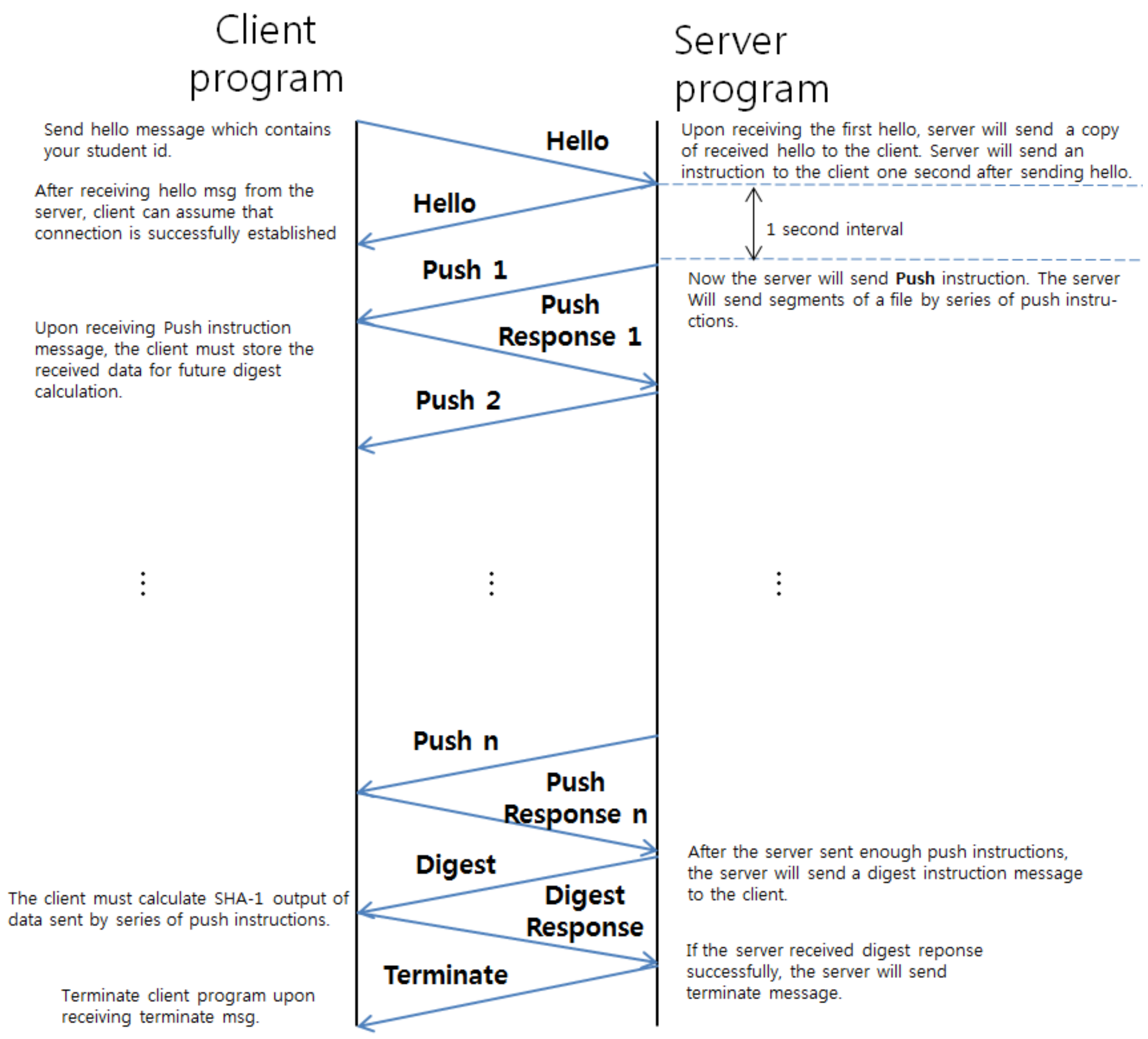
Socket Programming HW3

We will upload a server executable file on the BlackBoard. You can check whether your code is correct or not in the Ubuntu terminal using the server executable file.

0) Overview:

The goal of Socket programming HW3 is to calculate a message digest based on **IR (Instruction-Response) protocol defined in HW2**.


Client program
Server program
Send hello message which contains your student id.
After receiving hello msg from the server, client can assume that connection is successfully established
Upon receiving Push instruction message, the client must store the received data for future digest calculation.
The client must calculate SHA-1 output of data sent by series of push instructions.
Terminate client program upon receiving terminate msg.
Hello
Hello
Push 1
Push Response 1
Push 2
Push n
Push Response n
Digest
Digest Response
Terminate
Upon receiving the first hello, server will send a copy of received hello to the client. Server will send an instruction to the client one second after sending hello.
1 second interval
Now the server will send Push instruction. The server Will send segments of a file by series of push instructions.
After the server sent enough push instructions, the server will send a digest instruction message to the client.
If the server received digest reponse successfully, the server will send terminate message.

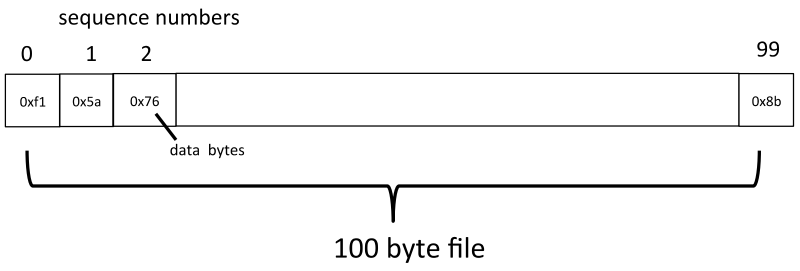
In this homework, we will define new instructions as follows:

① Op type 0x03 : This represents PUSH instruction. We assume that the server has a file which is divided into multiple segments. By using PUSH instruction, the server will transmit a segment of that file. In each PUSH packet sent by the server, there will be some data placed in the ‘data’ field of IR protocol packet format. When the client receives PUSH instruction, the client must store the data for later use.

② Op type 0x04 : This represents DIGEST instruction. By using DIGEST instruction, the server asks the client to calculate the message digest of the file.

Some detailed operation is given as follows.

* PUSH instruction: We assume there exists a file at the server. Each byte of the file is assigned a sequence number. The first byte of the file will be assigned with the sequence number of 0. The next byte is 1, and the next is 2, and so on. For example, if the file is 100 bytes long, the bytes are indexed by the sequence numbers 0 to 99.



The server will divide the file into multiple segments. The PUSH instruction is intended to transmit a segment to the client. The server will send multiple segments to the client, and the client must store the received segments.

The PUSH instruction uses the following handshake:

(1) C <-- PUSH <-- S

(2) C --> PUSH (response) --> S

The server will send a packet with PUSH instruction as follows.

① The ‘I’ bit is set to 1
② Op type is set to 0x03
③ The sequence number field represents the sequence number of the first byte of the data carried by the packet.
④ The data length field represents the length of the data carried by the packet.

For each PUSH instruction, the client must send the **PUSH response** as follows:

① The ‘R’ bit is set to 1

② Op type is set to 0x03
③ The sequence number field and data length field must be set to zero.
④ The data field must be empty.

Note that the PUSH instruction is initiated by the server. The client must wait for the incoming packets, in other words, the client must be ready to receive incoming packets, using recv() function.

Example: Suppose the server has 1000-byte file. The server may create 4 segments where each segment may have different length, and transmit 4 PUSH instructions in the following way:

PUSH packet #1: sequence number 0, data length 50 bytes

(the data in packet #1ranges sequence number 0~49 of the file)

PUSH packet #2: sequence number 50, data length 49 bytes

(the data in packet #2 ranges sequence number 50~98 of the file)

PUSH packet #3: sequence number 99, data length 501 bytes

(the data in packet #3 ranges sequence number 99~599 of the file)

PUSH packet #4: sequence number 600, data length 400 bytes

(the data in packet #4 ranges sequence number 600~999 of the file)

* DIGEST instruction: After the server sends all the segments, the server will issue DIGEST instruction to the client:

(1) C <-- DIGEST <-- S

(2) C --> DIGEST --> S

The server will send a packet with DIGEST instruction as follows.

① The 'I' bit is set to 1
② Op type is set to 0x04
③ The sequence number and data length will be set to 0
④ There is no data in this instruction packet.

The client must respond with the following **DIGEST response** packet.

① The 'R' bit is set to 1
② Op type is set to 0x04

③ The sequence number is set to 0
④ The data length must be set to 20.
⑤ The data field must contain the 20-byte message digest.

We will use SHA-1 algorithm (refer to the Security chapter in the textbook) as the cryptographic hashing function. SHA-1 algorithm will take any input data of arbitrary length, and will return 20-byte message digest.

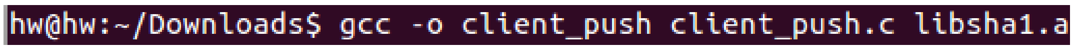
3) Hint

① We will provide the library file for you, where the library contains the implementation of SHA-1 function. Of course, you may choose to use other source code/library available on Internet, and even you may implement SHA-1 on your own. The important thing is that you must produce the correct result. As long as you compute the correct message digest, there will be no differences in the grade.

② Use libsha1.a file (uploaded at Blackboard) to use SHA1() function. If you wrote your client source code "client.c", use below command to compile "client.c" :

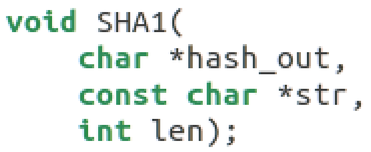
```
gcc -o output_file_name client.c libsha1.a
```

Otherwise you may encounter compile error! Below is captured image of example compile command.



```
hw@hw:~/Downloads$ gcc -o client_push client_push.c libsha1.a
```

③ After you link to the libsha1.a, you can simply invoke SHA-1 function in your client code. Below is the declaration of SHA1() function :



```
void SHA1(
    char *hash_out,
    const char *str,
    int len);
```

str is input data to sha-1 algorithm. Note that **str** do not need to be terminated by null character. **len** is data length of **str** in bytes. The calculated hash or message digest will be stored memory area **hash_out**. Note that **hash_out** should be 20 bytes long, because the message digest output of SHA-1 algorithm is always 20 bytes.

④ File size will be randomly determined on execution of server program. The maximum file size is 12288 (12*1024) bytes. Segment size range is 1 to file size(in bytes).

⑤ SHA-1 hash value examples

- SHA-1 output of 1byte data 0x61 (= a) : 0x86f7e437faa5a7fce15d1ddcb9eaeaea377667b8
- SHA-1 output of 1byte data 0x62 (= b) : 0xe9d71f5ee7c92d6dc9e92ffdad17b8bd49418f98
- SHA-1 output of 5byte data 0x6161616161 (= aaaaa) : 0xdf51e37c269aa94d38f93e537bf6e2020b21406c
- SHA-1 output of 5byte data 0x6161616162 (= aaaab) : 0x0b6af663352ee0c8c74c90d4b20b6c7724b0547b

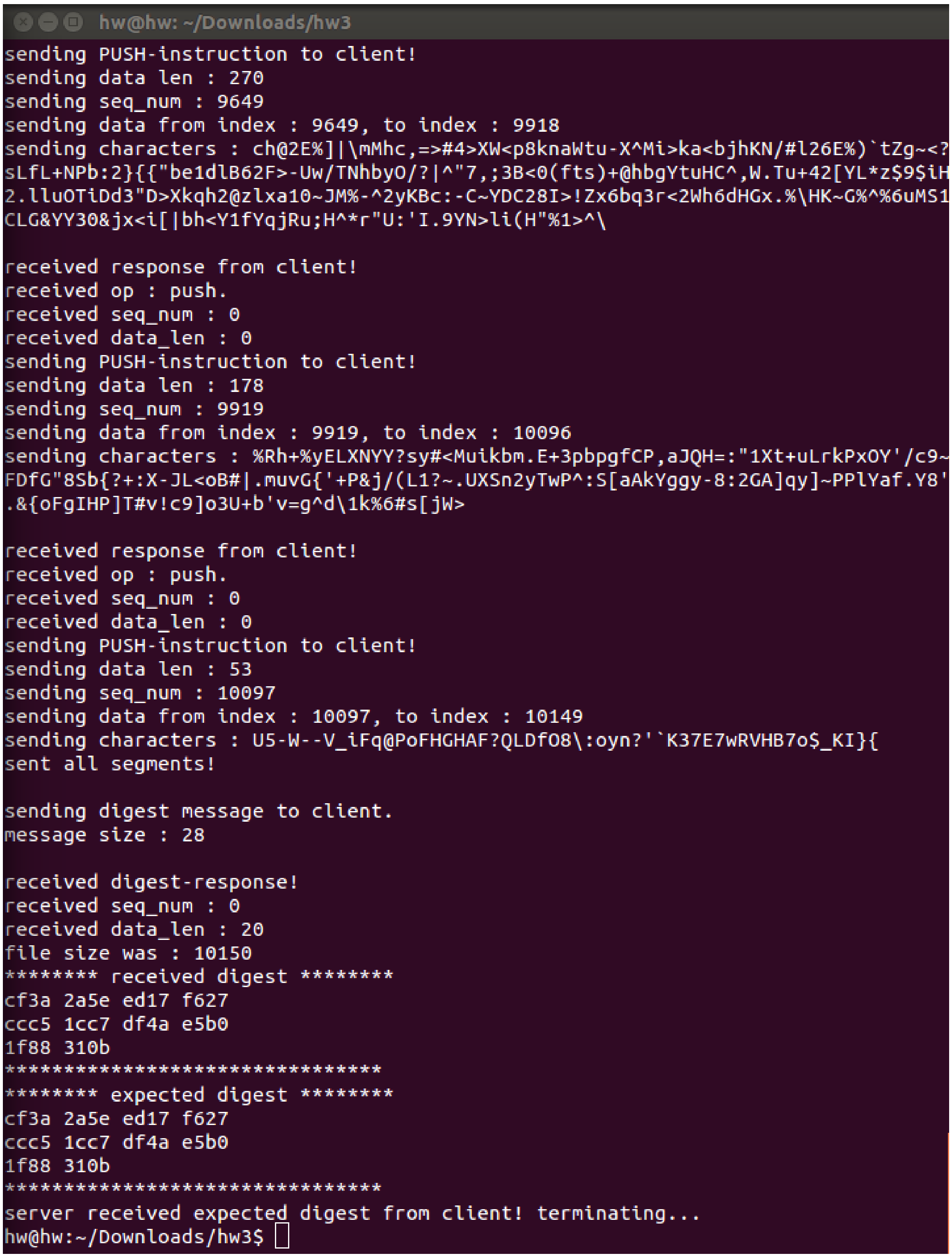
4) Requirements

1. Use c language.
2. Your client program code should not receive any command line arguments.
3. Do not use any function that takes input from terminal. (scanf(), fgets(), etc.)
4. Submit your c source code by uploading it to BlackBoard by due date. The name of your file must be your student id.
   ex) 2016001111.c
5. Use port number 47500.

5) Grading:

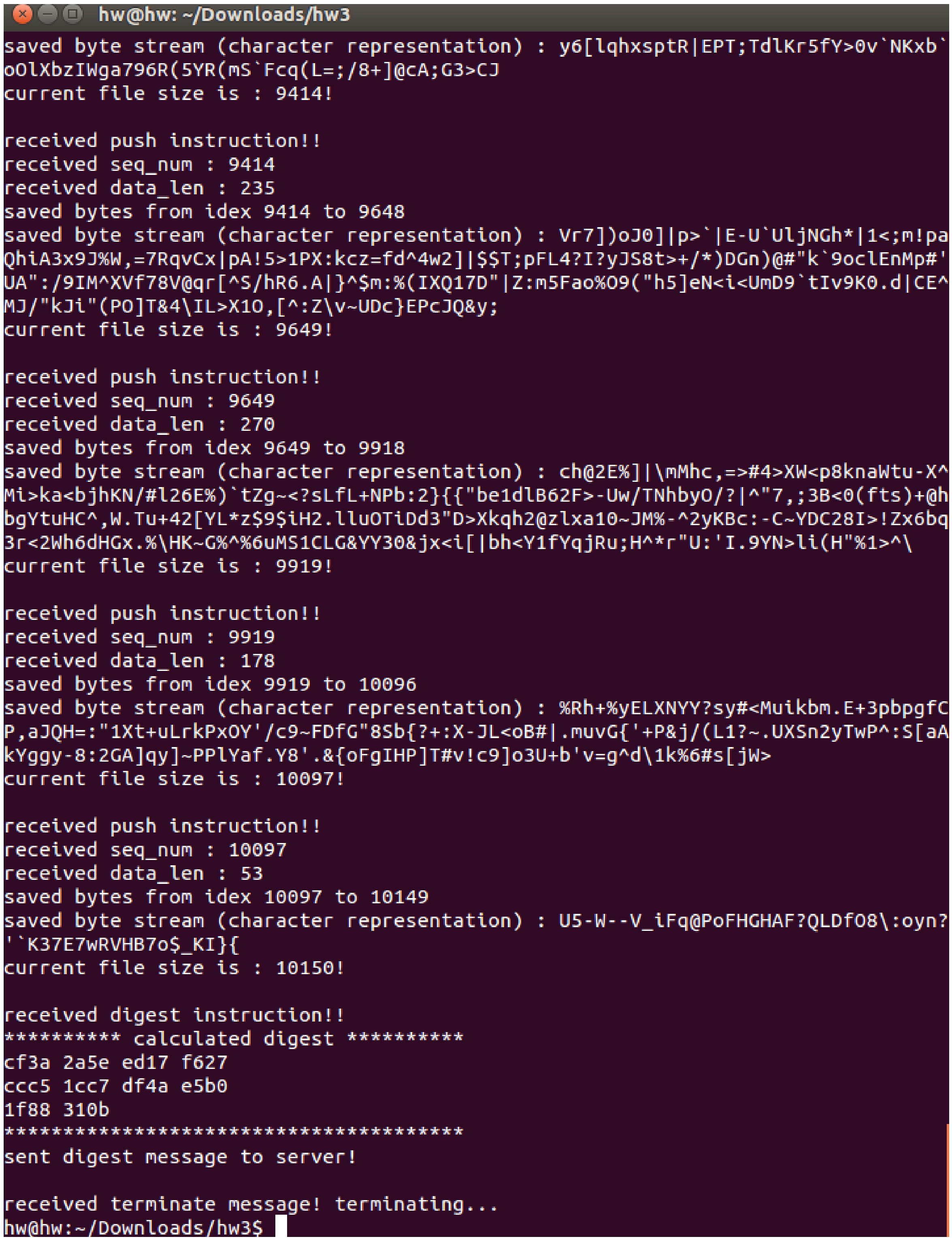
1. connect success : +2 points.
2. Hello message send and receive : +3 points.
3. PUSH-RESPONSE more than once : +5 points.
4. match digest : +10 points.

* Example testing results (screen shots)
Server result example



```
sending PUSH-instruction to client!
sending data len : 270
sending seq_num : 9649
sending data from index : 9649, to index : 9918
sending characters : ch@2E%]|\mMhc,=>#4>XW<p8knaWtu-X^Mi>ka<bjhKN/#l26E%)`tZg~<?
sLfL+NPb:2}{{"be1dlB62F>-Uw/TNhbyO/?|^"7,;3B<0(fts)+@hbgYtuHC^,W.Tu+42[YL*z$9$iH
2.lluOTiDd3"D>Xkqh2@zlxa10~JM%-^2yKBc:-C~YDC28I>!Zx6bq3r<2Wh6dHGx.%\HK~G%^%6uMS1
CLG&YY30&jx<i[|bh<Y1fYqjRu;H^*r"U:'I.9YN>li(H"%1>^\

received response from client!
received op : push.
received seq_num : 0
received data_len : 0
sending PUSH-instruction to client!
sending data len : 178
sending seq_num : 9919
sending data from index : 9919, to index : 10096
sending characters : %Rh+%yELXNYY?sy#<Muikbm.E+3pbpgfCP,aJQH=:"1Xt+uLrkPxOY'/c9~
FDfG"8Sb{?+:X-JL<oB#|.muvG{'+P&j/(L1?~.UXSn2yTwP^:S[aAkYggy-8:2GA]qy]~PPlYaf.Y8'
.&{oFgIHP]T#v!c9]o3U+b'v=g^d\1k%6#s[jW>

received response from client!
received op : push.
received seq_num : 0
received data_len : 0
sending PUSH-instruction to client!
sending data len : 53
sending seq_num : 10097
sending data from index : 10097, to index : 10149
sending characters : U5-W--V_iFq@PoFHGHAF?QLDfO8\:oyn?'`K37E7wRVHB7o$_KI}{
sent all segments!

sending digest message to client.
message size : 28

received digest-response!
received seq_num : 0
received data_len : 20
file size was : 10150
******** received digest ********
cf3a 2a5e ed17 f627
ccc5 1cc7 df4a e5b0
1f88 310b
*********************************
******** expected digest ********
cf3a 2a5e ed17 f627
ccc5 1cc7 df4a e5b0
1f88 310b
*********************************
server received expected digest from client! terminating...
hw@hw:~/Downloads/hw3$ 
```

Client result example



```
saved byte stream (character representation) : y6[lqhxsptR|EPT;TdlKr5fY>0v`NKxb`
oOlXbzIWga796R(5YR(mS`Fcq(L=;/8+]@cA;G3>CJ
current file size is : 9414!

received push instruction!!
received seq_num : 9414
received data_len : 235
saved bytes from idex 9414 to 9648
saved byte stream (character representation) : Vr7])oJ0]|p>`|E-U`UljNGh*|1<;m!pa
QhiA3x9J%W,=7RqvCx|pA!5>1PX:kcz=fd^4w2]|$$T;pFL4?I?yJS8t>+/*)DGn)@#"k`9oclEnMp#'
UA":/9IM^XVf78V@qr[^S/hR6.A|}^$m:%(IXQ17D"|Z:m5Fao%O9("h5]eN<i<UmD9`tIv9K0.d|CE^
MJ/"kJi"(PO]T&4\IL>X1O,[^:Z\v~UDc}EPcJQ&y;
current file size is : 9649!

received push instruction!!
received seq_num : 9649
received data_len : 270
saved bytes from idex 9649 to 9918
saved byte stream (character representation) : ch@2E%]|\mMhc,=>#4>XW<p8knaWtu-X^
Mi>ka<bjhKN/#l26E%)`tZg~<?sLfL+NPb:2}{{"be1dlB62F>-Uw/TNhbyO/?|^"7,;3B<0(fts)+@h
bgYtuHC^,W.Tu+42[YL*z$9$iH2.lluOTiDd3"D>Xkqh2@zlxa10~JM%-^2yKBc:-C~YDC28I>!Zx6bq
3r<2Wh6dHGx.%\HK~G%^%6uMS1CLG&YY30&jx<i[|bh<Y1fYqjRu;H^*r"U:'I.9YN>li(H"%1>^\
current file size is : 9919!

received push instruction!!
received seq_num : 9919
received data_len : 178
saved bytes from idex 9919 to 10096
saved byte stream (character representation) : %Rh+%yELXNYY?sy#<Muikbm.E+3pbpgfC
P,aJQH=:"1Xt+uLrkPxOY'/c9~FDfG"8Sb{?+:X-JL<oB#|.muvG{'+P&j/(L1?~.UXSn2yTwP^:S[aA
kYggy-8:2GA]qy]~PPlYaf.Y8'.&{oFgIHP]T#v!c9]o3U+b'v=g^d\1k%6#s[jW>
current file size is : 10097!

received push instruction!!
received seq_num : 10097
received data_len : 53
saved bytes from idex 10097 to 10149
saved byte stream (character representation) : U5-W--V_iFq@PoFHGHAF?QLDfO8\:oyn?
'`K37E7wRVHB7o$_KI}{
current file size is : 10150!

received digest instruction!!
********** calculated digest **********
cf3a 2a5e ed17 f627
ccc5 1cc7 df4a e5b0
1f88 310b
***************************************
sent digest message to server!

received terminate message! terminating...
hw@hw:~/Downloads/hw3$
```