California State University, Sacramento
College of Engineering and Computer Science

Computer Science 35: Introduction to Computer Architecture

Fall 2020 – Lab 7 – *House Elf Work*

Overview

In the ***Wizarding World of Harry Potter***, house elves do all the hard work – cleaning, cooking, and all other monotonous tasks wizards prefer not to do. This, fortunately, changed after the Second Wizarding War.

Anyway, at the beginning of the semester, you wrote a lab that implemented the classic Hello World program. For the entire semester, you relied on the CSC35.o library which hid the details of the operating system from you. Well, it's time to do the hard work – talk to the operating system directly.

The House Elf's Task

Your challenge is to print text to the screen using Linux/UNIX Kernal calls. It's just you and the operating system. So, you won't use the library.

In fact, you are not allowed to use the library!

You will use Kernal calls to do all the following:

1. Print "Hello World!" to the screen.
   You can use either **`.ascii`** or a series of **`.byte`** directives to create it in memory.

2. Print what you plan to do over Winter Break.
   You can use either **`.ascii`** or a series of **`.byte`** directives to create it in memory.

3. Print the name of fun class you would like to take. It doesn't have to be a Computer Science one.
   You can use either **`.ascii`** or a series of **`.byte`** directives to create it in memory.

4. Print your name to the screen.
   But, for this one, you were a bad elf and need to punish yourself. So, when you create this buffer, you must use **`.byte`** directives to create it in memory. You must use ASCII codes. There is an example below.

5. End the program

Tips

- Like all labs, **build it in pieces**. Get the "Exit" call to work first before working on the writes.
- You have to setup all registers – **each time** – before you call the kernal.
- Pay close attention to the order of your instructions. Syscall only after you have all the registers ready.

Creating a String 1 Byte at a Time

When you print your name, you have to create a buffer using ASCII codes. The following contains the string **"Dobby\n"** – he's a free elf. The newline character has the ASCII code 10.

```
Name:
     .byte 68
     .byte 111
     .byte 98
     .byte 98
     .byte 121
     .byte 10
```

Talking to the Kernel

Let's start off by seeing if we output "Hello, World!" using UNIX Kernal calls. The basic calls, that you will need for this lab, are below.

Call	rax	rdi	rsi	rdx	Results
Write	1	File Descriptor (1 = screen)	Source address	Total bytes to write	**rax** contains the number of bytes successfully written.
Exit	60	Error Code *(0 = all okay)*	*none*	*none*	*none*

WARNING:
rcx and **r11** will be destroyed whenever 64-bit UNIX is called. Do not use them for running data.

Link the Objects

Since you are not using the CSC 35 library. When you link, you will only specify the object you created.

```
ld –o a.out lab7.o
```

Submitting Your Lab

This activity may only be submitted in Intel Format. Using AT&T format will result in a zero.

To submit your lab, you must run Alpine by typing the following and, then, enter your username and password.

```
alpine
```

Please send an e-mail to yourself (on your Outlook, Google account) to check if Alpine is working. To submit your lab, send the assembly file (do not send the a.out or the object file). Send the .asm file to:

```
dcook@csus.edu
```

Requirements

You must think of a solution on your own. The requirements are as follows:

1. Print "Hello World!" to the screen.
2. Print some text – like a quote or your plans for the Winter Break.
3. Print your name using the .byte directive.
4. End the program
5. Use a separate Kernal call for each requirement: 1 to 4. You will receive a zero if you don't.
6. Use the 64-bit system calls. You will receive a zero if you don’t.

UNIX Commands

Editing

Action	Command	Notes
Edit File	**nano** *filename*	"Nano" is an easy to use text editor.
E-Mail	**alpine**	"Alpine" is an e-mail application.
Assemble File	**as -o** *object source*	Don't mix up the *object* and *source* fields
Link File	**ld -o** *exe object(s)*	Can link multiple 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 the current a file path
List files	**ls**	Lists the files in current directory.
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.