Description File

Name: Xiaoran Tang
EID: x_t9
ID: A04824445

Implemented features

- **The Frames**

```
struct frame {
   int   seq;        // sequence number
   int   src;        // source address
   int   dest;       // destination address
   char  data[10];   // data section
}
```

Assume that frames are of fixed size (4 * 3 + 10 = 22 bytes). Each frame has a sequence number `seq`, a source address `src`, a destination address `dest`, and a data section `data`.

- **Frame Types and Formats**

SP frame format

-	seq	src	dest	data
request frame	seq	+	+	data
data frame	seq	+	-	data

CSP frame format

-	seq	src	dest	data
data frame	seq	+	+	data
pos reply frame	seq	-	+	data
neg reply frame	seq	-	-	data

Explanation:
There are only 2 types of frames sent by a SP: request frame and data frame. The value of source address is always positive (+). If the value of destination in a frame is positive (+), then this frame is a request frame. If the value of destination in a frame is negative (-), then this frame is a data frame.

There are 3 frame types sent by the CSP: data frame, positive reply frame, negative reply frame. If the value of source address is positive (+) and the value of destination address is positive (+) in a frame, then this frame is a data frame. If the value of source address is negative (-) and the value of destination address is positive (+) in a frame, then this frame is a positive reply frame. If the

value of source address is negative (-) and the value of destination address is negative (-) in a frame, then this frame is a negative reply frame.

- **CSP (Communication Switch Process)**

CSP runs as the “server” role in the TCP model.

On first connection with an new SP, CSP receives the index of the SP `sp_id` from this SP and store the socket descriptor `connfd` of this SP with `sp_id` as key-value pairs in array `sp`.

When the CSP receives a frame, it will inspect the destination address first. A positive destnition address indicates that this is a request frame. (If the destination SP is not connected with CSP, CSP would send a negative reply frame.) Otherwise, it is a data frame.

Every time a request frame is received, the CSP will either send a reply to the SP so that the SP will send a frame, or hold it in its request frame queue if it is busy. The maximum length of the requeest frame queue is 10 by default (defined by `MAXBUF` variable). The current length of the request frame queue is recorded by a counter `curr`. A data frame queue is maintained by the operating system implicitly.

When a request frame is received, and if request frame queue is not full (`curr < MAXBUF`), the CSP will send a positive reply to allow the source SP to send its data frame. If the request frame queue is full (`curr = MAXBUF`), a reject reply will to send back the source SP. When CSP finish to send out one data frame, the request frame queue will have one more vacancy.

- **SP (Station Processes)**

SP runs as the “client” role in the TCP model. The code for each SP is the same.

SP actions:
(0) If there is a incoming data frame, receive it.
(1) It reads a line from its simulation input data file and format it as a request frame.
(2) It then sends a request frame to the CSP asking permission to send its data frame just formatted and wait for reply.
(3) If there is a reject reply from the CSP, it has to retransmit the request frame. The maximum number of retransmissions is `MAXREQ` (= 3 by default). The current number of retransmissions is `curr`.
(4) If there is a positive reply from the CSP, it will modify the positive reply frame to data frame and proceed to send the data frame to the CSP.

- **Multiplexing**

My code uses `select` and `shutdown` to handle multiplexing.

The `select` function allows the CSP to instruct the kernel to wait for any one of multiple events (including request/data frames from multiple SP’s, send replies, and forward data frames) to occur and to wake up CSP only when one or more of these events occurs. Similarly, `select` helps SP to send data/request frames, read from command line (`stdin`), and receive data frames sent by other SP’s, at the same time.

With `shutdown`, we can initiate TCP’s normal connection termination sequence, regardless of the reference count. Also, we can terminates only “writing” direction of data transfer and keep “reading” from the connection. In addition, we are able to tell the other end that we have finished

sending, even though that end might have more data to send us.

- **Data Structure**

-	fd0	fd1	fd2	fd3	fd4	fd5	...
rest	0	0	0	1	0	0	…

The server maintains a read descriptor set. Since the server is started in the foreground, descriptors 0, 1, and 2 are set to standard input, output, and error. Therefore, the first available descriptor for the listening socket is 3.

i	0	1	2	3	4	...	SETSIZE - 1
`client[i]`	-1	-1	-1	-1	-1	…	-1

An array of integers `client` contains the connected socket descriptor for each client. All elements in this array are initialized to -1.

sp_id	1	2	3	4	5	6	...	100
`sp[sp_id]`	-1	-1	-1	-1	-1	-1	…	-1

An array of integers `sp` contains the connected socket descriptor for each SP. This array acts as a mapping between `sp_id` and socket descriptors. All elements in this array are initialized to -1. This array is mainly used to find the socket descriptor in `Writen()` function. For example, `Writen(sp[src], &frame, sizeof(frame))` write the frame values to the source SP, where `src` is the ID of the source SP.

The data structure shown above demonstrates the condition where CSP has no connection with any SP’s yet.

When SP6 is connected to CSP, the data structure becomes the following:

-	fd0	fd1	fd2	fd3	fd4	fd5	...
rest	0	0	0	1	1	0	…

`rest[fd4]` changes from 0 to 1

i	0	1	2	3	...	SETSIZE - 1
`client[i]`	4	-1	-1	-1	…	-1

`client[0]` changes from -1 to 4, meaning that “the first SP has the socket fd value 4”

sp_id	1	2	3	4	5	6	...	100
`sp[sp_id]`	-1	-1	-1	-1	-1	4	…	-1

`sp[6]` changes from -1 to 4, meaning that “SP6 has the socket fd value 4”

After SP6 has connected to CSP, SP1 connects to CSP. When SP1 is conneced to CSP, the data structure becomes the following:

-	fd0	fd1	fd2	fd3	fd4	fd5	...
rest	0	0	0	1	1	1	…

`rest[fd5]` changes from 0 to 1

i	0	1	2	3	...	SETSIZE - 1
`client[i]`	4	5	-1	-1	…	-1

`client[1]` changes from -1 to 5, meaning that “the second SP has the socket fd value 5”

sp_id	1	2	3	4	5	6	...	100
`sp[sp_id]`	5	-1	-1	-1	-1	4	…	-1

`sp[1]` changes from -1 to 5, meaning that “SP1 has the socket fd value 5”

Then SP6 has disconnected from CSP. The data structure becomes the following:

-	fd0	fd1	fd2	fd3	fd4	fd5	...
rest	0	0	0	1	0	1	…

`rest[fd4]` changes from 1 to 0

i	0	1	2	3	...	SETSIZE - 1
`client[i]`	-1	5	-1	-1	…	-1

`client[0]` changes from 4 to -1

sp_id	1	2	3	4	5	6	...	100
`sp[sp_id]`	5	-1	-1	-1	-1	-1	…	-1

`sp[6]` changes from 4 to -1, meaning that “SP6 has no socket fd”

Not implemented features

- an explicit data frame queue
- the procedure to process the content of the data[] section in the data frame
- the ability for SP to wait, e.g. “Wait for receiving 2 frames”

Assumptions

- SP ID’s are from 1 to 100 (e.g. SP1 - SP100)
  - can be modified by changing `MAXSPNO` in `csp.c`
- max number of SP’s is 10
  - can be modified by changing `SETSIZE` in `csp.c`
- max number of buffer length for request queue in CSP is 10
  - can be modified by changing `MAXBUF` in `csp.c`
- max amount of data sent by a frame is 10 char letters
  - can be modified in `struct frame` definition
- The data is read from the STDIN directly (by typing line by line)

Controlling the Simulation Using Events

Here are several sample lines that could be in the simulation file for SP 1:
Frame 1, To SP 3
Frame 2, To SP 8
Frame 3, To SP 6
Frame 4, To SP 2

If the input format is incorrect, then the command line prompt
`usage: Frame <number>, To SP <number>` and `Invalid input: <wrong_input_line>`

Simulation Result

The activity log files can be maintained by each SP and CSP by using the Linux *output redirection*.

```
[<host1>]$ ./csp > CSP_log.txt
[<host2>]$ ./sp 127.0.0.1 9 > SP9_log.txt
[<host3>]$ ./sp 127.0.0.1 7 > SP7_log.txt
```

- **Send frame to unconnected SP**

Command line input in SP9: (saved in `input_test1.txt`)

```
Frame 1, to SP 7
```

`[<host1>]$ ./csp > CSP_log.txt`

```
new connection from SP9: 127.0.0.1, port 40828
Receive request from SP 9
Send negative reply to SP 7
Receive request from SP 9
Send negative reply to SP 7
Receive request from SP 9
Send negative reply to SP 7
Receive request from SP 9
Send negative reply to SP 7
```

`[<host2>]$ ./sp 127.0.0.1 9 > SP9_log.txt`

```
This Station Process is SP9
Frame 1, to SP 7
Send request to CSP to send data frame 1 to SP 7
```

```
Receive reject reply from CSP to send data frame 1 to SP 7
Retransmit request to CSP to send data frame 1 to SP 7 (count: 1)
Receive reject reply from CSP to send data frame 1 to SP 7
Retransmit request to CSP to send data frame 1 to SP 7 (count: 2)
Receive reject reply from CSP to send data frame 1 to SP 7
Retransmit request to CSP to send data frame 1 to SP 7 (count: 3)
```

- **Send frame to connected SP**

Command line input in SP9: (saved in `input_test1.txt`)

```
Frame 1, to SP 7
```

`[<host1>]$ ./csp > CSP_log.txt`

```
new connection from SP9: 127.0.0.1, port 40836
new connection from SP7: 127.0.0.1, port 40838
Receive request from SP 9
Send postive reply to SP 7
Receive data frame 1 from SP 9 (to SP 7)
Forward data frame 1 (from SP 9) to SP 7
```

`[<host2>]$ ./sp 127.0.0.1 9 > SP9_log.txt`

```
This Station Process is SP9
Send request to CSP to send data frame 1 to SP 7
Receive positive reply (permission) from CSP to send data frame 1 to SP 7
Send (via CSP) data frame 1 to SP 7
```

`[<host3>]$ ./sp 127.0.0.1 7 > SP7_log.txt`

```
This Station Process is SP7
Receive (via CSP) a data frame 1 from SP 9
```

- **Send frames among multiple SP's**

Command line input in SP5: (saved in `input_test1.txt`)

```
Frame 1, to SP 7
```

Command line input in SP7: (saved in `input_test2.txt`)

```
Frame 1, to SP 26
Frame 2, to SP 14
Frame 3, to SP 5
Frame 4, to SP 5
```

Notice: the Frame 4 above is sent after SP5 is disconnected from CSP.

Command line input in SP26: (saved in `input_test3.txt`)

```
Frame 1, to SP 5
Frame 2, to SP 7
Frame 3, to SP 5
```

`[<host1>]$ ./csp > CSP_log.txt`

```
new connection from SP5: 127.0.0.1, port 40944
new connection from SP7: 127.0.0.1, port 40946
new connection from SP26: 127.0.0.1, port 40950
Receive request from SP 5
Send postive reply to SP 7
Receive data frame 1 from SP 5 (to SP 7)
Forward data frame 1 (from SP 5) to SP 7
Receive request from SP 7
Send postive reply to SP 26
Receive data frame 1 from SP 7 (to SP 26)
Forward data frame 1 (from SP 7) to SP 26
Receive request from SP 7
Send negative reply to SP 14
Receive request from SP 7
Send negative reply to SP 14
Receive request from SP 7
Send negative reply to SP 14
Receive request from SP 7
Send negative reply to SP 14
Receive request from SP 7
Send postive reply to SP 5
Receive data frame 3 from SP 7 (to SP 5)
Forward data frame 3 (from SP 7) to SP 5
Receive request from SP 26
Send postive reply to SP 5
Receive data frame 1 from SP 26 (to SP 5)
Forward data frame 1 (from SP 26) to SP 5
Receive request from SP 26
Send postive reply to SP 7
Receive data frame 2 from SP 26 (to SP 7)
Forward data frame 2 (from SP 26) to SP 7
Receive request from SP 26
Send postive reply to SP 5
Receive data frame 3 from SP 26 (to SP 5)
Forward data frame 3 (from SP 26) to SP 5
Receive request from SP 7
Send negative reply to SP 5
Receive request from SP 7
Send negative reply to SP 5
Receive request from SP 7
Send negative reply to SP 5
Receive request from SP 7
Send negative reply to SP 5
```

`[<host2>]$ ./sp 127.0.0.1 5 > SP5_log.txt`

```
This Station Process is SP5
Send request to CSP to send data frame 1 to SP 7
Receive positive reply (permission) from CSP to send data frame 1 to SP 7
Send (via CSP) data frame 1 to SP 7
```

```
Receive (via CSP) a data frame 3 from SP 7
Receive (via CSP) a data frame 1 from SP 26
Receive (via CSP) a data frame 3 from SP 26
```

`[<host3>]$ ./sp 127.0.0.1 7 > SP7_log.txt`

```
This Station Process is SP7
Receive (via CSP) a data frame 1 from SP 5
Send request to CSP to send data frame 1 to SP 26
Receive positive reply (permission) from CSP to send data frame 1 to SP 26
Send (via CSP) data frame 1 to SP 26
Send request to CSP to send data frame 2 to SP 14
Receive reject reply from CSP to send data frame 2 to SP 14
Retransmit request to CSP to send data frame 2 to SP 14 (count: 1)
Receive reject reply from CSP to send data frame 2 to SP 14
Retransmit request to CSP to send data frame 2 to SP 14 (count: 2)
Receive reject reply from CSP to send data frame 2 to SP 14
Retransmit request to CSP to send data frame 2 to SP 14 (count: 3)
Send request to CSP to send data frame 3 to SP 5
Receive positive reply (permission) from CSP to send data frame 3 to SP 5
Send (via CSP) data frame 3 to SP 5
Receive (via CSP) a data frame 2 from SP 26
Send request to CSP to send data frame 4 to SP 5
Receive reject reply from CSP to send data frame 4 to SP 5
Retransmit request to CSP to send data frame 4 to SP 5 (count: 1)
Receive reject reply from CSP to send data frame 4 to SP 5
Retransmit request to CSP to send data frame 4 to SP 5 (count: 2)
Receive reject reply from CSP to send data frame 4 to SP 5
Retransmit request to CSP to send data frame 4 to SP 5 (count: 3)
```

`[<host4>]$ ./sp 127.0.0.1 26 > SP26_log.txt`

```
This Station Process is SP26
Receive (via CSP) a data frame 1 from SP 7
Send request to CSP to send data frame 1 to SP 5
Receive positive reply (permission) from CSP to send data frame 1 to SP 5
Send (via CSP) data frame 1 to SP 5
Send request to CSP to send data frame 2 to SP 7
Receive positive reply (permission) from CSP to send data frame 2 to SP 7
Send (via CSP) data frame 2 to SP 7
Send request to CSP to send data frame 3 to SP 5
Receive positive reply (permission) from CSP to send data frame 3 to SP 5
Send (via CSP) data frame 3 to SP 5
```