Objective

Understand the interprocess communication in UNIX using stream sockets.

Walsh Codes

A Walsh code of length n = 2^k is a set of perfectly orthogonal codewords that can be defined and generated by the rows of a $2^k \times 2^k$ Hadamard matrix. Starting with a 1×1 matrix $\boldsymbol{H}_1$ = [0], higher-order Hadamard matrices can be generated by the following recursion:

$$\boldsymbol{H}_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & \overline{H_{2^{k-1}}} \end{bmatrix}$$

For our program, we will use $\boldsymbol{H}_4$. Given $\boldsymbol{H}_0$ = [0], we have that

$$\boldsymbol{H}_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

so that the Walsh codes of length 4 represented in a bipolar form (±1), and using positive logic (0 equal to -1, and 1 equal to 1) are: $\boldsymbol{w}_0$ =[-1,-1,-1,-1], $\boldsymbol{w}_1$ =[-1,+1,-1,+1], $\boldsymbol{w}_2$ =[-1,-1,+1,+1], and $\boldsymbol{w}_3$ = [-1,+1,+1,-1]. Note that these codes are mutually orthogonal; that is, the dot product of any pair of codes is zero.

Specifications

You must write two programs to allow communication between three processes using the encoding/decoding mechanism based on Walsh codes.

The sender / receiver program: the user will execute this program using the following syntax:

./exec_filename hostname port_no < input_filename

where exec_filename is the name of your executable file, hostname is the address where the encoder program is located, port_no is the port number used by the encoder program, and input_filename is the name of your input file.

This program receives the information about the hostname and the port number from the command line and reads the information about the transmission requests from a file using I/O redirection.

Input Format: the input file will always have three lines representing the request from process 1, 2, and 3 respectively.

```
3 4     // Process 1 sending to process 3 the integer value 4
1 5     // Process 2 sending to process 1 the integer value 5
2 7     // Process 3 sending to process 2 the integer value 7
```

- The first value of each line represents the id of the process which the current process wants to contact.
- The second value of each line represents the data (integer value) that the current process wants to send. The range of this value will be between zero and seven. Based on this restriction the encoding/decoding mechanism will always represent the data field with three bits.

After reading the input file, the sender/receiver program will create three child processes (representing processes 1, 2, and 3). Each child process will create a socket to: a) send the request to the encoder program with the destination process and data; b) receive the encoded signal (twelve integer values), and the code (four integer values); c) Decode the message (integer value between zero and seven) using the received information; and d) Print the encoded message, the code, and the decoded message.

The encoder program: the user will execute this program using the following syntax:

./exec_filename port_no

where exec_filename is the name of your executable file, and port_no is the port number to create the socket.

The tasks of this program are: a) Receive the request from each child process from the sender / receiver program; b) Generate the encoded message using Walsh codes; c) Send the encoded message and corresponding code to all the child processes (waiting one second between replies); and d) Finish its execution.

The encoding process:

The encoder program will use Walsh codes of size four. Each child process will have a Walsh code assigned in the encoder program ($\boldsymbol{w}_1$ for child 1, $\boldsymbol{w}_2$ for child 2, $\boldsymbol{w}_3$ for child 3). To generate the encoded message, the encoder program has to transform the integer value sent by a client to its binary representation (three bits). Each bit of the value sent by the client will be encoded ("spread") by the corresponding Walsh code. Based on the input file we have:

For child process 1:
Value = 4.
Binary = +1 -1 -1 (bipolar form, positive logic)
Encoded Message 1 (EM_1):
Binary * $\boldsymbol{w}_1$ =

```
        +1               -1               -1
   -1 +1 -1 +1      -1 +1 -1 +1      -1 +1 -1 +1
```

EM_1= -1 +1 -1 +1 +1 -1 +1 -1 +1 -1 +1 -1

For child process 2:

Value = 5.

Binary = +1 -1 +1 (bipolar form, positive logic)

Encoded Message 2 (EM_2):

Binary * $\boldsymbol{w}_2$ =

```
        +1               -1               +1
  -1 -1 +1 +1      -1 -1 +1 +1      -1 -1 +1 +1
```

EM_2= -1 -1 +1 +1 +1 +1 -1 -1 -1 -1 +1 +1

For child process 3:

Value = 7.

Binary = +1 +1 +1 (bipolar form, positive logic)

Encoded Message 3 (EM_3):

Binary * $\boldsymbol{w}_3$ =

```
        +1               +1               +1
  -1 +1 +1 -1      -1 +1 +1 -1      -1 +1 +1 -1
```

EM_3= -1 +1 +1 -1 -1 +1 +1 -1 -1 +1 +1 -1

Encoded message (EM) = EM_1+ EM_2 + EM_3

```
EM =  –1 +1 –1 +1 +1 –1 +1 –1 +1 –1 +1 –1    EM1
      –1 –1 +1 +1 +1 +1 –1 –1 –1 –1 +1 +1    EM2
      –1 +1 +1 –1 –1 +1 +1 –1 –1 +1 +1 –1    EM3

EM =  –3 +1 +1 +1 +1 +1 +1 –3 –1 –1 +3 –1
```

The decoding process:

Each child process will receive the encoded message and the code corresponding to the sender process. For the proposed input file, we have:

Child process 1:

EM = –3 +1 +1 +1 +1 +1 +1 –3 –1 –1 +3 –1

$\boldsymbol{w}_2$ = –1 –1 +1 +1 (code of child process 2).

Decoded message 1 (DM_1)= EM * $\boldsymbol{w}_2$

```
      –3 +1 +1 +1 +1 +1 +1 –3 –1 –1 +3 –1
      –1 –1 +1 +1 –1 –1 +1 +1 –1 –1 +1 +1
DM1 = +3 –1 +1 +1 –1 –1 +1 –3 +1 +1 +3 –1
```

Because each bit of the original message was spread using codes of size four, we have that:

```
DM1 =
+3 –1 +1 +1     –1 –1 +1 –3     +1 +1 +3 –1
     +4              –4              +4
```

Then dividing each element by the size of the code, we have:

```
     +4/4            –4/4            +4/4        =
     +1              –1              +1          =
                      5
```

Child process 2:

EM = –3 +1 +1 +1 +1 +1 +1 –3 –1 –1 +3 –1

$\boldsymbol{w}_3$ = –1 +1 +1 –1 (code of child process 3).

Decoded message 1 (DM_2)= EM * $\boldsymbol{w}_3$

```
      –3 +1 +1 +1 +1 +1 +1 –3 –1 –1 +3 –1
      –1 +1 +1 –1 –1 +1 +1 –1 –1 +1 +1 –1
DM2 = +3 +1 +1 –1 –1 +1 +1 +3 +1 –1 +3 +1
```

Because each bit of the original message was spread using codes of size four, we have that:

```
DM2 =
+3 +1 +1 –1     –1 +1 +1 +3     +1 –1 +3 +1
     +4              +4              +4
```

Then dividing each element by the size of the code, we have:

```
     +4/4            +4/4            +4/4        =
     +1              +1              +1          =
                      7
```

Child process 3:

EM = –3 +1 +1 +1 +1 +1 +1 –3 –1 –1 +3 –1

$\boldsymbol{w}_1$ = –1 +1 –1 +1 (code of child process 1).

Decoded message 1 (DM_3)= EM * $\boldsymbol{w}_1$

```
      –3 +1 +1 +1 +1 +1 +1 –3 –1 –1 +3 –1
      –1 +1 –1 +1 –1 +1 –1 +1 –1 +1 –1 +1
DM3 = +3 +1 –1 +1 –1 +1 –1 –3 +1 –1 –3 –1
```

Because each bit of the original message was spread using codes of size four, we have that:

```
DM3 =
+3 +1 –1 +1     –1 +1 –1 –3     +1 –1 –3 –1
     +4              –4              –4
```

Then dividing each element by the size of the code, we have:

```
     +4/4            –4/4            –4/4        =
     +1              –1              –1          =
                      4
```

NOTES:

- Use a ***single-threaded*** server to avoid critical sections inside the server.
- Since your server will not fork any child processes, you should not worry about handling zombies and can safely ignore the fireman() call in the primer.
- You can safely assume that:
  a) The input files will always be in the proper format.
  b) The number of lines in the input file is always three.
  c) The first value of each line in the input file (destination process) will always be different.
  d) Child 1 will always use the Walsh code $\boldsymbol{w}_1$, Child 2 will always use the Walsh code $\boldsymbol{w}_2$, and Child 3 will always use the Walsh code $\boldsymbol{w}_3$
  e) Naming convention for your source file: first name_lastname.cpp/ first name_lastname.c

Expected output:

Encoder program:

```
Here is the message from child 1: Value = 4, Destination = 3
Here is the message from child 2: Value = 5, Destination = 1
Here is the message from child 3: Value = 7, Destination = 2
```

Sender / Receiver program:

```
Child 1, sending value: 4 to child process 3
Child 2, sending value: 5 to child process 1
Child 3, sending value: 7 to child process 2

Child 1
Signal:-3 1 1 1 1 1 1 -3 -1 -1 3 -1
Code: -1 -1 1 1
Received value = 5

Child 2
Signal:-3 1 1 1 1 1 1 -3 -1 -1 3 -1
Code: -1 1 1 -1
Received value = 7

Child 3
Signal:-3 1 1 1 1 1 1 -3 -1 -1 3 -1
Code: -1 1 -1 1
Received value = 4
```