UNIVERSIDADE ESTADUAL DE CAMPINAS

LABORATÓRIO DE REDES - MC823

Tarefa 1

Miguel Francisco Alves de Mattos Gaiowski

RA 076116

Guillaume Massé

RA 107888

Prof. Paulo Lício de Geus

Campinas, 30 de agosto de 2010

Sumário

1 Introdução

Esta primeira tarefa pedia que um servidor de eco fosse implementado usando a linguagem C. Além disso, um cliente também deveria ser feito com o intuito de se conectar ao servidor e enviar mensagens e receber o que o servidor manda de volta.

Além de um servidor simples, que só aceita uma conexão por vez, deveríamos também implementar um que funcionasse de forma concorrente, aceitando várias conexões ao mesmo tempo e fazendo o tratamento adequado.

2 Objetivos

Depois dos servidores e do cliente prontos, deveríamos poder fazer testes simples, como enviar e receber de volta de a mesma mensagem, como também fazer o *pipe* de arquivos para a entrada e saída do cliente e analisar o tempo levado para a transmissão dos dados. Com isso também conseguiríamos averiguar o funcionamento correto dos dois lados da conexão, se o cliente realmente recebe de volta exatamente o que mandou.

3 Desenvolvimento

Protótipos do cliente e servidor foram disponibilizados para que modificassemos. A parte do cliente foi relativamente fácil, bastando pegar a conexão fazer a entrada e saída de dados. Os dados da entrada padrão foram jogados no socket para o servidor, e após fechar a conexão alguns dados estatísticos da 'conversa' entre o cliente e o servidor são impressos na saída de erro.

Para o servidor de eco simples, que só aceita uma conexão por vez, tivemos que pegar o protótipo e fazer com que a conexão não fosse fechada após receber a primeira mensagem, como estava implementado. Ao invés disso a conexão fica aberta até que o cliente a feche (número de bytes recebidos seja zero). Além disso, após o fechamento da conexão o servidor deve entrar a em loop e aguardar novos pedidos de conexão.

Após receber cada mensagem, o servidor envia exatamente a mesma coisa para o cliente. E quando o cliente fecha a conexão, o servidor joga na saída de erro algumas informações sobre a quantidade de dados enviada e recebida.

Com o servidor concorrente, após entender como funciona o processor de fork para que o processo filho cuide de cada conexão recebida enquanto o processo pai fica aguardando novas conexões, o trabalho foi basicamente o mesmo, de implementar as funcionalidades básicas requeridas, de fazer eco dos dados recebidos e imprimir algumas informações ao final. O protótipo usado foi o fornecido pelo Beej [1].

Primeiros testes foram feitos na mesma máquina, com o servidor ouvindo numa porta e o cliente se conectando nessa porta em localhost. Algumas mensagens foram enviadas e recebidas, e depois o teste foi feito usando arquivos para entrada e saída, permitindo comparar com diff e fazer contagem de caracteres com wc.

4 Dificuldades

Como o enunciado pedia que testes fossem feitos em sub-redes distintas, colocamos o servidor rodando na máquina bugu, do IC-3, e o cliente rodando numa máquina conectada a rede do LAS. A porta 3401, que estava designada nos protótipos não pode ser acessada de uma rede fora da do IC-3, e por isso acabamos fazendo um túnel SSH e jogando os dados no túnel, que chegavam até o servidor

e voltavam. Isso acabou fazendo com que os dados levassem mais tempo para ir e voltar, devido ao overhead de cifragem e de encapsular os pacotes.

5 Conclusões

Pudemos aprender sobre o funcionamento de sockets e de conexões concorrentes. O mais interessante foi ver quão simples um servidor de eco concorrente pode ser, e que mesmo assim problemas básicos acontecem e um entendimento da estrutura da rede, com suas camadas e estruturas de programação, deve ser obtido para que o trabalho possa ser concluído.

O resultado do teste de mandar o arquivo services para o servidor pode ser visto abaixo:

```
$ ./client localhost < /etc/services > teste
Received: Conectado, envie uma mensagem que eu devolvo.
Tempo total de transferencia: 6.858977 s
Numero de linhas enviadas: 13921
Numero de linhas recebidas: 13921
Numero de caracteres na maior linha: 118
Total de caracteres enviados: 677959
Total de caracteres recebidos: 677959
$ wc /etc/services
13921   68850  677959 /etc/services
$
```

E o que o lado do servidor mostra:

```
[ra076116@bugu lab1]$ ./server_echo
Servidor: Conectado com 127.0.0.1
Cliente fechou a conexão.
  Numero de linhas recebidas: 13921
  Total de caracteres recebidos: 677959
```

Mostramos assim que o servidor de eco simples, para só um cliente por vez funciona como esperado.

Para mostrar o funcionamento do servidor concorrente, fizemos dois túneis. Ambos direcionados a porta 9000 do servidor, mas um saindo da porta 9000 e outro da porta 9001 da máquina onde estava o cliente.

Um dos clientes mostrou a saída:

```
$ ./client localhost < /etc/services > teste
Received: Conectado, envie uma mensagem que eu devolvo.
Tempo total de transferencia: 7.096509 s
Numero de linhas enviadas: 13921
Numero de linhas recebidas: 13921
Numero de caracteres na maior linha: 118
Total de caracteres enviados: 677959
Total de caracteres recebidos: 677959
caesar:lab1 miguelgaiowski$ diff teste /etc/services
$ wc /etc/services
13921   68850  677959 /etc/services
$
```

E o outro:

```
$ ./client localhost < /etc/services > teste2
Received: Conectado, envie uma mensagem que eu devolvo.
Tempo total de transferencia: 6.858977 s
Numero de linhas enviadas: 13921
Numero de linhas recebidas: 13921
Numero de caracteres na maior linha: 118
Total de caracteres enviados: 677959
Total de caracteres recebidos: 677959
caesar:lab1 miguelgaiowski$ diff teste2 /etc/services
$ wc /etc/services
13921   68850  677959 /etc/services
$
```

O servidor reportou:

```
[ra076116@bugu lab1]$ ./serverb
Servidor: Conectado com 127.0.0.1
Servidor: Conectado com 127.0.0.1
Cliente fechou a conexão.
 Numero de linhas recebidas: 13921
 Total de caracteres recebidos: 677959
Cliente fechou a conexão.
 Numero de linhas recebidas: 13921
 Total de caracteres recebidos: 677959
```

Podemos notar que o envio do mesmo arquivo para o servidor simples demorou mais do que mandar ao mesmo tempo duas vezes para o servidor concorrente. Concluímos que algum tráfego na rede inesperado possa ter causado a demora adicional.

6 Bibliografia

Referências

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

7 Anexos

Os códigos dos programas seguem anexos.

```c
/* client.c */

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

#define PORT 9001    /* the port client will be connecting to */
#define MAXDATASIZE 1000 /* max number of bytes we can get at once
*/

int main(int argc, char *argv[]) {
  int sockfd, numbytes;
  char inbuf[MAXDATASIZE], outbuf[MAXDATASIZE];
  struct hostent *he;
  struct sockaddr_in their_addr; /* connector's address information
*/
  int numInLines = 0, numOutLines = 0, maxlinesize = 1, totalInChars
= 0, totalOutChars = 0;

  if (argc != 2) {
    fprintf(stderr,"usage: client hostname\n");
    exit(1);
  }
  if ((he=gethostbyname(argv[1])) == NULL) {  /* get the host info
*/
    perror("gethostbyname");
    exit(1);
  }
  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }

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

```
  if (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct
sockaddr)) == -1) {
    perror("connect");
    exit(1);
  }
  if ((numbytes=recv(sockfd, inbuf, MAXDATASIZE, 0)) == -1) {
    perror("recv");
    exit(1);
  }
  inbuf[numbytes] = '\0';       /* Adiciona caracter marcando fim se
string */
  fprintf(stderr, "Received: %s",inbuf);

  struct timeval start;
  gettimeofday( &start, NULL );

  while (fgets(outbuf, MAXDATASIZE, stdin) != NULL) {
    if ((numbytes = send(sockfd, outbuf, strlen(outbuf), 0)) == -1)
{
      perror("send");
      exit(1);
    }
    numOutLines++;
    maxlinesize = maxlinesize > strlen(outbuf) ? maxlinesize :
strlen(outbuf);
    totalOutChars += numbytes;// - 1;
    if ((numbytes=recv(sockfd, inbuf, MAXDATASIZE, 0)) == -1) {
      perror("recv");
      exit(1);
    }
    numInLines++;
    totalInChars += numbytes;// - 1;

    inbuf[numbytes] = '\0';
    printf("%s", inbuf);
  }

  struct timeval end;
  gettimeofday( &end, NULL );
  double time_elapsed = ( (double)( end.tv_usec - start.tv_usec ) )
/ 1.0e+6 + ( (double)( end.tv_sec - start.tv_sec ) );
  fprintf( stderr, "Tempo total de transferencia: %lf s\n",
time_elapsed );

  fprintf(stderr, "Numero de linhas enviadas: %d\n", numOutLines);
  fprintf(stderr, "Numero de linhas recebidas: %d\n", numInLines);
  fprintf(stderr, "Numero de caracteres na maior linha: %d\n",
```

```
maxlinesize - 1);
  fprintf(stderr, "Total de caracteres enviados: %d\n",
totalOutChars);
  fprintf(stderr, "Total de caracteres recebidos: %d\n",
totalInChars);

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

```c
/* server_echo.c - Servidor simples */

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

#define MYPORT 4001    /* the port users will be connecting to */
#define BACKLOG 10     /* how many pending connections queue will
hold */
#define BUFF_SIZE 1000

int main() {
  int sockfd, new_fd;  /* listen on sock_fd, new connection on
new_fd */
  struct sockaddr_in my_addr;    /* my address information */
  struct sockaddr_in their_addr; /* connector's address information
*/
  int sin_size, nbytes;
  char buffer[BUFF_SIZE];
  int numInLines = 0, totalInChars = 0;

  if ((sockfd = socket(AF_INET, SOCK_STREAM, 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 */
  bzero(&(my_addr.sin_zero), 8);        /* zero the rest of the
struct */

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

```
    perror("listen");
    exit(1);
  }

  while(1) {  /* main accept() loop */
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr,
(socklen_t *)&sin_size)) == -1) {
      perror("accept");
      continue;
    }
    numInLines = 0; totalInChars = 0;
    fprintf(stderr, "Servidor: Conectado com
%s\n",inet_ntoa(their_addr.sin_addr));

    char SendText[] = "Conectado, envie uma mensagem que eu
devolvo.\n";
    if (send(new_fd, SendText, sizeof(SendText), 0) == -1){
      perror("send");
      exit(1);
    }
    while ( (nbytes = recv(new_fd, buffer, BUFF_SIZE, 0)) > 0 ) { //
recebe msg do cliente
      numInLines++;
      totalInChars += nbytes;
      send(new_fd, buffer, nbytes, 0); // devolve a mesma coisa.
    }
    if (nbytes == 0) {
      fprintf(stderr, "Cliente fechou a conexão.\n");
      fprintf(stderr, "  Numero de linhas recebidas: %d\n",
numInLines);
      fprintf(stderr, "  Total de caracteres recebidos: %d\n",
totalInChars);
    }
    close(new_fd);
  }
  return 0;
}
```

```c
/* serverb.c - Servidor Concorrente */

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

#define MYPORT 4001    /* the port users will be connecting to */
#define BACKLOG 10     /* how many pending connections queue will
hold */
#define BUFF_SIZE 1000

int main() {
  int sockfd, new_fd;  /* listen on sock_fd, new connection on
new_fd */
  struct sockaddr_in my_addr;    /* my address information */
  struct sockaddr_in their_addr; /* connector's address information
*/
  int sin_size, nbytes;
  int yes = 1;
  char buffer[BUFF_SIZE];

  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }
  if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes,
sizeof(int)) == -1) {
    perror("setsockopt");
    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 */
  bzero(&(my_addr.sin_zero), 8);        /* zero the rest of the
struct */

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

```
    perror("listen");
    exit(1);
  }

  while(1) {  /* main accept() loop */
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr,
(socklen_t *)&sin_size)) == -1) {
      perror("accept");
      continue;
    }
    printf("Servidor: Conectado com
%s\n",inet_ntoa(their_addr.sin_addr));
    char SendText[] = "Conectado, envie uma mensagem que eu
devolvo.\n";
    int numInLines = 0, totalInChars = 0;
    if (!fork()) { /* this is the child process */
      if (send(new_fd, SendText, sizeof(SendText), 0) == -1) {
        perror("send");
        exit(1);
      }
      numInLines = 0; totalInChars = 0;
      while ( (nbytes = recv(new_fd, buffer, BUFF_SIZE, 0)) > 0 ) {
// recebe msg do cliente
        numInLines++;
        totalInChars += nbytes ;
        send(new_fd, buffer, nbytes, 0); // devolve a mesma coisa.
      }
      if (nbytes == 0) {
        fprintf(stderr, "Cliente fechou a conexão.\n");
        fprintf(stderr, "  Numero de linhas recebidas: %d\n",
numInLines);
        fprintf(stderr, "  Total de caracteres recebidos: %d\n",
totalInChars);
      }
      close(new_fd);
      exit(0);
    }
    close(new_fd);  /* parent doesn't need this */

    while(waitpid(-1,NULL,WNOHANG) > 0); /* clean up all child
processes */
  }
}
```