EECS 22: Assignment 1

Prepared by: Tim Schmidt, Prof. Quoc-Viet Dang, Prof. Rainer Dömer

September 21, 2017

Due Wednesday October 11, 2017 at 6:00 pm

Contents

1 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. You should be able to login using your regular UCInetID credentials. If you have any problems, please contact your EECS 22 TA: eecs22@eecs.uci.edu

The name of the instructional servers are `bondi.eecs.uci.edu` and `laguna.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. In the follwing sections, server `bondi.eecs.uci.edu` is used to illustrate the instructions, but you can use also `laguna.eecs.uci.edu` or `crystalcove.eecs.uci.edu` to do your assignment.

1.1 Software and commands for remote login

You can connect to `bondi.eecs.uci.edu` from virtually any computer anywhere that has internet access. What you need is a client program for *remote login*.
We will 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 network traffic is safe and secure. For file transfers, you may use **`sftp`** or **`scp`**, which are secure as well.

Depending on what computer you use, it may have a different *implementation* of **`ssh`**, but the basic function underneath is the same. Check the resources tab on the course web site for details:
`https://eee.uci.edu/17y/18010/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 bondi.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
`bondi% _`

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

You should change your password using the **`passwd`** 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 or 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.

1.4 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://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.

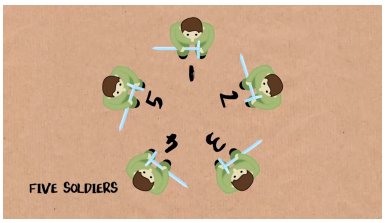
There is nothing to turn in for this part. However, it is critical that you get enough practice with your editor, so that you can do the homework for this class.

2 The Josephus Problem (100 points)

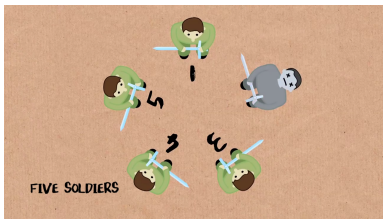
In computer science and mathematics, the Josephus problem (or Josephus permutation) is a theoretical problem related to a certain counting-out game.) People are standing in a circle waiting to be executed. Counting begins at a specified point in the circle and proceeds around the circle in a specified direction. After a specified number of people are skipped, the next person is executed. The procedure is repeated with the remaining people, starting with the next person, going in the same direction and skipping the same number of people, until only one person remains, and is freed. The problem - given the number of people, starting point, direction, and number to be skipped - is to choose the position in the initial circle to avoid execution. [source: `https://en.wikipedia.org/wiki/Josephus_problem`]

Instead of using bit-manipulating operations (as indicated on the Wikipedia web page, for example), implement your program by using an array to represent the circle of soldiers and loop and conditional constructs to "simulate" the counting-out game.)

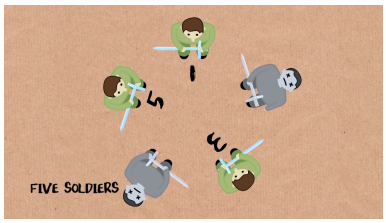
Throughout this assignment (and the entire EECS22 course), you may assume that the user only enters valid input. Thus, there is no need for you to "bullet-proof" your implementation against invalid data entered by the user. Here specifically, you may assume that the number of soldiers is a positive natural number below 1,000.



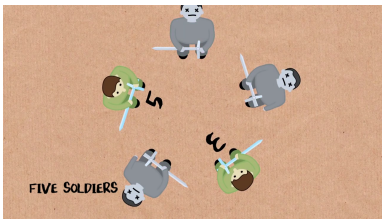
(a) all alive



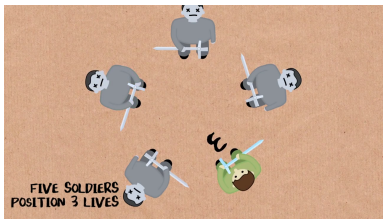
(b) 1 kills 2



(c) 3 kills 4



(d) 5 kills 1



(e) 3 kills 5

Figure 1: Example with 5 persons taken from `https://www.youtube.com/watch?v=uCsD3ZGzMgE`

The problem is fully animated in `https://www.youtube.com/watch?v=uCsD3ZGzMgE`.
Your program takes as input the number of people in the circle and outputs the safe spot. The interface of the C program should look like the following:

```
./josephus
```

```
How many people are in the circle? 41
The spot 19 is safe.
```

2.1 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 `josephus.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.

Note: Please write your C code using proper indentation (either tabs or white spaces).

2.2 Compiling your code

To test your program, it must be compiled with the **`gcc`** command. This command will report any errors in your code. Please compile your C code using **-ansi -std=c99 -Wall** options as below to specify c99 code with all warnings:

Below is an example of how you would compile and execute your program for the Josephus problem:

```
% gcc josephus.c -ansi -std=c99 -Wall -o josephus
% ./josephus
program executes
% _
```

3 Bonus (5 points)

You will get 5 extra points, if your program also prints the most executions by an individual person.

As an example, the interface of the C program with bonus points should look like the following:

```
./josephus
How many people are in the circle?5
The spot 3 is safe.
Most executions by an individual are 2.
```

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

4 Submission

Only compilable submissions can get up 100 points.
Not compilable submissions get **at most** 50 points!
We compile all submissions with `-ansi -std=c99 -Wall`.
Warnings will reduce your points.

Here is a checklist of the files you should have for submission. In the `hw1` directory, you should have the following files in your linux account:

- `josephus.c`
  The complete program that is compilable
- `josephus.txt`
  Briefly describe what your program does and why you implemented it in this way

- `josephus.script`
  The typescript shows that your program runs correctly with the given input.

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:
`% ~eecs22/bin/turnin.sh`
which will guide you through the submission process.

You will be asked if you want to submit each 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:

```
bondi% ls  # This step is just to make sure that you are in the correct directory that contains hw1/
hw1/
bondi%  ~eecs22/bin/turnin.sh
============================================================
EECS 22 Fall 2016:
Assignment "hw1" submission for eecs22
Due date:  Thu Oct 6 18:00:00 2016
** Looking for files:
** josephus.c
** josephus.txt
** josephus.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 my original work accordingly."                 *
Please type YES to confirm.  YES
============================================================
File josephus.c exists, overwrite?  [yes, no] yes
File josephus.c has been overwritten
Submit josephus.txt [yes, no]?  yes
File josephus.txt has been submitted
Submit josephus.script [yes, no]?  yes
File josephus.script has been submitted
============================================================
Summary:
============================================================
Submitted on Tue Sep 20 17:00:21 2016
You just submitted file(s):
josephus.c
josephus.txt
josephus.script
% _
```

4.1 Verify your submission

This step is optional, but recommended. If you want to confirm which files you have submitted, call the following command:
`% ~eecs22/bin/listfiles.py`

This command lists your submitted files. We will only look at the files with defined names (here: `josephus.c`, `josephus.txt` and `josephus.script`) and ignore other files.

5 Typescript

A typescript is a text file that captures an interactive session within 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.