```
1  /* Uses threads to handle multiple simultaneous clients */
2  //run with port # as the first argument
3  //talk to using netcat, e.g.:
4  //nc 127.0.0.1 10689
5  //supports multiple concurrent TCP clients
6
7  #include <pthread.h>
8  #include "common.h"
9  #include "1_util.h"
10 #include "2_threadserver_handle.h"
11
12 #define MAXKIDS 10
13 int stop = 0;
14 int sock = 0;
15 struct addrinfo *listen_addr = NULL;
16 char *output = NULL;
17 int children = 0;
18 int total_children = 0;
19 pthread_t mychildren[MAXKIDS] = { 0 };
20 char lastmsg[MAXLENGTH] = { '\0' };
21
22 void register_tcp_handlers()
23 {
24         struct sigaction actinfo;
25         actinfo.sa_handler = handler2;
26         sigfillset(&actinfo.sa_mask); //todo check for error
27         actinfo.sa_flags = 0;
28         sigaction(SIGINT, &actinfo, 0); //todo check for error
29         sigaction(SIGHUP, &actinfo, 0); //todo check for error
30         sigaction(SIGTERM, &actinfo, 0); //todo check for error
31         actinfo.sa_handler = SIG_IGN;
32         sigaction(SIGPIPE, &actinfo, 0); //todo check for error
33 }
34
35 void prepare_generic_socket(int argc, char *argv[],
36                 int family, int flags, int type, int protocol)
37 {
38         struct addrinfo lookup_addr;
39         memset(&lookup_addr, 0, sizeof(struct addrinfo));
40         lookup_addr.ai_family = family;
41         lookup_addr.ai_flags = flags;
42         lookup_addr.ai_socktype = type;
43         lookup_addr.ai_protocol = protocol;
44
45         if (getaddrinfo(NULL, argv[1], &lookup_addr, &listen_addr) != 0)
46         {
47                 exit_with_error("getaddrinfo failed");
48         }
49
50         sock = socket(listen_addr->ai_family, listen_addr->ai_socktype,
51                         listen_addr->ai_protocol);
52         if (sock < 0)
53         {
54                 exit_with_error("socket failed");
55         }
56         if (bind(sock, listen_addr->ai_addr,
57                                 listen_addr->ai_addrlen) < 0)
58         {
59                 exit_with_error("bind failed");
60         }
61
62 }
63
```

```
64  //this function sends a generic data buffer via tcp
65  void send_tcp_data(FILE* rx, FILE *tx, void *data, int datalen)
66  {
67          if (tx)
68          {
69                  size_t bytes_sent = fwrite(data, 1, datalen, tx);
70                  if (errno == EPIPE)
71                  {
72                          errno = 0;
73                          fclose(tx);
74                          fclose(rx);
75                          exit_with_error("pipe error\n");
76                  }
77                  else if (bytes_sent < 0)
78                  {
79                          fclose(tx);
80                          fclose(rx);
81                          exit_with_error("send failed");
82                  }
83                  fflush(tx);
84          }
85          else
86          {
87                  sprint_hex(data, datalen);
88          }
89  }
90
91  void join_all_children()
92  {
93          int children_to_join = total_children;
94          while (children_to_join)
95          {
96                  int i;
97                  for (i = 0; i < MAXKIDS; i++)
98                  {
99                          if (mychildren[i] != 0)
100                         {
101                                 pthread_join(mychildren[i], NULL);
102                                 children_to_join--;
103                                 printf("Child closed I now have %d "\
104                                                 "kids left to join\n",
105                                         children_to_join);
106                         }
107                 }
108         }
109 }
110
111 void receive_multi_tcp_clients()
112 {
113         while (!stop)
114         {
115                 struct sockaddr_storage client_addr;
116                 socklen_t addr_len = sizeof(struct sockaddr_storage);
117
118                 int client = accept(sock,
119                                         (struct sockaddr *) &client_addr,
120                                         &addr_len);
121                 if (stop)
122                 {
123                         break;
124                 }
125
126                 if (errno == EINTR && client < 0)
```

```
127                     {
128                             printf("accept got EINTR, re-attempting\n");
129                             continue;
130                     }
131
132                     if (client < 0)
133                     {
134                             exit_with_error("accept failed");
135                     }
136
137                     int *arg = malloc(sizeof(int));
138                     *arg = client;
139                     int result = pthread_create(&mychildren[total_children],
140                                     NULL, handle_tcp_client, arg);
141                     if (result != 0)
142                     {
143                             exit_with_error("pthread_create failed");
144                     }
145
146                     total_children++;
147                     children++;
148                     printf("started child thread I now have %d kids and I had %d\n",
149                                     children, total_children);
150                     if (children == MAXKIDS)
151                     {
152                             break;
153                     }
154             }
155
156             join_all_children();
157     }
158
```

```
1  #include "common.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <ctype.h>
5  #include <string.h>
6  #include <assert.h>
7  #include "2_threadserver_handle.h"
8
9  void connected_mode(FILE *tx, FILE *rx, int client, char username[])
10 {
11 connect_mode:;
12         char *buffer = NULL;
13         size_t len = 0;
14         int timer = 0;
15         char email_port[] = "10691";
16         char dropbox_port[] = "10692";
17         char *port = NULL;
18         buffer = "Choose one of the following:\n";
19         ssize_t bytes_received = strlen(buffer);
20         send_tcp_data(rx, tx, buffer, bytes_received);
21         buffer = NULL;
22         buffer = "Email\tDropbox\n";
23         bytes_received = strlen(buffer);
24         send_tcp_data(rx, tx, buffer, bytes_received);
25         buffer = NULL;
26         int trial = 0;
27 read_data:bytes_received = getline(&buffer, &len, rx);
28
29         if (bytes_received < 0 && errno != EINTR)
30         {
31                 goto close;
32         }
33
34         if (bytes_received > 0 || errno == EINTR)
35         {
36                 while (bytes_received == 0 && errno == EINTR)
37                 {
38                         printf("getline got EINTR, re-attempting\n");
39                         bytes_received = getline(&buffer, &len, rx);
40                 }
41                 buffer[bytes_received] = '\0';
42
43                 if ((bytes_received == 1) && (buffer[bytes_received] = '\0'))
44                 {
45                         char display_3[] = "Invalid Input. Try again\n";
46                         send_tcp_data(rx,tx,display_3,25);
47                         goto read_data;
48                 }
49                 char message[MAXLENGTH];
50                 int i =0;
51                 for (i=0;buffer[i+1]!='\0';i++)
52                 {
53                         message[i] = buffer[i];
54                 }
55                 message[i]='\0';
56
57                 if (trial == 0)
58                 {
59                         if (strcmp(message,"Email") == 0)
60                         {
61                                 port = email_port;
62                         }
63                         else if (strcmp(message,"Dropbox") == 0)
```

```
64                              {
65                                      port = dropbox_port;
66                              }
67                              else if (strcmp(message,"Quit") == 0)
68                              {
69                                      goto close;
70                              }
71                              else
72                              {
73                                      timer++;
74                                      if (timer <= 5)
75                                      {
76                                              char display_1[] = "Invalid option. Try Again\n"
    ;
77                                              send_tcp_data(rx, tx, display_1, 27);
78                                              goto read_data;
79                                      }
80                                      else
81                                      {
82                                              char display_2[] = "You have exceeded maximu
    m number of attempts. Goodbye!\n";
83                                              send_tcp_data(rx, tx, display_2, 56);
84                                              goto close;
85                                      }
86                              }
87
88                              int initiate = 0;
89                              initiate = initialize_client(port);
90
91                              if (initiate == 1)
92                              {
93                                      char display_4[] = "Sorry! Email or Dropbox program unavai
    lable\n";
94                                      send_tcp_data(rx, tx, display_4,33);
95                                      goto close;
96                              }
97                              int status_username = 0;
98                              status_username = query_email_or_dropbox(message,port,us
    ername,trial,rx,tx);
99                              if (status_username == 2)
100                             {
101                                     printf("got status 2 after sending username\n");
102                                     goto close;
103                             }
104                             else
105                             {
106                                     ;
107                             }
108                     }
109                     else
110                     {
111                             if (strcmp(message,"Quit") == 0)
112                                     goto close;
113                             int status = 0;
114                             status = query_email_or_dropbox(message,port,username,tr
    ial,rx,tx);
115                             if ((status == 2) ||(status == 3))
116                             {
117                                     //printf("control connect received status = %d\n
    ",status);
118                                     goto connect_mode;
119                             }
120                             else if (status == 1)
```

```
121                                     goto close;
122                             else
123                                     ;
124                     }
125                     trial++;
126             }
127             goto read_data;
128
129
130     close:
131             if (bytes_received <= 0)
132             {
133                     printf("client closed the connection!\n");
134             }
135             if (buffer != NULL)
136             {
137                     free(buffer);
138             }
139             return;
140     }
```

```
1  #include <stdio.h>
2  #include "2_threadserver_handle.h"
3  #include "common.h"
4  #include <stdlib.h>
5  #include <sys/socket.h>
6  #include <sys/types.h>
7  #include <netdb.h>
8  #include <unistd.h>
9
10 FILE *tx;
11 FILE *rx;
12 struct addrinfo *send_addr;
13
14 int initialize_user()
15 {
16         struct addrinfo lookup_addr;
17         memset(&lookup_addr, 0, sizeof(struct addrinfo));
18         lookup_addr.ai_family = AF_UNSPEC;
19         lookup_addr.ai_socktype = SOCK_STREAM;
20         lookup_addr.ai_protocol = IPPROTO_TCP;
21
22         //struct addrinfo *send_addr;
23         if (getaddrinfo("127.0.0.1", "10690", &lookup_addr, &send_addr) != 0)
24         {
25                 #ifdef DEBUG
26                 perror("getaddrinfo failed");
27                 #endif
28                 return 1;
29         }
30
31         int sock = socket(send_addr->ai_family, send_addr->ai_socktype,
32                         send_addr->ai_protocol);
33         if (sock < 0)
34         {
35                 #ifdef DEBUG
36                 perror("socket failed");
37                 #endif
38                 return 1;
39         }
40
41         if (connect(sock, send_addr->ai_addr, send_addr->ai_addrlen) < 0)
42         {
43                 #ifdef DEBUG
44                 perror("connect failed");
45                 #endif
46                 return 1;
47         }
48
49         tx = fdopen(sock, "w");
50         rx = fdopen(dup(sock),"r");
51
52         return 0;
53
54 }
55
56 int query_user_mgmt(char username[], char password[],FILE *rx_client, FILE *tx_c
   lient)
57 {
58         int status = initialize_user();
59         if (status == 1)
60         {
61                 char display[] = "Sorry! User management server unavailable\n";
62                 send_tcp_data(rx_client, tx_client, display, strlen(display));
```

```
63                      return 2;
64              }
65
66              server_id = 1;
67              size_t bytes_sent = fwrite(&server_id,1,4,tx);
68              if (bytes_sent != 4)
69              {
70                      #ifdef DEBUG
71                      perror("sendto failed");
72                      #endif
73                      return 1;
74              }
75              fflush(tx);
76
77              int datalen = sizeof(struct users);
78              struct users *tosend = malloc(datalen);
79              tosend->online = 1;
80              strcpy(tosend->username,username);
81              strcpy(tosend->password,password);
82
83              bytes_sent = fwrite(tosend,1,datalen,tx);
84
85              if(bytes_sent != (datalen))
86              {
87                      #ifdef DEBUG
88                      perror("sendto failed");
89                      #endif
90                      free(tosend);
91                      return 1;
92              }
93              fflush(tx);
94
95              char torecv = '\0';
96              ssize_t bytes_received = fread(&torecv,1,1,rx);
97
98              if(errno == EPIPE)
99              {
100                     errno = 0;
101                     #ifdef DEBUG
102                     perror("Pipe Error\n");
103                     #endif
104                     if (send_addr)
105                     {
106                             freeaddrinfo(send_addr);
107                     }
108                     free(tosend);
109                     fclose(tx);
110                     fclose(rx);
111             }
112
113             else if(bytes_received <= 0)
114             {
115                     #ifdef DEBUG
116                     perror("recvfrom failed\n");
117                     #endif
118                     free(tosend);
119                     fclose(rx);
120                     fclose(tx);
121                     return 1;
122             }
123             else if(bytes_received > 1)
124             {
125                     free(tosend);
```

```
126                     fclose(rx);
127                     fclose(tx);
128                     return 1;
129             }
130             else
131             {
132                     free(tosend);
133                     fclose(rx);
134                     fclose(tx);
135                     return 0;
136             }
137             fclose(rx);
138             free(tosend);
139             fclose(tx);
140
141             return 0;
142     }
```

```
1  #ifndef COMMON_H
2  #define COMMON_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <sys/socket.h>
8  #include <sys/types.h>
9  #include <netdb.h>
10 #include <unistd.h>
11 #include <assert.h>
12 #include <errno.h>
13 #include <signal.h>
14
15 #define MAXLENGTH 2000
16
17 /**
18   * This struct is used to pass information between all the programs
19   * in the system
20   * code = 3 -- corresponds to information conserning the user (ex. username)
21   * code = 4 -- corresponds to all infromation to and from the email program
22   * code = 5 -- corresponds to all infromation to and from the drive program
23   */
24 struct data{
25         uint8_t code;           ///< used to identity the type of data
26         int64_t len;            ///< holds the length of the corresponding messa
   ge
27         char msg;               ///< points to the start of the message
28 } __attribute__((packed));
29
30 /**
31   * This structs is used to hold information about each user
32   */
33 struct users{
34         int8_t online;          ///< used to check if the user is online
35         char username[10];      ///< points to the user name
36         char password[10];              ///< holds the hash value of the user's
   password
37 } __attribute__((packed));
38
39 /**
40   * This struct is used for storing and viewing email information
41   */
42 struct email{
43         char to[10];            ///< the user the mail is addressed to
44         char from[10];          ///< the user that originates the mail
45         char subject[100];      ///< subject of the mail
46         char msg[2000];         ///< body of the mail
47 }__attribute__((packed));
48
49 int server_id;
50 int initialize();
51 int client_func(char *user,int option);
52 int initialize_client(char *port);
53 void close_up();
54 int check_user(struct data *recv, FILE *rx);
55 int get_message(FILE *rx);
56 int check_inbox(FILE *rx,FILE *tx);
57 int compose_mail(FILE *rx, FILE *tx);
58 #endif
```

```
1  #include "common.h"
2
3  #ifndef UTIL_H_3
4  #define UTIL_H_3
5
6  extern int stop;
7  extern int sock;
8  extern struct addrinfo *listen_addr;
9  extern char *output;
10
11 void sprint_hex(uint8_t *data, size_t length);
12
13 void handler2(int signal);
14
15 void exit_with_error(char *msg);
16
17 void register_handler();
18
19 void cleanup();
20
21 #endif
```

```
1  #include "1_util.h"
2
3  //this function prints a generic data buffer to
4  //the "output" char array (for unit testing)
5  void sprint_hex(uint8_t *data, size_t length)
6  {
7          char myoutput[MAXLENGTH];
8          char tmp[MAXLENGTH];
9          assert(length < 100);
10         myoutput[0] = '\0';
11
12         int i;
13         for (i = 0; i < length; i++)
14         {
15                 sprintf(tmp, "%02x ", data[i]);
16                 strcat(myoutput, tmp);
17                 if (i % 16 == 15)
18                 {
19                         strcat(myoutput, "\n");
20                 }
21         }
22         if (myoutput[strlen(myoutput)-1] != '\n')
23         {
24                 strcat(myoutput, "\n");
25         }
26
27         output = strdup(myoutput);
28 };
29
30 void handler2(int signal)
31 {
32         stop = 1;
33         if (sock != 0)
34         {
35                 close(sock);
36         }
37 }
38
39 void cleanup()
40 {
41         if (listen_addr)
42                 freeaddrinfo(listen_addr);
43         if (sock)
44                 close(sock);
45 }
46
47 void exit_with_error(char *msg)
48 {
49         perror(msg);
50         cleanup();
51         exit(1);
52 }
```

```
#include <pthread.h>
#include "common.h"
#include "1_util.h"
#include "2_threadserver_handle.h"


int main(int argc, char *argv[])
{
        register_tcp_handlers();

        prepare_generic_socket(argc, argv, AF_INET, AI_PASSIVE, SOCK_STREAM, IPPROTO_TCP);

        if (listen(sock, 1) < 0)
        {
                exit_with_error("listen failed");
        }

        receive_multi_tcp_clients();

        cleanup();

        return 0;
}

```

```
1  #include "1_util.h"
2  #include "2_threadserver_handle.h"
3  #include <stdio.h>
4  #include <string.h>
5  #include "common.h"
6
7  //This function processes the data received by a single client
8  void *handle_tcp_client(void *arg)
9  {
10         int client = *((int *) arg);
11         free(arg);
12
13         char *buffer = NULL;
14         size_t len = 0;
15
16         uint8_t flag = 1;
17
18         FILE *rx = fdopen(client, "r");
19         FILE *tx = fdopen(dup(client), "w");
20         uint8_t no_of_attempts = 0;
21         uint8_t count = 0;
22         char *username = NULL;
23         char *password = NULL;
24
25         buffer = "Username : ";
26         ssize_t bytes_received = 11;
27         send_tcp_data(rx, tx, buffer, bytes_received);
28
29 read_data:buffer = NULL;
30         bytes_received = getline(&buffer, &len, rx);
31
32         if (bytes_received < 0 && errno != EINTR)
33         {
34                 goto close;
35         }
36
37         while (bytes_received > 0 || errno == EINTR)
38         {
39                 while (bytes_received == 0 && errno == EINTR)
40                 {
41                         printf("getline got EINTR, re-attempting\n");
42                         bytes_received = getline(&buffer, &len, rx);
43                 }
44                 buffer[bytes_received] = '\0';
45
46                 if ((bytes_received == 1) && (buffer[0] = '\0'))
47                 {
48                         goto error_msg;
49                 }
50                 char credential[MAXLENGTH];
51                 int i =0;
52                 for (i=0;buffer[i+1]!='\0';i++)
53                 {
54                         credential[i] = buffer[i];
55                 }
56                 credential[i]='\0';
57
58                 int length = strlen(credential);
59                 if (length > 10)
60                 {
61                         char disp_err[] = "Please enter less than or equal to 10 characters\n";
62                         send_tcp_data(rx,tx,disp_err,strlen(disp_err));
63                         no_of_attempts++;
```

```
64                              if (no_of_attempts > 5)
65                              {
66                                      char disp[] = "You have exceeded maximum number of attempt
      s. Goodbye!";
67                                      send_tcp_data(rx,tx,disp,strlen(disp));
68                                      goto close;
69                              }
70                              else
71                              {
72                                      goto read_data;
73                              }
74                      }
75
76                      if (count == 0)
77                      {
78                              username = malloc(sizeof(char *)*(length+1));
79                              strcpy(username,credential);
80                              if (username[0]!='\0')
81                                      flag = 0;
82                              else
83                                      flag = 1;
84                      }
85                      else if ((count == 1)&&(flag == 0))
86                      {
87                              flag = 1;
88                              password = malloc(sizeof(char *)*(length+1));
89                              strcpy(password,credential);
90
91                              flag = query_user_mgmt(username,password,rx,tx);
92                              if (flag == 0)
93                              {
94                                      connected_mode(tx, rx, client, username);
95                              }
96                              else if (flag == 2)
97                              {
98                                      goto close;
99                              }
100                             else
101                             {
102                                     goto error_msg;
103                             }
104                     }
105                     count++;
106                     if (count < 2)
107                     {
108                             buffer = "Password : ";
109                             bytes_received = 11;
110                             send_tcp_data(rx, tx, buffer, bytes_received);
111                             buffer = NULL;
112                             bytes_received = getline(&buffer, &len, rx);
113                     }
114                     else
115                     {
116                             goto close;
117                     }
118             }
119
120     error_msg:
121             {
122             char data[] = {"Invalid username or password. Goodbye!"};
123             send_tcp_data(rx,tx,data,strlen(data));
124             goto close;
125             }
```

```
126
127 close:
128         if (bytes_received <= 0)
129         {
130                 printf("client closed the connection!\n");
131         }
132
133         fclose(tx);
134         fclose(rx);
135
136         if (buffer != NULL)
137         {
138                 free(buffer);
139         }
140         free(username);
141         free(password);
142         children--;
143
144         return NULL;
145 }
146
147
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include "common.h"
5  #include "2_threadserver_handle.h"
6  #include <sys/socket.h>
7  #include <sys/types.h>
8  #include <unistd.h>
9  #include <netdb.h>
10
11 FILE *tx = NULL;
12 FILE *rx = NULL;
13 struct addrinfo *send_addr;
14
15 int initialize_client(char *port)
16 {
17         struct addrinfo lookup_addr;
18         memset(&lookup_addr, 0, sizeof(struct addrinfo));
19         lookup_addr.ai_family = AF_UNSPEC;
20         lookup_addr.ai_socktype = SOCK_STREAM;
21         lookup_addr.ai_protocol = IPPROTO_TCP;
22
23         struct addrinfo *send_addr;
24         if (getaddrinfo("127.0.0.1", port, &lookup_addr, &send_addr) != 0)
25         {
26                 #ifdef DEBUG
27                 perror("getaddrinfo failed");
28                 #endif
29                 printf("getaddrinfo has failed\n");
30                 return 1;
31         }
32
33         int sock = socket(send_addr->ai_family, send_addr->ai_socktype,
34                         send_addr->ai_protocol);
35         if (sock < 0)
36         {
37                 #ifdef DEBUG
38                 perror("socket failed");
39                 #endif
40                 printf("socket has failed\n");
41                 return 1;
42         }
43
44         if (connect(sock, send_addr->ai_addr, send_addr->ai_addrlen) < 0)
45         {
46                 #ifdef DEBUG
47                 perror("connect failed");
48                 #endif
49                 printf("connect has failed\n");
50                 return 1;
51         }
52
53         tx = fdopen(sock, "w");
54         rx = fdopen(dup(sock),"r");
55
56         return 0;
57 }
58
59 int query_email_or_dropbox(char message[] ,char *port,char username[],int trial,
    FILE *rx_client, FILE *tx_client)
60 {
61         char *email_port = "10691";
62         char *dropbox_port = "10692";
```

```
63          uint64_t datalen = 0;
64          struct data *tosend = NULL;
65          //int status = 0;
66          if (trial == 0)
67          {
68                  printf("username is %d bytes\n",strlen(username));
69                  datalen = sizeof(struct data) + strlen(username) - 1;
70                  tosend = malloc(datalen);
71                  tosend->code = 3;
72                  tosend->len = strlen(username);
73                  strncpy(&(tosend->msg),username,strlen(username));
74          }
75          else
76          {
77                  datalen = sizeof(struct data) + strlen(message) - 1;
78                  tosend = malloc(datalen);
79                  if (strcmp(port,email_port) == 0)
80                  {
81                          tosend->code = 4;
82                  }
83                  else if (strcmp(port,dropbox_port) == 0)
84                  {
85                          tosend->code = 5;
86                  }
87                  else
88                  {
89                          printf("received incorrect port no\n");
90                  }
91                  tosend->len = strlen(message);
92                  strncpy(&(tosend->msg),message,strlen(message));
93          }
94
95          size_t bytes_sent = fwrite(tosend,1,datalen,tx);
96
97          if(bytes_sent != (datalen))
98          {
99                  #ifdef DEBUG
100                 perror("sendto failed");
101                 #endif
102                 printf("fwrite failed\n");
103                 free(tosend);
104                 return 1;
105         }
106         fflush(tx);
107
108         struct data *torecv = malloc(sizeof(struct data));
109         size_t bytes_received = fread(torecv,1,sizeof(struct data)-1,rx);
110
111         if ((bytes_received < 0)&&(errno != EINTR))
112         {
113                 fclose(tx);
114                 fclose(rx);
115                 printf("fread failed\n");
116                 close_up();
117         }
118
119         while ((bytes_received > 0))
120         {
121                 if ((bytes_received == 0) && (errno != EINTR))
122                 {
123                         errno = 0;
124                         printf("fread got EINTR, re-attempting\n");
125                         close_up();
```

```
126                             return 3;
127                     }
128                     if ((bytes_received == 0) && (errno = EINTR))
129                     {
130                             errno = 0;
131                             printf("fread got EINTR, re−attempting\n");
132                             close_up();
133                             return 2;
134                     }
135
136                     if (torecv->code == 4)
137                     {
138                             printf("msg received from email server\n");
139                     }
140                     else if (torecv->code == 5)
141                     {
142                             printf("msg received from dropbox server\n");
143                     }
144                     else
145                     {
146                             printf("invalid code received\n");
147                             close_up();
148                     }
149
150                     uint64_t recv_len = torecv->len;
151
152                     char *received_string = malloc(recv_len + 1);
153                     bytes_received = fread(received_string,1,recv_len,rx);
154                     received_string[recv_len] = '\0';
155
156                     if (bytes_received != recv_len)
157                     {
158                             printf("wrong size of received string\n");
159                             close_up();
160                     }
161                     else if ((bytes_received == 1) && atoi(received_string) == 1)
162                     {
163                             return 0;
164                     }
165                     else
166                     {
167                             send_tcp_data(rx_client,tx_client,received_string,recv_l
    en+1);
168                             bytes_received = fread(torecv,1,sizeof(struct data)−1,rx
    );
169                     }
170             }
171             return 0;
172     }
173
174     void close_up()
175     {
176             freeaddrinfo(send_addr);
177             fclose(tx);
178             fclose(rx);
179     }
```

2_threadserver_handle.h


```
1  #ifndef _SERVERFILE_HANDLE_
2  #define _SERVERFILE_HANDLE_
3
4  #include <stdio.h>
5  #include "common.h"
6
7  extern int children;
8
9  void register_tcp_handlers();
10
11 void send_tcp_data(FILE* rx, FILE *tx, void *data, int datalen);
12
13 void prepare_generic_socket(int argc, char *argv[],
14                 int family, int flags, int type, int protocol);
15
16 void join_all_children();
17
18 void receive_multi_tcp_clients();
19
20
21 void *handle_tcp_client(void *arg);
22
23 int query_user_mgmt(char username[], char password[],FILE *rx_client, FILE *tx_c
   lient);
24
25 void connected_mode(FILE *tx, FILE *rx, int client, char username[]);
26
27 int initialize_user();
28
29 int query_email_or_dropbox(char message[] ,char *port,char username[],int trial,
    FILE *rx_client, FILE *tx_client);
30
31 uint8_t validate_username_password(char username[], char password[]);
32
33 void convert_to_uppercase(char message[], uint64_t msg_len);
34
35
36 #endif
```

```
1  #include "1_util.h"
2  #include "2_threadserver_handle.h"
3  #include <stdio.h>
4  #include <string.h>
5  #include "common.h"
6  #include "drive.h"
7  //creating database
8  struct users Deepak = {0, .username = "deepak", .password = "deepak"};
9  struct users Deepika = {0,.username = "deepika", .password = "deepika"};
10 struct users Pooja = {0,.username = "pooja", .password = "pooja"};
11 struct users Justin = {0,.username = "justin", .password = "justin"};
12
13
14 char user_list[4][10] = {"deepak","deepika","pooja","justin"};
15
16 /*
17  *@brief function checks if the user exits in the database
18  *@parameter function takes username entered by the user
19  *@return function returns 0 if user exits else 1
20 */
21
22 uint8_t validate_username(char username[])
23 {
24         Deepak.online = 0;
25         Deepika.online = 0;
26         Pooja.online = 0;
27         Justin.online = 0;
28
29         if (strncmp(username,Deepak.username,6) == 0)
30         {
31                 printf("Deepak is online\n");
32                 Deepak.online = 1;
33                 return 0;
34         }
35         else if (strncmp(username,Deepika.username,7) == 0)
36         {
37                 printf("Deepika is online");
38                 Deepika.online = 1;
39                 return 0;
40         }
41         else if (strncmp(username, Pooja.username,5) == 0)
42         {
43                 printf("Pooja is online\n");
44                 Pooja.online = 1;
45                 return 0;
46         }
47         else if (strncmp(username, Justin.username,6) == 0)
48         {
49                 printf("Justin is online");
50                 Justin.online = 1;
51                 return 0;
52         }
53         else
54         {
55                 printf("noone is online\n");
56                 return 1;
57         }
58 }
59
60 /*
61  *@brief function checks if the password is correct
62  *@parameter function takes username and password entered by the user
63  *@return function returns 0 if user exits else 1
```

```
64  */
65
66  uint8_t validate_password(char password[])
67  {
68          if ((Deepak.online == 1) && (strncmp(Deepak.password,password,6) == 0))
69          {
70                  printf("Deepak has been authenticated\n");
71                  return 0;
72          }
73          else if ((Deepika.online == 1 ) && (strncmp(Deepika.password,password,7)
     == 0))
74          {
75                  printf("Deepika has been authenticated\n");
76                  return 0;
77          }
78          else if ((Pooja.online == 1 ) && (strncmp(Pooja.password,password,5) ==
    0))
79          {
80                  printf("Pooja has been authenticated\n");
81                  return 0;
82          }
83          else if ((Justin.online == 1 ) && (strncmp(Justin.password,password,6) =
    = 0))
84          {
85                  printf("Justin has been authenticated\n");
86                  return 0;
87          }
88          else
89          {
90                  printf("noone is authenticated\n");
91                  return 1;
92          }
93  }
94
95  /**
96    * This function verifies if a user is valid or not
97    * rx points to the receiving socket
98    * tx points to the transmitting socket
99    */
100 int verify_user(FILE *rx, FILE *tx, struct data *recv){
101
102         if(recv->code != 4)
103                 return 1;
104         char *store = malloc(recv->len + 1);
105         fread(store,1,recv->len,rx);
106         store[recv->len] = '\0';
107         char present[2] = {'0','\0'};
108         int i = 0;
109         for(i = 0;i < 3;i++){
110                 if(strcmp(user_list[i],store) == 0){
111                         present[0] = '1';
112                 }
113         }
114         free(store);
115         struct data *tosend = malloc(sizeof(struct data));
116         tosend->code = 4;
117         tosend->len = strlen(present);
118         strcpy(&(tosend->msg),present);
119         send_tcp_data(rx,tx,tosend,10);
120         free(tosend);
121         return 0;
122 }
123
```

```
124
125 /**
126   * This function checks transfer func
127   * @param rx points to the receiving socket
128   * @param tx points to the transmitting socket
129   * @param recvmsg user msg
130   * @param username name of the user
131   * @param original_dirpath from which the user gets the file
132   * @return 0/1 Success/Failure
133   */
134
135 int check_transfer_func(FILE *tx,FILE *rx,char *recvmsg,char *username,char *ori
    ginal_dirpath)
136 {
137       char splitmsg1[MAXLENGTH] = {0};
138       char splitmsg2[MAXLENGTH] = {0};
139       char splitmsg3[MAXLENGTH] = {0};
140       sscanf(recvmsg,"%s %s %s",splitmsg1,splitmsg2,splitmsg3);
141
142         int changedir = chdir(original_dirpath);
143         if((changedir == -1) && (errno == ENOTDIR))
144         {
145                 errno = 0;
146                 printf("Cannot change directory\n");
147                 return 1;
148         }
149         char *curworkdir = NULL;
150         char buf[PATH_MAX+1];
151         curworkdir = getcwd(buf,PATH_MAX+1);
152         if(curworkdir != NULL)
153         {
154         }
155
156         int pathlen1 = strlen(curworkdir) + 1;
157         char *dirpath = NULL;
158         dirpath = strncat(curworkdir,"/",pathlen1);
159         changedir = chdir(dirpath);
160         if((changedir == -1) && (errno == ENOTDIR))
161         {
162                 errno = 0;
163                 printf("Cannot change directory\n");
164                 return 1;
165         }
166
167         curworkdir = getcwd(buf,PATH_MAX+1);
168         if(curworkdir != NULL)
169         {
170         }
171
172         FILE *sendfile = NULL;
173         if(strcmp(splitmsg1,"Transfer")==0)
174         {
175         sendfile = fopen(splitmsg2,"r");
176                 if(sendfile!=NULL)
177                 {
178
179                 char tempfile[2000]="";
180                 while(1)
181                 {
182                 fread(tempfile,1,2000,sendfile);
183                 size_t bytes_sent = fwrite(tempfile, 1,sizeof(tempfile),tx);
184                 if (bytes_sent != sizeof(tempfile))
185                 {
```

```
186                             printf("%d\n", bytes_sent);
187                             perror("fwrite 1 failed");
188                             fclose(tx);
189                             return 1;
190                     }
191                     fflush(tx);
192                     if(feof(sendfile))
193                     {
194                     break;
195                     }
196                     }
197
198                     }
199             fclose(sendfile);
200             }
201
202
203     return 0;
204     }
205
206
207
208     //This function processes the data received by a single client
209     void *handle_tcp_client(void *arg)
210     {
211             int client = *((int *) arg);
212             free(arg);
213             FILE *rx = fdopen(client, "r");
214             FILE *tx = fdopen(dup(client), "w");
215
216             ssize_t bytes_received = fread(&server_id,1,4,rx);
217
218             if (server_id == 1)     //control program
219             {
220
221                     struct users *buffer = malloc(sizeof(struct users));
222                     ssize_t bytes_received = fread(buffer,1,sizeof(struct users),rx)
    ;
223
224                     if (bytes_received < 0 && errno != EINTR)
225                     {
226                             goto close;
227                     }
228                     if ((bytes_received < 0) || (bytes_received != sizeof(struct use
    rs)))
229                     {
230                             fclose (tx);
231                             fclose (rx);
232                             exit_with_error ("fread failed");
233                     }
234
235                     if (bytes_received > 0)
236                     {
237                             int flag_username = validate_username(buffer->username);
238                             if (flag_username == 0)
239                             {
240                                     int flag_pwd = validate_password(buffer->passwor
    d);
241                                     if (flag_pwd == 0)
242                                     {
243                                             ssize_t bytes_sent = fwrite(&flag_pwd,1,
    4,tx);
244
```

```
245                                             if (bytes_sent != 4)
246                                             {
247                                                     fclose (tx);
248                                                     fclose (rx);
249                                                     exit_with_error ("fwrite failed");
250                                             }
251                                     }
252                                     else
253                                     {
254                                             printf("Password is invalid\n");
255                                             goto close;
256                                     }
257                             }
258                             else
259                             {
260                                     printf("Username is unknown\n");
261                                     goto close;
262                             }
263                     }
264             }
265             else if (server_id == 4)
266             {
267                     struct data *recv = malloc(sizeof(struct data));
268                     bytes_received = fread(recv,1,sizeof(struct data) -1,rx);
269
270                     while(bytes_received > 0){
271                             printf("here\n");
272
273                             printf("email program\n");          //email program
274                             if(verify_user(rx,tx,recv) != 0)
275                                     goto close;
276
277
278                             bytes_received = fread(recv,1,sizeof(struct data) -1,rx)
    ;
279                     }
280             }
281
282             else if (server_id == 5)
283             {
284
285                     printf("dropbox prgram");          //dropbox program
286                     //Receiving username
287                     struct data *torecv = NULL;
288                     torecv = malloc(sizeof(struct data)-1);
289                     bytes_received = fread(torecv,1,sizeof(struct data)-1,rx);
290                     uint8_t length = 0;
291                     length = torecv->len;
292                     char *recvmsg = malloc(length+1);
293                     char *username = malloc(length+1);
294                     if(fread(recvmsg,1,length,rx) != length)
295                     {
296                     exit(1);
297                     }
298                     recvmsg[length] = '\0';
299                     strcpy(username,recvmsg);
300
301
302     //Receiving Original path
303                     struct data *torecv1 = NULL;
304                     torecv1 = malloc(sizeof(struct data)-1);
305                     bytes_received = fread(torecv1,1,sizeof(struct data)-1,rx);
306                     length = torecv1->len;
```

```
307                     char *recvmsg1 = malloc(length+1);
308                     char *original_dirpath = malloc(length+1);
309                     if(fread(recvmsg1,1,length,rx) != length)
310                     {
311                     exit(1);
312                     }
313                     recvmsg1[length] = '\0';
314                     strcpy(original_dirpath,recvmsg1);
315
316     //Receivigng recvmsg transfer operation
317                     struct data *torecv2 = NULL;
318                     torecv2 = malloc(sizeof(struct data)-1);
319                     bytes_received = fread(torecv2,1,sizeof(struct data)-1,rx);
320                     length = torecv2->len;
321                     char *recvmsg2 = malloc(length+1);
322                    if(fread(recvmsg2,1,length,rx) != length)
323                     {
324                     exit(1);
325                     }
326                     recvmsg2[length] = '\0';
327                     check_transfer_func(tx,rx,recvmsg2,username,original_dirpath);

328                     free(torecv);
329                     free(torecv1);
330                     free(torecv2);
331                     free(recvmsg);
332                     free(recvmsg1);
333                     free(recvmsg2);
334                     free(username);
335                     free(original_dirpath);
336             }
337
338             else
339             {
340                     printf("Invalid server id");
341                     goto close;
342             }
343
344
345     close:  if (bytes_received <= 0)
346             {
347                     printf("connection with user management server closed!\n");
348             }
349             fclose(tx);
350             fclose(rx);
351             children--;
352             printf("now there are %d children\n", children);
353             return 0;
354     }
```

```
1  //author Deepak Ramadass
2  #include <stdio.h>
3  #include <string.h>
4  #include <sys/socket.h>
5  #include <sys/types.h>
6  #include <netdb.h>
7  #include <unistd.h>
8  #include "common.h"
9
10 #define MAXLENGTH 2000
11
12 FILE *tx = NULL;
13 FILE *rx = NULL;
14 struct addrinfo *send_addr = NULL;
15
16 int initialize(){
17
18         char *IP = "127.0.0.1";
19         char *sock_add = "10690";
20         struct addrinfo lookup_addr;
21         memset(&lookup_addr, 0, sizeof(struct addrinfo));
22         lookup_addr.ai_family = AF_UNSPEC;
23         lookup_addr.ai_socktype = SOCK_STREAM;
24         lookup_addr.ai_protocol = IPPROTO_TCP;
25
26         if (getaddrinfo(IP,sock_add, &lookup_addr, &send_addr) != 0)
27         {
28                 perror("getaddrinfo failed");
29                 return 1;
30         }
31
32         int sock = socket(send_addr->ai_family, send_addr->ai_socktype,
33                         send_addr->ai_protocol);
34         if (sock < 0)
35         {
36                 perror("socket failed");
37                 return 1;
38         }
39
40         if (connect(sock, send_addr->ai_addr, send_addr->ai_addrlen) < 0)
41         {
42                 perror("connect failed");
43                 return -1;
44         }
45
46         tx = fdopen(sock, "w");
47         rx = fdopen(dup(sock),"r");
48
49         server_id = 4;
50         fwrite(&server_id,1,4,tx);
51
52         return 0;
53
54 }
55
56 /*
57 void close_up(struct addrinfo *send_addr,FILE *rx,FILE *tx){
58         freeaddrinfo(send_addr);
59         fclose(tx);
60         fclose(rx);
61 }
62 */
63
```

```
64  void close_up(){
65          freeaddrinfo(send_addr);
66          fclose(tx);
67          fclose(rx);
68  }
69  //run with ip and port as arguments
70  //talk to using netcat, e.g.:
71  //nc -l -k -v 127.0.0.1 10689
72  //nc -l -k -v ::1 10689
73  int client_func(char *user, int option)
74  {
75
76          size_t len = sizeof(struct data) + strlen(user)-1;
77          struct data *tosend = malloc(len);
78          tosend->code = 4;
79          tosend ->len = strlen(user);
80          strncpy(&(tosend->msg),user,tosend->len);
81          size_t bytes_sent = fwrite(tosend, 1, len, tx);
82          if (bytes_sent != (len))
83          {
84                  printf("%d\n", bytes_sent);
85                  perror("fwrite 1 failed");
86                  fclose(tx);
87                  return 1;
88          }
89
90          fflush(tx);
91          free(tosend);
92
93          struct data *recv = malloc(sizeof(struct data));
94          size_t bytes_received = fread(recv,1,sizeof(struct data)-1,rx);
95
96          if (bytes_received == 0 && errno == EPIPE )
97          {
98                  errno = 0;
99                  perror("fread 2 failed");
100                 free(recv);
101                 fclose(rx);
102                 fclose(tx);
103                 return 1;
104         }
105
106         if(recv->code !=4){
107                 free(recv);
108                 fclose(rx);
109                 fclose(tx);
110                 return 1;
111         }
112
113         char *temp = malloc(recv->len + 1);
114         bytes_received = fread(temp,1,(recv->len),rx);
115
116         if (bytes_received == 0 && errno == EPIPE )
117         {
118                 errno = 0;
119                 perror("fread 2 failed");
120                 free(recv);
121                 fclose(rx);
122                 fclose(tx);
123                 return 1;
124         }
125         temp[recv->len] = '\0';
126
```

```
127             if(atoi(temp) != 1){
128                     free(recv);
129                     fclose(rx);
130                     fclose(tx);
131                     return 1;
132             }
133             free(temp);
134     //      freeaddrinfo(send_addr);
135     //      fclose(tx);
136     //      fclose(rx);
137
138             return 0;
139     }
```

```
1  //author Deepak Ramadass
2  #include "1_util.h"
3  #include "2_threadserver_handle.h"
4  #include "common.h"
5
6  char *user;
7  size_t length = sizeof(struct data);
8  char msg[MAXLENGTH] = "";
9  char *store = NULL;
10 ssize_t bytes_received = 0;
11 struct email *users_email[4][5] = {{NULL}};
12 int user_num = 0;
13 int close_client = 0;
14
15 /**
16   * This function is used to check if the code is 4
17   * @param code is the code recieved
18   * @return 0/1 success/failure
19   */
20 int code_check(int8_t code){
21         if(code != 4)
22                 return 1;
23         else
24                 return 0;
25 }
26
27 /**
28   * This function is used to free memory
29   */
30 void clean_up(){
31         int i = 0;
32         int j = 0;
33         for(i = 0; i< 4;i++){
34                 for(j = 0; j < 5; j++){
35                         if(users_email[i][j] != NULL)
36                                 free(users_email[i][j]);
37                 }
38         }
39 }
40
41 /**
42   * This function is used to assign a number to a user
43   * @ return num the number assigned
44   */
45 int find_user(char *user){
46         int num = 0;
47
48         if(strcmp(user,"deepak") == 0)
49                 num = 0;
50         else if(strcmp(user,"deepika") == 0)
51                 num = 1;
52         else if(strcmp(user,"pooja") == 0)
53                 num = 2;
54         else if(strcmp(user,"justin") == 0)
55                 num = 3;
56         else
57                 return -1;
58
59         return num;
60 }
61
62 /**
63   * This function checks and stores username
```

```
64    * @param recv points to the recieved data
65    * @param rx is used to read the file name
66    * @return 0/1 success/failure
67    */
68  int check_user(struct data *recv, FILE *rx){
69          if(recv == NULL)
70                  return 1;
71
72          if(recv->code != 3)
73                  return 1;
74
75          //printf("code %d\n",recv->code);
76          int64_t user_len = (recv->len);
77          //printf("len %lld\n",user_len);
78          user = malloc(user_len+1);
79          if(fread(user,1,user_len,rx) != user_len)
80                  return 1;
81          user[user_len] = '\0';
82
83          if((user_num = find_user(user)) < 0)
84                  return 1;
85          printf("User %s is logged in\n",user);
86
87          return 0;
88  }
89
90  /**
91    * This funtion gets the message from the socket
92    * @param rx is used to read data from the control program
93    * @return 0/1 success/failure
94    */
95  int get_message(FILE *rx){
96          struct data *recv = malloc(length - 1);
97          if(fread(recv,1,length-1,rx) != 9)
98                  return 1;
99
100         if((recv->code) != 4)
101                 return 1;
102         store = malloc(recv->len + 1);
103         if(fread(store,1,recv->len,rx) != recv->len)
104                 return 1;
105
106         store[recv->len] = '\0';
107         free(recv);
108         recv = NULL;
109         return 0;
110 }
111
112 /**
113   * This fuction is used to construct the packet to be sent
114   * @param rx is used to read data from the control program
115   * @param tx is used to send data to the control program
116   * @param msg contains the data to be sent
117   * @param tosend points to the response
118   */
119 void send_message(FILE *rx,FILE *tx,char *msg){
120         int len = strlen(msg) + sizeof(struct data);
121         struct data *tosend = malloc(len);
122         tosend->code = 4;
123         tosend->len = strlen(msg);
124         strncpy(&(tosend->msg),msg,tosend->len);
125         //printf("msg %s\n",&(tosend->msg));
126
```

```
127         send_tcp_data(rx,tx,tosend,len-1);
128         free(tosend);
129         tosend = NULL;
130 }
131 /**
132   * This function check if the request is for inbox
133   * and display the inbox
134   * @param rx is used to read data from the control program
135   * @param user holds the user identity
136   * @return 0/1 success/failure
137   */
138 int check_inbox(FILE *rx,FILE *tx){
139         int i = 0;
140
141         if(users_email[user_num][i] == NULL){
142                 sprintf(msg,"No Mails\n");
143                 send_message(rx,tx,msg);
144                 return 1;
145         }
146
147         while((i < 5) && (users_email[user_num][i] != NULL)){
148
149                 send_message(rx,tx,"\n\0");
150                 sprintf(msg,"No: %d\n",i+1);
151                 send_message(rx,tx,msg);
152
153         /*      sprintf(msg,"To");
154                 send_message(rx,tx,msg);
155                 send_message(rx,tx,users_email[user_num][i]->to);
156
157                 sprintf(msg,"From");
158                 send_message(rx,tx,msg);
159                 send_message(rx,tx,users_email[user_num][i]->from);
160         */
161                 sprintf(msg,"Subject:");
162                 send_message(rx,tx,msg);
163                 send_message(rx,tx,users_email[user_num][i]->subject);
164                 send_message(rx,tx,"\n\n\0");
165         /*      sprintf(msg,"Message");
166                 send_message(rx,tx,msg);
167                 send_message(rx,tx,users_email[user_num][i]->msg);
168         */
169                 i++;
170         }
171         sprintf(msg,"Choose one of the following:\nOpen  Delete  Quit\n");
172         send_message(rx,tx,msg);
173         send_message(rx,tx,"1");
174          return 0;
175 }
176
177 /**
178   * This function is used to write mails
179   * @param rx is used to read data from the control program
180   * @param tx is used to send data to the control program
181   * @param recv points to the recieved data
182   * @return 0/1 success/failure
183   */
184 int compose_mail(FILE *rx, FILE *tx){
185
186 here:   sprintf(msg,"Send To:");
187         send_message(rx,tx,msg);
188         send_message(rx,tx,"1");
189
```

```
190         if(get_message(rx) != 0)
191                 return 1;
192
193         int send_user = find_user(store);
194         close_client = initialize();
195         close_client = client_func(store,1);
196
197         if(close_client == 1){
198                 sprintf(msg,"User Unknown\n");
199                 send_message(rx,tx,msg);
200                 goto here;
201         }
202
203         int i = 0;
204         for(i = 0; i < 5; i++){
205                 if(users_email[send_user][i] == NULL)
206                         break;
207         }
208
209         if(i == 5){
210                 free(users_email[send_user][0]);
211                 i = 0;
212         }
213
214
215         users_email[send_user][i] = malloc(sizeof(struct email));
216         strncpy(users_email[send_user][i]->to,store,strlen(store));
217
218         strncpy(users_email[send_user][i]->from,user,strlen(user));
219
220         if(store != NULL);
221                 free(store);
222                 store = NULL;
223
224         sprintf(msg,"Subject:");
225         send_message(rx,tx,msg);
226         send_message(rx,tx,"1");
227
228         if(get_message(rx) != 0)
229                 return 1;
230
231         strncpy(users_email[send_user][i]->subject,store,strlen(store));
232         if(store != NULL);
233                 free(store);
234                 store = NULL;
235
236         sprintf(msg,"Message:");
237         send_message(rx,tx,msg);
238         send_message(rx,tx,"1");
239
240         if(get_message(rx) != 0)
241                 return 1;
242
243         strncpy(users_email[send_user][i]->msg,store,strlen(store));
244         if(store != NULL);
245                 free(store);
246                 store = NULL;
247
248         return 0;
249 }
250
251 /**
252   * This funstion displays an email
```

```
253    * @param rx is used to read data from the control program
254    * @param tx is used to send data to the control program
255    * @param i refers the email number
256    */
257  void show_mail(FILE *rx, FILE*tx, int i){
258          sprintf(msg,"From: ");
259          send_message(rx,tx,msg);
260          send_message(rx,tx,users_email[user_num][i]->from);
261          send_message(rx,tx,"\n\0");
262
263          sprintf(msg,"Subject: ");
264          send_message(rx,tx,msg);
265          send_message(rx,tx,users_email[user_num][i]->subject);
266          send_message(rx,tx,"\n\0");
267
268          sprintf(msg,"Message: ");
269          send_message(rx,tx,msg);
270          send_message(rx,tx,users_email[user_num][i]->msg);
271          send_message(rx,tx,"\n\0");
272
273  }
274
275  /**
276    * This function check if the request is for inbox
277    * and display the inbox
278    * @param rx is used to read data from the control program
279    * @param tx is used to send data to the control program
280    * @param recv points to the recieved data
281    * @return 0/1 success/failure
282    */
283  int manage_inbox(FILE *rx, FILE *tx){
284          char *token = strtok(store," ");
285          char *temp =  strtok(NULL," ");
286          int i = 0;
287
288          if(temp != NULL)
289                  i = atoi(temp);
290
291          if(strncmp(token,"Open",strlen("Open")) == 0){
292                  if((users_email[user_num][i-1] != NULL))
293                          show_mail(rx,tx,i-1);
294                  else
295                          return 1;
296
297                  check_inbox (rx,tx);
298                  return 0;
299          }
300
301          else if(strncmp(token,"Delete",strlen("Delete")) == 0){
302                  if((users_email[user_num][i-1] != NULL)){
303                          free(users_email[user_num][i-1]);
304                          users_email[user_num][i-1] = NULL;
305                  }
306                  check_inbox (rx,tx);
307                  return 0;
308          }
309
310          else if(strncmp(token,"Quit",strlen("Quit")) == 0){
311                  return 2;
312          }
313
314          else{
315                  sprintf(msg,"Invalid Entry\n");
```

```
316                     send_message(rx,tx,msg);
317                     check_inbox (rx,tx);
318                     return 2;
319             }
320
321             return 0;
322
323     }
324
325     /**
326       * This function is used to handle all email operations
327       * @param recv points to the recieved data
328       * @param tosend points to the response
329       * @param rx is used to read data from the control program
330       * @param tx is used to send data to the control program
331       * @return 0/1 success/failure
332       */
333     int manage_mail(FILE *rx, FILE *tx, struct data *recv,int step){
334
335             if(step == 1){
336                     sprintf(msg,"Choose one of the following:\nInbox\tCompose\n");
337                     send_message(rx,tx,msg);
338                     step++;
339             }
340
341             if(code_check(recv->code) != 0)
342                     return -1;
343
344             if(step == 2){
345
346                     int64_t len = recv->len;
347                     store = malloc(len+1);
348
349                     if(fread(store,1,len,rx) != len)
350                             return -1;
351                     store[len] = '\0';
352                     if(strcmp(store,"Compose") == 0){
353                             compose_mail(rx,tx);
354                             step--;
355                     }
356
357                     else if(strcmp(store,"Inbox") == 0){
358
359                             if(check_inbox (rx,tx) != 0){
360
361                                     step--;
362                             }
363                             else
364                                     step++;
365                     }
366
367                     else if(strcmp(store,"Quit") == 0)
368                             return -1;
369
370                     else{
371                             sprintf(msg,"Invalid Entry\n");
372                             send_message(rx,tx,msg);
373                             send_message(rx,tx,"1");
374                     }
375
376                     free(store);
377                     store = NULL;
378                     return step;
```

```
379         }
380
381         if(step == 3){
382
383                 int64_t len = recv->len;
384                 store = malloc(len+1);
385
386                 if(fread(store,1,len,rx) != len)
387                         return -1;
388                 //store[len] = '\0';
389
390                 int catch = manage_inbox(rx,tx);
391
392                 if(catch == 1)
393                         return -1;
394
395                 else if(catch == 2){
396                         step--;
397                         sprintf(msg,"Choose one of the following:\nInbox\tCompose\n");
398                         send_message(rx,tx,msg);
399                         send_message(rx,tx,"1");
400                 }
401
402                 free(store);
403                 store = NULL;
404         }
405         //step = 0;
406         return step;
407 }
408
409 //This function processes the data received by a single client
410 void *handle_tcp_client(void *arg)
411 {
412         int client = *((int *) arg);
413         free(arg);
414
415         int step = 0;
416         FILE *rx = fdopen(client, "r");
417         FILE *tx = fdopen(dup(client), "w");
418         struct data *recv = calloc(1, sizeof(struct data));
419         //struct data *tosend = calloc(1,length);
420         //struct data *recv = malloc(length);
421         bytes_received = fread(recv,1,length-1,rx);
422
423         if (bytes_received < 0 && errno != EINTR)
424         {
425                 goto close;
426         }
427
428
429
430         while (bytes_received >= 0 || errno == EINTR)
431         {
432                 if(bytes_received == 0 && errno == EINTR){
433                         errno = 0;
434                         goto close;
435                 }
436
437                 if(step == 0){
438                         if(check_user(recv,rx) != 0)
439                                 goto close;
440                         else{
441                                 step++;
```

```
442                                     goto label1;
443                             }
444                     }
445
446                     sleep(2);
447
448                     if((step = manage_mail(rx,tx,recv,step)) < 0){
449                             //send_message(rx,tx,"quit");
450                             bytes_received = -1;
451                             goto close;
452                     }
453                     //send_tcp_data(rx, tx, buffer, bytes_received);
454
455             label1: if(step == 1){
456                             sprintf(msg,"Choose one of the following:\nInbox\tCompose\n");
457                             send_message(rx,tx,msg);
458                             send_message(rx,tx,"1");
459                             step++;
460                     }
461
462                     bytes_received = fread(recv,1,length-1,rx);
463             }
464
465     close:
466             if (bytes_received <= 0)
467             {
468                     printf("client closed the connection!\n");
469             }
470             if (recv != NULL && bytes_received == 0)
471             {
472                     free(recv);
473             }
474
475             if (user != NULL && bytes_received == 0){
476                     free(user);
477                     user = NULL;
478             }
479
480             if(close_client >= 0)
481                     close_up();
482
483             clean_up();
484             fclose(tx);
485             fclose(rx);
486             children--;
487             printf("now there are %d children\n", children);
488
489             return NULL;
490     }
```