# CS 360 Special Topics: Computer Networking

**Spring 2007** **DUE: Mar 20, 2007**
**Assignment #3** **20 points**

## Programming a Multithreaded TCP Server

The focus of this assignment is to understand and use Internet-domain sockets and TCP for communication between a client and server. The server produced for this assignment will be multithreaded.

## The Network Calculator

You will need to take the code produced for your first assignment and adapt it to use TCP rather than UDP. Additionally, the server needs to be multithreaded (using the pthread library) to allow it to handle simultaneous connections. The protocol and MathPacket will change for this assignment. These changes are detailed below.

Previously, the client sent a single packet to the server to request that a mathematical operation be performed. In the new protocol, the client can send a number ( < 17) of packets requesting mathematical operations to be performed. This series of packets is called a *calculation stream*. The packets are to be handled in order and the result of packet #N will be used as Operand One for the operation requested by packet #(N+1), etc. This requires that the server retain some state (the result of the previous operation and the sequence number for the next expected packet) for each connection. Bits in the MathPacket will denote the sequence number for each packet, if that packet is the first packet in a connection stream, and if that packet is the last packet in a connection stream.

When the last packet in a connection stream is received by the server, the server needs to send a "Result of [Operation]" packet to the client. The PackType of that result packet should be appropriate for the PackType of the final packet received by the server. When the server receives a packet that is not the final packet in a connection stream, the server needs to send a "Result Continue" packet back to the client. The Operand fields of a "Result Continue" packet are undefined.

Additionally, the client needs to have the ability to store a named value (an integer value with a name of exactly four printable characters) on the server and refer to that value by name for the remainder of the connection. The stored value will be globally accessible. Any calculation stream that connects to the server can access these named values. If a calculation stream tries to create a named value that already exists, the new value overwrites the old value. The server should allows its clients to store an arbitrary number of named values. The final packet in a connection stream may not be a "Set named value" packet.

The client should only print out the result of the calculation if it receives a "Result of a[n] [Operation]" packet. If it receives a "Result Continue" packet, the client should not print anything.

## The MathPacket

The layout of the MathPacket is as follows, each of the following dashes represents one bit:

```
|<-------------32 bits ------------->|
|--------|--------|--------|----|----|
|PackType|Version |Flags   |Seq#|xxxx|
|                Operand One         |
|                Operand Two         |
|------------------------------------|
```

The bits marked with an *x* are unused and may be used in a future version of the packet.

`Version` should be set to: `00000010` for this assignment.

`PackType` is as follows:

```
00000001         Request Add
```

|  |  |
|---|---|
| `00000010` | `Request Subtract` |
| `00000100` | `Request Multiply` |
| `00001000` | `Request Divide` |
| `11111110` | `Result of an Add(integer result)` |
| `11111101` | `Result of a Subtract(integer result)` |
| `11111011` | `Result of a Multiply(integer result)` |
| `11111111` | `Result of an Add(float result)` |
| `11111100` | `Result of a Subtract (float result)` |
| `11111010` | `Result of a Multiply (float result)` |
| `11110111` | `Result of a Divide (integer result)` |
| `11110110` | `Result of a Divide (float result)` |
| `10000001` | `Result Continue` |
| `11100111` | `Set named value` |
| `01111111` | `Error: invalid PackType` |
| `10111111` | `Error: invalid Version` |
| `11101111` | `Error: invalid named value` |
| `11011111` | `Error: invalid sequence number` |
| `10011111` | `Error: invalid stream start` |
| `10111110` | `Error: invalid Flags` |

`Operand One` and `Operand Two` are both 32 bit fields. A request packet that is not preceded in the calculation stream by another request packet will provide an integer in each of the `Operand One` and `Operand Two` fields. A request packet that is preceded by a request packet will supply data only in `Operand Two`, the result of the previous request packet will be used for `Operand One`. The `Result` packets will return the result of the operation in the `Operand One` field. The value in the `Operand Two` field in `Result` packets is unspecified and should be ignored by the client.

When setting a named value, `Operand One` will contain the 4 character name and `Operand Two` will contain the integer value to be associated with that name.

The `Flags` field denotes the start and end of a calculation stream and which, if any, of the operands in the packet contain the name of a named value to be used in the calculation. The Flags field should be the OR of one or more of the calculation stream settings and zero or more of the named value settings.

**Flags**
**Calculation stream settings**

|  |  |
|---|---|
| `00000001` | `Calculation stream Start` |
| `00000010` | `Calculation stream Middle` |
| `00000100` | `Calculation stream End` |

**Named value settings**

|  |  |
|---|---|
| `10000000` | `Operand One contains the name of a stored value` |
| `01000000` | `Operand Two contains the name of a stored value` |

For example, if the value of `Flags` is: `10000101 Operand One` contains the name of a stored value and the packet is the start of a connection stream, and the end of a connection stream.

The `Seq#` field is a 4 bit field that records the order, within the calculation sequence, of the packet.

## The Client

The client should be an interactive application that takes 2 command line arguments: the DNS address of the server (not the IP address) and the port the server is listening on. The user should be prompted for the operation to perform (a, s, d, m, n), and two integers to serve as `Operand One` and `Operand Two`, in that order where appropriate. When prompting the user for each operand, the application needs to give the user the option of supplying a named value. Before sending the MathPacket to the server, the user needs to be prompted to determine if this is the last packet in the calculation stream. See the sample input/output below. The client should print the following message when the result of the calculation stream is returned, where the # is replaced with the value returned from the server:

```
          Result: #
```

Upon receiving an Error packet back from the server, the client should display one of the following error messages:

```
          Error: Invalid Packet Version
          Error: Invalid Packet Type
          Error: Invalid Stream Start
          Error: Invalid Flags
          Error: Invalid named value
          Error: Invalid Sequence Number
```

**The Server**

The server should take one command line argument: the port on which to listen. The server should loop forever handling requests. The server should return a packet of type `Error` if the request packet has an invalid `PackType`, invalid `Version,`contains and undefined named value, has an invalid value in `Flags,`or there is an error with the stream (wrong sequence number, the first packet is not marked as a stream start packet, more than one stream start packet is encountered). Sending the client an Error packet should terminate the connection on both ends. The server should not print anything to the screen.

**What and How to Submit**

You must submit TWO .c files, one for the client and one for the server, and ONE .h file that contains the data structure for the MathPacket and any extra data or `#define`s you deem necessary. The source code should be well documented and conform to the Coding Standards. You will also need to submit ONE Makefile that will build your client as an executable file named `client_TCP` and build your server as an executable file named `server_TCP`. The names of the targets for these two files should be client_tcp and server_tcp, respectively. The Makefile should also build both client and server by default if no command line arguments are given to the `(g)make` command. You are free to add any additional .c or .h files.

You will need to roll all FOUR of these files, and the .svn directory, into a gzipped tar file named **PUNetID_cs360s07_PA3.tar.gz** and submit that electronically using the submit script on **zeus** as detailed in lecture. You also need to turn in a hard copy of all FOUR files at the beginning of class on March 20th. Both the electronic and paper submissions must be made by 1pm, March 20th. Please do not create a directory in you .tar.gz file!

**How will this be graded?**

See Programming Assignment #1!

**Hints**

You will need to look at the `man` pages for `socket, bind, accept, listen, connect`, etc.

Reference servers will be up at: zeus.cs.pacificu.edu:9998 and Circe circe.cs.pacificu.edu:9998.

Use your assigned ports http://zeus.cs.pacificu.edu/chadd/cs360s07/ports.html.

Note the intermediate value on the server may be an integer or float. Once a float is created in the calculation stream, the end result of the stream will always be a float.

A named value will always be set to an integer. It will never be set to another named value.
In a complex project like this, you should commit to Subversion often! Name your project **PUNetID_cs360s07_PA3** in your Subversion repository so that I will be able to look in your repository and review your commits and commit messages. **This will be a graded portion of the assignment**.

Be sure to use your mutexes efficiently!

You MUST break up the source code for your client and server into multiple functions. If you have trouble breaking this down, come see me!

Sample Road Map:

Convert project 1 to TCP
Add the calculation stream
Add the named values
Add threads to the server

Be sure to test each step before moving on!

**Extra Credit**

+5 points: allow your client to store two types of named values on the server. Global named values are described above and are available to all connections to a server. Local named values are named values that are only available to the calculation stream that sets them. When that calculation stream ends, they should be removed from memory. When performing a calculation, a local named value should take precedence over a global named value. Use the following `PackType` to set a local named value. Use the command **`v`** in the interactive client to set a local named value.

```
11000011        Set named value locally
```

**Sample client input/output:**

```
[chadd@zeus]$ ./client_tcp zeus.cs.pacificu.edu 9998

Calculation Stream Client (v1.0)

Calculation command (a, s, m, d, n): n

      What is the name of the value: mass
      Please enter an integer: 10

Calculation command (a, s, m, d, n): a

      Is OperandOne a named value (y/n): n
      Please enter an integer: 2

      Is OperandTwo a named value (y/n): y
      Please enter a named value: mass

      Is this the last calculation in the stream (y/n): n

Calculation command (a, s, m, d, n): a

      Is OperandTwo a named value (y/n): n
      Please enter an integer: 98

      Is this the last calculation in the stream (y/n): y

Result: 110

[chadd@zeus]$
```

**Start on this today! Be prepared to demo your client and server in class on March 20th!**