CALIFORNIA STATE POLYTECHNIC UNIVERSITY

**Department of Computer Science**

***CS431***

**N.D.**

**HWK1**

Before you get started on the Homework make sure you have Lubuntu setup on a laptop or desktop. Please follow the guides listed in the O.S. Project within blackboard if you have not done so.

This class will be using the C programming language for homework and project assignments. As such you can find helpful guides on some of the differences in your book, specifically chapter 1 “The World according to C.” Additionally, I have uploaded a pdf document, from Cornell University, on blackboard that will go into more specifics for programmers with a background in the JAVA programming language.

**To get started on the homework and in class assignment starting Thursday, 1/14, please follow the guide below.**

First we need to understand a little about the environment we are using. At this point you are used to using an IDE, Integrated Development Environment, a program that allows you to write your code, compile, assemble, link and debug all within a nice and neat gui enabled interface.

In our class we will write code in an editor, emacs, similar to notepad++ or wrangler. We will then compile, assemble, link, load and debug our programs using commands within the shell(bash) via the LXterminal. This will give you a great respect for IDE’s and help you become more familiar with using Linux.

In reality when you setup your O.S. project you downloaded source code from blackboard in a compressed format, the tar files, next we uncompressed them, compiled and installed them using the *make* command.

Let’s start by writing a C source file. First we need a new directory to work in. Within Lubuntu open LXterminal and type:

mkdir Workspace 🡪 this will create a new folder

cd Workspace 🡪 go into the Workspace directory

At this point we need to install emacs:

sudo apt-get install emacs24 🡪 Press ‘Y’ when prompted

Lets create our first program, by typing:

emacs test.c

a new window will appear type in the following:

#include <stdio.h>

int main()

{

printf(“Hello CS431\n”);

return 0;

}

Now save the file and quit

Lets discuss what everything is

#include <stdio.h> //This is a standard library included with C Language

//the main() function is where PC, program counter, will //initially point to hence my thread of control within

int main() //a process

{

printf(“Hello CS431\n”); //predefined function within stdio.h that

// will print to monitor

return 0; //return value for main ‘0’ means no error other values //usually mean exiting with an error

}

Back at the command line type:

ls 🡪 you should see the file test.c SOURCE CODE

type:

gcc –c test.c 🡪running program **gcc** telling it to compile and assemble test.c

type:

ls 🡪 you should see test.o OBJECT FILE

type:

gcc –o testone test.o 🡪 Linking object file to create executable testone

type:

ls 🡪you should see testone (probably green) this is your EXECUTABLE FILE

to run or execute the program type:

./testone 🡪./ will load and run your program hence becoming a process in RAM!

to clean up your folder you can remove the object file as you no longer need it by typing:

rm test.o

To automate this process of compiling, linking and cleaning up we can create a very simple make file by typing:

gedit Makefile

a new window will appear type in the following:

testmake: test.o

gcc $(CFLAGS) –o testmake test.o

test.o: test.c

gcc $(CFLAGS) –c test.c

clean:

rm –f \*.o testmake

Now save the file and quit

Lets discuss what everything is

//here we tell the Makefile that we want to name the program testmake and we //should link the object file, test.o

//CFLAGS, don’t worry about this at this point but, in short it is a variable //that will allow us to enable/disable features when you run the make script

testmake: test.o

gcc $(CFLAGS) –o testmake test.o

//here we tell the Makefile how to make the object file test.o by compling //the source code test.c

test.o: test.c

gcc $(CFLAGS) –c test.c

//here we tell the Makefile how to clean the directory up so we can rebuild //the program.

clean:

rm –f \*.o testmake

In LXterminal type:

make 🡪this will automatically compile and link the file to an *executable file*

In LXterminal type(yes do it again):

make 🡪this time Makefile realizes the file has not changed, touched, so it will not rebuild the code

In LXterminal type:

make clean 🡪remove all object files and the executable file testmake

In LXterminal type:

make 🡪rebuild the *program*

Let’s install your debugger GDB by typing:

sudo apt-get install gdb 🡪 Type ‘y’ when prompted

There is a guide in blackboard, under course documents you can refer to learn how to use the debugger. Please recall **gdb** will use the debugger for the INTEL/AMD x86 code in your homework and **cs431-gdb** will run the debugger for the MIPS R3000 simulated code in your projects

# Assignment #1

1. Look up ELF executable files and describe to me in no more then one page, roughly 3 paragraphs, what it is and how it works. You must also show a diagram, that you created! That you will refer to in your explanation. You should explain the material as if it was to a child, assume the reader knows nothing.
2. In your book in chapter 2 a problem regarding producers-consumers is presented to you.
   1. Solve the producer-consumer problem with a single threaded solution.
   2. Solve the producer-consumer problem using a multithreaded approach specifically using Semaphores

NOTE: Make sure you include the semaphore library in your source code 🡪 #include <semaphore.h>

Additionally, *when linking make sure you link the object file of your source code and the pthread library 🡪 gcc –o processname objectfile.o -lpthread*

* 1. Solve the producer-consumer problem using a multithread approach

specifically using Mutexes

NOTE: *When linking make sure you link the object file of your source code and the pthread library 🡪 gcc –o processname objectfile.o -lpthread*

**Submitting Your Work**

Your group will submit two files.

groupname.txt and groupname-HW1.tar.gz

**(Where groupname = your group name)**

Include a file called groupname.txt with all of the group member names. Additionally, as a group you will compress your folder and submit one copy to me via blackboard.

To compress your folder use the following command:

**(Recall groupname = your group name)**

tar –zcvf groupname-ASST1.tar.gz (DIRECTORY)

Your DIECTORY should include groupname.txt and 3 sub-directories, labeled a, b, and c.