CSL303 | Assignment-1 | Due 14/Sep/2021 11:59 PM | 100 points

- Important instructions for code submissions are here: https://goo.gl/IMWvdF
- Grading scheme to be followed is available here: https://goo.gl/52D82g
- Assignment description may be underspecified to allow some room for exploration and creativity.
- Your submission should be packaged as a zip file named **exactly** in this format: CSL303-[your entry no.]-[assignment no.].zip.

We would like to implement a program that mimics how processes are managed by the *dispatcher* in an OS. The high-level structure of the program is shown in Fig. 1 below. The program has to be developed in C programming language on a linux OS.

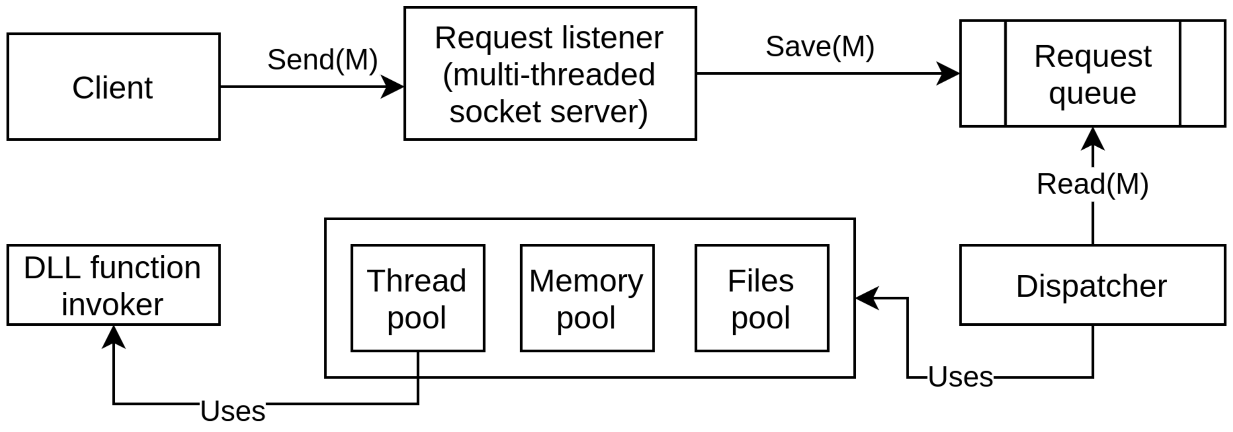

Fig. 1 (High-level context)

The program has the following main modules:

Request listener

It is a program which listens for incoming socket connections from clients. A client sends request messages to the request listener over the socket connections. The request message has the following structure:

```
struct request {
     char* dll_name; // Name of a dynamically loaded library
     char* func_name; // Name of a function to call from the DLL
     char** func_args; // Arguments to pass to the function
};
```

A request message represents a request to invoke a function from a dynamically loaded library.

On receiving a socket connection from a client, the listener spawns a new thread to handle the connection and the messages received via that connection. This allows the listener to handle multiple requests from clients without blocking. The request messages received over a socket connection will be stored in a request queue. The client is sent an acknowledgement on successfully enqueuing the request. The port number on which the listener listens should be supplied as a CLI argument to the program.

Request queue
It is an in-memory data structure which will hold the requests received by the request listener. It will be a bounded queue having a fixed maximum size, say 100.

Dispatcher
This is the main part of the whole program. Its main purpose is to take out request messages from the request queue, and then execute the function specified in the request message. The function has to be executed in a separate thread. There is only a fixed number (e.g., 8) of threads allowed to be used. Similarly, the amount of memory (e.g. 10MB) and the number of files (e.g. 10) that can be open at any point of time in these threads (taken together, not individually) are fixed. These numbers should be taken as CLI arguments when running the program.

The functions to be executed (i.e. the one specified in the request message) are to be loaded dynamically (see https://tldp.org/HOWTO/Program-Library-HOWTO/dl-libraries.html).

Dynamically loaded library (DLL) function invoker
The functionality of DLL function invocation should be encapsulated in a module of its own for better understanding and maintainability. For testing the program, you should use one of the standard DLLs (e.g. the math library at /lib/x86_64-linux-gnu/libm.so.6) available on a linux OS.

You have to write proper unit tests to allow complete testing of your program.