalcove.eecs.uci.edu` You can log into your account with your EECS user name and password. Your account also comes with a certain amount of disk space. You can use this space to store homework assignment files, and you don't need to bring your own disks or other storage media.

1.1 Software and commands for remote login

You can connect to `zuma.eecs.uci.edu` or `crystalcove.eecs.uci.edu` from virtually any computer anywhere that has internet access. What you need is a client program for *remote login*.

Previously, people used **`rlogin`** or **`telnet`** to connect to the server, and **`ftp`** or **`rcp`** to transfer files. However, these protocols are insecure, because your keystrokes or output are in clear text and can be *snooped* by others. This means your account name and password can be stolen this way. So, for security reasons, do not use either of these programs.

Instead, use **`ssh`** as the primary way to connect to the server. **`ssh`** stands for *secure shell*, and it encrypts your network communication, so that your data cannot be understood by snoopers. For file transfers, use **`sftp`** or **`scp`**, which are secure.

Depending on what computer you use, it may have a different *implementation* of **`ssh`**, but the basic function underneath are all the same. Check the course web site on SSH:
`https://eee.uci.edu/11f/18056/resources.html`

- If you are logging in from a Windows machine, you can use **`SecureCRT`** or **`PuTTY`**.
- MacOS X already has this built-in (use Terminal or X11 to run a Linux shell). Most Linux distributions also bundle **`ssh`**.
- If you are logging in from an X terminal, you can use the command

  ```
  % ssh zuma.eecs.uci.edu -X -l yourUserName
  ```

  (note: `%` is the prompt, not part of your command) It will prompt you for your password. Note that the `-X` option allows you to run programs that open X windows on your screen.

1.2 Linux Shell

By now you should be logged in, and you should be looking at the prompt
`zuma% _`

Note: in the following writeup, we will show just
`%`
for the prompt, instead of
`zuma%`

You should change your password using the **`yppasswd`** command.

Try out the following commands at the shell prompt (See reference to the Linux Guide in section 1.3 for more details about these commands.).

`ls`	list files
`cd`	(change working directory)
`pwd`	(print working directory)
`mkdir`	(make directory)
`mv`	(rename/move files)
`cp`	(copy files)
`rm`	(remove files)
`rmdir`	(remove directory)
`cat`	(print the content of a file)
`more`	(print the content of a file, one screen at a time)
`echo`	(print the arguments on the rest of the command line)

Most commands take one or more file names as parameters. When referring to files, you may need to qualify the file name with directory references, absolute vs. relative paths:

`.`	(current directory)
`..`	(one level higher)
`~`	(home directory)
`/`	the root (top level) directory

1.3 Follow the Linux Guide

The best bet may be to search online for something like "linux user tutorial," "linux user guide," "unix command line" or "unix shell command" and check a few results to see what is agreeable to you. From those links, the following may be reasonable:
`http://linux.org.mt/article/terminal`
`http://www.linux-tutorial.info/modules.php?name=MContent&pageid=49`
or `ftp://metalab.unc.edu/pub/Linux/docs/linux-doc-project/users-guide/user-beta-1.pdf.zip` (3.3.1-2, and chapter 4)
Learn basic shell commands: list files, change directory, rename files, move files, copy files, show file content.

There is nothing to turn in for this part.

2 Learn to use a text editor

There are three editors that are available on nearly all Linux systems that you may choose from.
`pico` is the easiest to get started with. A guide for **`pico`** can be found at:
`http://www.dur.ac.uk/resources/its/info/guides/17Pico.pdf.`
`vi` is a very powerful editor, but is arguably a bit more difficult to learn. Follow the **`vi`** guide at:
`http://dcssrv1.oit.uci.edu/indiv/gdh/vi/vi-SunWorld-article.html`
or `http://www.ece.uci.edu/~chou/vi.html`
Finally, **`emacs`** is another editor that you may use. **`emacs`** is also a powerful editor, but is a bit easier to learn than **`vi`**. Follow the **`emacs`** guide at:

http://www.gnu.org/software/emacs/tour/.

Learn how to edit a file, move the cursor, insert text, insert text from file, delete words, delete lines, cut/paste, save changes, save to another file, quit without saving.

There is nothing to turn in for this part. However, it is critical that you get enough practice with your editor, so that you can do the homework for this class.

3 Part 2: Blackjack (100 points)

This is a Blackjack program. The idea is to simulate Blackjack in order to make programming more fun! ;-)

Here is the overview of the implementation: To simulate the shuffled stack of cards, we use a pseudo random number generator that generates a random number in the range of 1 to 13. This represents the cards numbered 1 through 10, plus the jack, queen and king, respectively. If the card number is 1 to 10, it directly represents the value of the card. If the card number is 11 to 13, the card represents the jack, queen, and king, which all have the face value 10.

Player's round: The dealer draws an initial card for the player and shows it. The player then can choose to draw additional cards as many times as he wants. If his cards have a combined value of more than 21, he loses immediately. If the player decides not to draw any more cards, it's the dealer's turn.

The interface should look like the following:

```
**********************************
** Welcome to EECS22 BlackJack! **
**********************************
Your first card is: 7
Do you want another card?
Type 1 for Yes, 0 for No: 1
Your next card is: 8
Your combined value is: 15
Do you want another card?
Type 1 for Yes, 0 for No: 1
Your next card is: 9
Your combined value is: 24
Sorry. You lose!
```

Dealer's round: The dealer draws his own cards until he reaches one of the following conditions: If his combined value reaches more than 21, the dealer loses. If his combined value is the same as the player's value, the dealer wins. If his combined value is higher than the player's value, the dealer wins. An example code is shown below:

```
**********************************
** Welcome to EECS22 BlackJack! **
**********************************
Your first card is: 7
Do you want another card?
Type 1 for Yes, 0 for No: 1
Your next card is: 8
Your combined value is: 15
Do you want another card?
Type 1 for Yes, 0 for No: 0
Dealer draws another card.
Dealer's card is: 10
Dealer's value is 10, you have 15.
```

```
Dealer draws another card.
Dealer’s card is: 4
Dealer’s value is 14, you have 15.
Dealer draws another card.
Dealer’s card is: 10
Dealer’s value is 24, you have 15.
Dealer loses. You win!
```

You should submit your program code as file `blackjack.c`, a text file `blackjack.txt` briefly explaining how you designed your program, and a typescript `blackjack.script` which shows that you compile and run your program. Please run it twice so that the script shows that you and the dealer win one time each.

HINT: To generate the initial random number, you have to use a random number generator which is provided by the C standard function rand(). This function generates a random number of type int in the range of 0 to 32767. This function is provided in the header file `stdlib.h`.

In practice, no computer function can produce truly random data; they only produce pseudo-random numbers. These are computed from the formula and the number sequences they produce are repeatable. A seed value is usually used by the random number generator to generate a number. Therefore, if you use the same seed value all the time, the same sequence of ”random” numbers will be generated (i.e. your program will always produce the same ”random” number in every program run). To avoid this, we can use the current time of the day to set the random seed, as this will always be changing with every program run. With this trick, your program will produce different numbers every time you run it.

To set the seed value, you have to use the function **srand()**, which is also defined in the header file `stdlib.h`. For the current time of the day, you can use the function time(), which is defined in the header file `time.h` (`stdlib.h` and `time.h` are header files just like the `stdio.h` file that we have been using so far).

In summary, use the following code fragments to generate the random number for the game:

1. Include the `stdlib.h` and `time.h` header files at the beginning of your program:
   #include <stdlib.h>
   #include <time.h>

2. Include the following lines at the beginning of your main function:
   /* initialize the random number generator with the current time */
   srand(time(NULL));

3. To simulate drawing a card from the shuffled deck, use the following statement:
   /* draw a random card */
   card = rand() % 13 + 1; // You need to define the variable card.

   The integer variable ’card’ then will have a random value in the range from 1 through 13.

3.1 Writing your code

First create a subdirectory named `hw1` (for homework one). Change into the created directory `hw1`. Then, use your editor to create a C file named `blackjack.c`. Do not use a word processor and transfer or paste the content. The C file should state your name and exercise number as a comment at the top of the file.

3.2 Compiling your code

To test your program, it must be compiled with the **gcc** command. This command will report any errors in your code. To call **gcc**, use the following template:

```
% gcc sourcefile -o targetfile
```
Then, simply execute the compiled file by typing the following:
```
% ./targetfile
```
Note: Please compile your C code using **-ansi -Wall** options as below to specify ANSI code with all warnings:

Below is an example of how you would compile and execute your program for ”Guess the number” game:
```
% gcc blackjack.c -ansi -Wall -o blackjack
% ./blackjack
program executes
% _
```

4 Bonus Problem

Extend the blackjack program. To make the game more real, for each ace card (1), the player can choose the value to be either 1 or 11 for best interest. The decision can only be made once the card is issued and cannot be changed afterwards.

To submit, use the same files as in Part 2, i.e. blackjack.c, blackjack.txt, and blackjack.script.

5 Submission

To submit your work, you have to be logged in the server `zuma` or `crystalcove`.

Here is a checklist of the files you should have:

In the `hw1` directory, you should have the following files in your linux account:

- `blackjack.c`
- `blackjack.txt`
- `blackjack.script`

We do require these *exact* file names. If you use different file names, we will not see your files for grading. Now, you should change the current directory to the directory containing the `hw1` directory. Then type the command:
```
%  /ecelib/bin/turnin22
```
which will guide you through the submission process.
You will be asked if you want to submit the script file. Type yes or no. If you type “n” or “y” or just plain return, they will be ignored and be taken as a no. You can use the same command to update your submitted files until the submission deadline.
Below is an example of how you would submit your homework:

```
% ls # This step is just to make sure that you are in the correct directory that contains hw1/
hw1/
% /ecelib/bin/turnin22
============================================================
EECS 22 Fall 2014:
Assignment "hw1" submission for eecs22
Due date:  Tue Oct 14 23:00:00 2014
** Looking for files:
** blackjack.c
** blackjack.txt
** blackjack.script
```

```
==============================================================
* Please confirm the following:                              *
* "I have read the Section on Academic Honesty in the        *
* UCI Catalogue of Classes (available online at              *
* http://www.editor.uci.edu/catalogue/appx/appx.2.htm#gen0)  *
* and submit myoriginal work accordingly."                   *
Please type YES to confirm.  Y
==============================================================
File blackjack.c exists, overwrite?  [yes, no] y
File blackjack.c has been overwritten
Submit blackjack.txt [yes, no]?  y
File blackjack.txt has been submitted
Submit blackjack.script [yes, no]?  y
File blackjack.script has been submitted
==============================================================
Summary:
==============================================================
Submitted on Mon Sep 29 17:00:21 2014
You just submitted file(s):
blackjack.c
blackjack.txt
blackjack.script
% _
```

5.1 Verify your submission

This step is optional, but recommended. If you want to confirm which files you have submitted, call the following command:

```
% /users/grad2/doemer/eecs22/bin/listfiles.py
```

This command lists your submitted files. Don't worry if you submitted too many files. We will only look at the files with defined names (here: `blackjack.c`, `blackjack.txt` and `blackjack.script`) and ignore other files.

6 Typescript

A typescript is a text file that captures an interactive session with the Linux shell. Very often you are required to turn in a typescript to show that your program runs correctly. To create a typescript, use the **`script`** command. Here is an example:

- Type the command
  ```
  % script
  ```
  into the shell. It should say
  ```
  Script started, file is typescript
  % _
  ```
  This means it is recording every key stroke and every output character into a file named "typescript", until you hit **`^D`** or type **`exit`**.

- Type some shell commands. But don't start a text editor!

- Stop recording the typescript by typing **`exit`**.
  ```
  % exit
  Script done, file is typescript
  % _
  ```

- Now you should have a text file named `typescript`. Make sure it looks correct.

  ```
  % more typescript
  Script started on Mon 29 Sep 2014 04:58:45 PM PDT
  ...
  ...
  ```

You should immediately rename the typescript to another file name. Otherwise, if you run **`script`** again, it will overwrite the `typescript` file.

Note: If you backspace while in script, it will show the `^H` (control-H) character in your typescript. This is normal. If you use **`more`** to view the typescript, then it should look normal.