Contents

Chapter 1

Overview

Welcome to BMASockets Core Server. The server core was developed to provide a quick path to developing your server requirements on a high performance Linux platform network. The design can be used to develop customing gaming platforms or you can use any of the existing protocol session handlers to implement your own compact high performance server using existing known protocols.

The BMASockets Core Server provides existing handlers for the following protocols:

1. HTTP and HTTPS
2. SIP and SIPS
3. HTTP streaming server
4. HTTP Web Sockets handler

The focus of the design is to extend the capabilities of two core objects to create the interface to your own implementations.

Chapter 2

Linux epoll Interactions

Linux provides an set of system calls to open a socket that is used to manage the networking requests for multiple sockets from a single process. You can find plenty of materials on the internet on epoll and how this function works. The Core Server is designed around the concept of the socket (BMASocket) and the handling of accepting connections on a binding socket and managing the individual client sockets that are *accepted*.

TCP and UDP are both supported by the Core Server. The differences in managing these socket types is abstracted through the use of the sessions object (BMASession). This abstract class cannot be instantiated but is used instead to extend into a customizable session object that will be used to manage the protocol for the connected session. At the session level there is no difference between the underlying socket type, whether it be UDP or TCP.

The TCP side of the fence incorporates the connection oriented design to provide the sessions to the higher levels. Each client represents a socket that has connected through the *bind* and *accept* system call method. Conversing with a client in a TCP session returns the appropriate data through that socket connection.

The UDP side of the fence incorporates session objects that are based upon connectionless packets sent to a single receiving socket. The UDP server (BMAUDPServerSocket) maintains a list of sessions which are managed according the sending address of the packets received. Each remote address represents a different client and interactions back to the session are sent through the single socket to the corresponding remote address. This provides a seamless session to the higher level activities of the server.

The interface provided through the session appears the same regardless of the socket type. This affords the developer the opportunity to write UDP or TCP socket handlers with no knowledge of the those protocols. Building a game server would be the same for either type and both types could be enabled simultaneously.

Chapter 3

The Core Server

In order to provide flexibility to the Core Server several design factors have been put in place. Creating a new server that handles a custom protocol requires the extension of only two objects. These are the BMATCPServerSocket or BMAUDPServerSocket, depending on the type desired, and the BMASession object.

When extending the BMATCPServerSocket all that is needed is to override the getSocketAccept() method to return an extended BMASession object. This basically tells the server to spawn a new session of a particular type for every new connection to the bound TCP port.

The extended BMASession object can override the onDataReceived() method to handle the incoming requests for the socket. An entire application structure could be built upon this mechanism to handle complex protocols with the client.

Chapter 4

Sample Server Example

This chapter specifies the approach to extending and creating a customized server. This example is focusing on a server used in gaming environments. The gaming clients connect to the server to interact with other clients.

The BMA Server Core can provide all the features needed of a multiport high demand server. The implementation of epoll combined with the ability to manage job control load through the use of multiple threads provides a robust request handling environment for all socket based architectures.

As the BMAEPoll object is started it will spawn the specified number of threads which will in turn begin the socket handling functions necessary to manage all network requests in the program. The main program itself is not used but must not be allowed to return or end so it can be used to handle other non sockets related processing in the application.

Additionally, this server example provides a console that is accessible through a Telnet style server and does not currently incorporate any encryption (TLS) or login authentication. This will be changing in the future.

4.1 The Server

The server provides a few interesting features that may not be readily apparent when reading through the documentation.

Since each BMATCPServerSocket also inherits from BMACommand the server can override a routine to output data relivant to the type of server you are creating.

When creating the server you are asked to provide a command name as a parameter. The inheriting server object can then obtain a list of connected clients from a console object by typing this name in on a command line.

```
#ifndef BMAConsoleServer_h__
#define BMAConsoleServer_h__

#include "includes"
#include "BMATCPServerSocket.h"
```

```
#include "BMACommand.h"
class BMATCPSocket;

class BMAConsoleServer : public BMATCPServerSocket {

 public:
   BMAConsoleServer(BMAEPoll &ePoll, std::string url, short int
    port);
   ~BMAConsoleServer();

   BMASession * getSocketAccept();

   void registerCommand(BMACommand &command);

   int processCommand(BMASession *session) override; ///<Output the
     consoles array to the console.

   std::vector<BMACommand *> commands;

};

#endif
```

```
#include "BMAEPoll.h"
#include "BMAConsoleServer.h"
#include "BMAConsoleSession.h"
#include "BMACommand.h"

BMAConsoleServer::BMAConsoleServer(BMAEPoll &ePoll, std::string url
    , short int port)
    : BMATCPServerSocket(ePoll, url, port, "consoles") {
//    ePoll.log.registerConsole(*this);
//    ePoll.log.write(0) << "BMAConsole initializing..." << endl;
}

BMAConsoleServer::~BMAConsoleServer() {

}

BMASession * BMAConsoleServer::getSocketAccept() {
   return new BMAConsoleSession(ePoll, *this);
}

void BMAConsoleServer::registerCommand(BMACommand &command) {
   commands.push_back(&command);
}

int BMAConsoleServer::processCommand(BMASession *session) {

   std::stringstream out;
   int sequence = 0;

   for(BMASession *session : sessions) {
      out << "|" << ++sequence;
      out << "|" << session->getClientAddressAndPort();
      out << "|" << std::endl;
   }

   session->write((char *)out.str().c_str(), out.str().size());

   return 0;
}
```

4.2 The Session

```
#include "includes"
#include "BMAEPoll.h"
#include "BMAMP3File.h"
#include "BMAConsoleServer.h"
#include "BMATCPServerSocket.h"
#include "BMAStreamServer.h"
#include "BMAHTTPServer.h"
#include "BMASIPServer.h"
#include "BMAHTTPRequestHandler.h"
#include "BMAMP3StreamContentProvider.h"
#include "BMATimer.h"

int main(int argc, char **argv) {

   std::string ipAddress = "0.0.0.0";

   BMAEPoll ePoll;
   ePoll.start(4, 1000);

   //------------------------
   // Testing TCP server.
   //------------------------

   BMATCPServerSocket tcpServer(ePoll, ipAddress, 1028, "users");

   //--------------------------
   // MP3 Streaming Server
   //--------------------------

   BMAStreamServer stream(ePoll, ipAddress, 1032, "listeners");
   BMAMP3File tester(stream, "../extravaganza.mp3");

   //---------------
   // HTTP Server
   //---------------

   BMAHTTPServer http(ePoll, ipAddress, 1080, "http");
   BMAHTTPRequestHandler handler1(http, "/");

   //--------------
   // SIP Server
   //--------------

   BMASIPServer sip(ePoll, ipAddress, 5061, "sip");

   //-----------------------------------------------------------------
   // Console controller so the program can be monitored through
   // a telnet session. BMATCPServerSocket can be registered as
   // a command and will report its status when the command is
   // entered on the console.
   //-----------------------------------------------------------------

   BMAConsoleServer console(ePoll, ipAddress, 1027);
   console.registerCommand(ePoll);
   console.registerCommand(console);
   console.registerCommand(http);
   console.registerCommand(sip);
   console.registerCommand(stream);

   ePoll.start(4, 1000);

   while(true)
     sleep(300);

   ePoll.stop();

}
```