Lab 2 Report

Thursday, March 23, 2017 4:26 PM

Running the program

Please refer to the README file for this project.

Server side design

Some features

- Always-on server
  - Restart when get into errors
  - Stop until intervened
- Reliable data transfer
  - Retransmission when message loss
- Retry and terminate if client can not be recovered.

Message Format

Message Structure

MsgType\n rcv_port\n \r\n MsgContent

Fields in the Message

MsgType	Char	1 Byte	Type of message
rcv_port	Char	5 Byte	The receiving port of the message sender
seq_no	Char		The sequence no of current message. (only valid for FILE_TRANSFER messages.)
MsgContent		char/binary	Message content, same as pay load. Can be modified through MAXBUFLEN

Message Types

There are 6 types of message in this protocol and application.

```
27  #define FILE_REQUEST 1  // client side
28  #define FILE_TRANSFER 2 // Server side
29  #define ACK0 3 // client side
30  #define ACK1 4 // client side
31  #define MISSION_OVER 5 // Server side
32  #define ERR 6  // Server side
```

Server side program procedures

- Setp 1: Create socket for Receiver on Port: rcv_port.
- Step 2: Wait for the Request message.
  - if msg_type != FILE_REQUEST , print error message.
  - else, then go to Step2: enter Finite State Machine to start transfer file
- Step 3: Create Sender socket for sending data.
- Step 4: Enter Finite State Machine to start transfer file.
- Step 5: Upon finish, send a MISSION_OVER Message to client and close the sockets.
- Go back to Step 1 and listening.

Finite State Machine of Rdt3.0

ERR state was added into the finite state machine for handling errors.

FSM States

```
#define ERR 6  // file not exists and so on

#define WAIT_FOR_0 100
#define WAIT_FOR_ACK0 200
#define WAIT_FOR_1 300
#define WAIT_FOR_ACK1 400
```

FSM Simplified Code

```
461         sleep(0.1); // wait for client to get prepared
462         printf("Sender Start ::");
463         // enter FSM
464         while (1) {
465             n++;
466             switch (current_state) {
467                 case WAIT_FOR_0:
468                     // some code
469                 case WAIT_FOR_ACK0:
470                     // some code
471                 case WAIT_FOR_1:
472                     // some code
473                 case WAIT_FOR_ACK1:
474                     // some code
475                 case ERR:
476                     iscomplete = false;
477                     iserror = true;
478                     printf("[sending] some error happend. %s\n", err_msg);
479                     break;
480                 default:
481                     perror("No such state in rdt3.0.");
482                     break;
483             }
484             if(iscomplete || iserror) break; // leave FSM
485         }
486
487         if(iserror){  // restart server
488             close(sockfd_rcv);
489             close(sockfd_snd);
490             continue;
491         }
```

Client side design

Some features

- Package loss implemented at client side

```
38  /* package loss probability k% */
39  #define LOSS_PROB 20
```

- Reliable data transfer
  - Use ACK0 / ACK1 to indicate the status of receiving
  - Coordinated with server using rdt3.0
- Close connection when timeout.

Message Format

Consistent with server side.

Client side program procedures

- Step 1: Create socket for Sending to Port: server_IP, server_port
- Step 2: Send File Request Msg.
- Step 3: create rcv socket for receiving file from server: rcv_port
- Step 4: Enter FSM and Receive Message.
- Step 5: Upon received MISSION_OVER Message, close the sockets.
- Finish.

Finite State Machine of Rdt3.0

```
393     while(!iscomplete)
394     {
395         n++;
396         memset(in_msg_buf, 0, sizeof in_msg_buf);
397         bzero(data_buf, sizeof data_buf);
398         bzero(out_msg_buf, sizeof out_msg_buf);
399         // Finite State Machine
400         switch (current_state)
401         {
402             case WAIT_FOR_0:
403                 // Receive msg with loss (back to receiving state when loss)
404                 // ... some code
405                 if(parseMsgType(in_msg_buf) == MISSION_OVER){
406                     iscomplete = true;
407                     break;
408                 }
409                 //logic for rdt3.0 based on seq_no
410                 seq_no = parseSequenceNumber(msg);
411                 switch (seq_no)
412                 {
413                     case 0:
414                         // write into file
415                         // ... some code
416                         // Send ACK 0
417                         // ... some code
418                         current_state = WAIT_FOR_1;
419                         break;
420                     case 1:
421                         // send ACK 1
422                         // ... some code
423                         current_state = WAIT_FOR_0;
424                 }
425                 break;
426
427             case WAIT_FOR_1:
428                 // Reset buffer
429                 // ... some code
430                 // Receive msg with loss (back to receiving state when loss)
431                 // ... some code
432                 if(parseMsgType(in_msg_buf) == MISSION_OVER){
```

```
430                 // Receive msg with loss (back to receiving state when loss)
431                 // ... some code
432                 if(parseMsgType(in_msg_buf) == MISSION_OVER){
433                     iscomplete = true;
434                     break;
435                 }
436                 //logic for rdt3.0 based on seq_no
437                 seq_no = parseSequenceNumber(in_msg_buf);
438                 switch (seq_no)
439                 {
440                     case 0:
441                         // Send ACK 0
442                         // ... some code
443                         current_state = WAIT_FOR_1;
444                         break;
445                     case 1:
446                         // write into file
447                         // ... some code
448                         // send ACK 1
449                         // ... some code
450                         current_state = WAIT_FOR_0;
451                 }
452                 break;
453             } // end of swtich case
454 
455         } // end of while
456         printf("Total number of packages received: %d", n);
```