```
// TODO add current sending pointer

// TODO add buffer for datagram sizes
// TODO add file reading before wait
// TODO retransmission and dropped packets
// TODO RTT
// TODO implement slow start -> until a datagram has been lost, window *2 each
// time; when a datagram is lost, divide window by 2, and then linear phase
// TODO implement connect with UDP

#include "serveur1-PerformancesRadicalementSuperieures.h"

int desc, data_desc;
FILE *file;
struct sockaddr_in adresse;

pthread_mutex_t ack_mutex;
pthread_cond_t ack_cond;
int file_sent = FALSE;
int last_ack = 0;
time_t snd_time[BUFFER_SIZE];

socklen_t addr_len;
int data_desc_open = FALSE;
int file_open = FALSE;
int desc_open = FALSE;

void end_handler() {
#if DEBUG
printf("Entering end handler\n");
#endif /* if DEBUG */

send_disconnect_message(data_desc, adresse);

if (data_desc_open) close(data_desc);

if (desc_open) close(desc);

if (file_open) fclose(file);
exit(EXIT_SUCCESS);
}

void alrm_handler() {}
```

void* send_thread(void *args) {
ADDRESS *client_address = args;
int data_desc = client_address->desc;
struct sockaddr_in adresse = client_address->addr;

char segment_buffer[BUFFER_SIZE][RCVSIZE];
int bytes_read_buffer[BUFFER_SIZE];
int sequence_nb = 1;
int snd;
int i = 0;
int datagram_size;
int p_buff;
int retransmission = FALSE;

do {
i = 0;

```
do {
  p_buff = sequence_nb % BUFFER_SIZE;

  if (!retransmission) {
    memset(segment_buffer[p_buff], '\0', RCVSIZE);
    sprintf(segment_buffer[p_buff], "%06d", sequence_nb);
    bytes_read_buffer[p_buff] = fread(segment_buffer[p_buff] + HEADER_SIZE,
                                      1,
                                      DATA_SIZE,
                                      file);
  }

  pthread_mutex_lock(&ack_mutex);

  if (last_ack != sequence_nb - 1) pthread_cond_wait(&ack_cond, &ack_mutex);
  retransmission = last_ack != (sequence_nb - 1);

  if (retransmission) {
    sequence_nb = last_ack + 1;
    p_buff      = sequence_nb % BUFFER_SIZE;
  }
  file_sent = bytes_read_buffer[p_buff] < DATA_SIZE;
  pthread_mutex_unlock(&ack_mutex);


  datagram_size = bytes_read_buffer[p_buff] + (HEADER_SIZE * sizeof(char));

  if (bytes_read_buffer[p_buff] == -1) perror("Error reading file\n");
  else if (bytes_read_buffer[p_buff] > 0) {
    snd = sendto(data_desc,
                 segment_buffer[p_buff],
```

```
                    datagram_size,
                    0,
                    (struct sockaddr *)&adresse,
                    sizeof(adresse));

      pthread_mutex_lock(&ack_mutex);
      snd_time[p_buff] = time(0);
      pthread_mutex_unlock(&ack_mutex);

      if (snd < 0) {
        perror("Error sending segment\n");
        pthread_exit(NULL);
      }
      #if DEBUG
      printf("Sent segment %06d\n", sequence_nb);
      #endif /* if DEBUG */
      sequence_nb++;
    }
    i++;
  } while (i < WINDOW && bytes_read_buffer[p_buff] != 0);
```

} w hile (bytes_read_buffer[p_buff] != 0);
pthread_exit(NULL);
}

void* ack_thread(void *args) {
ADDRESS *client_address = args;
int data_desc = client_address->desc;

char buffer[ACK_SIZE + 1];
struct sockaddr_in src_addr;
int rcv;
int end;
int parsed_ack = 0;
long rto = 50000;
long srtt = 50000;
long rtt;
long rttvar = 0;

do {
memset(buffer, '\0', ACK_SIZE + 1);
#if DEBUG
printf("Waiting ACK %d\n", parsed_ack + 1);
#endif /* if DEBUG */
rcv = recvfrom(data_desc,
buffer,

ACK_SIZE,
0,
(struct sockaddr *)&src_addr,
&addr_len);

```
if (rcv < 0) {
  if (errno == EWOULDBLOCK) {
    perror("Blocked");
    pthread_mutex_lock(&ack_mutex);
    pthread_cond_signal(&ack_cond);
    pthread_mutex_unlock(&ack_mutex);
  }
  else {
    perror("Error receiving ACK\n");
    pthread_exit(NULL);
  }
}
else {
  parsed_ack = atoi(buffer + 3);
  pthread_mutex_lock(&ack_mutex);
  last_ack = parsed_ack;
  end      = file_sent;
  pthread_cond_signal(&ack_cond);
  pthread_mutex_unlock(&ack_mutex);
  set_timeout(data_desc, 1, 0);
  #if DEBUG
  printf("Received ACK %d\n", parsed_ack);
  #endif /* if DEBUG */
}
```

} w hile (!end);
pthread_exit(NULL);
}

int main(int argc, char const *argv[]) {
signal(SIGTSTP, end_handler);

struct sockaddr_in src_addr;
socklen_t addr_len = sizeof(src_addr);
char buffer[RCVSIZE] = { 0 };

int port;

if (argc == 2) port = atoi(argv[1]);
else port = 4242;
desc = create_socket(port);

```
desc_open = TRUE;

memset(&src_addr, 0, addr_len);
data_desc = my_accept(desc, &src_addr);
data_desc_open = TRUE;

#if DEBUG
printf("Waiting for file name\n");
#endif /* if DEBUG */

if (recvfrom(data_desc, buffer, sizeof(buffer), 0, (struct sockaddr *)&src_addr,
&addr_len) == -1) {
perror("Error receiving file name\n");
end_handler();
}
adresse = src_addr;

file = fopen(buffer, "r");

if (file == NULL) {
perror("Error opening file\n");
end_handler();
}
file_open = TRUE;

addr_len = sizeof(adresse);

pthread_t snd;
pthread_t ack;
pthread_mutex_init(&ack_mutex, NULL);
pthread_cond_init(&ack_cond, NULL);

ADDRESS addr;
addr.addr = adresse;
addr.desc = data_desc;

if (pthread_create(&snd, NULL, send_thread, (void *)&addr) != 0) {
perror("Error creating send thread");
exit(EXIT_FAILURE);
}

if (pthread_create(&ack, NULL, ack_thread, (void *)&addr) != 0) {
perror("Error creating ack thread");
```

```
exit(EXIT_FAILURE);
}

pthread_join(snd, NULL);
pthread_join(ack, NULL);
#if DEBUG
printf("Joining threads\n");
#endif /* if DEBUG */

end_handler();
return 0;
}
```