California State University, Sacramento
College of Engineering and Computer Science

Computer Science 35: Introduction to Computer Architecture

Spring 2019 – Lab 7 – *Wacky Words*

Overview

As kids we all got to play fun word games. Of course, we started with the classic *See N' Say*®, but eventually we moved on to other fun games such as *MadLibs*®.

Often, the player would enter various nouns, verbs. These words would then be inserted into a story. Because the player didn't know the context of the words, the resulting story can be silly, nonsensical, or simply weird.

For this lab, you are going to a very basic form of this type of game. You will have the player input three words and your program will print out a wacky sentence. The sentence is completely up to you!

Sample Run

The following is a sample run of the program. The user's input is printed in **blue**. The data outputted from your buffers is printed in **red**. Don't use this (rather boring) sentence in your program. Use your own!!!!

Prompt the player

```
Enter a name.
Joe
Enter a noun.
chicken
Enter a noun.
desk
The chicken raised its desk and threw it at Joe!
```

Output from the read data

Part 1 – Using Your Last Lab

Like that last lab, you aren't allowed to use that handy-dandy CSC35.o library. Instead, use will reuse some of your from Lab 6 to make this lab much easier.

You created a subroutine that was used to print a string to the screen. The name was up to use – as was the two registers that you used. All that being said, if your subroutine was named "MyExcellentWrite" and used register rax (for the address) and rbx (for the length), calling it would look like this:

```
mov $Hello, %rax
mov $23, %rbx
call MyExcellentWrite
```

Let's make a copy of your last lab and modify it. Type the following at the UNIX prompt. I'm assuming your Lab 6 is named "lab6.s". If it is something else, use that name:

```
cp lab6.s lab7.s
```

Presto! You copied the file. Open and edit this one.

Part 2 – Your Read Subroutine

Running the UNIX Kernal call to read a string is pretty much like writing one. To create an empty buffer, you can use the .space directive. It will create specified number of bytes in memory.

```
Noun:
   .space 50      #Fifty character buffer

NounLength:
   .quad 0        #One long integer with an initial value of 0
```

You will call the UNIX "read" command with the buffer address and the maximum size of the buffer. UNIX will return the number of characters that the user entered in the %rax register. This is the length of the string. Also note, this length includes the new line. **You should subtract one**.

UNIX Read		
Register	**Value**	**Description**
`rax`	0	The value 0 is the UNIX command number to read data. UNIX returns the *actual* number of bytes that were read in this register. This number also counts the newline character.
`rdi`	0	rdi is used to specify the source. Standard input, the keyboard, is 0.
`rsi`	*address*	This is the address where UNIX will store the bytes.
`rdx`	*max bytes*	To prevent a buffer overflow, rdx stores the maximum number of bytes that will be stored.

So, let's assume – once again – that you will write a subroutine called "MyAwesomeRead" and it uses the rax register (for the buffer), and rbx (for the maximum number of character). Calling it would look like this:

```
mov $Noun, %rax
mov $30, %rbx
call MyExcellentRead
mov %rbx, NounLength      #Save the returned length
```

Your subroutine will return the number of characters read in a register. This this example, it was rbx. **Your main program will need to save this value** – you will need it later.

Part 3 – The Game

This lab has several parts you need to get working before continuing on. If you attempt to write the entire program at one time, **it will be nearly impossible to find bugs.** If you try to write it all at one time, I might (during the lab) simply recommend you start over.

Remember: incremental design!

1. First, you need to be able to read data from the keyboard.
2. Now see if you can print off the buffer you just filled from the keyboard. Make sure to use the length of the string in the buffer rather than the entire buffer.
3. Working? Excellent! Now see if you can do it with another variable. You need a second buffer and variable to hold the number of characters UNIX read.
4. Now let's see if you can make your program print the entire sentence on one line.

Requirements

You must think of a solution on your own. Looking at another student's solution may result in a zero on this assignment. The requirements are as follows:

1. Read in three words from the player.
2. Prompt the player each time.
3. Print the sentence to the screen. This must be your creation! Use your imagination!
4. Show one your classmates your creation!

UNIX Commands

Editing

Action	Command	Notes
Edit File	`nano` *filename*	"Nano" is an easy to use text editor.
E-Mail	`alpine`	"Alpine" is text-based e-mail application. You will e-mail your assignments it.
Assemble File	`as -o` *objectfile asmfile*	Don't mix up the *objectfile* and *asmfile* fields. It will destroy your program!
Link File	`ld -o` *exefile objectfiles*	Link and create an executable file from one (or more) object files

Folder Navigation

Action	Command	Description
Change current folder	`cd` *foldername*	"Changes Directory"
Go to parent folder	`cd ..`	Think of it as the "back button".
Show current folder	`pwd`	Gives a file path
List files	`ls`	Lists the files in current directory.

File Organization

Action	Command	Description
Create folder	`mkdir` *foldername*	Folders are called directories in UNIX.
Copy file	`cp` *oldfile newfile*	Make a copy of an existing file
Move file	`mv` *filename foldername*	Moves a file to a destination folder
Rename file	`mv` *oldname newname*	Note: same command as "move".
Delete file	`rm` *filename*	Remove (delete) a file. There is **no** undo.

Just in case you need it…

The following is the UNIX call for write. You used this in Lab 6.

UNIX Write		
Register	**Value**	**Description**
`rax`	`1`	The value 1 is the UNIX command number to write data.
`rdi`	`1`	rdi is used to specify the destination. The screen is 1.
`rsi`	*address*	This is address where UNIX will read from bytes from.
`rdx`	*byte count*	rdx contains the number of bytes to write from the buffer. This can be the entire buffer or just part of it.