COMP90015 Distributed Systems Assignment 1

Student Name: Qingyang Hong
Student ID: 629379
Supervised by: Professor Rajkumar Buyya
Date: 25 August 2013

Multithreaded Dictionary Server

Section 1: Problem Context

The task aims to build a client-server architecture remote dictionary system, involving fundamental technologies of TCP/UDP Sockets and Multithreading. In term of the functionality of a remote dictionary system, user should be able to look up words on a machine different from the one, which's running the searching program. Most of time a remote application will be accessible to multiple users, as it doesn't only run for a local user.

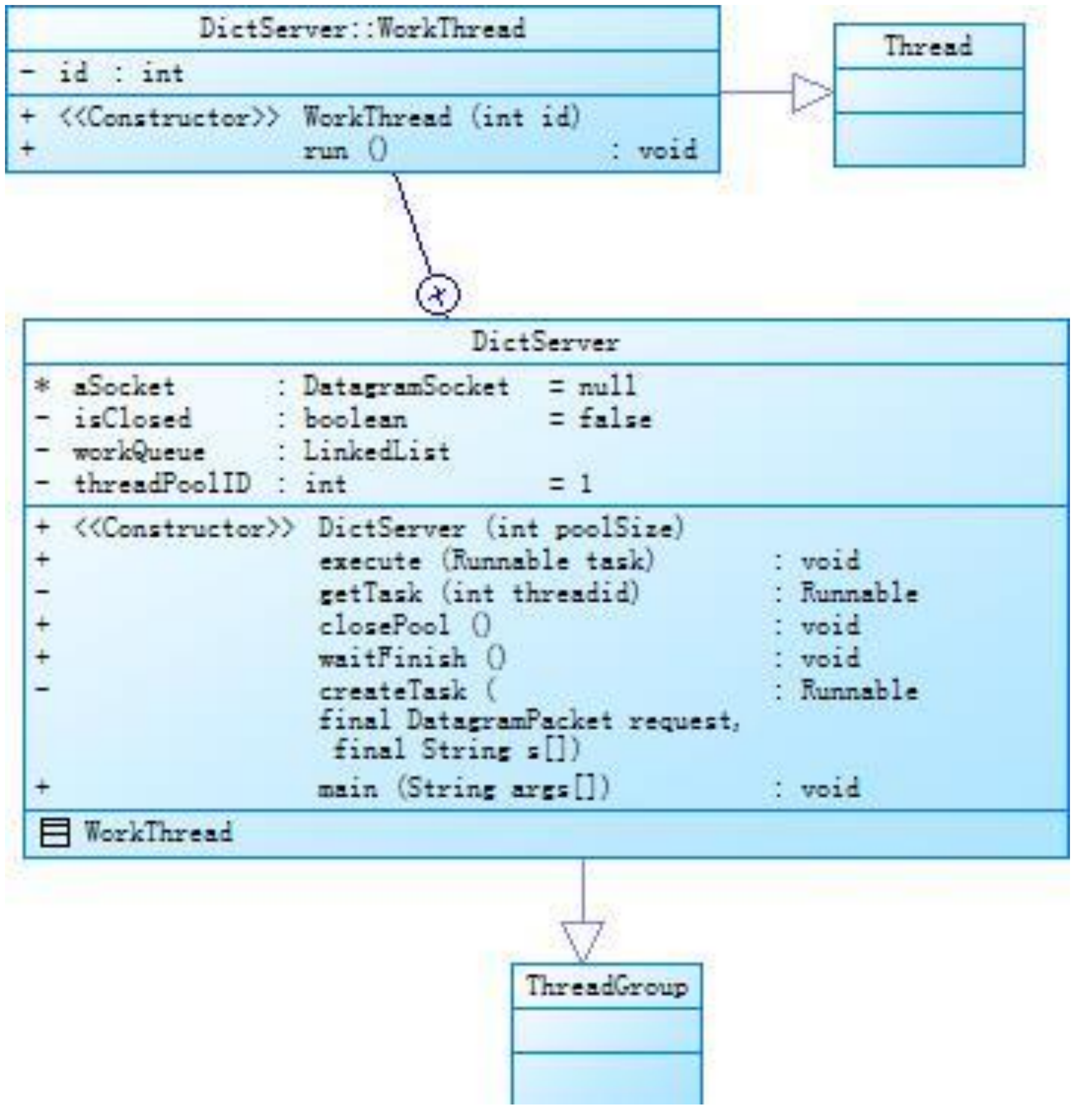
To manage the first part of the problem, a client-server architecture is introduced in this context. The system is based on a remote message transmission mechanism. Two machines can communicate by sending messages to each other, when being aware of opposites' IP addresses. So the network becomes a very fundamental medium for communication in this case. As it is also better to run on multiple platforms, the system is programmed in JAVA language.

To manage the second part of the problem, with considering of the design in first part, a UDP Socket is adopted. UDP does not maintain connection state. For this reason, a server can typically support more clients with UDP rather than TCP. Also in the multi-user context, a multithreaded worker-pool architecture is adopted to the server. The main functionality of the worker-pool is to create several worker-threads in the worker-pool to execute multiple tasks from different clients. The system will perform concurrency in the server worker-pool. In consideration of the limited resource on the server, a manageable number of worker-threads are established, so that scheduling mechanism is also needed to support the worker-pool.

Some other problems are generated by using UDP. Since it isn't based on connections, data packets aren't guaranteed to transmit between clients and the server. In this new context, a resending mechanism should be introduced in the system.

Section 2: Architecture

In this section, explicit representations of the system architecture are introduced by using both UML Class Diagram and Sequence Diagram (Interaction Diagram).


DictServer::WorkThread
- id : int
+ <<Constructor>> WorkThread (int id)
+ run () : void
Thread
DictServer
* aSocket : DatagramSocket = null
- isClosed : boolean = false
- workQueue : LinkedList
- threadPoolID : int = 1
+ <<Constructor>> DictServer (int poolSize)
+ execute (Runnable task) : void
- getTask (int threadid) : Runnable
+ closePool () : void
+ waitFinish () : void
- createTask (final DatagramPacket request, final String s[]) : Runnable
+ main (String args[]) : void
WorkThread
ThreadGroup

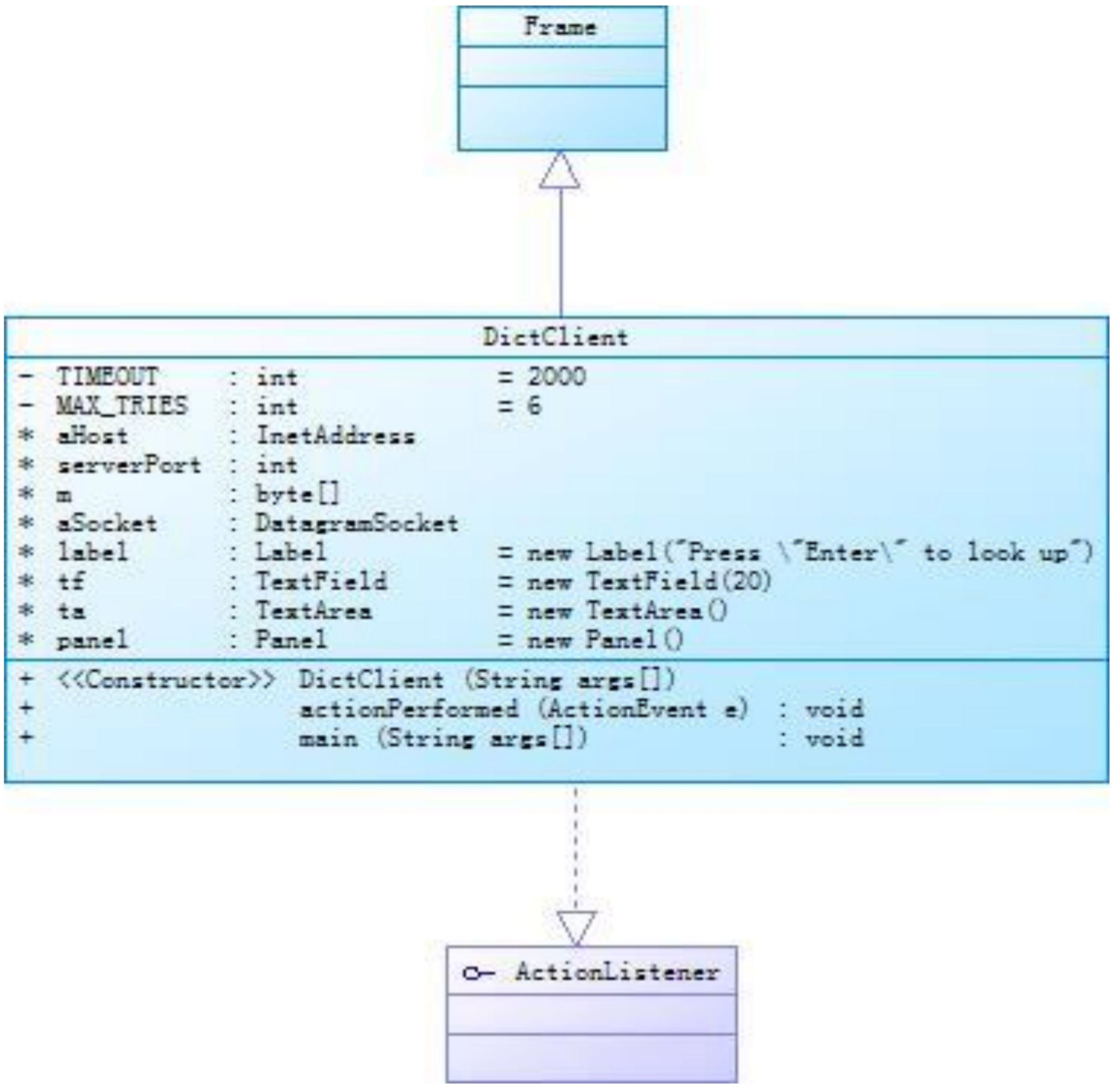
Information from the Server side Class Diagram:

1.By using UDP, DictServer Class instantiates DatagramSocket Class as a socket.

2.A workqueue is defined to support scheduling tasks.

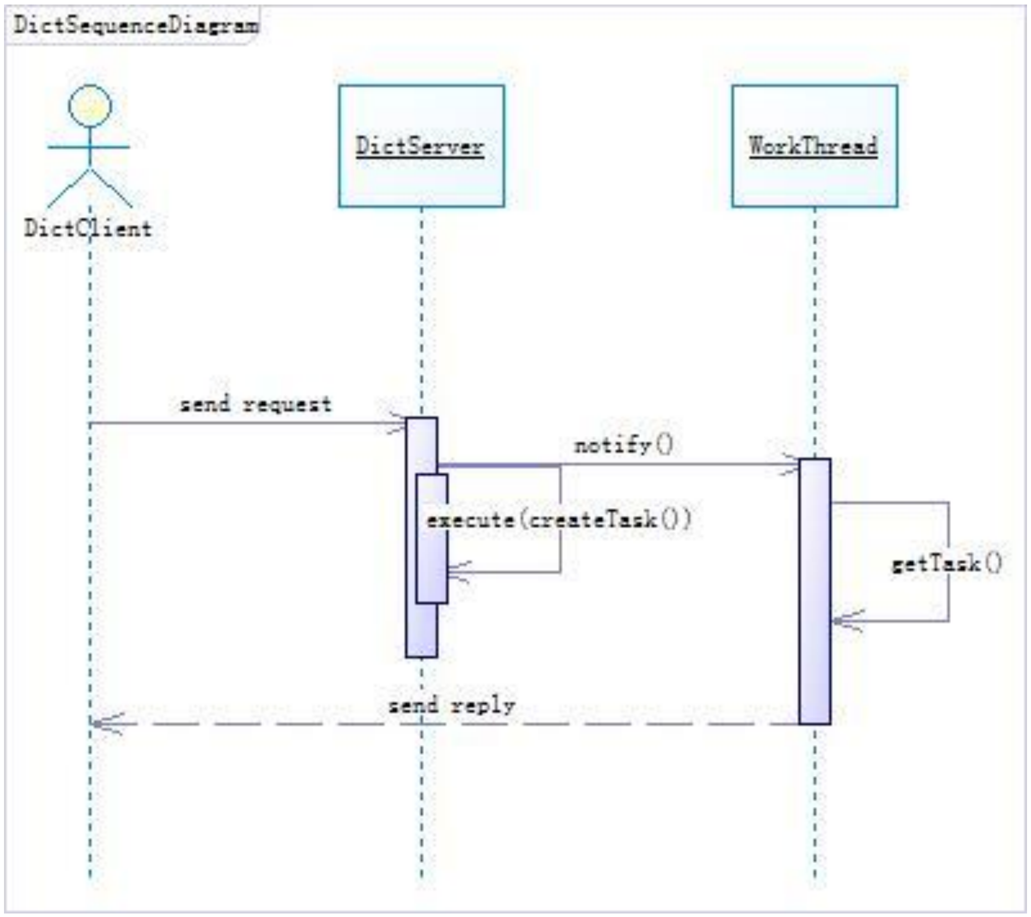
3. DictServer Class has a inner class, WorkThread to instantiate workthread objects based on pool size.

4.Though DictServer Class extends the ThreadGroup Class, most methods are customized in DictServer Class. Two inherited methods are used in auxiliary methods, closePool() and waitFinish().


Frame
DictClient
- TIMEOUT : int = 2000
- MAX_TRIES : int = 6
* aHost : InetAddress
* serverPort : int
* m : byte[]
* aSocket : DatagramSocket
* label : Label = new Label("Press \"Enter\" to look up")
* tf : TextField = new TextField(20)
* ta : TextArea = new TextArea()
* panel : Panel = new Panel()
+ <<Constructor>> DictClient (String args[])
+ actionPerformed (ActionEvent e) : void
+ main (String args[]) : void
ActionListener

Information from the Client side Class Diagram:

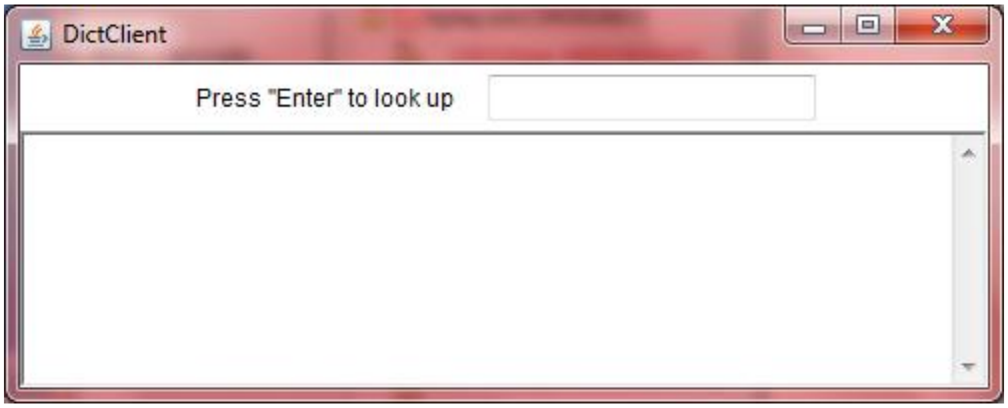
1. DictClient uses a GUI design.
2. Instants TIMEOUT and MAX_TRIES are defined for resending.
3. aHost and serverPort are to support sending requests.

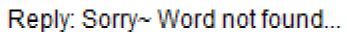


Communication sequence from the Sequence Diagram: When DictClient sends a searching request to DictServer, DictServer invokes execute, which adds a new task into the workQueue (a LinkedList) and notifies a WorkThread to getTask. A WorkThread is in waiting state after started, as no task exists. When being notified, the WorkThread invokes getTask. If it can get a task from the workQueue, it will be in running state. If the workQueue is empty, the WorkThread keeps waiting. After executing a task, the WorkThread sends a reply.

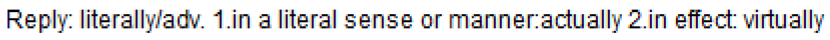
Section 3: Achievement

Achievements are demonstrated by screenshots and a critical analysis for functionality. The server program is set up on "cat.csse.unimelb.edu.au" to simulate remote context.

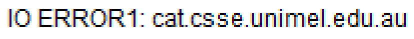




Reply: Sorry~ Word not found...



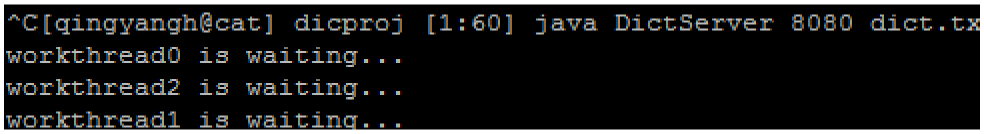
Reply: literally/adv. 1.in a literal sense or manner:actually 2.in effect: virtually



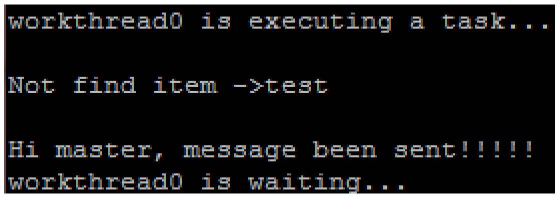
IO ERROR1: cat.csse.unimel.edu.au

• DictClient GUI has three kinds of replies, a not-found reply, a found reply or a error reply. As using a UDP architecture, the system has involved a resending mechanism. If a packet can't be received after reaching the TIMEOUT , DictClient will try to resend it again. When failing to receive a packet after trying resending MAX_TRIES times,

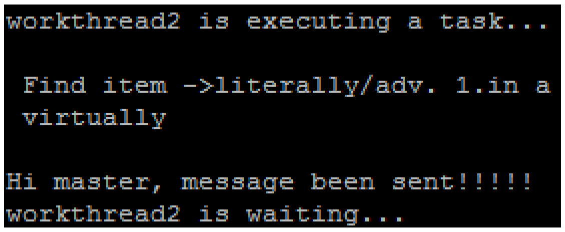
DictClient will believe and return "No response, host unaccessible...".



```
^C[qingyangh@cat] dicproj [1:60] java DictServer 8080 dict.tx
workthread0 is waiting...
workthread2 is waiting...
workthread1 is waiting...
```

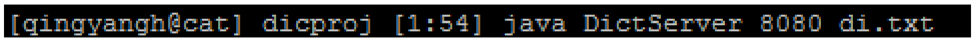


```
workthread0 is executing a task...

Not find item ->test

Hi master, message been sent!!!!!
workthread0 is waiting...
```

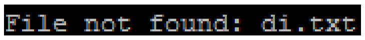


```
workthread2 is executing a task...

 Find item ->literally/adv. 1.in a
 virtually

Hi master, message been sent!!!!!
workthread2 is waiting...
```

• DictServer creates three worker-threads (a large number of 350 has been tested on department server). Worker-threads are waiting when no task is in workQueue. Each task has two result states, found or not found. After sending the reply worker-threads will inform to has completed sending and go back to waiting state.

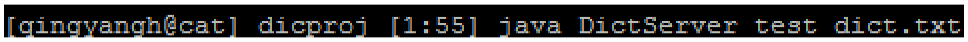


```
[qingyangh@cat] dicproj [1:54] java DictServer 8080 di.txt
```

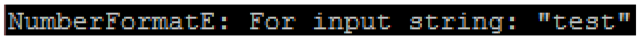


```
File not found: di.txt
```

• handling a FileNotFoundException

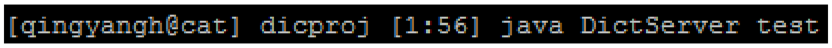


```
[qingyangh@cat] dicproj [1:55] java DictServer test dict.txt
```

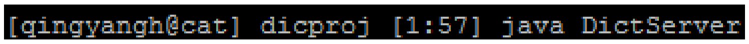


```
NumberFormatE: For input string: "test"
```

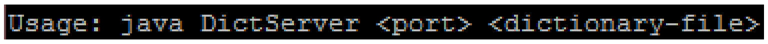
• handling a NumberFormatException



```
[qingyangh@cat] dicproj [1:56] java DictServer test
```



```
[qingyangh@cat] dicproj [1:57] java DictServer
```



```
Usage: java DictServer <port> <dictionary-file>
```

• handling the error when the number of command-line arguments aren’t correct

• DictClient and DictServer are also competent to handle a SocketException, a IOException or a InterruptedException.

Section 4: Conclusion

To summarize, the system firstly cater to the fundamental requirements of using UDP socket, a thread pool with reasonable concurrency and proper handling of errors. With UDP, a client won't get an error, when the server is shut down. Secondly, a refreshing designed GUI is provided with concise interaction between clients and the server. To cope with the UDP-unguaranteed problem, a resending mechanism is added to avoid flaws in transmission and help to verify connectivity. Thirdly, a workQueue structure is in the system to support task scheduling. Besides, some necessary auxiliary methods are incorporated. As a result, the whole system is portable, secure, user-friendly and of high focus on utility.