Jean-Xavier BOURGON
Rodolphe LOURD

DICOM

****

Dossier de réalisation

Responsable de l'UV : O. Baala

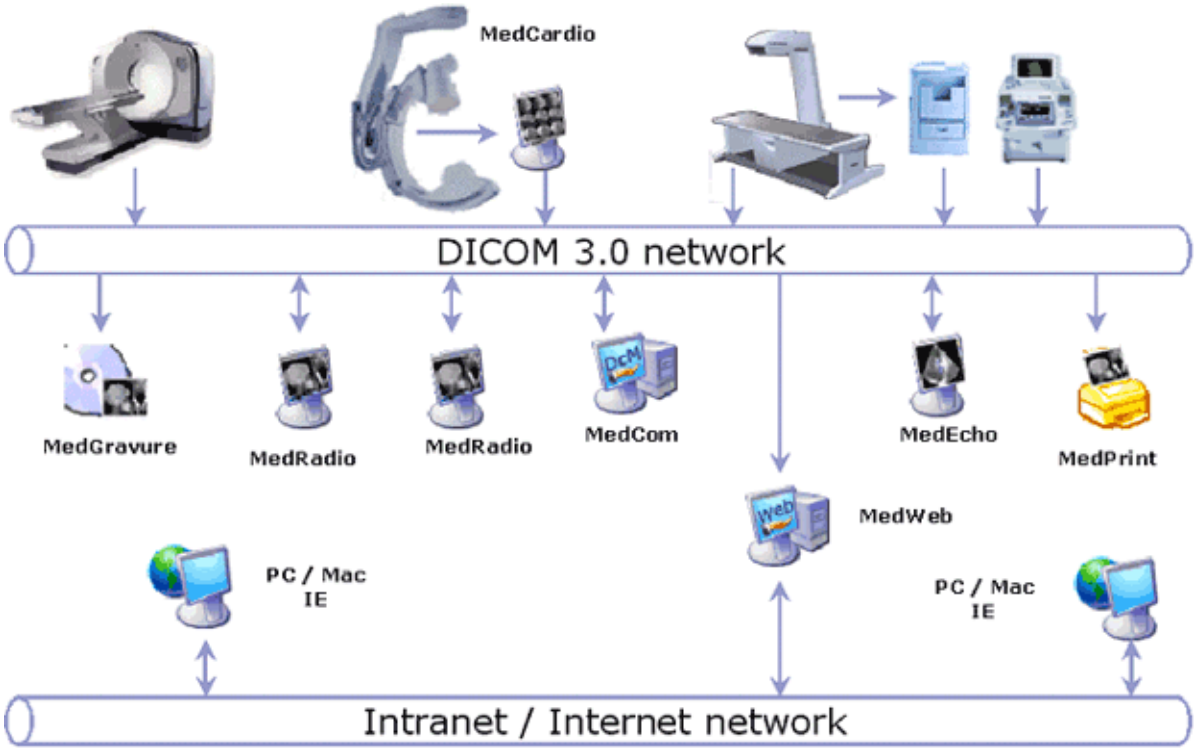
Semestre d'automne 2004

1. Objectif

Le but est de découvrir une méthode de communication selon la norme DICOM et de mettre en place ce protocole à l'aide des enseignements suivis en filière Réseaux et Télécommunications.

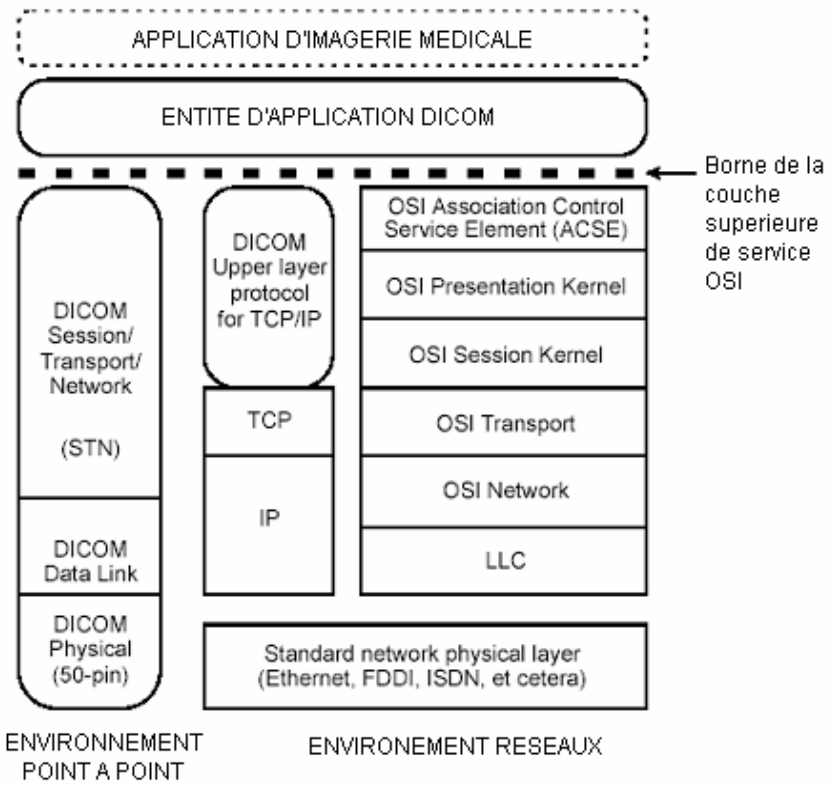
2. Rappel du sujet : La norme DICOM

Elle définit une méthode de communication pour les différents équipements d'imagerie médicale numérique. Cette norme a été créée pour faire communiquer plusieurs appareils médicaux et leur permettre de s'échanger des images et des données quel que soit leur emplacement physique (dans le même hôpital ou entre des hôpitaux) tout en assurant une compatibilité avec les protocoles réseaux déjà existants notamment dans le réseau Internet.



La figure suivante représente la réalisation des échanges de données. Ce modèle comporte trois types de piles de protocoles pour le support de communication entre des entités d'application DICOM :

- *communication selon le modèle point à point*
- *communication selon le modèle OSI*
- *communication selon le modèle TCP/IP*



On va s'intéresser uniquement à la communication selon le modèle TCP/IP, vu qu'elle est la plus utilisée dans les communication réseaux. Les composants représentés par des rectangles arrondis correspondent aux éléments développés dans la norme DICOM.

3. Notre réalisation

D'après les spécificités décrites dans notre cahier des charges et notre cahier de spécifications, nous nous sommes attelés à la programmation du projet.

Nous avons donc comme prévu travaillé en langage C sous une station Linux, les objectifs n'ont pas été modifiés par rapport à ce que nous avions écrit.

Au lancement de notre projet, un menu propose à l'utilisateur de définir si il est :

- Serveur (tapez 1)
- Emetteur (tapez 2)
- Recepteur (tapez 3)

Il est évident qu'en l'absence de serveur l'émetteur et le récepteur ne pourra pas se connecter. Dans une utilisation normale il convient donc tout d'abord de définir une machine qui jouera ce rôle (un compromis entre stabilité et puissance) et sur laquelle on lancera notre programme en premier.

L'émetteur, une fois lancé, aura par la suite à rentrer le nom (adresse IP) du serveur sur lequel il souhaitera diffuser ses messages, les récepteurs en feront de même. Nous stockons tous les

noms et adresses des récepteurs et émetteurs dans un tableau qui accepte au maximum 64 machines (32 émetteurs et 32 récepteurs) ce qui est largement suffisant dans notre exemple.

L'émetteur sera alors ensuite invité à spécifier le destinataire du message parmi les récepteurs connectés au même serveur. Le message sera alors envoyé :

1. tout d'abord de l'émetteur au serveur
2. puis du serveur au bon récepteur

En cas de déconnexion volontaire de l'émetteur ou du serveur les connexions sont fermées proprement. Les différentes parties en sont averties et un message est diffusé aux utilisateurs pour leur faire part de la situation. Ainsi les utilisateurs ne seront pas laissés dans le flou en cas de maintenance par exemple. Dans le cas d'une déconnexion brutale un message sera également envoyé. Ainsi l'émetteur sait si son message a bien été reçu.

Le serveur peut en outre connaître à tout moment le nombre de personnes connectées. La procédure technique est expliquée dans l'analyse de notre code source, en gras sont notés les commentaires qui permettent de comprendre la structure de notre programme. Nous avons détaillé précisément les objectifs de chaque fonction ainsi que les variables pour que le code soir clair.

Le début est constitué des déclarations pour le serveur, le récepteur et l'émetteur, dans certains cas nous réemployons certaines d'entre elles pour optimiser notre programme.

/* contenu du fichier message.h ***/**

```
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>

#define PORT_RM 3000       // port de reception de message
#define PORT_DC 3001    // port de demande de connexion
#define TAILLE_MAX 4000 // taille max du contenu des messages
```

/* définition des types de messages */

```
#define C_CLIENT 1 // demande de connexion du récepteur
#define D_CLIENT 2// demande de déconnexion du récepteur
#define M_DCLIENT 3        // message de l'émetteur
#define D_DCLIENT 4        // demande de déconnexion de l'émetteur
#define D_SERVEUR 5        // déconnexion du serveur
#define INFO_SERVEUR 6  // informations sur le serveur
#define INFO_DCLIENT 7   // informations sur le récepteur

#define DEST_DCLIENT 8  // informations sur le destinataire
```

```
#define BRABRABRA 9      // destinataire inexistant

typedef struct MESSAGE_STRUCT
{
  int type;        // type du message
  char mess[TAILLE_MAX];// contenu du message
} t_message;

t_message message;

        /*************************************************************/
                        /*              emettre_message()              */
                    /*  fonction permettant d'envoyer un message à un récepteur  */
        /*************************************************************/

void emettre_message(int client)
{
  if (write(client,&message,sizeof(message))==-1)
  {
    perror("Erreur write : ");
    exit(1);
  }
}


/*** contenu du fichier dicom.c ***/

#include "message.h"

/*** déclarations du serveur ***/

int sock_rm,sock_dc,acc_rm;  // descripteurs de sockets
int reuse=1;                 // port réutilisable
t_message message;
int nbclients=0;             // indique le nombre de clients connectés
char *thosts[64];            // tableau contenant les noms des clients
char nom_dclient[20];        // nom de l'émetteur
char nom_dest[20];           // nom du destinataire
int test_dest;               // test de présence du destinataire
char mess_diff[20];          // message à diffuser

struct sockaddr_in serveur_rm;  // socket de réception des messages
struct sockaddr_in serveur_dc;  // socket de diffusion des messages
struct hostent *hp;

extern void emettre_message(int client);
int recevoir_message(int client);
void diffuser(t_message msg);
void fermer_connexions();
void attente_diffuseur();
```

```
void traitant_FIN();

/*** déclarations de l'émetteur ***/

int sock;       // descripteur de socket
char nom[20];   // nom de l'émetteur

void recevoir_message_dif(int client);
void traitant_SIGPIPE(int n);
void traitant_SIGINT(int n);

/*** déclarations du récepteur ***/

int at;           // descripteur de socket
char nom_cli[20];  // nom du récepteur

extern void emettre_message(int client);
void recevoir_message_cli(int client);
void traitant_SIGNAL(int n);
```

/*** fonctions du serveur ***/

/* diffuser : fonction permettant de diffuser un message à tous les récepteurs connectés dans le cas ou il nécessaire de prévenir tout le monde en même temps */

```
void diffuser(t_message msg)
{
 int i;
 for (i=0;i<32;i++)
 {
  if (thosts[i]!=NULL)
  {
   emettre_message(i);
  }
 }
}
```

/* fermer_connexions : fonction permettant de fermer toutes les connexions avec les récepteurs */

```
void fermer_connexions()
{
 int i,c;
 for (i=0;i<32;i++)
 {
  if (thosts[i]!=NULL)
  {
   close(i);
   free(thosts[i]);
   thosts[i]=NULL;
  }
```

```
  }
  nbclients=0;
}

/* attente_diffuseur : fonction permettant de mettre le serveur en attente d'un émetteur
*/
void attente_diffuseur()
{
  int lg;
  printf("\nEmetteur non connecté...\n");
  lg=sizeof(serveur_rm);
  acc_rm=accept(sock_rm,(struct sockaddr *)&serveur_rm,(int *)&lg);
  if (acc_rm==-1)
  {
    perror("Erreur accept : ");
    exit(1);
  }
}

/* traitant_FIN : fonction permettant d'arrêter le serveur proprement */
void traitant_FIN(int n)
{
  message.type=D_SERVEUR;
  diffuser(message);
  close(sock_dc);
  close(sock_rm);
  close(acc_rm);
  printf("\nA bientôt...\n");
  exit(1);
}

/* recevoir_message : fonction permettant de recevoir un message d'un
émetteur/récepteur */
int recevoir_message(int client)
{
  int test;
  test=read(client,&message,sizeof(message));
  if (test==-1)
  {
    perror("Erreur read : ");
    exit(1);
  }
  return (int)test;
}
```

/*** fonctions de l'émetteur ***/

```
/* recevoir_message_dif : fonction permettant de recevoir un message du serveur */
void recevoir_message_dif(int client)
```

```
{
 int test;
 test=read(client,&message,sizeof(message));
 if (test==-1)
 {
  perror("Erreur read : ");
  exit(1);
 }
 if (test==0)
 {
  printf("\nConnexion avec le serveur coupée.\n");
  close(sock);
  exit(1);
 }
}
```

/* traitant_SIGPIPE : fonction appelée par SIGPIPE permettant de détecter la déconnexion brutale du serveur */

```
void traitant_SIGPIPE(int n)
{
 printf("\nConnexion avec le serveur perdue...\n");
 close(sock);
 raise(SIGPIPE);
 exit(1);
}
```

/* traitant_SIGINT : fonction appelée par SIGINT permettant de réaliser la déconnexion de l'émetteur auprès du serveur */

```
void traitant_SIGINT(int n)
{
 message.type=D_DCLIENT;
 strcpy(message.mess,"");
 emettre_message(sock);
 close(sock);
 printf("\nDéconnexion...\n");
 exit(1);
}
```

/*** fonctions du récepteur ***/

/* recevoir_message_cli : fonction permettant de recevoir un message du serveur */

```
void recevoir_message_cli(int client)
{
 int test;
 test=read(client,&message,sizeof(message));
 if (test==-1)
 {
  perror("Erreur read : ");
  exit(1);
 }
```

```
  if (test==0)
  {
    printf("\nConnexion avec le serveur coupée.\n");
    close(sock);
    exit(1);
  }
}
```

/* traitant_SIGNAL : fonction appelée par SIGINT permettant de réaliser la déconnexion du récepteur auprès du serveur */

```
void traitant_SIGNAL(int n)
{
  message.type=D_CLIENT;
  emettre_message(sock);
  close(sock);
  printf("\nDéconnexion...\n");
  exit(1);
}
```

/***************************/**

/* Programme principal ***/**

/***************************/**

```
void main()
{
  /* variables globales */

  char choix;
  choix='0';

  /* variables du serveur */

  int lg,c,test,acc_dc,retval,i;
  struct timeval tv;
  char nom[20];  // nom du serveur
  fd_set fdmask;
  struct sigaction action;

  /* variables de l'émetteur */

  char hote[50];  // nom de l'hôte
  struct sockaddr_in serveur;
  struct hostent *hp;

  /* boucle de choix du poste utilisé */

  while ((choix!='1')&&(choix!='2')&&(choix!='3'))
  {
```

```
   printf("\n* Bienvenue sur l'application Dicom - par J.X. Bourgon et R. LOURD *\n");
   printf("\n* taper 1 pour être serveur");
   printf("\n* taper 2 pour être émetteur");
   printf("\n* taper 3 pour être récepteur\n");
   scanf("%c",&choix);
  }

  switch (choix) // Il y a 3 case, un pour chaque choix (serveur, émetteur et récepteur)
  {
   case '1':   // choix du serveur

     /* installation d'un handler pour le signal SIGINT */
     signal(SIGINT,traitant_FIN);
     gethostname(nom,sizeof(nom));
     printf("Serveur démarré ; le nom de la machine est : %s\n",nom);
     hp=gethostbyname(nom);

     /* crée la socket d'écoute pour communiquer avec les récepteurs */
     sock_dc=socket(AF_INET,SOCK_STREAM,0);       // socket de demande de connexion
des récepteurs
     if (sock_dc==-1)
     {
      perror("Erreur socket_dc : ");
      exit(1);
     }

     /* crée la socket d'écoute pour communiquer avec l'émetteur */
     sock_rm=socket(AF_INET,SOCK_STREAM,0);       // socket de réception des messages
*/
     if (sock_rm==-1)
     {
      perror("Erreur socket_rm : ");
      exit(1);
     }

     /* initialisation de la structure serveur_rm côté émetteur */
     bzero(&serveur_rm,sizeof(serveur_rm));
     serveur_rm.sin_family=AF_INET;
     serveur_rm.sin_port=htons(PORT_RM);
     bcopy(hp->h_addr,(char *)&serveur_rm.sin_addr,hp->h_length);

     /* initialisation de la structure serveur_dc côté récepteur */
     bzero(&serveur_dc,sizeof(serveur_dc));
     serveur_dc.sin_family=AF_INET;
     serveur_dc.sin_port=htons(PORT_DC);

     /* attachement de sock_rm */
     setsockopt(sock_rm,SOL_SOCKET,SO_REUSEADDR,(int *)&reuse,sizeof(reuse));
     if (bind(sock_rm,(struct sockaddr *)&serveur_rm,sizeof(serveur_rm))==-1)
     {
```

```
  perror("Erreur bind_rm : ");
  exit(1);
}

/* attachement de sock_dc */
setsockopt(sock_dc,SOL_SOCKET,SO_REUSEADDR,(int *)&reuse,sizeof(reuse));
if (bind(sock_dc,(struct sockaddr *)&serveur_dc,sizeof(serveur_dc))==-1)
{
  perror("Erreur bind_dc : ");
  exit(1);
}

/* création de la file d'attente de la demande de connexion de l'émetteur */
if (listen(sock_rm,1)==-1)
{
  perror("Erreur listen_rm : ");
  exit(1);
}

/* création de la file d'attente des demandes de connexion des récepteurs */
if (listen(sock_dc,5)==-1)
{
  perror("Erreur listen_dc : ");
  exit(1);
}

/* attente de la connexion d'un émetteur */
attente_diffuseur();
printf("\nConnexion d'un émetteur en cours ; attente de son nom...\n");

/* boucle du serveur */
while (1)
{
  FD_ZERO(&fdmask);  // effacer tous les descripteurs associés a fdmask
  FD_SET(sock_dc,&fdmask);      // ajout du descripteur sock_dc
  FD_SET(acc_rm,&fdmask);       // ajout du descripteur sock_rm

  /* ajout des descripteurs de tous les récepteurs connectés */
  for (i=0;i<33;i++)
  {
    if (thosts[i]!=NULL)
    {
      FD_SET(i,&fdmask);
    }
  }

  /* écoute de tous les descripteurs de sockets pendant 15 secondes maxi */
  tv.tv_sec=15;
  tv.tv_usec=0;
  retval=select(32,&fdmask,NULL,NULL,&tv);
```

```
if (retval<0)
{
  printf("Erreur select number %i: \n",errno);
  perror("Erreur select ");
  exit(1);
}

/* si réception de l'émetteur */
if (FD_ISSET(acc_rm,&fdmask))
{
  test=recevoir_message(acc_rm);

  /* test si réception d'informations de l'émetteur */
  if (message.type==INFO_DCLIENT)
  {
    printf("\nConnexion de l'émetteur %s effectuée...\n",message.mess);
    strcpy(nom_dclient,message.mess);

    /* récupération du nom de l'émetteur qui vient de se connecter */
    printf("\nnom_dclient : %s\n",nom_dclient);
    thosts[32]=(char *)malloc(strlen(message.mess)*sizeof(char));
    strcpy(thosts[32],message.mess);
    printf("\nValeur de thosts[32] : %s\n",thosts[32]);
    test=recevoir_message(acc_rm);
    strcpy(mess_diff, message.mess);
  }

  /* informations sur le destinataire du message */
  if (message.type==DEST_DCLIENT)
  {
    printf("\nLe destinataire du message est %s\n",message.mess);
    strcpy(nom_dest,message.mess);
    test_dest=0;
    for (i=0;i<32;i++)
    {
      if (thosts[i]!=NULL)
      {
        //printf ("\nDEBUT FOR\n");
        if (strcmp(thosts[i],nom_dest)==0)
        {
          //printf ("\nDEBUT IF\n");
          test_dest=1;
        }
      }
    }
    if (test_dest==1)
    {
      //printf ("\nDEBUT IF2\n");
      printf("\nLe récepteur destinataire %s est connecté\n",nom_dest);
    }
```

```
          else
          {
           //printf ("\nDEBUT ELSE2\n");
           printf("\nLe récepteur destinataire %s n'est pas connecté\n",nom_dest);

           message.type=BRABRABRA;
           strcpy(message.mess,"brabrabra");
           diffuser(message);
           printf("\nNotification envoyée au diffuseur\n");
          }
          test=recevoir_message(acc_rm);
          strcpy(mess_diff, message.mess);
         }

         /* test si le message doit être envoyé à tous les récepteurs */
         if (message.type==M_DCLIENT)
         {
          printf("Message envoyé : %s\n",message.mess);
          diffuser(message);
          strcpy(mess_diff,message.mess);
         }

         /* test si l'émetteur envoie une demande de déconnexion */
         if (message.type==D_DCLIENT || test==0)
         {
          printf("L'émetteur %s vient de se déconnecter...\n",nom_dclient);
          message.type=D_DCLIENT;
          fermer_connexions();
          close(acc_rm);
          attente_diffuseur();
         }
        }

       /* test si un récepteur envoie une demande de déconnexion (socket d'écoute) */
       if (FD_ISSET(sock_dc,&fdmask))
       {
        /* création d'un descripteur de socket pour les récepteurs qui demandent une
connexion puis établissement de celle-ci */
         printf("\nAttente de connexion d'un récepteur...\n");
         lg=sizeof(serveur_dc);
         acc_dc=accept(sock_dc,(struct sockaddr *)&serveur_dc,(int *)&lg);
         if (acc_dc==-1)
         {
          perror ("Erreur accept sur sock_dc : ");
          exit(1);
         }
         test=recevoir_message(acc_dc);
         if (message.type==C_CLIENT)
         {
          printf("Le message à recevoir est : %s\n",mess_diff);
```

```
        thosts[acc_dc]=(char *)malloc(strlen(message.mess)*sizeof(char));
        strcpy(thosts[acc_dc],message.mess);
        nbclients++;
        printf("Le récepteur %s est connecté...\n", thosts[acc_dc]);
        printf("Il y a %d récepteurs connectés...\n\n",nbclients);
        message.type=INFO_SERVEUR;
        strcpy(message.mess,nom);
        emettre_message(acc_dc);
        message.type=INFO_DCLIENT;
        strcpy(message.mess,nom_dclient);
        emettre_message(acc_dc);
        message.type=M_DCLIENT;
        strcpy(message.mess,mess_diff);
        emettre_message(acc_dc);
      }
     }

     /* test si on reçoit la déconnexion d'un récepteur */
     for (i=0;i<32;i++)
     {
      if (thosts[i]!=NULL)
      {
       if (FD_ISSET(i,&fdmask))
       {
        test=recevoir_message(i);
        if (message.type==D_CLIENT || test==0)
        {
         printf("\nLe récepteur %s est déconnecté.\n",thosts[i]);
         thosts[i]=NULL;
         close(i);
         nbclients--;
         printf("\nIl y a %d récepteurs connectés.\n",nbclients);
         free(thosts[i]);
        }
       }
      }
     }
    }

    break;

   case '2': // choix de l'émetteur

    printf("\nVous êtes émetteur ; veuillez indiquer le nom d'hôte ou l'adresse IP du serveur...\n");
    scanf("%s",&hote);

    /* installation d'un handler pour le signal SIGINT */
    signal(SIGINT,traitant_SIGINT);
```

```
/* installation d'un handler pour le signal SIGPIPE */
signal(SIGPIPE,traitant_SIGPIPE);
hp=gethostbyname(hote);

/* création de la socket de communication avec le serveur */
sock=socket(AF_INET,SOCK_STREAM,0);
if (sock==-1)
{
  perror("Erreur socket : ");
  exit(1);
}

/* initialisation de la structure serveur */
bzero(&serveur,sizeof(serveur));
serveur.sin_family=AF_INET;
serveur.sin_port=htons(PORT_RM);
bcopy(hp->h_addr,(char *)&serveur.sin_addr,hp->h_length);

/* demande de connexion sur le serveur */
printf("\nConnexion sur le serveur de diffusion %s...\n",hp->h_name);
if (connect(sock,(struct sockaddr *)&serveur,sizeof(serveur))==-1)
{
  perror("Erreur connect ");
  exit(1);
}

/* envoie le nom de l'émetteur au serveur */
message.type=INFO_DCLIENT;
printf("Entrez votre identité :\n");
gets(nom);
strcpy(message.mess,nom);
printf("le nom du client_diff est %s\n",message.mess);
emettre_message(sock);

/* envoi du destinataire auquel diffuser le message */
message.type=DEST_DCLIENT;
printf("Nom du destinataire : \n");
gets(nom);
strcpy(message.mess,nom);
//printf("le nom du destinataire est %s\n",message.mess);
emettre_message(sock);

/* boucle de traitement des messages envoyés au serveur */
while (1)
{
  message.type=M_DCLIENT;
  printf("Message à envoyer : ");
  gets(message.mess);
  emettre_message(sock);
```

```
      /* messages reçus du serveur */
      if (message.type==BRABRABRA)
      printf("\nDestinataire inconnu\n");
    }

    break;

  case '3': // choix du récepteur

    printf("\nVous êtes récepteur ; veuillez indiquer le nom d'hôte ou l'adresse IP du serveur...\n");
    scanf("%s",&hote);

    /* installation d'un handler pour le signal SIGINT */
    signal(SIGINT,traitant_SIGNAL);
    hp=gethostbyname(hote);

    /* création de la socket de communication avec le serveur */
    sock=socket(AF_INET,SOCK_STREAM,0);
    if (sock==-1)
    {
      perror("Erreur socket : ");
      exit(1);
    }

    /* initialisation de la structure serveur */
    bzero(&serveur,sizeof(serveur));
    serveur.sin_family=AF_INET;
    serveur.sin_port=htons(PORT_DC);
    bcopy(hp->h_addr,(char *)&serveur.sin_addr,hp->h_length);

    /* demande de connexion sur le serveur */
    printf("\nDemande de connexion ...\n");
    if (connect(sock,(struct sockaddr *)&serveur,sizeof(serveur))==-1)
    {
      perror("Erreur connect ");
      exit(1);
    }

    /* envoie le nom du récepteur au serveur */
    message.type=C_CLIENT;
    printf("Entrez votre identité :\n");
    gets(nom_cli);
    strcpy(message.mess,nom_cli);
    emettre_message(sock);

    /* boucle de traitement des messages arrivant du serveur */
    while (1)
    {
      /* lecture du message */
```

```
      recevoir_message_cli(sock);

      /* réception d'informations concernant le serveur */
      if (message.type==INFO_SERVEUR)
      {
        printf("\nBienvenue sur le serveur %s\n",message.mess);
      }

      /* réception d'informations concernant l'émetteur */
      if (message.type==INFO_DCLIENT)
      {
        printf("\nL'émetteur est %s\n",message.mess);
      }

      /* si le message vient de l'émetteur alors on l'affiche */
      if (message.type==M_DCLIENT)
      {
        printf("\nLe message est %s\n",message.mess);
      }

      /* si le serveur se déconnecte alors on déconnecte le récepteur */
      if (message.type==D_SERVEUR)
      {
        printf("\nConnexion coupée par le serveur...\n");
        close(sock);
        exit(1);
      }

      /* si l'émetteur se déconnecte alors on déconnecte le récepteur */
      if (message.type==D_DCLIENT)
      {
        printf("\nLe client diffuseur vient de se deconnecter ...\n\n");
        close(sock);
        exit(1);
      }
    }

    break;

  }
}
```

4. Conclusion

Ce projet nous a donc permis de mettre en pratiques nos connaissances acquises lors des séances de travaux dirigés et des TP, notamment en ce qui concerne la communication par sockets. Nous avons ainsi pu travailler sur la partie réseau d'un projet concret d'imagerie médicale.