Anexo 1

Código fuente utilizado

Cliente de Flujos

tomado de http://es.tldp.org/Tutoriales/PROG-SOCKETS/prog-sockets/x459.html

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
/* netbd.h es necesitada por la estructura hostent ;-) */

#define PORT 3550
/* El Puerto Abierto del nodo remoto */

#define MAXDATASIZE 100
/* El número máximo de datos en bytes */

int main(int argc, char *argv[])
{
   int fd, numbytes;
   /* ficheros descriptores */

   char buf[MAXDATASIZE];
   /* en donde es almacenará el texto recibido */

   struct hostent *he;
   /* estructura que recibirá información sobre el nodo remoto */

   struct sockaddr_in server;
   /* información sobre la dirección del servidor */

   if (argc !=2) {
      /* esto es porque nuestro programa sólo necesitará un
      argumento, (la IP) */
      printf("Uso: %s <Dirección IP>\n",argv[0]);
      exit(-1);
   }

   if ((he=gethostbyname(argv[1]))==NULL){
      /* llamada a gethostbyname() */
      printf("gethostbyname() error\n");
      exit(-1);
   }

   if ((fd=socket(AF_INET, SOCK_STREAM, 0))==-1){
      /* llamada a socket() */
      printf("socket() error\n");
      exit(-1);d
   }

   server.sin_family = AF_INET;
   server.sin_port = htons(PORT);
   /* htons() es necesaria nuevamente ;-o */
   server.sin_addr = *((struct in_addr *)he->h_addr);
   /*he->h_addr pasa la información de ``*he'' a "h_addr" */
   bzero(&(server.sin_zero),8);
```

```
   if(connect(fd, (struct sockaddr *)&server,
      sizeof(struct sockaddr))==-1){
      /* llamada a connect() */
      printf("connect() error\n");
      exit(-1);
   }

   if ((numbytes=recv(fd,buf,MAXDATASIZE,0)) == -1){
      /* llamada a recv() */
      printf("Error en recv() \n");
      exit(-1);
   }

   buf[numbytes]='\0';

   printf("Mensaje del Servidor: %s\n",buf);
   /* muestra el mensaje de bienvenida del servidor =) */

   close(fd);   /* cerramos fd =) */

}
```

Servidor de Flujos

http://es.tldp.org/Tutoriales/PROG-SOCKETS/prog-sockets/x453.html

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
/* netbd.h es necesitada por la estructura hostent ;-) */

#define PORT 3550
/* El Puerto Abierto del nodo remoto */

#define MAXDATASIZE 100
/* El número máximo de datos en bytes */

int main(int argc, char *argv[])
{
   int fd, numbytes;
   /* ficheros descriptores */

   char buf[MAXDATASIZE];
   /* en donde es almacenará el texto recibido */

   struct hostent *he;
   /* estructura que recibirá información sobre el nodo remoto */

   struct sockaddr_in server;
   /* información sobre la dirección del servidor */

   if (argc !=2) {
      /* esto es porque nuestro programa sólo necesitará un
      argumento, (la IP) */
      printf("Uso: %s <Dirección IP>\n",argv[0]);
      exit(-1);
   }
```

```
    if ((he=gethostbyname(argv[1]))==NULL){
       /* llamada a gethostbyname() */
       printf("gethostbyname() error\n");
       exit(-1);
    }

    if ((fd=socket(AF_INET, SOCK_STREAM, 0))==-1){
       /* llamada a socket() */
       printf("socket() error\n");
       exit(-1);d
    }

    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);
    /* htons() es necesaria nuevamente ;-o */
    server.sin_addr = *((struct in_addr *)he->h_addr);
    /*he->h_addr pasa la información de ``*he'' a "h_addr" */
    bzero(&(server.sin_zero),8);

    if(connect(fd, (struct sockaddr *)&server,
       sizeof(struct sockaddr))==-1){
       /* llamada a connect() */
       printf("connect() error\n");
       exit(-1);
    }

    if ((numbytes=recv(fd,buf,MAXDATASIZE,0)) == -1){
       /* llamada a recv() */
       printf("Error en recv() \n");
       exit(-1);
    }

    buf[numbytes]='\0';

    printf("Mensaje del Servidor: %s\n",buf);
    /* muestra el mensaje de bienvenida del servidor =) */

    close(fd);   /* cerramos fd =) */

}
```

streamcount_echoclient.c

tomado de Newmarch Jan, Linux Journal October 2007

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

#define ECHO_PORT 2013

char *usage_msg = "usage: streamcount_echo_client ip-addr istreams
ostreams";
char *msg = "hello";
void usage () {
      fprintf(stderr, "%s\n", usage_msg);
      exit(1);
}

int main(int argc, char *argv[]){
      int sockfd;
      int len;
      struct sockaddr_in serv_addr;
      int port = ECHO_PORT;
      struct sctp_initmsg initmsg;
      struct sctp_status status;

      if (argc != 4) usage ();

      /* create endpoint */

      sockfd = socket (AF_INET, SOCK_STREAM, IPPROTO_SCTP);

      if (sockfd <0) {
            perror("socket creation");
            exit (2);
      }

      /* connect to server */

      serv_addr.sin_family = AF_INET;
      serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
      serv_addr.sin_port = htons(port);

      memset (&initmsg, 0 ,sizeof (initmsg));
      initmsg.sinit_max_instreams = atoi (argv[2]);
      initmsg.sinit_num_ostreams = atoi (argv[3]);
      printf("Asking for: input streams: %d, output streams: %d\n",
initmsg.sinit_max_instreams,
      initmsg.sinit_num_ostreams);

      if (setsockopt (sockfd, IPPROTO_SCTP, SCTP_INITMSG, &initmsg,
sizeof (initmsg))) {
            perror("set sock opt\n");
      }
```

```
      if (connect(sockfd, (struct sockaddr *) &serv_addr,
      sizeof(serv_addr)) < 0) {
            perror("connectx");
            exit(3);
      }
      len = sizeof(status);
      memset(&status, 0, len);

      if (getsockopt(sockfd, IPPROTO_SCTP, SCTP_STATUS, &status, &len)
== -1) {
            perror("get sock opt");
      }
      printf("Got: input streams: %d, output streams: %d\n",
      status.sstat_instrms,   status.sstat_outstrms);

      /* give the server time to do something */
      sleep(2);
      /* no reads/writes are done */
      close(sockfd);
      exit(0);
}
```

streamcount_echoserver.c

tomado de Newmarch Jan, Linux Journal October 2007

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/sctp.h>
#define ECHO_PORT 2013

char *usage_msg = "usage: streamcount_echo_server istreams ostreams";
void usage() {
      fprintf(stderr, "%s\n", usage_msg);
      exit(1);
}

int main(int argc, char *argv[]) {
      int sockfd, client_sockfd;
      int len;
      struct sockaddr_in serv_addr, client_addr;
      int port = ECHO_PORT;
      struct sctp_initmsg initmsg;
      struct sctp_status status;

      if (argc != 3) usage();

      /* CREATE ENDPOINT */

      sockfd = socket(AF_INET, SOCK_STREAM,IPPROTO_SCTP);

      if (sockfd < 0) {
            perror("socket");
            exit(2);
      }
```

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

      if (bind(sockfd, (struct sockaddr *) &serv_addr,
sizeof(serv_addr)) == -1) {
            perror("sctp bind");
            exit(2);
      }

      memset(&initmsg, 0, sizeof(initmsg));
      initmsg.sinit_max_instreams = atoi (argv[1]);
      initmsg.sinit_num_ostreams = atoi (argv[2]);
      printf("Asking for: input streams: %d, output streams: %d\n",
initmsg.sinit_max_instreams, initmsg.sinit_num_ostreams);

      if (setsockopt (sockfd, IPPROTO_SCTP, SCTP_INITMSG, &initmsg,
sizeof(initmsg))) {
            perror("set sock op/n");
      }

      /*specify queue */

      listen(sockfd, 5);

      for (;;) {
            len = sizeof(client_addr);
            client_sockfd = accept(sockfd, (struct sockaddr *)
      &client_addr, & len);

            if (client_sockfd == -1) {
                  perror(NULL) ; continue;
            }

            memset(&status, 0, sizeof(status));
            len = sizeof(status);

            if (getsockopt(client_sockfd, IPPROTO_SCTP, SCTP_STATUS,
&status, &len) == -1){
                  perror("get sock opt");
            }

            printf("Got: input streams: %d, ouput streams: %d\n",
status.sstat_instrms, status.sstat_outstrms);

            /* give the client time to do something */

            sleep(2);
            close(client_sockfd);
      }
}
```

streamsend_echoclient.c

tomado de Newmarch Jan, Linux Journal October 2007

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

#include <netinet/sctp.h>
/*include extras*/

#include <errno.h>
#include <string.h>


/*fin include extras*/

#define SIZE 1024
char buf [SIZE];
#define ECHO_PORT 2013
char *usage_msg = "Usar: ./app direccionIP numero_streams_input
numero_streams_output stream\n";

void usage() {
      fprintf(stderr, "%s\n", usage_msg);
      exit(1);
}
/*funciones*/
int
sctp_connectx(int fd, struct sockaddr *addrs, int addrcnt)
{
      void *addrbuf;
      struct sockaddr *sa_addr;
      socklen_t addrs_size = 0;
      int i;

      addrbuf = addrs;
      for (i = 0; i < addrcnt; i++) {
            sa_addr = (struct sockaddr *)addrbuf;
            switch (sa_addr->sa_family) {
            case AF_INET:
                  addrs_size += sizeof(struct sockaddr_in);
                  addrbuf += sizeof(struct sockaddr_in);
                  break;
            case AF_INET6:
                  addrs_size += sizeof(struct sockaddr_in6);
                  addrbuf += sizeof(struct sockaddr_in6);
                  break;
            default:
                  errno = EINVAL;
                  return -1;
            }
      }

      return setsockopt(fd, SOL_SCTP, SCTP_SOCKOPT_CONNECTX, addrs,
addrs_size);
```

```
}

/* This library function assists the user with the advanced features
 * of SCTP.  This is a new SCTP API described in the section 8.7 of
the
 * Sockets API Extensions for SCTP. This is implemented using the
 * sendmsg() interface.
 */
int
sctp_sendmsg(int s, const void *msg, size_t len, struct sockaddr *to,
           socklen_t tolen, uint32_t ppid, uint32_t flags,
           uint16_t stream_no, uint32_t timetolive, uint32_t context)
{
      struct msghdr outmsg;
      struct iovec iov;
      char outcmsg[CMSG_SPACE(sizeof(struct sctp_sndrcvinfo))];
      struct cmsghdr *cmsg;
      struct sctp_sndrcvinfo *sinfo;

      outmsg.msg_name = to;
      outmsg.msg_namelen = tolen;
      outmsg.msg_iov = &iov;
      iov.iov_base = (void *)msg;
      iov.iov_len = len;
      outmsg.msg_iovlen = 1;

      outmsg.msg_control = outcmsg;
      outmsg.msg_controllen = sizeof(outcmsg);
      outmsg.msg_flags = 0;

      cmsg = CMSG_FIRSTHDR(&outmsg);
      cmsg->cmsg_level = IPPROTO_SCTP;
      cmsg->cmsg_type = SCTP_SNDRCV;
      cmsg->cmsg_len = CMSG_LEN(sizeof(struct sctp_sndrcvinfo));

      outmsg.msg_controllen = cmsg->cmsg_len;
      sinfo = (struct sctp_sndrcvinfo *)CMSG_DATA(cmsg);
      memset(sinfo, 0, sizeof(struct sctp_sndrcvinfo));
      sinfo->sinfo_ppid = ppid;
      sinfo->sinfo_flags = flags;
      sinfo->sinfo_stream = stream_no;
      sinfo->sinfo_timetolive = timetolive;
      sinfo->sinfo_context = context;

      return sendmsg(s, &outmsg, 0);
}

/* This library function assist the user with sending a message
without
 * dealing directly with the CMSG header.
 */
int
sctp_send(int s, const void *msg, size_t len,
          const struct sctp_sndrcvinfo *sinfo, int flags)
{
      struct msghdr outmsg;
      struct iovec iov;

      outmsg.msg_name = NULL;
      outmsg.msg_namelen = 0;
      outmsg.msg_iov = &iov;
```

```
	iov.iov_base = (void *)msg;
	iov.iov_len = len;
	outmsg.msg_iovlen = 1;
	outmsg.msg_controllen = 0;

	if (sinfo) {
		char outcmsg[CMSG_SPACE(sizeof(struct sctp_sndrcvinfo))];
		struct cmsghdr *cmsg;

		outmsg.msg_control = outcmsg;
		outmsg.msg_controllen = sizeof(outcmsg);
		outmsg.msg_flags = 0;

		cmsg = CMSG_FIRSTHDR(&outmsg);
		cmsg->cmsg_level = IPPROTO_SCTP;
		cmsg->cmsg_type = SCTP_SNDRCV;
		cmsg->cmsg_len = CMSG_LEN(sizeof(struct sctp_sndrcvinfo));

		outmsg.msg_controllen = cmsg->cmsg_len;
		memcpy(CMSG_DATA(cmsg), sinfo, sizeof(struct
sctp_sndrcvinfo));
	}

	return sendmsg(s, &outmsg, flags);
}

/*fin funciones*/

int main(int argc, char *argv[1]) {
	int sockfd;
	int len;
	struct sockaddr_in serv_addr;
	struct sockaddr_in *addresses;
	int addr_size = sizeof(struct sockaddr_in);
	int addr_count = argc - 1;
	int port = ECHO_PORT;

	char *message = "Enviando mensaje SCTP al servidor\n";
	struct sctp_initmsg initmsg;
	struct sctp_status status;
	struct sctp_sndrcvinfo sinfo;

	int ochannel;

	if (argc != 5) usage();

	/* create endpoint */

	sockfd = socket(AF_INET, SOCK_STREAM,IPPROTO_SCTP);

	if (sockfd < 0) {
		perror (NULL);
		exit(2);
	}

	/* connect to server */
	addresses = malloc(addr_size);

	if (addresses == NULL) {
		exit(1);
	}
```

```
      serv_addr.sin_family = AF_INET;
      serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
      serv_addr.sin_port = htons(port);

      memcpy(addresses, &serv_addr, addr_size);

      memset(&initmsg, 0, sizeof(initmsg));
      initmsg.sinit_max_instreams = atoi(argv[2]);
      initmsg.sinit_num_ostreams = atoi(argv[3]);
      printf("Negociando numero de streams...\n");
      printf("Cliente tiene disponible: input streams: %d, output
streams: %d\n",
            initmsg.sinit_max_instreams, initmsg.sinit_num_ostreams);

      if (setsockopt(sockfd, IPPROTO_SCTP, SCTP_INITMSG, &initmsg,
sizeof(initmsg))) {
            perror("set sock opt\n");
      }
      if (sctp_connectx(sockfd, (struct sockaddr *) addresses, 1) <
0){
            perror("connectx\n");
            exit(3);
      }

      memset(&status,0,sizeof(status));
      len=sizeof(status);
      status.sstat_assoc_id = 1;

      if (getsockopt(sockfd, IPPROTO_SCTP, SCTP_STATUS, &status, &len)
== -1) {
            perror("get sock opt/n");
      }

      printf("Se llega al acuerdo:\n input  streams: %d, output
streams: %d\n", status.sstat_instrms, status.sstat_outstrms);

      /* sanity check channel */

      ochannel = atoi(argv[4]);

      if (ochannel >= status.sstat_outstrms)
            printf("writing on illegal channel %d\n", ochannel);

      /* transfer data */

      bzero(&sinfo, sizeof(sinfo));
      sinfo.sinfo_stream = ochannel;
      sctp_send(sockfd, message, strlen(message), &sinfo, 0);
      sinfo.sinfo_flags = SCTP_EOF;
      sctp_send(sockfd, NULL, 0, &sinfo, 0);

      close(sockfd);
      exit(0);
}
```

streamsend_echoserver.c

tomado de Newmarch Jan, Linux Journal October 2007

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

#include <netinet/sctp.h>

#define SIZE 1024
char buf [SIZE];
#define TIME_PORT 2013


/*include extras*/
#include <string.h>
#include <errno.h>
/*fin extra include*/
/*Funciones*/
int sctp_recvmsg(int s, void *msg, size_t len, struct sockaddr *from,
             socklen_t *fromlen, struct sctp_sndrcvinfo *sinfo,
             int *msg_flags)
{
      int error;
      struct iovec iov;
      struct msghdr inmsg;
      char incmsg[CMSG_SPACE(sizeof(struct sctp_sndrcvinfo))];
      struct cmsghdr *cmsg = NULL;

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

      iov.iov_base = msg;
      iov.iov_len = len;

      inmsg.msg_name = from;
      inmsg.msg_namelen = fromlen ? *fromlen : 0;
      inmsg.msg_iov = &iov;
      inmsg.msg_iovlen = 1;
      inmsg.msg_control = incmsg;
      inmsg.msg_controllen = sizeof(incmsg);

      error = recvmsg(s, &inmsg, 0);
      if (error < 0)
            return error;

      if (fromlen)
            *fromlen = inmsg.msg_namelen;
      if (msg_flags)
            *msg_flags = inmsg.msg_flags;

      if (!sinfo)
            return error;

      for (cmsg = CMSG_FIRSTHDR(&inmsg); cmsg != NULL;
                        cmsg = CMSG_NXTHDR(&inmsg, cmsg)){
            if ((IPPROTO_SCTP == cmsg->cmsg_level) &&
                (SCTP_SNDRCV == cmsg->cmsg_type))
                  break;
```

```
	}

	  /* Copy sinfo. */
	if (cmsg)
		memcpy(sinfo, CMSG_DATA(cmsg), sizeof(struct
sctp_sndrcvinfo));

	return (error);
}
int
sctp_bindx(int fd, struct sockaddr *addrs, int addrcnt, int flags)
{
	int setsock_option = 0;
	void *addrbuf;
	struct sockaddr *sa_addr;
	socklen_t addrs_size = 0;
	int i;

	switch (flags) {
	case SCTP_BINDX_ADD_ADDR:
		setsock_option = SCTP_SOCKOPT_BINDX_ADD;
		break;
	case SCTP_BINDX_REM_ADDR:
		setsock_option = SCTP_SOCKOPT_BINDX_REM;
		break;
	default:
		errno = EINVAL;
		return -1;
	}

	addrbuf = addrs;
	for (i = 0; i < addrcnt; i++) {
		sa_addr = (struct sockaddr *)addrbuf;
		switch (sa_addr->sa_family) {
		case AF_INET:
			addrs_size += sizeof(struct sockaddr_in);
			addrbuf += sizeof(struct sockaddr_in);
			break;
		case AF_INET6:
			addrs_size += sizeof(struct sockaddr_in6);
			addrbuf += sizeof(struct sockaddr_in6);
			break;
		default:
			errno = EINVAL;
			return -1;
		}
	}

	return setsockopt(fd, SOL_SCTP, setsock_option, addrs,
addrs_size);
}

/*fin funciones*/

char *usage_msg = "Usar: ./app direccionIP numero_streams_input
numero_streams_output\n";
void usage() {
	fprintf(stderr, "%s\n", usage_msg);
}
int main(int argc, char *argv[]) {
	int sockfd, client_sockfd;
```

```
      int nread, len;
      struct sockaddr_in serv_addr, client_addr;
      int addr_count=1;
      int port = TIME_PORT;
      struct sctp_initmsg initmsg;
      struct sctp_status status;
      struct sctp_sndrcvinfo sinfo;
      int flags;
      if (argc != 4) usage();

      /* create endpoint */
      sockfd = socket (AF_INET, SOCK_STREAM, IPPROTO_SCTP);
      if (sockfd < 0) {
            perror(NULL);
            exit(2);
      }
      serv_addr.sin_family = AF_INET;
      serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
      serv_addr.sin_port = htons(port);

      if (sctp_bindx(sockfd, (struct sockaddr *) &serv_addr,
addr_count,
      SCTP_BINDX_ADD_ADDR)== -1) {
            perror("sctp bindx");
            exit(2);
      }
      memset(&initmsg,0, sizeof(initmsg));
      initmsg.sinit_max_instreams = atoi(argv[2]);
      initmsg.sinit_num_ostreams = atoi(argv[3]);
      printf("Negociando numero de streams...\n");
      printf("Servidor tiene disponible: %d, output atreams; %d\n",
      initmsg.sinit_max_instreams,
      initmsg.sinit_num_ostreams);
      if (setsockopt(sockfd, IPPROTO_SCTP,
      SCTP_INITMSG, &initmsg, sizeof(initmsg))) {
            perror("set sock opt\n");
      }
      /* specity queue,*/
      listen(sockfd, 5);
      for (;;) {
            len = sizeof(client_addr);
            client_sockfd = accept (sockfd, (struct sockaddr *)
&client_addr,
            &len);

            if (client_sockfd == -1) {
                  perror(NULL); continue;
            }

            memset(&status, 0, sizeof(status));
            len = sizeof(status);
            status.sstat_assoc_id= 0;

            if (getsockopt(client_sockfd, IPPROTO_SCTP, SCTP_STATUS,
&status, &len) == -1){
                  perror("get sock opt/n");
            }
            printf("Se llega al acuerdo:\n input streams:   %d, output
streams: %d\n",
            status.sstat_instrms,
            status.sstat_outstrms);
```

```
            for(;;) {
                  /* tranfer data */
                  len = sizeof(client_addr);
                  bzero(&sinfo,sizeof(sinfo));
                  nread = sctp_recvmsg(client_sockfd, buf, SIZE,
                  (struct sockaddr *) &client_addr, &len,   &sinfo,
&flags);

                  if (nread == 0) {
                        break;
                  }
                  printf("read %d bytes on channel %hd\n", nread,
sinfo.sinfo_stream);
                  printf("sinfo flags: %d\n", sinfo.sinfo_flags);
                  write(1, buf, nread);
            }
            close(client_sockfd);
      }
}
```

GeneraPaquetes.c

```
/*************************************************************
   GeneraPaquetes.c - Aplicacion que genera y recibe los paquetes
Traduccion del codigo de Gustavo Do Carmo
*************************************************************/

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netinet/sctp.h>
#include <arpa/inet.h>
#include <errno.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
#include <pthread.h>
#include "./Misc.c"

//-------------------------------------------------------------
//Estruturas de parametros para funciones - threads
//-------------------------------------------------------------
struct parametroT1
{
  int sock_fd;
  struct sockaddr_in direccionIP;
};

//-------------------------------------------------------------
//Protótipos
//-------------------------------------------------------------
//Funciones para control general para mostrar los resultados
void uso (void);
void verificaEntrada (void);
void parser (int argc, char **argv);
void calculaTiempo (void);
void geraScriptGnuPlot (void);

//Funciones para la creacion de la comunicaion
void creaConexionTCP (void);
void creaAsociacionSCTP (void);

//Funcion para envio y recepcion de mensajes
void *enviaMensajesTCP (struct parametroT1 *);
void *enviaMensajesSCTP (struct parametroT1 *);
void *recibeMensajesTCP (struct parametroT1 *);
void *recibeMensajesSCTP (struct parametroT1 *);

//Funciones especificas para TCP
int writeN (int fd, const void *buffer, int n);
int readN (int fd, void *buffer, int n);

//-------------------------------------------------------------
//Variables globales
```

```
//---------------------------------------------------------------
int flujos = 0;
int fila = 0;
char *direccionIPLocal = NULL;
char *direccionIPRemoto = NULL;
int puertoLocal = 0;
int puertoRemoto = 0;
int tamanoBuffer = 0;
int numeroMensajesEnviar = 30;
int tipoDeProtocolo = 0;
int tamanoMensaje = 20;
int mensajeDesordenado = 0;
int intervaloEntreMensajes = 0;
int verbose = 1;
FILE *archivo;
FILE *archivo2;

//Representacion de mensajes Gnutella
extern char *gnutella[5];

//estrutura enviada via socket
struct tipao
{
  int id;
  char sendline[100];
};

//Estruturas y variables para el calculo de tiempo
struct paraContar
{
  int id;
  int tamano;
  struct timeval cronometroIda;
  struct timeval cronometroVuelta;
};

struct timeval tiempoGlobalInicial;
struct timeval tiempoGlobalFinal;
int cantidadEnviadaGlobal = 0;
struct paraContar matriz[NUMERO_MAX_MENSAJES];

//Miscelania
double latenciaMedia;
double velocidadMedia;
double velocidadGeneral = 0;
double latenciaGeneral = 0;
double latenciaTotal;
double velocidadTotal;
double latenciaIndividual[NUMERO_MAX_MENSAJES];
double velocidadIndividual[NUMERO_MAX_MENSAJES];

//---------------------------------------------------------------
//Programa principal
//---------------------------------------------------------------
int
main (int argc, char **argv)
{
  parser (argc, argv);
  verificaEntrada ();

  //Archivos para almacenar los dados de desempeno
```

```
  char nombrearchivo1[50];
  char nombrearchivo2[50];

  sprintf (nombrearchivo1,
         "./ResultadosLatencias_T%d_M%d_S%d_F%d_D%d_U%d.txt",
         tipoDeProtocolo, numeroMensajesEnviar, tamanoMensaje,
         flujos, mensajeDesordenado, intervaloEntreMensajes);

  sprintf (nombrearchivo2,
         "./ResultadosVelocidad_T%d_M%d_S%d_F%d_D%d_U%d.txt",
         tipoDeProtocolo, numeroMensajesEnviar, tamanoMensaje,
         flujos, mensajeDesordenado, intervaloEntreMensajes);

  archivo = fopen (nombrearchivo1, "w");
  archivo2 = fopen (nombrearchivo2, "w");

  int i;

  //Ejecuta el programa por un número definido de veces
  if (tipoDeProtocolo == 1)
    for (i = 1; i <= verbose; i++)
      {                        //Protocolo escojido fue TCP
      creaConexionTCP ();
      }
  else
    for (i = 1; i <= verbose; i++)
      {                        //Protocolo escojido fue SCTP
      creaAsociacionSCTP ();
      }

  fclose (archivo);
  fclose (archivo2);

  return 0;
}                       //fin de main


//------------------------------------------------------------------
//Adquiriendo los datos de la linea de comandos
//------------------------------------------------------------------
void
parser (int argc, char **argv)
{
  int getoptC;
  int bufferAux;
  int numAux;
  int tamanoAux;
  int desorden;
  int intervalo;
  int verb;

  while (argc)
    {
      getoptC = getopt (argc, argv, "H:P:h:p:t:x:f:b:m:s:d:u:v:");

      if (getoptC < 0)
      break;

      switch (getoptC)
      {
```

```
case 'H':
  direccionIPLocal = optarg;
  break;

case 'P':
  puertoLocal = atoi (optarg);
  break;

case 'h':
  direccionIPRemoto = optarg;
  break;

case 'p':
  puertoRemoto = atoi (optarg);
  break;

case 't':
  tipoDeProtocolo = atoi (optarg);
  break;

case 'x':
  flujos = atoi (optarg);
  break;

case 'f':
  fila = atoi (optarg);
  break;

case 'b':
  bufferAux = atoi (optarg);
  if ((bufferAux > 0) && (bufferAux < 1001))
    tamanoBuffer = bufferAux;
  break;

case 'm':
  numAux = atoi (optarg);
  if ((numAux > 0) && (numAux < 101))
    numeroMensajesEnviar = numAux;
  break;

case 's':
  tamanoAux = atoi (optarg);
  if ((tamanoAux > 1) && (tamanoAux < 101))
    tamanoMensaje = tamanoAux;
  break;

case 'd':
  desorden = atoi (optarg);
  if (desorden == 1)
    mensajeDesordenado = MSG_UNORDERED;
  break;

case 'u':
  intervalo = atoi (optarg);
  if ((intervalo > 0) && (intervalo < 1001))
    intervaloEntreMensajes = intervalo;
  break;

case 'v':
  verb = atoi (optarg);
  if ((verb > 1) && (verb < 101))
```

```
          verbose = verb;
        break;

      default:
        break;
      }                 //fin switch
    }                   //fin while argc
}                       //fin de parser


//---------------------------------------------------------------
//Revisando los datos de la linea de comandos
//---------------------------------------------------------------
void
verificaEntrada (void)
{
  if ((direccionIPLocal == NULL) ||
      (inet_aton (direccionIPLocal, NULL) == 0))
    {
      printf ("\nFalto colocar una direccion IP local\n");
      uso ();
      exit (0);
    }

  if (puertoLocal == 0)
    {
      printf ("\nFalto colocar un puerto local\n");
      uso ();
      exit (0);
    }

  if ((direccionIPRemoto == NULL) ||
      (inet_aton (direccionIPRemoto, NULL) == 0))
    {
      printf ("\nFalto colocar una direccion IP Remoto\n");
      uso ();
      exit (0);
    }

  if (puertoRemoto == 0)
    {
      printf ("\nFalto colocar un puerto Remoto\n");
      uso ();
      exit (0);
    }

  if ((tipoDeProtocolo != 1) && (tipoDeProtocolo != 2))
    {
      printf ("\nEscoja el protocolo!\n");
      printf ("\nTCP=1 o SCTP=2\n");
      uso ();
      exit (0);
    }

  if ((flujos < 1) && (tipoDeProtocolo == 2))
    {
      printf ("\nFluJo configurado para 1\n");
      flujos = 1;
    }

  if ((fila < 1) && (tipoDeProtocolo == 2))
```

```
    {
      printf ("\nFila seteada para 1\n");
      fila = 1;
    }

  printf ("\n");
  printf ("DireccionIP del host local:%s\n", direccionIPLocal);
  printf ("Puerto del host local:%d\n", puertoLocal);
  printf ("DireccionIP del host Remoto:%s\n", direccionIPRemoto);
  printf ("Puerto del host Remoto:%d\n", puertoRemoto);

  if ((tipoDeProtocolo == 1))       //Caso TCP
    printf ("Protocolo de transporte: TCP\n");
  else
    {                   //Caso SCTP
      printf ("Protocolo de transporte: STCP\n");
      printf ("Numero de flujos:%d\n", flujos);
      printf ("Fila de Mensajes en cada flujo:%d\n", fila);

      if (tamanoBuffer == 0)
      printf ("Buffer de envio y recepcion configurado como
patron\n");
      else
      printf ("Buffer de envio y recepcion (kBytes):%d\n",
            tamanoBuffer);

      if (mensajeDesordenado == 0)
      printf ("Entrega de Mensajes: ordenada\n");
      else
      printf ("Entrega de Mensajes: desordenada\n");
    }                   //Fin caso SCTP

  printf ("Numero de Mensajes a enviar:%d\n",
        numeroMensajesEnviar);
  printf ("tamano de Mensajes (bytes):%d\n", tamanoMensaje);
  printf ("Intervalo entre Mensajes (milisegundos):%d\n",
        intervaloEntreMensajes);
  printf ("\n");
}                       //fin verificaEntrada


//---------------------------------------------------------------
//Imprime la forma de uso de la aplicacion o help
//---------------------------------------------------------------
void
uso (void)
{
  printf ("\n");
  printf ("Uso: ./GeneraPaquetes -H dirIPLoc -P ptLoc -h dirIPRem -p
ptRem -t tipProt \n");
  printf ("                  -x numFlujos -f fila -b buffer -m Mensajes
-s size\n");
  printf ("                  -d orden -u tiempo -v verbose\n\n");
  printf ("    -H dirIPLoc         - direccion IP del host local\n");
  printf ("    -P ptLoc          - puerto del host local\n");
  printf ("    -h dirIPRem         - direccion IP del host Remoto\n");
  printf ("    -p ptRem          - puerto del host Remoto\n");
  printf ("    -t tipProt        - protocolo de transporte - TCP=1 -
SCTP=2\n");
  printf ("    -x numFlujos        - número de flujos - min 1 - max
300 - SCTP\n");
```

```
  printf ("    -f fila           - fila de Mensajes en cada flujo -
min 1 - max 5 - SCTP\n");
  printf ("    -b buffer         - buffer de envio y recibo (kBytes)
");
  printf ("- min 1 - max 1000 - SCTP\n");
  printf ("    -m Mensajes      - número de Mensajes a enviar - min 1
- max 100\n");
  printf ("    -s size           - tamano da mensaje (bytes) - min 2 -
max 100\n");
  printf ("    -d orden          - 1 - envia los Mensajes de manera
desordenada\n");
  printf ("    -u tiempo          - tiempo entre Mensajes -
milisegundos - patron 50\n");
  printf ("    -v verbose        - verbose 1 o 2\n");
  printf ("\n");
}                       //fin help

//------------------------------------------------------------------
//------------------------------------------------------------------
void
creaAsociacionSCTP (void)
{
  int sock_fd;
  int erro;

  struct sockaddr_in servaddr, meuaddr, direccionIPEnviante;
  struct sctp_event_subscribe evnts;

  //Creacion del socket
  sock_fd = socket (AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);

  if (sock_fd < 0)
    {
      printf ("Fallo creacion del socket SCTP\n");
      exit (0);
    }

  //Configurando las direcciones IP
  bzero (&servaddr, sizeof (servaddr));
  servaddr.sin_family = AF_INET;
  inet_aton (direccionIPRemoto, &servaddr.sin_addr);
  servaddr.sin_port = htons (puertoRemoto);

  bzero (&meuaddr, sizeof (meuaddr));
  meuaddr.sin_family = AF_INET;
  inet_aton (direccionIPLocal, &meuaddr.sin_addr);
  meuaddr.sin_port = htons (puertoLocal);

  // notificaciones de eventos(Solo en SCTP)
  bzero (&evnts, sizeof (evnts));
  evnts.sctp_data_io_event = 1;

  erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_EVENTS,
                &evnts, sizeof (evnts));

  if (erro != 0)
    {
      printf ("Error al configurar la opcion de initmessage en el
socket\n");
      exit (0);
    }
```

```
  int desliga = 1;

  //Desactivando el algoritmo Nagle
  erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_NODELAY,
                &desliga, sizeof (desliga));

  if (erro == (-1))
    {
      printf ("Error al desconectar el algoritmo Nagle en SCTP\n");
      exit (0);
    }

  struct sctp_initmsg initmsg;

  //Configurando el número de flujos
  initmsg.sinit_num_ostreams = flujos;
  initmsg.sinit_max_instreams = flujos;
  initmsg.sinit_max_attempts = 3;
  initmsg.sinit_max_init_timeo = 30;

  erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_INITMSG,
                (char *) &initmsg,
                sizeof (struct sctp_initmsg));

  if (erro != 0)
    {
      printf ("Error al configurar la opcion de initmessage en el
socket\n");
      exit (0);
    }

  //Configurando el tamaño del buffer
  tamanoBuffer = tamanoBuffer * 1000;

  if (tamanoBuffer != 0)
    {
      erro = setsockopt (sock_fd, IPPROTO_SCTP, SO_RCVBUF,
                   (char *) &tamanoBuffer,
                   sizeof (tamanoBuffer));

      if (erro != 0)
      {
        printf ("Error al configurar la opcion de recepcion de buffer
en el socket\n");
        exit (0);
      }

      erro = setsockopt (sock_fd, IPPROTO_SCTP,
                   SO_SNDBUF, (char *) &tamanoBuffer,
                   sizeof (tamanoBuffer));

      if (erro != 0)
      {
        printf ("Error al configurar la opcion de envio de buffer en
el socket\n");
        exit (0);
      }
    }                    //fin tamanobuffer!=0

  //Creacion del bind
```

```
  erro = bind (sock_fd, (struct sockaddr *) &meuaddr,
               sizeof (meuaddr));

  if (erro != 0)
    {
      printf ("Error en la creacion de la funcion bind\n");
      exit (0);
    }

  //Esta parte es para contar el tiempo del establecimiento de la
asociacion SCTP
  //es decir, la latencia.
  struct sctp_sndrcvinfo sri;
  struct tipao mensaje;
  int cantidadEnviada;
  int cantidadRecibida;
  int msg_flags;

  bzero (&sri, sizeof (sri));
  sri.sinfo_stream = 0;

  int out_sz;

  socklen_t len;
  len = sizeof (struct sockaddr_in);

  char *sendline;
  sendline = malloc (25);

  bzero (mensaje.sendline, sizeof (100));
  bzero (sendline, sizeof (25));
  sprintf (mensaje.sendline, "Conectando");
  out_sz = strlen (mensaje.sendline) + sizeof (int);
  mensaje.id = 1000;

  cantidadEnviada =
        sctp_sendmsg (sock_fd, "Conectando", out_sz,
                  (struct sockaddr *) &servaddr,
                  sizeof (struct sockaddr), 0,
                  mensajeDesordenado, sri.sinfo_stream, 0,
                  0);

  if (cantidadEnviada == (-1))
    {
      printf ("Falla al enviar mensaje\n");
    }

  cantidadRecibida =
        sctp_recvmsg (sock_fd, sendline, 25,
                  (struct sockaddr *) &direccionIPEnviante,
                  &len, &sri, &msg_flags);

  if (cantidadRecibida == (-1))
    {
      printf ("Falla al recibir mensaje\n");
    }

  //Fin     del establecimiento de la asociacion SCTP

  struct parametroT1 valores1;
  struct parametroT1 valores2;
```

```
  //Llenando las estruturas de los parametros de los threads
  valores1.sock_fd = sock_fd;
  valores1.direccionIP = direccionIPEnviante;
  valores2.sock_fd = sock_fd;
  valores2.direccionIP = servaddr;

  pthread_t p_thread_envia;
  pthread_t p_thread_recibe;

  //Abre un thread que genera y envia mensajes
  pthread_create (&p_thread_envia, NULL,
              (void *) enviaMensajesSCTP, &valores2);

  //Abre un thread que recibe mensajes
  pthread_create (&p_thread_recibe, NULL,
              (void *) recibeMensajesSCTP, &valores1);

  //Guarda los términos de ambos threads
  pthread_join (p_thread_envia, NULL);
  pthread_join (p_thread_recibe, NULL);

  //Calcula y muestra resultados
  calculaTiempo ();
  geraScriptGnuPlot ();

  return;
}                       //fin de creacion de la asociacion SCTP

//------------------------------------------------------------------
//------------------------------------------------------------------
void
geraScriptGnuPlot (void)
{
  printf ("\n");
  printf ("#-----------------------------------------------------------
--------\n");
  printf ("#SCRIPT PARA GNUPLOT\n");
  printf ("#-----------------------------------------------------------
--------\n");

  printf ("\n");
  printf ("#-----------------------------------------------------------
--------\n");
  printf ("#MANUAL DE LECTURA\n");
  printf ("#-----------------------------------------------------------
--------\n");

  printf ("\n");
  printf ("#-latencia media: la media aritmética de todas las
latencias individuales\n");
  printf ("#-velocidad media:    la media aritmética de todas las
velocidades individuales\n");
  printf ("\n");
  printf ("#-latencia General: la suma de todas las latencias, sin
considerar los intervalos\n");
  printf ("#-velocidad General:    la suma de todas las velocidades,
sin considerar los intervalos\n");
  printf ("\n");
  printf ("#-latencia total: desde el primer mensaje enviado hasta el
ultimo recibido\n");
```

```
  printf ("#-velocidad total:    desde el primer mensaje enviado hasta
el ultimo recibido\n");

  printf ("\n");
  printf ("#-------------------------------------------------------------
--------\n");
  printf ("#INFORMACIOPN GENERAL\n");
  printf ("#-------------------------------------------------------------
--------\n");

  printf ("\n");
  printf ("#Direccion IP del host local:%s\n", direccionIPLocal);
  printf ("#Puerto del host local:%d\n", puertoLocal);
  printf ("#Direccion IP del host Remoto:%s\n", direccionIPRemoto);
  printf ("#Puerto del host Remoto:%d\n", puertoRemoto);

  if ((tipoDeProtocolo == 1)) //Caso TCP
    printf ("#Protocolo de transporte: TCP\n");
  else
    {                   //Caso SCTP
      printf ("#Protocolo de transporte: STCP\n");
      printf ("#Número de flujos:%d\n", flujos);
      printf ("#Fila de Mensajes en cada flujo:%d\n", fila);

      if (tamanoBuffer == 0)
      printf ("#Buffer de envio y recepcion configurado como
patron\n");
      else
      printf ("#Buffer de envio y recepcion (kBytes):%d\n",
            tamanoBuffer);
    }                   //Fin caso SCTP

  printf ("#Número de Mensajes a enviar:%d\n",
        numeroMensajesEnviar);
  printf ("#tamano de Mensajes (bytes):%d\n", tamanoMensaje);
  printf ("\n");
  printf ("#Round Trip Time:\n");
  printf ("#Tasa de perdida de paquetes:\n");
      printf ("\n");

  printf ("Latencia Media      %0.6f \n", latenciaMedia);
  printf ("Velocidad Media     %0.6f \n", velocidadMedia);
  printf ("Latencia General    %0.6f \n", latenciaGeneral);
  printf ("Velocidad General   %0.6f \n", velocidadGeneral);
  printf ("Latencia Total      %0.6f \n", latenciaTotal);
  printf ("Velocidad Total     %0.6f \n", velocidadTotal);
      printf ("\n");
  printf ("#--------------------------------------\n");
  printf ("#IdMsg:       Lat(s):     Vel(b/s):    \n");
  printf ("#--------------------------------------\n");

  printf ("\n");

  //printf ("#-------------------------------------------------------------
----------\n");
  //printf ("#Lat General(s) : VelGeneral(b/s) : Lat Total(s) :
VelTotal(b/s)\n");
  //printf ("#-------------------------------------------------------------
----------\n");
  //printf ("\n");
```

```
  int i;

  for (i = 0; i < numeroMensajesEnviar; i++)
    {
      printf ("%d         %0.6f    %0.6f\n", i, latenciaIndividual[i],
velocidadIndividual[i], latenciaMedia, velocidadMedia, latenciaGeneral,
velocidadGeneral, latenciaTotal, velocidadTotal);
    }



  fprintf (archivo, "%0.6f ", latenciaMedia);
  fprintf (archivo2, "%0.6f ", velocidadMedia);
}                        //fin de generaScript


//-------------------------------------------------------------------
//-------------------------------------------------------------------
void
calculaTiempo (void)
{
  int i, j;
  double resultadotiempoSec;
  double resultadotiempoMicro;
  double resultadoFinal;
  double velocidad;


  printf ("\n");
  printf ("-----------------------------------------------------\n");
  printf ("CÁLCULO DE LATENCIA Y VELOCIDAD POR MENSAJE ENVIADO\n");
  printf ("-----------------------------------------------------\n");

  for (i = 0; i < numeroMensajesEnviar; i++)
    {
      for (j = 0; j < numeroMensajesEnviar; j++)
      {
        if (i == matriz[j].id)
          {
            resultadotiempoSec =
                  (matriz[j].cronometroVuelta.tv_sec) -
                  (matriz[j].cronometroIda.tv_sec);

            resultadotiempoMicro =
                  (double) (matriz[j].cronometroVuelta.
                        tv_usec) -
                  (double) (matriz[j].cronometroIda.tv_usec);

            resultadoFinal =
                  resultadotiempoSec +
                  ((double) resultadotiempoMicro /
                   (double) 1000000);

            printf ("\n");
            printf ("Latencia por mensaje %d = %0.6f segundos\n", i,
resultadoFinal);

            latenciaIndividual[i] = resultadoFinal;

            velocidad = (double) (matriz[j].tamano) /
                  (double) resultadoFinal;
```

```
            printf ("Velocidad por mensaje %d = %0.6f bytes/seg\n",
                  i, velocidad);

            printf ("Cantidad de datos enviado: %d\n",
                  matriz[j].tamano);

            velocidadIndividual[i] = velocidad;

            j = numeroMensajesEnviar;
          }             //fin if
      }                 //fin for 2
    }                   //fin for 1

  //Cálculo da latencia y taza de transferencia globales
  resultadotiempoSec =
        (tiempoGlobalFinal.tv_sec) - (tiempoGlobalInicial.tv_sec);

  resultadotiempoMicro =
        (double) (tiempoGlobalFinal.tv_usec) -
        (double) (tiempoGlobalInicial.tv_usec);

  resultadoFinal =
        resultadotiempoSec +
        ((double) resultadotiempoMicro / (double) 1000000);

  printf ("\n");
  printf ("-----------------------------------------------------\n");
  printf ("\n");
  printf ("Latencia Total Final = %0.6f segundos\n",
        resultadoFinal);

  latenciaTotal = resultadoFinal;

  velocidad = (double) (cantidadEnviadaGlobal) /
        (double) resultadoFinal;

  printf ("Cantidad de datos enviado: %d bytes\n",
        cantidadEnviadaGlobal);

  printf ("Velocidad Total Final = %0.6f bytes/seg\n\n", velocidad);

  velocidadTotal = velocidad;

  for (i = 0; i < numeroMensajesEnviar; i++)
    {
      latenciaGeneral = latenciaIndividual[i] + latenciaGeneral;

      velocidadGeneral = velocidadIndividual[i] + velocidadGeneral;
    }

  printf ("Latencia General = %0.6f seg\n\n", latenciaGeneral);

  latenciaMedia =
        (double) latenciaGeneral / (double) numeroMensajesEnviar;

  printf ("Latencia Media = %0.6f seg\n\n", latenciaMedia);

  printf ("Velocidad General = %0.6f bytes/seg\n\n",
velocidadGeneral);
```

```
  velocidadMedia =
        (double) velocidadGeneral / (double) numeroMensajesEnviar;

  printf ("Velocidad Media = %0.6f bytes/seg\n\n", velocidadMedia);

}                       //fin de calculaTiempo


//-------------------------------------------------------------
//Funcion que genera y envia los mensajes a usando SCTP
//-------------------------------------------------------------
void *
enviaMensajesSCTP (struct parametroT1 *variables)
{
  printf ("Se ejecuta la funcion de enviar mensajes\n");

  struct tipao mensaje;

  int sock_fd = variables->sock_fd;
  struct sockaddr_in servaddr = variables->direccionIP;

  int loop;
  int indiceFlujos = 0;
  int out_sz;
  int indiceFila = fila;
  int indiceMensaje = 0;
  int cantidadEnviada;
  struct sctp_sndrcvinfo sri;

  //Esta estructura es usada para ver en que flujo vino el mensaje

  bzero (&sri, sizeof (sri));
  sri.sinfo_stream = 0;

  printf ("\n");
  printf ("-----------------------------------------------------\n");
  printf ("DADOS SOBRE EL ENVIO DE MENSAJES\n");
  printf ("-----------------------------------------------------\n");

  //loop de envio de Mensajes
  for (loop = 0; loop < numeroMensajesEnviar; loop++)
    {
      sri.sinfo_stream = indiceFlujos;
      bzero (mensaje.sendline, sizeof (100));
      snprintf (mensaje.sendline, tamanoMensaje,
            gnutella[loop % 5]);
      out_sz = strlen (mensaje.sendline) + 1;
      mensaje.id = loop;

      printf ("\n");
      printf ("ENVIO\n");
      printf ("ID Mensaje:%d\n", mensaje.id);
      printf ("Mensaje:%s\n", mensaje.sendline);
      printf ("Numero Flujo:%d\n", indiceFlujos);
      printf ("Indice Fila:%d\n", indiceFila);
      printf ("Numero Loop:%d\n", loop);

      //Tiempo entre los mensajes en milisegundos
      usleep (1000 * intervaloEntreMensajes);

      //Início del calculo del tiempo de ida  del mensaje
```

```
      matriz[loop].id = mensaje.id;

      gettimeofday (&matriz[loop].cronometroIda, NULL);

      //Enviando un mensaje
      cantidadEnviada =
            sctp_sendmsg (sock_fd, &mensaje, out_sz,
                      (struct sockaddr *) &servaddr,
                      sizeof (struct sockaddr), 0,
                      mensajeDesordenado,
                      sri.sinfo_stream, 0, 0);

      if (cantidadEnviada == (-1))
      {
        printf ("Falla al enviar mensaje\n");
      }

      //Para cálculo del total de bytes enviados
      cantidadEnviadaGlobal =
            cantidadEnviadaGlobal + cantidadEnviada;

      printf ("Bytes enviados:%d\n", cantidadEnviada);
      matriz[loop].tamano = cantidadEnviada;

      //Control de la fila en los flujos
      if (indiceFila == 1)
      {
        if (indiceFlujos == (flujos - 1))
          {
            indiceFlujos = 0;
          }
        else
          {
            indiceFlujos++;
          }
        indiceFila = fila;
      }
      else
      indiceFila--;

      if (indiceMensaje == 4)
      indiceMensaje = 0;
      else
      indiceMensaje++;
    }                   //fin del loop for de envio de mensajes

  return 0;
}                       //fin de envioMensajes


//------------------------------------------------------------------
//Funcion que recibe los mensajes usando SCTP
//------------------------------------------------------------------
void *
recibeMensajesSCTP (struct parametroT1 *variables)
{
  printf ("Se ejecuta la funcion de recibir mensajes\n");

  //Utilizado para recibir los mensajes
  struct tipao mensaje;
```

```
//Llenando las variables locales con los datos del parametro
int sock_fd = variables->sock_fd;
struct sockaddr_in direccionIPEnviante = variables->direccionIP;

int loop;
int cantidadRecibida;
struct sctp_sndrcvinfo sri;

char *sendline;
sendline = malloc (105);

socklen_t len;
len = sizeof (struct sockaddr_in);
int msg_flags;

//Estrutura utilizada para saber en flujo vino el mensaje
bzero (&sri, sizeof (sri));
sri.sinfo_stream = 0;

printf ("\n");
printf ("-----------------------------------------------------\n");
printf ("DADOS SOBRE A RECEPCION DE MENSAJES\n");
printf ("-----------------------------------------------------\n");

//loop de recepcion de mensajes
for (loop = 0; loop < numeroMensajesEnviar; loop++)
  {
    bzero (sendline, 105);

    //Recibiendo un mensaje
    cantidadRecibida =
          sctp_recvmsg (sock_fd, sendline, 105,
                    (struct sockaddr *) &direccionIPEnviante,
                    &len, &sri, &msg_flags);

    //Armacenando el momento de llegada del mensaje
    gettimeofday (&matriz[loop].cronometroVuelta, NULL);

    usleep (1);

    bzero (mensaje.sendline, 100);
    memcpy (&mensaje, sendline, cantidadRecibida);

    matriz[loop].id = mensaje.id;

    if (cantidadRecibida == (-1))
    {
      printf ("Falla al recibir mensaje\n");
      loop = loop - 1;
      continue;
    }

    if (loop == numeroMensajesEnviar - 1)
    {
      tiempoGlobalInicial = matriz[0].cronometroIda;
      tiempoGlobalFinal = matriz[loop].cronometroVuelta;
    }

    //Impresion de datos de la recepcion
    printf ("\n");
    printf ("RECIBI\n");
```

```
      printf ("ID=%d - Mensaje:%s\n",
            mensaje.id, mensaje.sendline);
      printf ("Numero Flujo:%d\n", sri.sinfo_stream);
      printf ("Cantidad Recibida:%d\n", cantidadRecibida);

    }                   //fin loop de recepcion de mensajes

  free (sendline);
  close (sock_fd);

  return 0;
}                       //fin recibe mensajes


//---------------------------------------------------------------
//Funcion para enviar N bytes usando TCP
//---------------------------------------------------------------
int
writeN (int fd, const void *buffer, int n)
{
  int nRestante;
  int nEscritos;
  const char *ptr;

  ptr = buffer;
  nRestante = n;

  while (nRestante > 0)
    {
      if ((nEscritos = write (fd, ptr, nRestante)) <= 0)
      {
        if (nEscritos < 0 && errno == EINTR)
          nEscritos = 0;
        else
          return (-1);
      }

      nRestante -= nEscritos;
      ptr += nEscritos;
    }

  return n;

}                       //fin writeN


//---------------------------------------------------------------
//Funcion para recibir N bytes usando TCP
//---------------------------------------------------------------
int
readN (int fd, void *buffer, int n)
{
  int nRestante;
  int nLeidos;
  char *ptr;

  ptr = buffer;
  nRestante = n;

  while (nRestante > 0)
    {
```

```
      if ((nLeidos = read (fd, ptr, nRestante)) < 0)
      {
        if (errno == EINTR)
          nLeidos = 0;
        else
          return (-1);
      }
      else if (nLeidos == 0)
      break;

      nRestante -= nLeidos;
      ptr += nLeidos;
    }

  return (n - nRestante);

}                       //fin readN

//-------------------------------------------------------------------
//Funcion para crear una conexion TCP con EchoServer
//-------------------------------------------------------------------
void
creaConexionTCP (void)
{
  int sock_fd;
  int erro;

  struct sockaddr_in servaddr, meuaddr;

  //Abre un socket
  sock_fd = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);

  if (sock_fd < 0)
    {
      printf ("Creacion del socket fallo\n");
      exit (0);
    }

  int desliga = 1;

  //Desliga o Nagle
  erro = setsockopt (sock_fd, IPPROTO_TCP, TCP_NODELAY,
                &desliga, sizeof (desliga));

  if (erro == (-1))
    {
      printf ("Error al desconectar el algoritmo Nagle en TCP\n");

    }

  //Definiendo las Direcciones IP
  bzero (&servaddr, sizeof (servaddr));
  servaddr.sin_family = AF_INET;
  inet_aton (direccionIPRemoto, &servaddr.sin_addr);
  servaddr.sin_port = htons (puertoRemoto);

  bzero (&meuaddr, sizeof (meuaddr));
  meuaddr.sin_family = AF_INET;
  inet_aton (direccionIPLocal, &meuaddr.sin_addr);
  meuaddr.sin_port = htons (puertoLocal);
```

```
  //Intentando conectar al EchoServer
  erro = connect (sock_fd, (struct sockaddr *) &servaddr,
              sizeof (servaddr));

  if (erro < 0)
    {
      printf ("Error al efectuar la conexion\n");
      exit (0);
    }

  struct parametroT1 valores1;
  struct parametroT1 valores2;

  valores1.sock_fd = sock_fd;
  valores2.sock_fd = sock_fd;

  pthread_t p_thread_envia;
  pthread_t p_thread_recibe;

  //Crea un thread que envia mensajes
  pthread_create (&p_thread_envia, NULL,
              (void *) enviaMensajesTCP, &valores2);

  //Crea un thread que recibe mensajes
  pthread_create (&p_thread_recibe, NULL,
              (void *) recibeMensajesTCP, &valores1);

  //Guarda los dos threads terminarem
  pthread_join (p_thread_envia, NULL);
  pthread_join (p_thread_recibe, NULL);

  //Calcula resultados  y los muestra
  calculaTiempo ();
  geraScriptGnuPlot ();

  return;

}                       //fin conexion TCP


//-------------------------------------------------------------
//Funcion para envio de mensajes usando TCP
//-------------------------------------------------------------
void *
enviaMensajesTCP (struct parametroT1 *variables)
{
  struct tipao mensaje;

  int sock_fd = variables->sock_fd;
  int out_sz;
  int loop;
  int cantidadEnviada;

  printf ("\n");
  printf ("-----------------------------------------------------\n");
  printf ("DADOS DE ENVIO DE MENSAJES\n");
  printf ("-----------------------------------------------------\n");

  //loop de envio de mensajes
  for (loop = 0; loop <= numeroMensajesEnviar; loop++)
    {
```

```
      bzero (mensaje.sendline, sizeof (100));
      snprintf (mensaje.sendline, tamanoMensaje,
            gnutella[loop % 5]);
      out_sz = strlen (mensaje.sendline) + 1;
      mensaje.id = loop;

      //Impresion de datos  sobre el envio
      printf ("\n");
      printf ("ENVIO\n");
      printf ("ID Mensaje:%d\n", mensaje.id);
      printf ("Mensaje:%s\n", mensaje.sendline);
      printf ("Número Loop:%d\n", loop);

      //Início do cálculo do tiempo de ida da mensaje
      matriz[loop].id = mensaje.id;
      gettimeofday (&matriz[loop].cronometroIda, NULL);

      usleep (1000 * intervaloEntreMensajes);

      cantidadEnviada = writeN (sock_fd, &mensaje, out_sz);

      if (cantidadEnviada == (-1))
      {
        printf ("Falla al enviar el mensaje\n");
      }

      //Para cálculo del total de bytes enviados
      cantidadEnviadaGlobal =
            cantidadEnviadaGlobal + cantidadEnviada;

      printf ("Bytes enviados:%d\n", cantidadEnviada);
      matriz[loop].tamano = cantidadEnviada;

    }                   //fin del loop for de envio de mensajes

  return 0;

}                       //fin de envia mensajes


//-------------------------------------------------------------
//Funcion que recibe mensajes usando TCP
//-------------------------------------------------------------
void *
recibeMensajesTCP (struct parametroT1 *variables)
{
  struct tipao mensaje;

  int sock_fd = variables->sock_fd;
  int loop;
  socklen_t len;
  int cantidadRecibida;

  len = sizeof (struct sockaddr_in);

  char *sendline;
  sendline = malloc (105);

  printf ("\n");
  printf ("-----------------------------------------------------\n");
  printf ("DADOS DE LA RECEPCION DE MENSAJES\n");
```

```
  printf ("-----------------------------------------------------\n");

  //loop de recepcion de mensajes
  for (loop = 0; loop <= numeroMensajesEnviar; loop++)
    {
      bzero (sendline, 105);
      cantidadRecibida =
            readN (sock_fd, sendline, matriz[loop].tamano);

      gettimeofday (&matriz[loop].cronometroVuelta, NULL);

      usleep (1);

      if (cantidadRecibida == (-1))
      {
        printf ("Falla al recibir mensaje\n");
        loop = loop - 1;
        continue;
      }

      if (cantidadRecibida == (0))
      {
        loop = loop - 1;
        continue;
      }

      bzero (mensaje.sendline, 100);
      memcpy (&mensaje, sendline, cantidadRecibida);

      if (loop == numeroMensajesEnviar - 1)
      {
        tiempoGlobalInicial = matriz[0].cronometroIda;
        tiempoGlobalFinal = matriz[loop].cronometroVuelta;
      }

      //Impresion de datos de la recepcion
      printf ("\n");
      printf ("RECEBI\n");
      printf ("ID=%d - mensaje:%s\n", mensaje.id,
            mensaje.sendline);
      printf ("Cantidad Recibida: %d\n", cantidadRecibida);
      printf ("Cantidad que era para recibir: %d\n",
            matriz[loop].tamano);

    }                   //fin loop de rececion de mensajes

  free (sendline);
  close (sock_fd);

  return 0;
}                       //fin de recibe mensajes

//---------------------------------------------------------------
//Fin de GeneraPaquetes.c
//---------------------------------------------------------------
```

EchoServer.c

```
/**************************************************************
         EchoServer.c - Echo Server
Traduccion del codigo de Gustavo Do Carmo

**************************************************************/
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netinet/sctp.h>
#include <netinet/tcp.h>
#include <errno.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

//--------------------------------------------------------------
//Protótipos
//--------------------------------------------------------------
void uso (void);
void parser (int argc, char **argv);
void verificaEntrada (void);
void echoSCTP (void);
void echoTCP (void);
int writeN (int fd, const void *buffer, int n);

//--------------------------------------------------------------
//Variables globales
//--------------------------------------------------------------
int tamanoBuffer = 0;
int puertoLocal = 0;
char *direccionLocal = NULL;
int mensajeDesordenado = 0;
int tipoProtocolo = 1;

//Estrutura usada para recibir mensajes
struct tipao
{
  int id;
  char sendline[100];
} mensagem;

//--------------------------------------------------------------
//Programa principal
//--------------------------------------------------------------
int
main (int argc, char **argv)

  parser (argc, argv);
  verificaEntrada ();

  if (tipoProtocolo == 1)
    echoTCP ();
  else if (tipoProtocolo == 2)
    echoSCTP ();
  else
```

```
    uso ();

  return 0;
}


//------------------------------------------------------------------
//EchoServer usando TCP
//------------------------------------------------------------------
void
echoTCP ()
{
  int sock_fd, sock_nuevo;
  int erro;

  struct sockaddr_in servaddr, cliaddr;

  sock_fd = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);

  if (sock_fd < 0)
    {
      printf ("Falla de creacion del socket\n");
      exit (0);
    }

  bzero (&servaddr, sizeof (servaddr));
  servaddr.sin_family = AF_INET;
  inet_aton (direccionLocal, &servaddr.sin_addr);
  servaddr.sin_port = htons (puertoLocal);

  int desconecta = 1;

  erro = setsockopt (sock_fd, IPPROTO_TCP, TCP_NODELAY,
                &desconecta, sizeof (desconecta));

  if (erro == (-1))
    {
      printf ("Error en el socket TCP\n");
    }

  erro = bind (sock_fd, (struct sockaddr *) &servaddr,
            sizeof (servaddr));

  if (erro != 0)
    {
      printf ("Error al crear la funcion bind\n");
      exit (0);
    }

  erro = listen (sock_fd, 1);

  if (erro != 0)
    {
      printf ("Error en la funcion listen\n");
      exit (0);
    }

  socklen_t len;

  len = sizeof (struct sockaddr_in);
```

```
  while (1)
    {
      sock_nuevo =
            accept (sock_fd, (struct sockaddr *) &cliaddr, &len);

      if (sock_nuevo < 0)
      {
        printf ("Error en la funcion accept\n");
        exit (0);
      }

      char *readbuf;
      int cantidadEnviada;
      int cantidadRecibida;

      readbuf = malloc (100);

      printf ("-----------------------------------------------\n");
      printf ("DATOS DE LA RECEPCION\n");
      printf ("-----------------------------------------------\n");

      while (1)
      {
        bzero (readbuf, 100);

        //Recibiendo segmento del cliente
        cantidadRecibida = read (sock_nuevo, readbuf, 100);

        if (cantidadRecibida == (-1))
          {
            printf ("Falla al recibir los bytes\n");
          }

        //El cliente termina la comunicacion
        if (cantidadRecibida == 0)
          {
            printf ("\n Cliente termina la comunicacion \n\n");
            close (sock_nuevo);
            free (readbuf);
            break;
          }

        printf ("\n");
        printf ("Número de bytes recibidos:%d\n",
              cantidadRecibida);
        printf ("Direccion IP remota:%s\n",
              inet_ntoa (cliaddr.sin_addr));

        //Enviando un segmento de regreso para el cliente
        cantidadEnviada =
              writeN (sock_nuevo, readbuf, cantidadRecibida);

        if (cantidadEnviada == (-1))
          {
            printf ("Falla al enviar los bytes\n");
          }
      }                 //fin while 1
    }

  return;
}                       //Fin EchoServer TCP
```

```
//------------------------------------------------------------------
//EchoServer usando SCTP
//------------------------------------------------------------------
void
echoSCTP ()
{

  int sock_fd;
  int erro;

  struct sockaddr_in servaddr, cliaddr;
  struct sctp_sndrcvinfo sri;
  struct sctp_event_subscribe evnts;

  sock_fd = socket (AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);

  if (sock_fd < 0)
    {
      printf ("Falla de creacion del socket\n");
      exit (0);
    }

  bzero (&servaddr, sizeof (servaddr));
  servaddr.sin_family = AF_INET;
  inet_aton (direccionLocal, &servaddr.sin_addr);
  servaddr.sin_port = htons (puertoLocal);

  bzero (&evnts, sizeof (evnts));
  evnts.sctp_data_io_event = 1;

  erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_EVENTS,
                &evnts, sizeof (evnts));

  if (erro != 0)
    {
      printf ("Error al configurar la opcion de eventos en SCTP\n");
      exit (0);
    }

  struct sctp_initmsg initmsg;

  initmsg.sinit_num_ostreams = 30;
  initmsg.sinit_max_instreams = 30;
  initmsg.sinit_max_attempts = 3;
  initmsg.sinit_max_init_timeo = 30;

  int desconecta = 1;

  erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_NODELAY,
                &desconecta, sizeof (desconecta));

  if (erro == (-1))
    {
      printf ("Error en el socket SCTP\n");
      exit (0);
    }

  erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_INITMSG,
                (char *) &initmsg,
```

```
                sizeof (struct sctp_initmsg));

  if (erro != 0)
    {
      printf ("Error al configurar la opcion initmsg en SCTP\n");
      exit (0);
    }

  tamanoBuffer = tamanoBuffer * 1000;

  if (tamanoBuffer != 0)
    {
      erro = setsockopt (sock_fd, IPPROTO_SCTP, SO_RCVBUF,
                  (char *) &tamanoBuffer,
                  sizeof (tamanoBuffer));

      if (erro != 0)
      {
        printf ("Error al configurar en el socket la opcion de recibir
buffer\n");
        exit (0);
      }

      erro = setsockopt (sock_fd, IPPROTO_SCTP, SO_SNDBUF,
                  (char *) &tamanoBuffer,
                  sizeof (tamanoBuffer));

      if (erro != 0)
      {
        printf ("Error al configurar en el socket la opcion de envio
de buffer\n");

        exit (0);
      }
    }                   //fin tamano del buffer

  erro = bind (sock_fd, (struct sockaddr *) &servaddr,
              sizeof (servaddr));

  if (erro != 0)
    {
        printf ("Error al crear la funcion bind\n");
      exit (0);
    }

  erro = listen (sock_fd, 1);

  if (erro != 0)
    {
      printf ("Error en la funcion listen\n");
      exit (0);
    }

  socklen_t len;
  int msg_flags;
  char *readbuf;
  int cantidadEnviada;
  int cantidadRecibida;

  len = sizeof (struct sockaddr_in);
  readbuf = malloc (105);
```

```
  while (1)
    {
      bzero (readbuf, 105);

      //Recibiendo mensajes del cliente
      cantidadRecibida = sctp_recvmsg (sock_fd, readbuf, 105,
                               (struct sockaddr *)
                               &cliaddr, &len, &sri,
                               &msg_flags);

      if (cantidadRecibida == (-1))
      {
        printf ("Falla al recibir los mensajes\n");
      }

      bzero (mensagem.sendline, 100);
      memcpy (&mensagem, readbuf, cantidadRecibida);

      //DATOS DE LA COMUNICACION
      printf ("\n");
      printf ("Id:%d\n", mensagem.id);
      printf ("buffer:%s\n", mensagem.sendline);
      printf ("Numero de stream:%d\n", sri.sinfo_stream);
      printf ("Numero de bytes recibidos:%d\n",
            cantidadRecibida);
      printf ("DireccionIP remota:%s\n",
            inet_ntoa (cliaddr.sin_addr));

      //enviando un mensaje de regreso para el cliente
      cantidadEnviada =
            sctp_sendmsg (sock_fd, readbuf, cantidadRecibida,
                      (struct sockaddr *) &cliaddr,
                      sizeof (struct sockaddr),
                      sri.sinfo_ppid, mensajeDesordenado,
                      sri.sinfo_stream, 0, 0);

      if (cantidadEnviada == (-1))
      {
        printf ("Falla al enviar los mensajes\n");
      }
    }                   //fin while 1

  free (readbuf);
  return;
}                       //fin EchoServer SCTP


//---------------------------------------------------------------
//Revisando los datos de la linea de comandos
//---------------------------------------------------------------
void
verificaEntrada (void)
{
  if ((direccionLocal == NULL) ||
      (inet_aton (direccionLocal, NULL) == 0))
    {
      printf ("\nFalto colocar la Direccion IP local\n");
      uso ();
      exit (0);
    }
```

```
  if (puertoLocal == 0)
    {
      printf ("\nFalto colocar el puerto local\n");
      uso ();
      exit (0);
    }

  printf ("\n");
  printf ("Direccion IP del host local:%s\n", direccionLocal);
  printf ("Puerto del host local:%d\n", puertoLocal);
  printf ("Protocolo de transporte = ");

  if (tipoProtocolo == 1)
    printf ("TCP\n");
  else
    printf ("SCTP\n");

  if (tamanoBuffer == 0)
    printf ("Buffer \n");
  else
    printf ("Buffer de envio y recibimiento (Kbytes):%d\n",
          tamanoBuffer);

  if (mensajeDesordenado == 0)
    printf ("Entrega de mensajes: ordenada\n");
  else
    printf ("Entrega de mensajes: desordenada\n");

  printf ("\n");

}                       //fin Revisando


//---------------------------------------------------------------------
//Adquiriendo los datos de la linea de comandos
//---------------------------------------------------------------------
void
parser (int argc, char **argv)
{
  int getoptC;
  int bufferAux;
  int desorden;
  int protocolo;

  while (argc)
    {
      getoptC = getopt (argc, argv, "H:P:t:b:d:");

      if (getoptC < 0)
      break;

      switch (getoptC)
      {
      case 'H':
        direccionLocal = optarg;
        break;

      case 'P':
        puertoLocal = atoi (optarg);
        break;
```

```
      case 't':
        protocolo = atoi (optarg);
        if ((protocolo == 1) || (protocolo == 2))
          tipoProtocolo = protocolo;
        break;

      case 'b':
        bufferAux = atoi (optarg);
        if ((bufferAux > 0) && (bufferAux < 1001))
          tamanoBuffer = bufferAux;
        break;

      case 'd':
        desorden = atoi (optarg);
        if (desorden == 1)
          mensajeDesordenado = MSG_UNORDERED;
        break;

      default:
        break;
      }                 //fin switch
    }             //fin while argc
}                       //fin de parser

//---------------------------------------------------------------
//Imprime la forma de uso de la aplicacion o help
//---------------------------------------------------------------
void
uso (void)
{
  printf ("\nUso:\n");
  printf ("./EchoServer -H dirIPLoc -P ptLoc -t tipProt -b buffer -d
orden\n");
  printf (" -H dirIPLoc       - direccionIP del host local\n");
  printf (" -P ptLoc          - puerto del host local\n");
  printf (" -t tipProt        - protocolo de transporte - TCP=1 -
SCTP=2\n");
  printf (" -b buffer         - buffer de envio y recepcion(Kbytes)
min 1 max 1000 ");
  printf (" -d orden          - 1 - envia los mensajes de forma
ordenada, usar solo en SCTP\n");
  printf ("\n");
}                       //fin forma de uso

//---------------------------------------------------------------
//Funcion para forzar la escritura de todos los bytes TCP
//---------------------------------------------------------------
int
writeN (int fd, const void *buffer, int n)
{
  int nRestante;
  int nEscritos;
  const char *ptr;

  ptr = buffer;
  nRestante = n;

  while (nRestante > 0)
    {
      if ((nEscritos = write (fd, ptr, nRestante)) <= 0)
```

```
      {
        if (nEscritos < 0 && errno == EINTR)
          nEscritos = 0;
        else
          return (-1);
      }

      nRestante -= nEscritos;
      ptr += nEscritos;
    }

  return n;
}                       //fin de writeN

//-------------------------------------------------------------------
//Fin de EchoServer.c
//-------------------------------------------------------------------
```