UNIVERSIDADE ESTADUAL DE CAMPINAS

LABORATÓRIO DE REDES - MC823

Tarefa 3 - UDP

Miguel Francisco Alves de Mattos Gaiowski

RA 076116

Guillaume Massé

RA 107888

Prof. Paulo Lício de Geus

Campinas, 25 de outubro de 2010

Sumário

1 Introdução

Enquanto as outras tarefas tratavam de conexões TCP, esta trata de UDP. Faremos um servidor de eco, e um cliente para interagir com ele. Além disso, usaremos diferentes métodos para utilizar UDP com a linguagem C. Isto é, usaremos as funções send(), recv(), sendto(), recvfrom() e connect().

2 Objetivos

Implementar um cliente e um servidor de eco usando datagramas. Verificar a falta de confiabilidade de UDP. Testar as diferentes combinações de métodos para a implementação.

3 Desenvolvimento

Implementamos um cliente em UDP. Basicamente ele lê linhas da entrada padrão e as transmite para o servidor. Após cada linha ele espera um resposta do servidor para enviar a próxima.

O servidor recebe as linhas e as envia para o cliente.

Vale notar que fizemos duas implementações diferentes para o cliente e para o servidor. Uma delas usa o par sendto e recvfrom. A outra usa send e recv após um connect. Sabemos que o termo connect não se aplica a UDP, sendo uma nomenclatura infeliz da biblioteca de C. Ainda assim, ela simula uma conexão após o primeiro datagrama ser recebido e permite que se possa continuar recebendo sem falar de quem, basta fazer uma conexão virtual antes.

No servidor, caso esteja recebendo dados de um cliente e passe mais de um segundo sem receber nada, um sinal de alarme é emitido e o servidor trata imprimindo as estatísticas obtidas até o momento e reiniciando a contagem. Isso simula que um novo cliente será o próximo a enviar dados. Isso pode acontecer quando um cliente deixa de enviar um pacote final com 0 bytes indicando fim de troca de dados entre eles.

4 Dificuldades

Tivemos algumas dificuldades neste laboratório para fazer o tratamento de sinais de alarme, já que nunca tínhamos feito algo parecido. Google foi nosso parceiro retornando exemplos de uso.

Tivemos problemas também usando um cliente com sendto/recvfrom e um servidor com connect ao mesmo tempo.

5 Experimentos

Testamos os programas em máquinas do IC-3. O tempo de transmissão do arquivo /etc/services variou conforme os diferente métodos usados. Já que o tempo foi muito pequeno, geramos um arquivo 10 vezes maior, fazendo concatenando o /etc/services nele mesmo. O arquivo que geramos tinha 106345 linhas.

Usando as funções sendto()/recvfrom() tanto no cliente quanto no servidor, obtivemos uma medição

de 59.356 segundos para envio do arquivo /etc/services entre duas máquinas do IC-3.

Com o cliente usando connect() e depois send()/recv(), o tempo foi de 58.812 segundos. Já com os papéis trocados, ou seja, servidor fazendo connect(), observamos um tempo de 59.311 segundos. Uma mudança praticamente desprezavel.

Fizemos um flood.c para testar a perda de pacotes. Deixamos vários processos fazendo cálculos inúteis na mesma máquina do servidor. No cliente, retiramos a espera pela resposta antes de envio da próxima linha. Além disso, omitimos todo tipo de output, para que enviasse todas as linhas o mais rápido possível.

O servidor não conseguiu receber todos os pacotes tão rápido quanto o cliente enviava e, já que não há reenvio, acabou perdendo vários pacotes. Ele recebeu apenas 77605 linhas, das 106345. Ou seja, apenas 72,9% dos pacotes.

6 Conclusões

Notamos que UDP não é confiável e é tão simples de ser usado quanto TCP. Podemos concluir, no entanto, que UDP pode ser muito bom para certas aplicações, tais como streaming de áudio e vídeo, graças a sua versatilidade e habilidade de manter uma velocidade constante de envio e outra de recebimento, bastando tratar a perda de pacotes.

7 Bibliografia

Referências

[1] Brian “Beej Jorgensen” Hall. *Beej’s Guide to Network Programming*. 2009.

8 Anexos

Os códigos dos programas seguem anexos.

```c
/*
** cludp_echo.c - Cliente de echo UDP
*/

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

#define SERVERPORT 4950 // the port users will be connecting to
#define MAXDATASIZE 1000

int main(int argc, char *argv[]) {
  int numbytes;
  int sockfd;
  struct sockaddr_in their_addr; // connector's address information
  struct hostent *he;
  char inbuf[MAXDATASIZE], outbuf[MAXDATASIZE];
  socklen_t addr_len;


  if (argc != 2) {
    fprintf(stderr,"usage: %s hostname\n", argv[0]);
    exit(1);
  }

  if ((he=gethostbyname(argv[1])) == NULL) {  // get the host info
    perror("gethostbyname");
    exit(1);
  }

  if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }

  their_addr.sin_family = AF_INET;       // host byte order
  their_addr.sin_port = htons(SERVERPORT); // short, network byte order
  their_addr.sin_addr = *((struct in_addr *)he->h_addr);
  memset(&(their_addr.sin_zero), '\0', 8);  // zero the rest of the struct

  addr_len = sizeof(struct sockaddr);

  while (fgets(outbuf, MAXDATASIZE, stdin) != NULL) {
    if ((numbytes = sendto(sockfd, outbuf, strlen(outbuf), 0,
                           (struct sockaddr *)&their_addr, addr_len)) == -1) {
      perror("sendto");
      exit(1);
    }
    if ((numbytes=recvfrom(sockfd, inbuf, MAXDATASIZE, 0,
                           (struct sockaddr *)&their_addr, &addr_len)) == -1) {
      perror("recv");
      exit(1);
    }
    inbuf[numbytes] = '\0';
    printf("%s", inbuf);
  }
  sendto(sockfd, "", 0, 0, (struct sockaddr *)&their_addr, addr_len);

  close(sockfd);

  return 0;
}
```

```c
/*
** srvudp_echo.c -- Servidor de echo UDP
*/

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

#define MYPORT 4950     // the port users will be connecting to

#define MAXBUFLEN 1000

int sockfd, numbytes, inlines = 0, inchars = 0;

void alarme() {
  /* Passou um segundo sem receber nada, sendo que ja tinha recebido algo antes.
     Vamos imprimir as estatisticas até agora e resetar a contagem. */

  fprintf(stderr, "Linhas recebidas: %d\nCaracteres recebidos: %d\n", inlines, inchars);
  inlines = 0; inchars = 0;
  fflush(stdout);
}

void quit() {
  close(sockfd);
  fprintf(stderr, "\rSinal capturado, saindo.\n");
  exit(0);
}

int main(void) {

  struct sockaddr_in my_addr;    // my address information
  struct sockaddr_in their_addr; // connector's address information
  socklen_t addr_len;
  char buf[MAXBUFLEN];           /* Buffer para receber msgs */

  if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }

  my_addr.sin_family = AF_INET;          // host byte order
  my_addr.sin_port = htons(MYPORT);      // short, network byte order
  my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
  memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

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

  addr_len = (socklen_t)sizeof(struct sockaddr);

  signal(SIGINT, quit);
  signal(SIGALRM, alarme);      /* Associando o sinal ao handler. */

  while(1) {                    /* Loop principal */
    inlines = 0; inchars = 0;
    while((numbytes = recvfrom(sockfd, buf, MAXBUFLEN-1 , 0, (struct sockaddr *)&their_addr, &add
r_len)) > 0) {
      if (numbytes == -1) {
        perror("recvfrom");
        exit(1);
      }
```

```
      inchars += numbytes;
      inlines++;
      buf[numbytes] = '\0';
      printf("%s", buf);
      if ((numbytes = sendto(sockfd, buf, strlen(buf), 0,
                             (struct sockaddr *)&their_addr, addr_len)) == -1) {
        perror("sendto");
        exit(1);
      }
      alarm(1);                 /* Se nao receber datagrama com 0 bytes em até um segundo
                                   ele desperta, e chama a funcao alarme() */
    }
    fprintf(stderr, "Linhas recebidas: %d\nCaracteres recebidos: %d\n", inlines, inchars);
    alarm(0);                   /* Terminou com um cliente, desliga o alarme. */
    fflush(stdout);             /* Forçar a saída */
  }

  return 0;
}
```

```c
/*
** c_cludp_echo.c - Cliente de echo UDP "connectado"
*/

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

#define SERVERPORT 4950 // the port users will be connecting to
#define MAXDATASIZE 1000

int main(int argc, char *argv[]) {
        int sockfd;
        struct sockaddr_in their_addr; // connector's address information
        struct hostent *he;
        int numbytes;
        char inbuf[MAXDATASIZE], outbuf[MAXDATASIZE];
        socklen_t addr_len;


        if (argc != 2) {
                fprintf(stderr,"usage: %s hostname\n", argv[0]);
                exit(1);
        }

        if ((he=gethostbyname(argv[1])) == NULL) {  // get the host info
                perror("gethostbyname");
                exit(1);
        }

        if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
                perror("socket");
                exit(1);
        }

        their_addr.sin_family = AF_INET;         // host byte order
        their_addr.sin_port = htons(SERVERPORT); // short, network byte order
        their_addr.sin_addr = *((struct in_addr *)he->h_addr);
        memset(&(their_addr.sin_zero), '\0', 8);  // zero the rest of the struct

        addr_len = sizeof(struct sockaddr);

        // UDP Connect
        if( ( connect( sockfd, (struct sockaddr *)&their_addr, addr_len ) ) == -1 )
        {
                  perror("connect");
                  exit(1);
        }

        while (fgets(outbuf, MAXDATASIZE, stdin) != NULL)
        {
                if ( ( numbytes = send(sockfd, outbuf, strlen(outbuf), 0) ) == -1 )
                {
                  perror("sendto");
                  exit(1);
                }
                if ((numbytes = recv(sockfd, inbuf, MAXDATASIZE, 0) ) == -1) {
                  perror("recv");
                  exit(1);
                }
                inbuf[numbytes] = '\0';
                printf("%s", inbuf);
        }

        // Wait a bit to close connection
```

```
        //sleep(1); for flood.c ?

        // Send 0 byte message
        if( ( send(sockfd, NULL, 0, 0 ) ) == -1 )
        {
          perror("sendto");
          exit(1);
        }

        // UDP (Dis)Connect
        their_addr.sin_family = AF_UNSPEC;
        connect( sockfd, (struct sockaddr *)&their_addr, addr_len );

        close(sockfd);

        return 0;
}
```

```c
/*
** c_srvudp_echo.c -- Servidor de echo UDP usando connect()
*/

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

#define MYPORT 4950     // the port users will be connecting to

#define MAXBUFLEN 1000

void alarme();

int sockfd, numbytes, inlines = 0, inchars = 0;
struct sockaddr_in their_addr; // connector's address information
socklen_t addr_len;

void alarme() {
  /* Passou um segundo sem receber nada, sendo que ja tinha recebido algo antes.
     Vamos imprimir as estatisticas até agora e resetar a contagem. */

  fprintf(stderr, "Linhas recebidas: %d\nCaracteres recebidos: %d\n", inlines, inchars);
  inlines = 0; inchars = 0;
  fflush(stdout);

  // UDP (Dis)Connect
  their_addr.sin_family = AF_UNSPEC;
  connect( sockfd, (struct sockaddr *)&their_addr, addr_len );
}

void quit() {

  // UDP (Dis)Connect
  their_addr.sin_family = AF_UNSPEC;
  connect( sockfd, (struct sockaddr *)&their_addr, addr_len );

  close(sockfd);
  fprintf(stderr, "\rSinal capturado, saindo.\n");
  exit(0);
}

int main(void) {

  struct sockaddr_in my_addr;   // my address information


  char buf[MAXBUFLEN];          /* Buffer para receber msgs */

  if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    {
      perror("socket");
      exit(1);
    }

  my_addr.sin_family = AF_INET;          // host byte order
  my_addr.sin_port = htons(MYPORT);      // short, network byte order
  my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
  memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

  if (bind(sockfd, (struct sockaddr *)&my_addr,
           sizeof(struct sockaddr)) == -1)
    {
      perror("bind");
      exit(1);
```

```
    }

  addr_len = (socklen_t)sizeof(struct sockaddr);

  signal(SIGINT, quit);
  signal(SIGALRM, alarme);      /* Associando o sinal ao handler. */

  while(1)
    {                   /* Loop principal */

      //Get first message/IP from client
      if( recvfrom(sockfd, buf, MAXBUFLEN-1 , 0, (struct sockaddr *)&their_addr, &addr_len)!= -1
)
        {
          printf("%s", buf);
          // UDP Connect
          if( ( connect( sockfd, (struct sockaddr *)&their_addr, addr_len ) ) == -1 )
            {
              perror("connect");
              exit(1);
            }
        }
      else
        {
          perror("recvfrom");
          exit(1);
        }

      alarm(1);
        if ((numbytes = send(sockfd, buf, strlen(buf), 0)) == -1)
        {
          perror("send");
          exit(1);
        }


      /* printf("Connected\n"); */

      inlines = 1; inchars = numbytes;
      while((numbytes = recv(sockfd, buf, MAXBUFLEN-1 , 0)) > 0)
        {
          if (numbytes == -1)
            {
              perror("recvfrom");
              exit(1);
            }
          inchars += numbytes;
          inlines++;
          buf[numbytes] = '\0';
          printf("%s", buf);
          if ((numbytes = send(sockfd, buf, strlen(buf), 0)) == -1)
            {
              perror("send");
              exit(1);
            }
          alarm(1);                     /* Se nao receber datagrama com 0 bytes em até um segundo
                                           ele desperta, e chama a funcao alarme() */

        }

      fprintf(stderr, "Linhas recebidas: %d\nCaracteres recebidos: %d\n", inlines, inchars);
      alarm(0);                 /* Terminou com um cliente, desliga o alarme. */
      fflush(stdout);           /* Forçar a saída */
    }

  return 0;
}
```