



### Objective of this assignment:

- Develop and implement a simple application using **UDP** sockets.

### What you need to do:

- Implement a simple **UDP** Client-Server application

### Objective:

The objective is to implement a simple client-server application using a safe method: start from a simple **working** code for the client and the server. You must slowly and carefully *bend* (modify) little by little the client and server **alternatively** until you achieve your ultimate goal. You must **bend** and **expand** the client and the server alternatively the way a blacksmith forges iron. From time to time save your working client and server code such that you can **rollback** to the latest working code in case of nasty bugs (hard bugs occur often when socket programming).

For this programming assignment, you are advised to start from the *Friend* client and server application. The *Friend* client-server code is available on Canvas with this programming assignment. **For this assignment, you must implement a simple polynomial calculator server using UDP.** For simplicity, we will limit this client-server application to compute 3<sup>rd</sup> degree polynomials:

$$P(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \text{ with } 0 \leq a_i \leq 64 \text{ and } 0 \leq x \leq 64 \\ \text{for all } i \text{ } 0 \leq i \leq 3.$$

### Part A: Datagram Socket Programming

The objective is to design and implement a **Polynomial Calculator Server (CS)**. This calculator server will compute the polynomial  $P(x)$  when the **client** sends it a 9-byte **request** with this format:

Field	Total Message Length (TML)	Request ID	$x$	$a_3$	$a_2$	$a_1$	$a_0$	Checksum
Size! (byte)	1	2	1	1	1	1	1	1

### Where

- TML** is the Total Message Length (in bytes) including TML. It is an integer representing the **total** number of bytes of the message. If the message is correct, TML should always be equal to 9. TML size is one byte long.
- Request ID** is the request ID. This number is generated by the client to differentiate requests. You may use a variable randomly initialized by the client and increment it each time the client generates a new request. Request ID size is 2 bytes long.
- $x$**  is a number is the number for which we want to compute  $P(x)$ .  $x$ 's size is one byte.
- $a_i$**  is the  $i^{\text{th}}$  polynomial coefficient  $a_i$  with  $0 \leq a_i \leq 64$  for all  $i \text{ } 0 \leq i \leq 3$ .  $a_i$ 's size is one byte.
- Checksum** is an 8-bit Internet checksum computed as specified in the appendix of this assignment. Checksum size is one byte.

Note that only a stream of 72 bits (9 bytes) will be sent.

**Hint:** create a class object *Request* like "Friend", but with the information needed for a request.

Below are two examples of requests

**Request 1:** suppose the Client wants to compute  $P(x) = x^3 + 6x + 12$  for  $x = 5$ .

TML	Request ID	$x$	$a_3$	$a_2$	$a_1$	$a_0$	Checksum
0x09	0x0001	0x05	0x01	0x00	0x06	0x0C	0xDB <sup>2</sup>

The client will then send the binary stream (expressed here byte by byte in hexadecimal for convenience):

0x09 0x00 0x01 0x05 0x01 0x00 0x06 0x0C 0xDB

**Request 2:** suppose the Client wants to compute  $P(x) = 15x^2 + 22x + 7$  for  $x = 13$ .

TML	Request ID	$x$	$a_4$	$a_3$	$a_2$	$a_1$	$a_0$	Checksum
--	--							--

Complete the above table to find the binary stream to be sent(expressed here byte by byte in hexadecimal ..):

<sup>1</sup> Note that given the polynomial definition, we could halve the size of a message. To be compatible, do not try to minimize

<sup>2</sup> Checksum may be inaccurate



0x09 .....

The **Server** will respond with a message with this format:

Total Message Length (TML)	Request ID	Error Code	Result	Checksum
one byte	2 bytes	1 byte	4 bytes	one byte

#### Where

- 1) **TML** is the Total Message Length (in bytes) including TML. It is an integer representing the **total** numbers of bytes in the message.
- 2) **Request ID** is the request ID. This number is the number that was sent as Request ID in the request sent by the client. This allows the client to match a response to its request.
- 3) **Error Code** is **0** if the request was valid, **127** if the request was invalid (Length not matching TML), and **63** if the checksum does not match..
- 4) **Result** is the value of the polynomial  $P(x)$  requested.
- 5) **Checksum** is an 8-bit Internet checksum computed as specified in the appendix of this assignment.

In response to **Request 1** above, the server will send back:

0x09	0x00	0x01	0x00	0x00	0x00	0x00	0x00	0xA7	0x4E
------	------	------	------	------	------	------	------	------	------

i.e, the stream 0x09 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0xA7 0x4E

In response to **Request 2**, the server will send back:

0x09									
------	--	--	--	--	--	--	--	--	--

i.e, the stream 0x09 -- --

- a) **Repetitive Server:** Write a **datagram Polynomial Calculator Server (ServerUDP.java)** in **java**. This server must respond to requests as described above. The server must run on port (10010+GID) and could run on any machine on the Internet. **GID** is your group ID that was assigned to you. The server must accept a command line of the form: **java ServerUDP portnumber** where **portnumber** is the port where the server should be working. For example, if your Group ID (GID) is 13 then your server must listen on Port # 10023.
  - b) Write a datagram **client (ClientUDP.java)** in java:
- 6) Accepts a command line of the form: **java ClientUDP servername PortNumber** where **servername** is the server name and **PortNumber** is the port number of the server. Your program must prompt the user to ask for the number **x**, and the coefficient  $a_i$  for all  $i$  such that  $0 \leq i \leq 3$ . For each entry from the user, your program must perform the following operations:
    - i. form a message m as described above
    - ii. send the message to the server and wait for a response
    - iii. print all the message one byte at a time in hexadecimal (for **debugging/grading** purpose)
    - iv. print out the response of the server in a manner convenient for a typical Facebook user: the original polynomial, the value of x and the numerical result  $P(x)$ .
    - v. print out the round trip time (time between the transmission of the request and the reception of the response)
    - vi. prompt the user for a new request. (Design some way to allow the user to end the client program)

Five points will be deducted for any requirement not met.

**How to get started?**

1) Download all files (UDP sockets) to run the "Friend" application used in Module 2 to illustrate how any class object can be exchanged: Friend.java, FriendBinConst.java, FriendDecoder.java, FriendDecoderBin.java, SendUDP.java, and RecvUDP.java.

- 2) Compile these files and execute the UDP server and client. Make sure they work
- 3) Create a new folder called Request and duplicate inside it ALL files related to the Friend class object
- 4) Inside the Folder Request, change ALL occurrences of "Friend" with "Request" including the file names.
- 3) Adapt each file to your application. Replace the fields used by Friend with the fields used by a request.
- 4) Aim to have the client send one request and have the server understand it (just like what we did with a friend object).
- 5) When your server will receive and print out correctly a request, then you need to send back a response...
- 6) Create a class object Response....

### Report

- Write a report. The report should not exceed half a page.
- Your report must state whether your programs work or not (this must be just ONE sentence). If your program does not work, explain the obstacles encountered.

### What you need to turn in:

- Electronic copy of each source program separately (standalone). **In addition**, put all the source programs in one folder that you name with your group ID. Zip the folder and submit it **TOO**.
- Electronic copy of the report (including your answers if appropriate) (standalone). Submit the file as a Microsoft Word or a PDF file.

### Grading

**Your code will be compiled, executed, tested, and graded on Engineering Tux machines. Insure that your code works well on Tux machines.**

- 1) UDP client is worth 40% if it works well: communicates with YOUR server.
  - 2) UDP client is worth 10% extra if it works well with a working server from any of your classmates.
- 
- 1) UDP server is worth 40% if it works well: communicates with YOUR client.
  - 2) UDP server is worth 10% extra if it works well with a working client from any of your classmates.

Note that five points will be deducted for any requirement not met.



## Appendix: How To Compute an n-bit Checksum

Consider a stream  $R$  of bits to send. An n-bit Internet Checksum of the stream  $S$  is computed as follows:

- 1) Break the stream  $R$  of bits in n-bit words  $w_1, w_2, \dots$ , and  $w_m$ .
- 2) Compute the n-bit word  $S = w_1 + w_2$ , if there is a carry then set  $S = S + 1$
- 3) Compute  $S = S + w_3$ , if there is a carry then set  $S = S + 1$

.....

- n) Compute  $S = S + w_n$ , if there is a carry then set  $S = S + 1$

Finally, the Internet Checksum =  $\sim S$  (one-complement of  $S$ )

Example: suppose you must send the stream  $R = 01011100011110100101$ .

To compute a 4-bit checksum,

- 1) Break the stream  $R$  of bits in 4-bit words  $w_1, w_2, \dots$ , and  $w_m$ . In this case, we will obtain:  
 $w_1, w_2, w_3, w_4$ , and  $w_5$  with  $w_1 = 0101, w_2 = 1100, w_3 = 0111, w_4 = 1010$  and  $w_5 = 0101$

Go to Step 2.... Compute the n-bit word  $S = w_1 + w_2$ , if there is a carry then set  $S = S + 1$  .....

**How to check that your checksum is right?:** if you add all words  $w_i$  and the checksum, the sum should be equal to  $111\dots1$  (all ones).