**Computer Systems Technology**

British Columbia Institute of Technology

COMP 8005 - Assignment2- Report

Albert Huang &

Aiyan Ma

Feb 26, 2018

Revision Control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Date** | **Version** | **Revised by** | **Remarks** |
| 1 | 2018-Feb-26 | V0.1 | Aiyan,Ma | The initial draft for Test plan and test case |
| 2 | 2018-Mar-03 | V1.0 | Albert Huang | Update for the test execution and report. |

# Table of Contents

Table of Contents 2

1 Purpose 3

2Background 3

3 Executions and Observations 4

4Summary 5

5Conclusion 5

6 Client Design 5

7 Further Work 5

1 The message middle ware 5

2 Other EPoll Provider 5

8 Reference 6

9Appendix 6

# 1 Purpose

The purpose of this work is to design and compare the 3 models of doing high performance servers:

1. Multi-thread ( traditional way)
2. Poll ( Level ttrigger)
3. EPoll ( Edge trigger)

To archive it, Three servers are designed and compared by using java nio library.

For EPoll and Poll server, message cache, state machine with java nio are utilized with the intention to enhance the scalability and performance ( see details in the design work).

For the Multi thread server, one thread per connection model is used to accept and handle the client request.

The client, named SuperClient, also designed to connect and send packets in such a way that connections and packets are generated as much as possible.

For these 3 models, the scalability(connection) and the performance both are analyzed.

# 2Background

Because of the computer technology evolution, the Multi thread is the traditional way in java j2SE1.1 nearly 20 years ago, then java nio launched in J2SE1.4 and become popular in Java5. From these facts, the hypothesis is, the later technology, as they declare, the more high performance can be archived.

Similar case happen for Python to introduce the multi-thread, Poll and Epoll technology, although maybe not so explicitly.

There fore, from scalablility and performance view, we can anticipate that

**EPoll>Poll>Multi-Thread.**

That’s our hypothesis.

We want to verify to confirm or get other findings thought our design and test.

To utmost the resource of the host can provide, the ulimit of

# 3 Executions and Observations

To execute it, in Java side, the following software modules are composed:

|  |  |  |  |
| --- | --- | --- | --- |
| Software Modules | Function | Keynote |  |
| EPoll.java | For EPoll server | EPoll optimized with message cache; state machine implemented. Details can be found in the documentation of design work. |  |
| Poll.java | For Poll Sever | Poll optimized with message cache; state machine implemented. |  |
| MTServer.java | For Multi-thread server | Traditional Server |  |
| SuperClient.java | For Client with nio, optimized for connection creatation | Client with java nio to have the ability to send packets: SIZE, interval, connections |  |
| Super2client.java | For Client with java.io, optimized on packet rate | Client with java io to have the ability to send packets: SIZE, interval, connections |  |
| Util.java, Util\_tasks,java | Util class for abover modules | CPUTasks with aes256 encryption/decryption; |  |

# 4Summary

# 5Conclusion

# 6 Client Design

Client plays important role in this task,

# 7 Further Work

The following technology and Epoll provide can be used for further work:

## 1 The message middle ware

The message middleware such as JMS ( Java Message Service <https://en.wikipedia.org/wiki/Java_Message_Service> ) can be leveraged to further bench mark purpose.

## 2 Other EPoll Provider

Other than the EPoll provider from SUN, there are several other vendors, per the research from intrenet, Java EPoll provider named wizzardo <https://github.com/wizzardo/epoll> also did a lot of work on Selector Provide for epoll.

Also, IBM also provides the epoll provider for java nio : -Dcom.ibm.nio.rdma.EPollSelectorProvider

<https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/appendixes/cmdline/dcomibmniordmaepollselectorprovider.html>

# 8 Reference

# 9Appendix