Inlab 5 - Build Tools, PyNetworking

Do the inlab in groups. Though the inlab is un-graded, the outlab tasks may directly extend the inlab tasks. Therefore you may want to do in-lab questions properly.

P1. C you again

C was one the first human readable programming languages designed for people in all domains (business and scientific community). C is an imperative procedural language created by Dennis Ritchie. Unix and linux kernels are built in C (and assembly) language. Due to this C code runs faster than any other language. But still C was a very low level language, meaning that bigger **abstractions** were not readily available in it.

To deal with this Bjarne Stroustrup created C++ with objectives of speed and abstraction in mind. The core of his creation was something called a **class.** Thus C++ became one of the first object oriented languages.

C has a simple architecture and few paradigms which makes it simple to use. Remember the zen of python (*There should be one—and preferably only one—obvious way to do it.*) But due to the sheer flexibility of C++ (with several paradigms it posses) it became very difficult to maintain standards in bigger C++ projects.

One of the biggest misconceptions in this area is that C and C++ are the same language. C is not object oriented while C++ is. Acknowledge the difference.

Today we will learn about compiling & linking C/C++ code in various ways using raw commands, Makefile and CMake. This is one of the ways to gain insights into how exactly things work in C/C++ world.

Create a Makefile named **rawmake** (can be run using **make -f rawmake**) and fill it up as instructed in the below tasks

Task 1

Add a rule to compile **helloworld.cpp** to form **helloworld** executable

Task 2

Add a rule to compile **usespthread.cpp** to form **usespthread** executable. This file uses pthread library used to create and manage threads. You will be using this a lot in your Operating Systems course.
In order to properly compile this file you need to special use a flag for g++.

Task 3

In this task we build a library using files **myengine.hpp** and **myengine.cpp.** There are actually two types of libraries categorized based on the way the library is linked to main file. Dynamic and Static libraries. This and This are good readups on compiling static and dynamic libraries. This is a good readup which says difference between them.

Use the given resources to add rules to make dynamic and static libraries. Dynamic library created should have name **libMyEngineDynamic.so** and similarly static **libMyEngineStatic.a**

Task 4

In this task we try to install the above built libraries into the system (requires sudo permission). Add a PHONY rule named **installdynamic** which installs dynamic version (.so file) to **/usr/local/lib/** and corresponding headers to **/usr/local/include/**

Do the same for static version (.a file) by creating a rule **installstatic**

Task 5

Now finally we use the installed libraries in **mygame.cpp**
Add a rule to compile **mygame.cpp** with the static library installed in the previous task and produce binary name **mygamestatic**.

Task 6

Add a rule to compile **mygame.cpp** with the dynamic library installed in the previous task and produce binary name **mygamedynamic**.

Task 7

Also add a PHONY rule to clean all the generated intermediates and binaries (.o .a. .so and other binaries)

P2. Request to Moodle

Task 1

Write a script **moodle.py** to login to your moodle account. Get the LDAP ID and Password from stdin. Print the message whether the login is successful or not.
(Hint: requests, getpass, re)

Example 1:
>$python moodle.py
>Enter LDAP ID: xxxx
>Enter Password: xxxx
>Logged in successfully.

Example 2:
>$python moodle.py
>Enter LDAP ID: xxxx
>Enter Password: xxxx
>Failed to log in.