Design Introduction

This program consists of one part: the peer node acted both as index server, file server and client. The client accepts user command and distribute operations about hash table like get, put to actual index server. The index server handle the hash table operations and give feedback to clients. And the file server provides file download.

For the whole program:
We try to use the OO design principle to complete this program. So there are some common class defined, such as socket class, thread class, config class and so on.
For each those class, they do their job automatically and the user can just call its interface to finish the job.

Generally, each peer node will have two processes. The purpose is that the first process can monitor the second process in where the multi threads work.
It is a robost consider. Also the server can work in the daemon mode.

For the peer index server and file server part:
There would be one thread listen on the port to receive the connections. I register epoll event for this listen. So the incoming packets can be reached immediately and the cpu time is low. Behind the listen thread, there is a thread pool consists of dynamic number of threads (specifed by config file, so it is static now), to handler all the requests. All the threads share one Resource Manager. The Resource Manager manage all the info collect from peer clients and usef for Register, Search and so on. It is thread safe, since it used the Mutex to lock and unlock each operation. The message between the listen thread and the thread pool is the connected socket. The thread pool would directly receive message from this socket, do processing, and sned back message through this socket. Each pool thread would wait for a sem to begin the job. Also between the client and server, there are types of commands they used to communicate. They are the self-defined struct.We would not close those connections as to maintain it as a long connection.

For the peer client part:
There would be one thread to handler the user interactive interface. We would maintain all the connections to server, so there is no need to reconnect each time.

About the backup mechanism:
In the assignment, we automatically choose the backup nodes for each file in each node. The strategy is that for each file, there will be exactly one index server. Thus we make use of this point and choose the backup nodes as those two following the index server. So when the index server go down, we can automatically try its following neghbor node. This is a relatively static method, while I may improve it to a better mechanism next time.

Some tradeoffs made:
Instead of for each connection to generate a new thread, this program choose the thread pool, in purpose of system resource management and control. In some case like there are hundreds of clients, for each client one thread would be perfect, and this would give all the burden to system resource limitation. The thread pool solution says that when we start the program, we can specify how many threads work in this pool. It needs us to evaluate this number. The advantage is that we may resue the resource more efficient.

Improvements and extensions:
For this DHT sytem, I think those aspects are important:
1. Peer Hash Tbale dynamic Backup:

In the DHT, each node is important and necessary. So we should have some strategy to keep the whole system robost.

I have implemented the static backup in which the backup node is known at very original, and the number of backups is hardcode. It would be more convinent to let the backup system dynamic, that means users do not need to concern about it.