Project [RFID READER -> LCD] Design

Benny Chao - A01270575
David Lee - A01251357
November 6th, 2022

Structures

State

Field	Purpose
in_redirect_regex	Find input redirection: < *path*
out_redirect_regex	Find output redirection: *[1]>[>] path*
err_redirect_regex	Find output redirection: 2*>[>] path*
path	An array of directories to search for external commands
prompt	Prompt to display to the user, defaults to $
max_line_length	The longest possible command line
current_line	The current command line
current_line_length	The length of the current command line
command	The command to execute
fatal_error	True if an error happened that should exit the shell

Command

line	The command line for this command
command	The command (e.g. ls, exit, cd, cat)
argc	The number of arguments passed to the command
argv	The arguments passed to the command
stdin_file	The file to redirect stdin from, or NULL
stdout_file	The file to redirect stdout to, or NULL
stdout_overwrite	Append to or truncate the stdout file
stderr_file	The file to redirect stderr to, or NULL
stderr_overwrite	Append to or truncate the stderr file
exit_code	The status returned from the command

Finite State Machine

Client

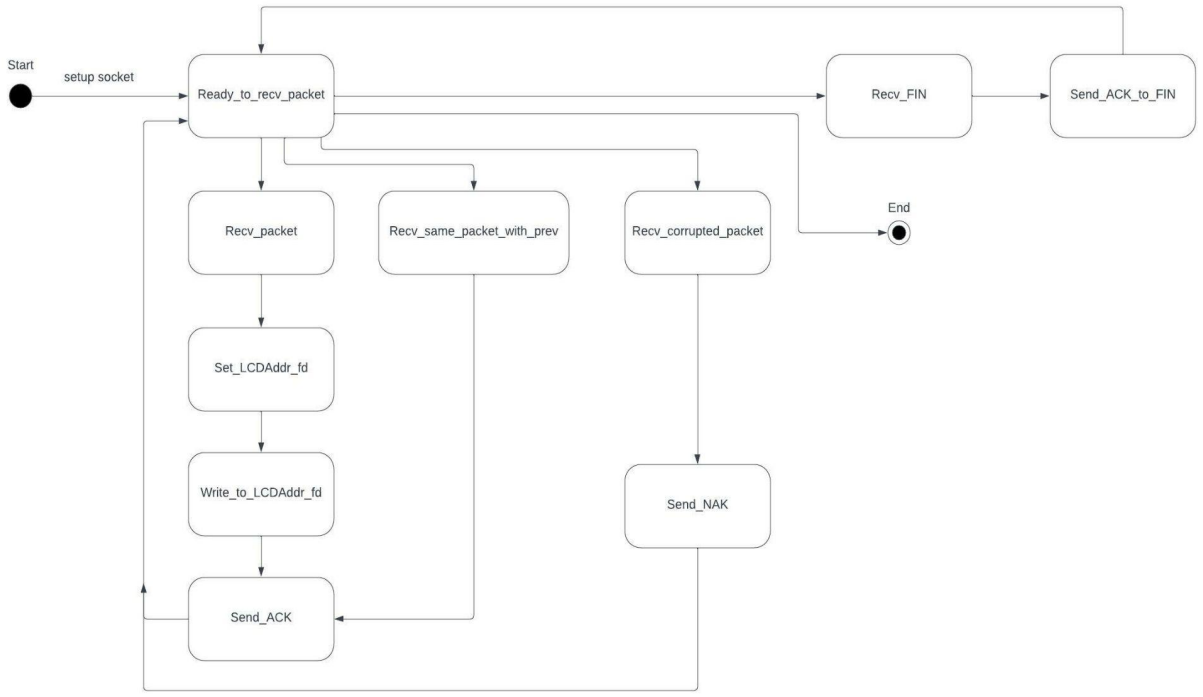
From State	To State	Action
START	Ready_to_read_user_input	`setup_socket`
Ready_to_read_user_input	Ready_to_send	`create_packet`
Ready_to_read_user_input	Ready_to_send_FIN	`create_packet_with_fin`
Ready_to_send	Wait_for_ACK	`sendto`
Wait_for_ACK	Ready_to_send	`timeout`
Wait_for_ACK	Receive_ACK	`recvfrom`
Receive_ACK	Ready_to_read_user_input	`deserialize_packet`
Ready_to_send_FIN	Wait_for_ACK_to_FIN	`sendto`
Wait_for_ACK_to_FIN	Ready_to_send_FIN	`timeout`
Wait_for_ACK_to_FIN	Receive_ACK_to_FIN	`recvfrom`
RECEIVE_ACK_TO_fin	END	`deserialize_packet`

Server

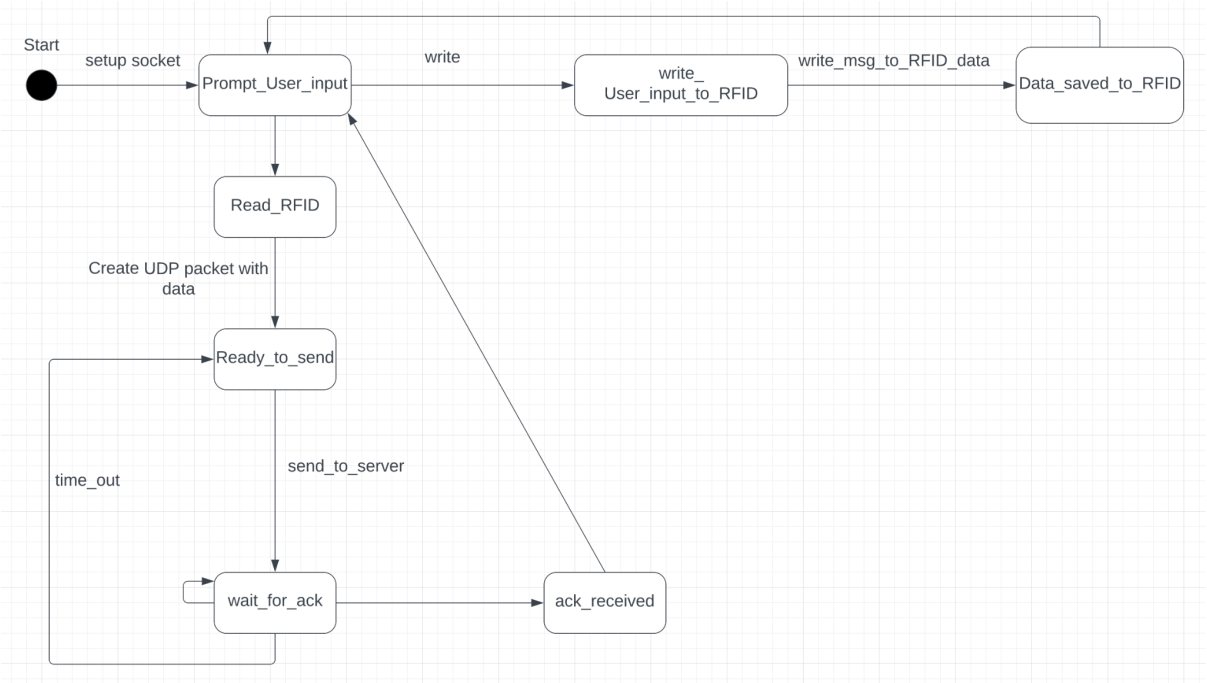
From State	To State	Action
Start	Ready_to_recv_packet	`setup_socket`
Ready_to_recv_packet	Recv_packet	`recvfrom`
Ready_to_recv_packet	Recv_same_packet_with_prev	`recvfrom`
Ready_to_recv_packet	Recv_corrupted_packet	`recvfrom`
Recv_packet	Print_data_to_stdout	`deserialize_packet`
Print_data_to_stdout	Send_ACK	`write_to_screen`
Recv_same_packet_with_prev	Send_ACK	`deserialize_packet, check_seqno`

Recv_corrupted_packet	Send_NAK	`deserialize_packet, check_checksums`
Send_ACK	Ready_to_recv_packet	`sendto`
Send_NAK	Ready_to_recv_packet	`sendto`
Ready_to_recv_packet	Recv_FIN	`recvfrom`
Recv_FIN	Ready_to_recv_packet	`sendto`
Ready_to_recv_packet	End	`terminate`

State Transition Diagram for Server



State Transition Diagram for Client



Functions

Server

socket

Purpose

Create a socket

Parameters

Domain, type, and protocol to set up.

Return

Type	Next State
Success	`READY_TO_RECV_PACKET`
Failure	`ERROR`

Pseudocode

```
If any errors occur
     set state.fatal_error to true and return ERROR

Set sock = socket(AF_INET, SOCK_STREAM, 0)
     If sock fails;
     Error.
```

bind

Purpose

Bind socket to port

Parameters

Socket descriptor, address, length of address

Return

Type	Next State
Success	`READY_TO_RECV_PACKET`
Failure	`ERROR`

Pseudocode

```
If any errors occur
     set state.fatal_error to true and return ERROR
```

```
Set bind(socket, (struct sockaddr *) &local, sizeof(struct
sockaddr_un)) == 1);
     If bind fail;
     Error.
```

listen

Purpose

Listen to the port.

Parameters

Socket, backlog

Return

Type	Next State
Success	READY_TO_RECV_PACKET
Failure	ERROR

Pseudocode

```
If any errors occur
     set state.fatal_error to true and return ERROR

listen(socket, backlog);
     If listen == -1
     Error.
```

accept

Purpose

Accept a socket, and begin listening to connections

Parameters

Socket, struct socketaddr * address, socklent_t * address len)

Return

Type	Next State
Success	READY_TO_RECV_PACKET
Failure	ERROR

Pseudocode

```
If any errors occur
     set state.fatal_error to true and return ERROR

     accept(socket, struct sockaddr *, socklen_t *restrict
address_len);
     If accept == -1;
     error.
```

ready_to_recv_packet

Purpose

Awaits a packet from Client

Parameters

Socket_fd, packet, sizeOf(Packet), Addr len

Return

Read	Next State

SUCCESS	`RECV_PACKET`
	`RECV_SAME_PACKET_WITH_PREVIOUS_HEADER,`
	`REC_CORRUPTED_PACKET,`
	`REC_FIN`

Failure	`ERROR`

Pseudocode

```
fprintf("listening on port: opts -> port_in");
nread = recvfrom( opts -> socket_fd), &packet,
sizeof(&packet),0, (struct sockaddr&, addr)

If (nread == -1)
error;
```

Recv_Packet

Purpose

To unpack the packet that was received

Parameters

The state object to store the commands in.

Return

Type	Next State
Success	`Set_LCDaddr_fd`

Failure	ERROR

Pseudocode

```
deserialize(Packet);
if(packet.type == RUDP)
     if(packet.header.seq.num == 1 and current.seq == -1){
     fprintf("Receiving message from client")}

     If checksum_in_packet == !packet_checksum) {
     create(Nak_packet);
     sendto(socket_fd, Nak_packet, sizeof(rudp_packet),0,
clientaddr);

if(packet.header.seq.num != current.seq.number) {
create(RUDP_ack);
sendto(socket_fd,RUDP_ack, sizeof(rudp_packet), 0,
clientaddr);
```

set_LCDaddr_fd

Purpose

Setting up the PI LCD to be ready for displaying a message

Parameters

The state object to store the command data into.

Return

Type	Next State
Success	write_to_LCDaddr_fd
Failure	ERROR

Pseudocode

```
send_command(0x33);
delay(5);
send_command(0x32);
```

```
delay(5);
send_command(0x28);
delay(5);
send_command(0x0C);
delay(5);
send_command(0x01);
wiringPiI2CWrite(fd, 0x08;
```

write_to_LCDaddr_fd

Purpose

Take message from packet and display onto LCD screen.

Parameters

Int x, y,

Char[] data;

Return

Type	Next State
Success	`rdy_to_recv_packet`
Failure	`ERROR`

Pseudocode

```
int addr, i;
int tmp;

if(x<0) x = 0;
if(x > 15) x = 15;
If(y < 0 ) y = 0;
If(y > 1) y =1;

Addr = 0x80 + 0x40 * y + x;
send_command(addr);

tmp = strlen(addr);

For (i=0; i < tmp; i++) {
send_data(data[i]);
```

send_ACK

Purpose

Sends ACK back to client. That we have received a message.

Parameters

Return

Type	Next State
Success	`SEND_ACK`
Failure	`ERROR`

Pseudocode

```
init_rudp_header(RUDP_ACK, packet.header.seq_no,
&response_packet_header);
Response_packet =
create_rudp_packet_malloc(*response_packet_header, 0, NULL);
sendto(opts-> socket_fd, response_packet,
sizeof(rudp_packet_t), 0, (const struct sock addr*)
proxy_addr, sizeof(struct sockaddr_in));
```

recv_same_packet_with_prev

Purpose

Server received a duplicate packet

Parameters

Return

Type	Next State

Success	SEND_ACK
Failure	ERROR

Pseudocode

```
If (packet.header.seq.no != current_seq_no) {

 init_rudp_header(RUDP_ACK,packet.header.seq_no,&respons
 e_packet_header);

 Response_packet =
 (create_rudp_packet_malloc(&response_packet_header, 0,
 NULL);

 sendto(opts->sock_fd, response_packet,
 sizeof(rudp_packet_t), 0, (const struct sockaddr *)
 proxy_addr, sizeof(struct sockaddr_in));
 free(response_packet);
 Continue;
```

recv_corrupted_packet

Purpose

Check if packet is corrupted.

Parameters

Return

Type	Next State
Success	SEND_NAK
Failure	ERROR

Pseudocode

```
Unit16_t check_sum_of_data_in_pack;
```

```
Check_sum_of_data_in_pack = generate_crc16(packet.data,
packet.data_length);
```

send_NAK

Purpose

Check if packet is corrupted.

Parameters

Return

Type	Next State
Success	`ready_to_recv_packet`
Failure	`ERROR`

Pseudocode

```
If (check_sum_of_data_in_pack != packet.check_sum)
{
 init_rudp_header(RUDP_NAK, packet.header.seq_no,
&response_packet_header);
Response_packet =
(create_rudp_packet_malloc(&response_packet_header, 0,
NULL);
sendto(opts->sock_fd, response_packet,
sizeof(rudp_packet_t), 0, (const struct sockaddr *)
proxy_addr, sizeof(struct sockaddr_in));
continue;
}
```

recv_fin

Purpose

Check if packet is a fin packet

Parameters

Return

Type	Next State
Success	`send_ack_to_fin`
Failure	`ERROR`

Pseudocode

```
If (packet.header.packet_type == RUDP_FIN)
{
 If (current_seq_no != -1) {
fprintf(stdout, “Finish the message transmission”);}
}

init_rudp_header(RUDP_ACK, packet.header.seq_no,
&response_packet_header);

Response_packet =
(create_rudp_packet_malloc(&response_packet_header, 0, NULL);

sendto(opts->sock_fd, response_packet, sizeof(rudp_packet_t),
0, (const struct sockaddr *) proxy_addr, sizeof(struct
sockaddr_in));
Current_seq_no = -1;
}
```

send_ack_to_fin

Purpose

Send acknowledgement to fin packet

Parameters

Return

Type	Next State
Success	`ready_to_recv_packet`
Failure	`ERROR`

Pseudocode

```
sendto(opts->sock_fd, response_packet, sizeof(rudp_packet_t), 0, (const struct sockaddr *) proxy_addr, sizeof(struct sockaddr_in));
Current_seq_no = -1;
}
```

Client

socket

Purpose

Create a socket

Parameters

Domain, type, and protocol to set up.

Return

Type	Next State
Success	`READY_TO_RECV_PACKET`
Failure	`ERROR`

Pseudocode

```
If any errors occur
     set state.fatal_error to true and return ERROR

Set sock = socket(AF_INET, SOCK_STREAM, 0)
     If sock fails;
     Error.
```

bind

Purpose

Bind socket to port

Parameters

Socket descriptor, address, length of address

Return

Type	Next State
Success	READY_TO_RECV_PACKET
Failure	ERROR

Pseudocode

```
If any errors occur
     set state.fatal_error to true and return ERROR

Set bind(socket, (struct sockaddr *) &local, sizeof(struct
sockaddr_un)) == 1);
     If bind fail;
     Error.
```

execute_commands

Purpose

Parse the commands to separate the command name, arguments, and I/O redirection.

Parameters

The state object to store the command data in.

Return

Type	Next State
“exit” command	`EXIT`
Success	`RESET_STATE`
Failure	`ERROR`

Pseudocode

```
if state.command.command is “cd”
     call builtin_cd()
otherwise, if state.command.command is “exit”
     return EXIT
otherwise, call execute()
     If execute has an error
          set state.fatal_error to true

print the state.command.exit_code to stdout

If state.fatal_error is true
     return ERROR

return RESET_STATE
```

prompt_user_input

Purpose

Get the input that the user wants to send

Parameters

Get from stdin

Return

Type	Next State
Success	`READ RFID, write_user_input_to_RFID`
Failure	`ERROR`

Pseudocode

```
printf("Enter data to be written");
scanf("%s", data);
printf("Reading..place the rfid card");

while(fgets(buffer, MAX_DATA_LEN, stdin) != NULL)...
```

write_user_input_to_RFID

Purpose

Write user input to RFID_CARD

Parameters

Get from stdin

Return

Type	Next State
Success	`data_saved_to_rfid`
Failure	`ERROR`

Pseudocode

```
while(1) {
Status = write_card_data(data);
If (status == MI_OK) {
break;
}
```

read_rfid

Purpose

Read to rfid card

Parameters

Get from stdin

Return

Type	Next State
Success	`ready_to_send`
Failure	`ERROR`

Pseudocode

```
while(fgets(buffer, MAX_DATA_LEN, stdin) != NULL) {
while(1) {
Status = write_card_data(data);
If (status == MI_OK) {
break;
}
```

ready_to_send

Purpose

Send packet to server

Parameters

Socket_fd, packet, sizeof(rudp_packet), const struct sockaddr * proxy addr, sizeof(struct sockaddr_in)

Return

Type	Next State
Success	`wait_for_ack`
Failure	`ERROR`

Pseudocode

```
Rudp_header_t header;
header.packet_type = RUDP_SYN;
header.seq_no = current_seq;

Nwrote = sendto(opts->socket_fd, packet,
sizeof(rudp_packet_t), 0, (const struct sockaddr *)
proxy_addr, sizeof(struct sockaddr_in));

If (nwrote == -1) {
```

```
free(packet);
Return MY_FAILURE_CODE;
}
```

wait_for_ack

Purpose

Wait for response from server

Parameters

Socket_fd, packet, sizeof(rudp_packet), const struct sockaddr * proxy addr, sizeof(struct sockaddr_in)

Return

Type	Next State
Success	`ack_received`
Failure	`wait_for_ack`

Pseudocode

```
nread = recvfrom(opts-> socketfd, &resp_packet,
sizeof(rudpPacket), 0, (struct sockaddr *)from_addr,
&from_addr_len);

if(nread == -1) {
fprintf(stdout, “time out!! Packet sent again”);

nwrote(sendto(opts->socket_fd, packet, sizeof(rudp_packet_t),
0, (const struct sockaddr *) proxy_addr, sizeof(struct
sockaddr_in));

If (nwrote == -1) {
free(packet);
Return failure_code;
Goto wait_response_packet;
```