Name: Fan Feng
Assignment: 3
Course: CPSC 433/533

~~- [25 pt in total] Protocol and functions~~
~~ a. [5 pt] Method: HTTP 1.0 GET method (URL mapping)~~
~~ b. [5 pt] Header: Last-Modified, If-Modified and User-Agent~~
~~ c. [10 pt in total] Support CGI program~~
~~ c.1. [4 pt] Detect executable~~
~~ c.2. [6 pt] Setup Environment variables and direct I/O~~
~~ d. [5 pt in total] Heartbeat monitoring~~

- [40 pt in total] Implementation structure
~~ a. [3 pt] Sequential~~
~~ b. [3 pt] Per request thread~~
~~ c. [3 pt] Thread pool with shared welcome thread~~
~~ d. [3 pt] Thread pool with a shared queue and busy wait~~
~~ e. [3 pt] Thread pool with a shared queue and suspension~~
 f. Asynchronous I/O using select [12 pt]
 g. Asynchronous I/O using Java 7 Async channel future/listener [13 pt]

~~ [15 pt in total] Performance benchmarking~~
~~ a. [6 pt] Clear description of benchmarking methodology~~
 b. [9 pt] Best throughput exceeds 10 Mbps

~~ [20 pt in total] Comparison and exploration of other designs~~
~~ a. 2 points for each question~~

- [35 pt in total potential] Bonus points
 a. [10 pt] Beat or close to Apache performance
 b. [25 pt] The fastest server

Part 1a:

SHTTPTestClient.java
RequestSender.java
Statistics.java

Part 1b:

Configuration.java
FileCache.java
HTTPServerSequential.java
HTTPServerPerThread.java
RequestHandler.java

HeartbeatMonitor.java

Design of heartbeat monitoring:
Monitor <MonitorClassName> is an optional parameter in config.txt.

In RequestHandler.java, when mapping URL to file, if a request queries a virtual URL (/load), the server randomly accepts/rejects the request with a pre-defined acceptance rate.

If a request is accepted, the server responds with status code 200 and "Heartbeat Monitor" as response body. Otherwise, status code 503 is returned.

See mapURL2File() in RequestHandler.java and HeartbeatMonitor.java for details.

Part 2a:
Thread pool with service threads competing on welcome socket:
HTTPServerSharedWelcomeSocket.java
ServiceThreadSharedWelcomeSocket.java

Thread pool with shared queue busy wait:
HTTPServerSharedQueueBusyWait.java
ServiceThreadSharedQueueBusyWait.java

Thread pool with shared queue suspension:
HTTPServerSharedQueueSuspension.java
ServiceThreadSharedQueueSuspension.java

Part 2b:

Part 2c:

Part 2d: Comparison of Designs
(1): Comparison with xsocket
(i).
http://xsocket.sourceforge.net/core/tutorial/V2/TutorialCore.htm:
xsocket allows (number of CPUs + 1) dispatchers.

Line 105 in IoSocketDispatcherPool.java:
The dispatchers share workload in a round-robin fashion.

(ii).
Line 208 – 258 in IoSocketDispatcher.java:
While the dispatcher is open, the selector times out for 5 seconds, performs the register handler tasks, and calls handleReadWriteKey() on new events. After that all socket handlers are deregistered, and the selector is closed.

(iii).
In EchoServerTest.java: EchoServer server = new EchoServer(0);
In EchoServer.java: private IServer server = null;
IHandler hdl = new EchoHandler();
server = new Server(listenPort, hdl);
Line 210 in HandlerAdapter.java:
performOnData(connection, taskQueue, ignoreException, handler);
Line 242 in HandlerAdapter.java: handler.onData(connection);

(iv).
xsocket uses IIdleTimeoutHandler to implement idle timeout of a connection.

(v).
The library tests one class at a time. Specifically, EchoServerTest.java tests server side and both sides respectively.

We can test the server with idle timeout by establishing a connection to it without sending anything. The server should be able to kill the idle connection automatically.

(2): Comparison with Netty
(i). https://netty.io/wiki/user-guide-for-4.x.html

There are two event loop groups. The first one, often called 'boss', accepts an incoming connection. The second one, often called 'worker', handles the traffic of the accepted connection once the boss accepts the connection and registers the accepted connection to the worker.

Synchronization among them is achieved my multithreading.

(ii). https://netty.io/4.0/api/io/netty/channel/ChannelFuture.html

sync() waits for this future until it is done. One can implement it using the isDone() method from the superinterface java.util.concurrent.Future.

(iii). https://github.com/netty/netty/wiki/Using-as-a-generic-library

One key difference between ByteBuf and ByteBuffer is that ByteBuf's life cycle is bound to its reference count, whereas ByteBuffer's life cycle is not, and it relies on Java's garbage collection.

(iv).
https://netty.io/4.0/xref/io/netty/example/http/helloworld/HttpHelloWorldServerInitializer.html
https://netty.io/4.0/xref/io/netty/example/http/snoop/HttpSnoopServerInitializer.html

HTTP Hello World Server: (SslHandler), HttpServerCodec, HttpHelloWorldServerhandler
HTTP Snoop Server: (SslHandler), HttpRequestDecoder, HttpResponseEncoder, HttpSnoopServerHandler

(v).
https://netty.io/5.0/api/io/netty/channel/ChannelPipeline.html

A ChannelPipeline contains a list of Channel Handlers which handles or intercepts inbound events and outbound operations of a Channel.

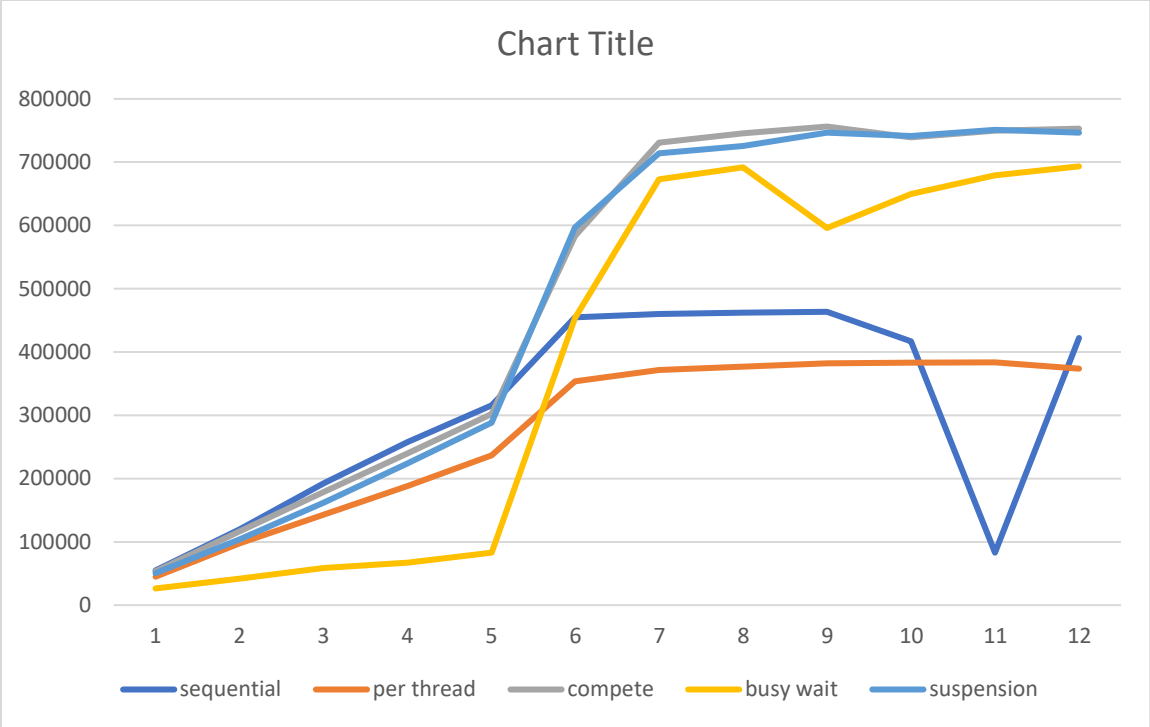
An I/O event is handled by a ChannelHandler and is forwarded by the ChannelHandler which handled the event to the ChannelHandler which is placed right next to it.
A ChannelHandler can also trigger an arbitrary I/O event if necessary. To forward or trigger an event, a ChannelHandler calls the event propagation methods defined in ChannelHandlerContext.

Part 2e: Performance Benchmarking

Benchmarking methodology:
For each server, run different number of threads (1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 50, 60) for 60 seconds. Record the data throughput, and plot data throughput against number of threads.

Run Apache server on zoo by making -server zoo.cs.yale.edu, and changing URLs to follow /classes/cs433/web/www-root/html-small/doc1.html. Add data throughput against number of threads to the above plot.

Conclusion:



Sequential server and per-thread server perform the worst. The three thread pool servers have better performance, and perform roughly the same.

Data throughput in the plot is in bytes per second. The best data throughput equals about 750 KB/s = 6Mbps.