PROGRAM CODE

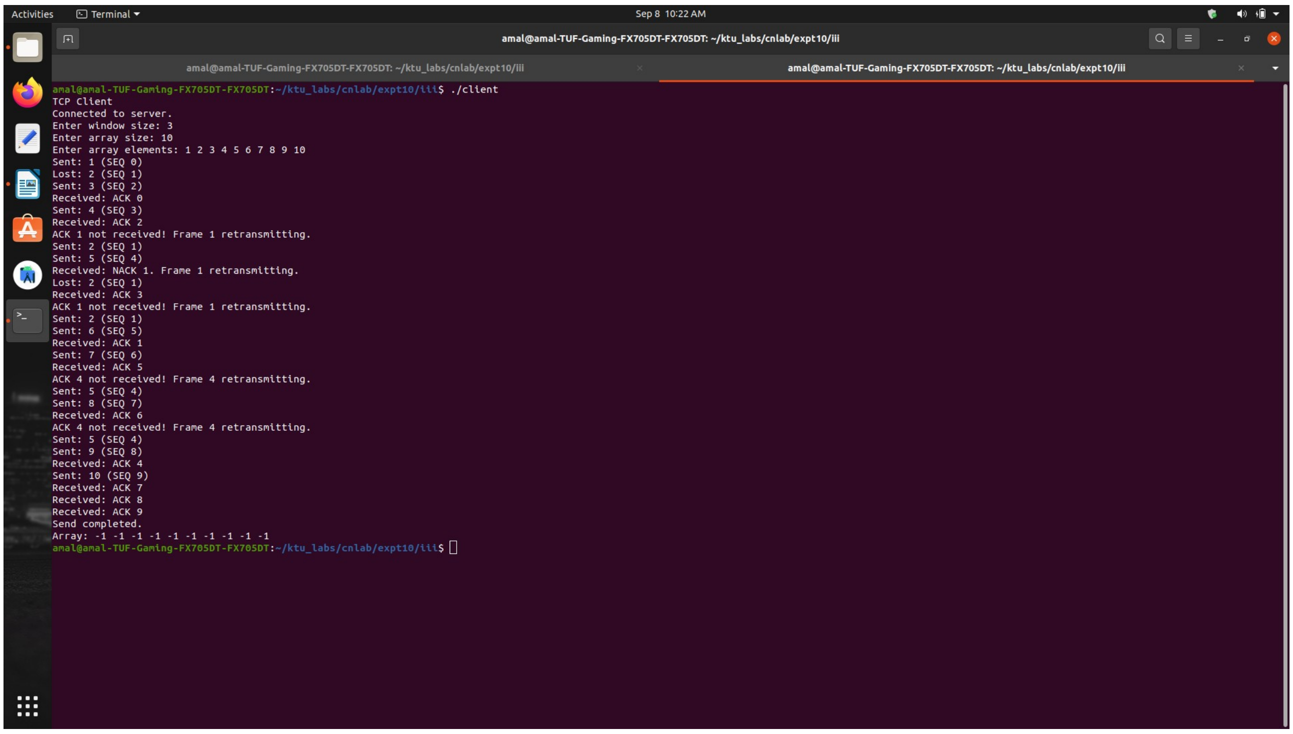
server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h>

#define PORT 8000
#define SIZE 100

typedef struct packet {
      int data;
      int type; // SEQ (0), ACK (1) or NACK(-1)
      int seq; // Sequence number
} packet;

int add(int* arr, int key, int index) {
      int flag = -1;

      for(int i = 0; i < index; i++) {
            if(arr[i] == -1) {
                  flag = i;
                  break;
            }
      }

      arr[index] = key;

      return flag;
}

void main() {
      int server_fd, client_fd;
      struct sockaddr_in address;
      int addrlen = sizeof(address);

      printf("Selective Repeat ARQ\nTCP Server\n");

      if((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
            printf("Socket creation failed!\n");
            exit(1);
      }

      address.sin_family = AF_INET;
      address.sin_addr.s_addr = INADDR_ANY;
      address.sin_port = htons(PORT);

      if(bind(server_fd, (struct sockaddr*) &address, addrlen) < 0) {
            printf("Socket binding failed!\n");
            exit(1);
      }

      if(listen(server_fd, 5) < 0) {
            printf("Listening failed!\n");
            exit(1);
      }
      if((client_fd = accept(server_fd, (struct sockaddr*) &address,
(socklen_t*) &addrlen)) < 0) {
            printf("Connection failed!\n");
            exit(1);
      } else {
            printf("Connected to client.\n");
      }

      packet p;

      int* arr = malloc(SIZE * sizeof(int));

      for(int i = 0; i < SIZE; i++)
            arr[i] = -1;

      while(1) {
            int status = recv(client_fd, &p, sizeof(packet), 0);

            if(status < 0) {
                  printf("Receive failed!\n");
            } else if (status == 0) {
                  printf("Receive completed.\nArray: ");

                  for(int i = 0; arr[i] != -1; i++) {
                        printf("%d ", arr[i]);
                  }

                  printf("\n");

                  break;
            } else {
                  printf("Received: %d (SEQ %d)\n", p.data, p.seq);

                  int index = add(arr, p.data, p.seq);

                  if(index != -1) {
                        int temp = p.seq;

                        p.type = -1;

                        p.seq = index;

                        if(rand() % 10 != 6) {
                              if(send(client_fd, &p, sizeof(packet), 0) < 0) {
                                    printf("Send failed!\n");
                              } else {
                                    printf("Sent: NACK %d\n", p.seq);
                              }
                        } else {
                              printf("Lost: NACK %d\n", p.seq);
                        }

                        p.seq = temp;
                  }

                  p.type = 1;

                  if(rand() % 10 != 6) {
                        if(send(client_fd, &p, sizeof(packet), 0) < 0) {
                              printf("Send failed!\n");
                        } else {
                              printf("Sent: ACK %d\n", p.seq);
                        }
                  } else {
                        printf("Lost: ACK %d\n", p.seq);
                  }
            }
      }

      close(server_fd);
      close(client_fd);
}
```

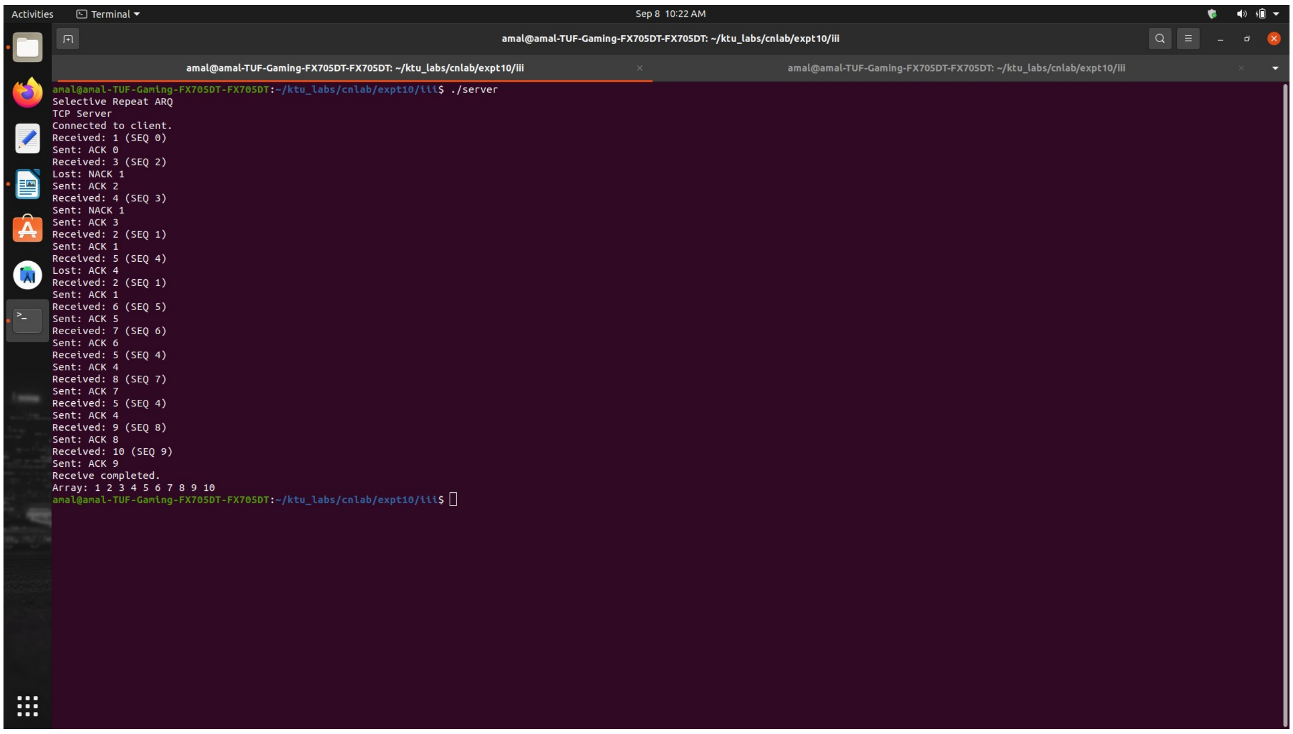
client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 8000

int count = 0;

typedef struct packet {
      int data;
      int type; // SEQ (0), ACK (1) or NACK(-1)
      int seq; // Sequence number
} packet;

typedef struct window {
      int size;
      int start;
      int end;
} window;

typedef struct data {
      int* arr;
      int n;
      int client_fd;
      packet* p;
      window* w;
} data;

int ackFrame(int* arr, int index) {
      int flag = -1;

      for(int i = 0; i < index; i++) {
            if(arr[i] != -1) {
                  flag = i;
                  break;
            }
      }

      arr[index] = -1;

      return flag;
}

void sendWindow(data d) {
      for(d.p->seq = d.w->start; d.p->seq <= d.w->end && d.p->seq < d.n; d.p-
>seq++) {
            d.p->type = 0;
            d.p->data = d.arr[d.p->seq];

            if(d.p->data == -1)
                  continue;
            if(rand() % 10 != 6) {
                  if(send(d.client_fd, d.p, sizeof(packet), 0) < 0) {
                        printf("Send failed!\n");
                  } else {
                        printf("Sent: %d (SEQ %d)\n", d.p->data, d.p->seq);
                  }
            } else {
                  printf("Lost: %d (SEQ %d)\n", d.p->data, d.p->seq);
            }
      }
}

void sendFrame(data d, int seq) {
      d.p->type = 0;
      int temp;

      if(seq == -1)
            d.p->data = d.arr[d.w->end];
      else {
            d.p->data = d.arr[seq];
            temp = d.p->seq;
            d.p->seq = seq;
      }

      if(d.p->data == -1)
            return;

      if(rand() % 10 != 6) {
            if(send(d.client_fd, d.p, sizeof(packet), 0) < 0) {
                  printf("Send failed!\n");
            } else {
                  printf("Sent: %d (SEQ %d)\n", d.p->data, d.p->seq);
            }
      } else {
            printf("Lost: %d (SEQ %d)\n", d.p->data, d.p->seq);
      }

      if(seq == -1)
            d.p->seq = d.p->seq + 1;
      else
            d.p->seq = temp;
}

void recvAck(data d) {
      data d1;
      packet p;
      d1.p = &p;

      if(recv(d.client_fd, d1.p, sizeof(packet), 0) < 0) {
            printf("Time out! Window retransmitting.\n");
            sendWindow(d);
            recvAck(d);
      } else {
            if(d1.p->type == 1) {
                  if(d.arr[d1.p->seq] == -1) {
                        recvAck(d);
                  } else {
                        printf("Received: ACK %d\n", d1.p->seq);

                        count++;

                        d.w->start++;
                        d.w->end++;
                        int index = ackFrame(d.arr, d1.p->seq);

                        if(index != -1) {
                              printf("ACK %d not received! Frame %d
retransmitting.\n", index, index);

                              sendFrame(d, index);
                        }

                        if(count == d.n) {
                              printf("Send completed.\nArray: ");

                              for(int i = 0; i < d.n; i++) {
                                    printf("%d ", d.arr[i]);
                              }

                              printf("\n");

                              close(d.client_fd);
                              exit(0);
                        }

                        if(d.w->end < d.n) {
                              sendFrame(d, -1);

                              recvAck(d);
                        }
                        else
                              recvAck(d);
                  }
            } else if(d1.p->type == -1) {
                  printf("Received: NACK %d. Frame %d retransmitting.\n", d1.p-
>seq, d1.p->seq);

                  sendFrame(d, d1.p->seq);

                  recvAck(d);
            }
      }
}

void main() {
      int client_fd;
      struct sockaddr_in serv_addr;

      printf("TCP Client\n");

      client_fd = socket(AF_INET, SOCK_STREAM, 0);

      if(client_fd < 0) {
            printf("Socket creation failed!\n");
            exit(1);
      }

      serv_addr.sin_family = AF_INET;
      serv_addr.sin_addr.s_addr = INADDR_ANY;
      serv_addr.sin_port = htons(PORT);

      if(connect(client_fd, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) <
0) {
            printf("Connection failed!\n");
            exit(1);
      } else {
            printf("Connected to server.\n");
      }

      struct timeval tv;
      tv.tv_sec = 1;
      tv.tv_usec = 0;
      setsockopt(client_fd, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof
tv);

      int n;
      window w;

      printf("Enter window size: ");
      scanf("%d", &w.size);

      w.start = 0;
      w.end = w.size - 1;

      printf("Enter array size: ");
      scanf("%d", &n);

      int arr[n];

      printf("Enter array elements: ");
      for(int i = 0; i < n; i++) {
            scanf("%d", &arr[i]);
      }

      packet p;
      data d;
      d.client_fd = client_fd;
      d.p = &p;
      d.w = &w;
      d.n = n;
      d.arr = arr;
      p.seq = 0;

      sendWindow(d);
      recvAck(d);
}
```

OUTPUT



```
amal@amal-TUF-Gaming-FX705DT-FX705DT:~/ktu_labs/cnlab/expt10/iii$ ./client
TCP Client
Connected to server.
Enter window size: 3
Enter array size: 10
Enter array elements: 1 2 3 4 5 6 7 8 9 10
Sent: 1 (SEQ 0)
Lost: 2 (SEQ 1)
Sent: 3 (SEQ 2)
Received: ACK 0
Sent: 4 (SEQ 3)
Received: ACK 2
ACK 1 not received! Frame 1 retransmitting.
Sent: 2 (SEQ 1)
Sent: 5 (SEQ 4)
Received: NACK 1. Frame 1 retransmitting.
Lost: 2 (SEQ 1)
Received: ACK 3
ACK 1 not received! Frame 1 retransmitting.
Sent: 2 (SEQ 1)
Sent: 6 (SEQ 5)
Received: ACK 1
Sent: 7 (SEQ 6)
Received: ACK 5
ACK 4 not received! Frame 4 retransmitting.
Sent: 5 (SEQ 4)
Sent: 8 (SEQ 7)
Received: ACK 6
ACK 4 not received! Frame 4 retransmitting.
Sent: 5 (SEQ 4)
Sent: 9 (SEQ 8)
Received: ACK 4
Sent: 10 (SEQ 9)
Received: ACK 7
Received: ACK 8
Received: ACK 9
Send completed.
Array: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
amal@amal-TUF-Gaming-FX705DT-FX705DT:~/ktu_labs/cnlab/expt10/iii$
```



```
amal@amal-TUF-Gaming-FX705DT-FX705DT:~/ktu_labs/cnlab/expt10/iii$ ./server
Selective Repeat ARQ
TCP Server
Connected to client.
Received: 1 (SEQ 0)
Sent: ACK 0
Received: 3 (SEQ 2)
Lost: NACK 1
Sent: ACK 2
Received: 4 (SEQ 3)
Sent: NACK 1
Sent: ACK 3
Received: 2 (SEQ 1)
Sent: ACK 1
Received: 5 (SEQ 4)
Lost: ACK 4
Received: 2 (SEQ 1)
Sent: ACK 1
Received: 6 (SEQ 5)
Sent: ACK 5
Received: 7 (SEQ 6)
Sent: ACK 6
Received: 5 (SEQ 4)
Sent: ACK 4
Received: 8 (SEQ 7)
Sent: ACK 7
Received: 5 (SEQ 4)
Sent: ACK 4
Received: 9 (SEQ 8)
Sent: ACK 8
Received: 10 (SEQ 9)
Sent: ACK 9
Receive completed.
Array: 1 2 3 4 5 6 7 8 9 10
amal@amal-TUF-Gaming-FX705DT-FX705DT:~/ktu_labs/cnlab/expt10/iii$
```