==========

TW1-Message queue

Sender:

```
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>

#define MAX 50
struct msg_buffer {
  long mesg_type;
  char mesg_text[100];
}
message;
int main() {
  key_t key;
  int msgid;
  key = ftok("progfile", 65);
  msgid = msgget(key, 0666 | IPC_CREAT);
  message.mesg_type = 1;
  printf("Write data: \n");
  fgets(message.mesg_text, MAX, stdin);
  msgsnd(msgid, & message, sizeof(message), 0);
  printf("Data sent is : %s\n", message.mesg_text);
  return 0;
}
```

Receiver:

```
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>

#define MAX 50
struct msg_buffer {
  long mesg_type;
  char mesg_text[100];
}
message;
int main() {
  key_t key;
  int msgid;
  key = ftok("progfile", 65);
  msgid = msgget(key, 0666 | IPC_CREAT);
  msgrcv(msgid, & message, sizeof(message), 1, 0);
  printf("Data read is: %s\n", message.mesg_text);
  msgctl(msgid, IPC_RMID, NULL);
  return 0;
}
```

==========

TW1-Pipe

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
```

```
#include <sys/wait.h>

int main() {
  int fd[2], n;
  char buffer[100];
  pid_t p;
  pipe(fd);
  p = fork();
  if (p > 0) {
    printf("Parent process pid: %d\n", getppid());
    printf("Child process pid: %d\n", p);
    printf("Passing value child\n");
    write(fd[1], "Hello World!\n", 13);
  } else {
    printf("Child process pid: %d\n", getpid());
    printf("Parent process pid: %d\n", getppid());
    n = read(fd[0], buffer, 100);
    printf("Data received by child process: \n");
    write(1, buffer, n);
  }
  return 0;
}
```

==========

TW2 and TW5

Server:

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

```
#include <strings.h>
#include <string.h>
#define PORT 4444

int main() {
    int listenfd, connfd;
    struct sockaddr_in servAddr, cliAddr;
    socklen_t clilen;
    char buffer[1024];

    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    printf("[+] Server socket created successfully\n");

    bzero(&servAddr, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_port = htons(PORT);
    servAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    bind(listenfd, (struct sockaddr *) &servAddr, sizeof(servAddr));
    printf("[+] Bind to PORT %d successful\n", PORT);

    listen(listenfd, 5);
    printf("[+] Listening...\n");

    connfd = accept(listenfd, (struct sockaddr *) &cliAddr, &clilen);

    strcpy(buffer, "Hello World!");
    send(connfd, buffer, strlen(buffer), 0);
    printf("[+] Data sent to client: %s\n", buffer);
```

```
    printf("[+] Closing the connection\n");
    return 0;
}
```

Client:

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define PORT 4444

int main() {
    int sockfd;
    struct sockaddr_in servAddr;
    char buffer[1024];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    printf("[+] Client socket created successfully\n");

    bzero(&servAddr, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_port = htons(PORT);
    servAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    connect(sockfd, (struct sockaddr *) &servAddr, sizeof(servAddr));
    printf("[+] Connected to server\n");
```

```
    recv(sockfd, buffer, 1024, 0);
    printf("[+] Data received from server: %s\n", buffer);
    printf("[+] Closing the connection\n");
    return 0;
}
```

==========

TW3

```
#include <stdio.h>
#define NODES 10
#define NO_ROUTE 999
#define NO_HOP 1000
int no;
struct node
{
  int a[NODES][4];
} router[NODES];
void
init (int r)
{
  int i;
  for (i = 1; i <= no; i++)
    {
      router[r].a[i][1] = i;
      router[r].a[i][2] = NO_ROUTE;
      router[r].a[i][3] = NO_HOP;
    }
  router[r].a[r][2] = 0;
  router[r].a[r][3] = r;
}
```

```
void
inp (int r)
{
  int i;
  printf ("\nEnter distance from node %d to other nodes\n", r);
  printf ("Enter 999 if there is no direct route\n");
  for (i = 1; i <= no; i++)
    {
      if (i != r)
    {
      printf ("Enter distance to node %d: ", i);
      scanf ("%d", &router[r].a[i][2]);
      router[r].a[i][3] = i;
    }
    }
}

void
display (int r)
{
  int i;
  printf ("\nThe routing table for node %d is as follows", r);
  for (i = 1; i <= no; i++)
    {
      if (router[r].a[i][2] == 999)
    printf ("\n%d \t no link \t no hop", router[r].a[i][1]);
      else
    printf ("\n%d \t %d \t %d", router[r].a[i][1],
        router[r].a[i][2], router[r].a[i][3]);
    }
```

```
}

void
dv_algo (int r)
{
  int i, j, z;
  for (i = 1; i <= no; i++)
    {
// r b�� source router
// i b�� step taken (via which router to reach the dest router)
// j b�� destination router
// cannot jump from the source router or to a router which is not
       reachable or from the source router
    if (router[r].a[i][2] != 999 && router[r].a[i][2] != 0)
    {
      for (j = 1; j <= no; j++)
        {
          z = router[r].a[i][2] + router[i].a[j][2];
          if (z < router[r].a[j][2])
        {
          router[r].a[j][2] = z;
          router[r].a[j][3] = i;
        }
        }
    }
    }
}

int
main ()
```

```
{
  int i, j, x, y;
  char choice = 'y';
  printf ("Enter the number of nodes: ");
  scanf ("%d", &no);
  for (i = 1; i <= no; i++)
    {
      init (i);
      inp (i);
    }
  printf ("\nThe routing tables of nodes after initialization is as follows");
  for (i = 1; i <= no; i++)
    display (i);
  printf ("\n\nComputing shortest paths...\n");
  for (i = 1; i <= no; i++)
    dv_algo (i);
  printf ("\nThe routing tables of nodes after computation of shortest paths is as follows");
  for (i = 1; i <= no; i++)
    display (i);
  printf ("\n");
  while (choice != 'n')
    {
      printf ("\nEnter the nodes between which shortest distance is to be found: ");
      scanf ("%d %d", &x, &y);
      getchar ();
      printf ("The length of the shortest path between nodes %d and %d is %d\n", x, y,
router[x].a[y][2]);
      printf ("Continue? (y/n): ");
      scanf ("%c", &choice);
    }
  return 0;
```

}

==========

TW4

Server:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT     8080
#define MAXLINE  1024

// Sender code
int main(void) {
    int sockfd,len,n;
    char buffer[MAXLINE], msg[MAXLINE]="exit";
    struct sockaddr_in servaddr, cliaddr;

    // Creating socket file descriptor
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    printf("[+]Server Socket is created.\n");

    memset(&servaddr, 0, sizeof(servaddr));
```

```
    memset(&cliaddr, 0, sizeof(cliaddr));

    // Filling server information
    servaddr.sin_family = AF_INET; // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);

    // Bind the socket with the server address
    if (bind(sockfd, (const struct sockaddr *)&servaddr,
            sizeof(servaddr)) < 0 ) {
       perror("bind failed");
       exit(EXIT_FAILURE);
    }
    printf("[+]Bind to port %d\n", 8080);

    len = sizeof(cliaddr); //len is value/result
    while(1) {
            recvfrom(sockfd, (char *)buffer, MAXLINE,
              MSG_WAITALL, (struct sockaddr *) &cliaddr, &len);
            printf("Client : %s\n", buffer);
            printf("Enter the message to reply to client:");
            fgets(buffer, sizeof(buffer),stdin);
            n = strlen(buffer);
            buffer[n-1]='\0';
            if (strcmp(buffer, msg)==0) {
                  printf("Disconnected from %s:%d\n", inet_ntoa(cliaddr.sin_addr),
ntohs(cliaddr.sin_port));
                  break;
                } else {
                 sendto(sockfd, buffer, sizeof(buffer),
                   MSG_CONFIRM, (const struct sockaddr *) &cliaddr, len);
```

```
            printf("Message sent.\n");
             }
        }
        close(sockfd);
    return 0;
}
```

Client:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT     8080
#define MAXLINE  1024

// Reciever code
int main(void) {
    int sockfd, len, n;
    char buffer[MAXLINE],msg[MAXLINE]="exit";
    struct sockaddr_in servaddr;

    // Creating socket file descriptor
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
```

```
    printf("[+]Client Socket is created.\n");

    memset(&servaddr, 0, sizeof(servaddr));

    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    while(1) {
        printf("Enter message to send to server:");
        fgets(buffer, sizeof(buffer), stdin);        //gets(buffer);
        n = strlen(buffer);
        buffer[n-1]='\0';
        sendto(sockfd, buffer, sizeof(buffer), MSG_CONFIRM,
            (const struct sockaddr *) &servaddr, sizeof(servaddr));
        printf("Message sent.\n");
        if (strcmp(buffer, msg) == 0) {
            close(sockfd);
          printf("[-]Closing server.\n");
          exit(1);
        } else
      recvfrom(sockfd, (char *)buffer, MAXLINE,
            MSG_WAITALL, (struct sockaddr *) &servaddr, &len);
      printf("Server:%s\n",buffer);
    }
    close(sockfd);
    return 0;
}
```

==========

TW6-TW10

Term Work 6: Steps: You need to use ubuntu ns3_working (6-8).
On desktop find a directory named ns-allinone-3.28.
Go to ns-3.28,examples,tutorial,first.cc(copy this file)
Go to ns-3.28,scratch (and paste first.cc here).
Now open first.cc and add this code.
source code:
(hash)include "ns3/netanim-module.h"

AnimationInterface anim("fist, xml"); // as per your name.
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll(ascii.CreateFileStream("first.tr")); // for 7 and 8 use second.tr and third.tr.
pointToPoint.EnablePcapAll("first"); // 7 and 8 use second and third.

Go to ns-3.28 directory and right click on open area and click open in terminal.
commands:
./waf build
./waf --run scratch/first
cd ..
cd netanim-3.108
./NetAnim (This opens the application NetAnim)

Open folders and select fist, xml
Done.

same for 7,8(just use second and third as per needed).

Term Work 9: Steps: You need to use ubuntu cooja (9-10)
cd contiki-ng (from this go to tools).
cd tools
cd cooja
ant run

Now we in application cooja:
New simulation , (name of simulation)Termwork9 create
Now select Motes , Add motes , create new mote type , sky mote , browse (examples,libs,ipv6-hooks,ipv6-hooks.c(file))
Now Compile and create.
Select number of motes as 4 and click add motes.
Now bring all the motes together in a radius
Now right click on mote 1 and select first option and then select serial socket server.

Similarly remaining motes make it client.
Go to all client and change its port number to server's port number.

Term Work 10: Steps
cd contiki-ng (from this go to tools).
cd tools
cd cooja
ant run

Now we in application cooja:
New simulation , (name of simulation)Termwork9 create
Now select Motes , Add motes , create new mote type , sky mote , browse (examples,rpl-udp,upd-server.c(file))
Now Compile and create.
Select number of motes as 1 and click add motes.
Now select Motes , Add motes , create new mote type , sky mote , browse (examples,rpl-udp,upd-client.c(file))
Now Compile and create.
rest same steps as per 9th.