Atividade 6 - Implementacao de Aplicacao Cliente/Servidor similar a telnet e concorrente com TCP- Códigos

Gabriel Hidasy Rezende

November 9, 2015

```
//Q1-2 cliente
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>
#include <string.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include "netutils.h"
#define MAXLINE 4096

int main(int argc, char **argv) {
   int    sockfd, n;
   char   recvline[MAXLINE + 1];
   char   error[MAXLINE + 1];
   struct sockaddr_in servaddr;

   if (argc != 3){
      strcpy(error,"uso: ");
      strcat(error,argv[0]);
      strcat(error," <IPaddress> <Port Number>");
      perror(error);
      exit(1);
   }
   int pnumber;
```

```
  sscanf(argv[2],"%d",&pnumber);
  /* Abre o socket */
  sockfd = Socket(AF_INET, SOCK_STREAM, 0);

  /* Prepara as estruturas que serao usadas */
  bzero(&servaddr, sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_port = htons(pnumber);

  /* Aqui e criada a estrutura que sera usada para a conexao */
  Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

  /* A conexao e feita */
  Connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
  printf("Connected to %s:%d\n",argv[1],pnumber);

  char command[1025];
  while (1) {
    /* Le a linha de comando */
    n = scanf(" %[^\n]",command);
    if (n == 0) {
      break;
    }
    /* Envia o comando */
    Write(sockfd,command,strlen(command));
    /* Recebe a resposta */
    n = Read(sockfd, recvline, MAXLINE);
    if (n == 0) {
      break;
    }
    recvline[n] = 0;
    printf("%s\n",recvline);
  }
  close(sockfd);
  exit(0);
}
```

```
//Q1-2 servidor
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
```

```
#include <netinet/in.h>
#include <sys/socket.h>
#include <time.h>
#include <unistd.h>
#include <arpa/inet.h>
#include "netutils.h"

#define LISTENQ 10
#define MAXDATASIZE 4096

void deal_with_client(int connfd)
{
  char command[MAXDATASIZE];
  int n;
  while(1) {
    n = Read(connfd,command,MAXDATASIZE-1);
    if (n == 0) {
      break;
    }
    command[n] = 0;
    //Thats a security flaw, try to chroot this server
    system(command);
    Write(connfd,command,n);
  }
}

int main (int argc, char **argv)
{
  int   listenfd, connfd;
  struct sockaddr_in servaddr;
  int pnumber = 12344;
  if (argc == 2) {
    sscanf(argv[1],"%d",&pnumber);
   }
   /* No trecho abaixo um socket e inicializado */
   listenfd = Socket(AF_INET, SOCK_STREAM, 0);

   /* Estruturas sao preparadas */
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family    = AF_INET;
   servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
   servaddr.sin_port      = htons(pnumber);

   /* A porta e reservada */
   Bind(listenfd,(struct sockaddr *) &servaddr, sizeof(servaddr));
   /* O servidor passa a receber conexoes */
   Listen(listenfd,LISTENQ);

   for ( ; ; ) {
     struct sockaddr_in client;
     connfd = Accept(listenfd,&client);
     char client_address[128];
     int client_port = ntohs(client.sin_port);
     inet_ntop(AF_INET,&client.sin_addr.s_addr,client_address,128);
     printf("Serving client %s:%d\n",client_address,client_port);

      int chid = fork();
      if (chid == 0) {
   deal_with_client(connfd);
   close(connfd);
   exit(0);
      } else {
   close(connfd);
      }
   }
   return(0);
}
```

```
//Q3 cliente
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>
#include <string.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include "netutils.h"
#define MAXLINE 4096

int main(int argc, char **argv) {
     int    sockfd, n;
     char   recvline[MAXLINE + 1];
     char   error[MAXLINE + 1];
```

```
struct sockaddr_in servaddr;
time_t rawtime;
struct tm * timeinfo;

if (argc != 3){
strcpy(error,"uso: ");
strcat(error,argv[0]);
strcat(error," <IPaddress> <Port Number>");
perror(error);
exit(1);
}
int pnumber;
sscanf(argv[2],"%d",&pnumber);
/* Abre o socket */
sockfd = Socket(AF_INET, SOCK_STREAM, 0);

/* Prepara as estruturas que serão usadas */
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(pnumber);

/* Aqui é criada a estrutura que será usada para a conexão */
Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

/* A conexão é feita */
Connect(sockfd, (struct sockaddr *) &servaddr,
    sizeof(servaddr));
time(&rawtime);
timeinfo = localtime(&rawtime);
printf("%s: connected to %s:%d\n",asctime
    (timeinfo),argv[1],pnumber);

char command[MAXLINE];
while (1) {
/* Le a linha de comando */
n = scanf(" %[^\n]",command);
if (n <= 0) {
     break;
}
if (!strcmp("Bye",command)) {
     break;
}
/* Envia o comando */
Write(sockfd,command,strlen(command));
/* Recebe a resposta */
```

```
    n = Read(sockfd, recvline, MAXLINE);
    if (n == 0) {
        break;
    }
    recvline[n] = 0;
    printf("%s\n",recvline);
    }
    close(sockfd);
    exit(0);
}
```

```
//Q3 Servidor
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <signal.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#include <arpa/inet.h>
#include "netutils.h"
#include <time.h>
#define LISTENQ 10
#define MAXDATASIZE 4096

FILE *logfd;
int listenfd;

struct client {
    int port;
    pid_t pid;
    char address[32];
};

struct client_list {
    struct client c;
    struct client_list *next;
};
```

```c
struct client_list *clist = NULL;

void timestamp(char *t)
{
    time_t rawtime;
    struct tm *timeinfo;
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    strcpy(t,asctime(timeinfo));
    t[24] = 0; //TODO, ensure \n is always at 24 or find it
}

void add_client(pid_t pid, char *address, int port)
{
    printf("Adding PID %d\n",pid);
    if(clist == NULL) {
    clist = malloc(sizeof(struct client_list));
    clist->next = NULL;
    }
    struct client_list *head = clist;
    struct client_list *prev = clist;

    head = clist;
    while (clist != NULL) {
    prev = clist;
    clist = clist->next;
    }
    clist = malloc(sizeof(struct client_list));
    prev->next = clist;
    clist->next = NULL;
    clist->c.port = port;
    strcpy(clist->c.address,address);
    clist->c.pid = pid;
    clist = head;
}

int remove_client(pid_t r, struct client *c)
{
    if (r == -1) {
    return 0;
    }
    printf("Removing PID %d\n",r);
    struct client_list *curr = clist->next;
    struct client_list *prev = clist;
    while (curr->c.pid != r) {
```

```
    curr = curr->next;
    prev = prev->next;
    if (curr == NULL)
        return 0;
    }
    c->port = curr->c.port;
    c->pid = curr->c.pid;
    strcpy(c->address,curr->c.address);
    prev->next = curr->next;
    free(curr);
    return 1;
}

void deal_with_client_chdhandler(int signal)
{
    /* Registering a sighandler for sigchd to interrupt reads when
       there is no
  output from the process */
    pid_t pid;
    pid = wait(NULL);
    printf("Child %d is dead\n",signal);
}

void deal_with_client(int connfd, char *address, int port)
{
    signal(SIGCHLD,deal_with_client_chdhandler);
    char time_now[25];
    timestamp(time_now);
    printf("%s: connected %s:%d\n",time_now,address,port);
    char command[MAXDATASIZE];
    int n;
    while(1) {
    int i;
    for(i = 0; i < MAXDATASIZE; i++) {
        command[i] = 0;
    }
    n = Read(connfd,command,MAXDATASIZE-1);
    if (n == 0) {
        break;
    }
    command[n] = 0;
    printf("%s:%d > %s\n",address,port,command);


    char output[MAXDATASIZE];
    FILE *process_output;
    process_output = popen(command,"r");
    n = fread(output,1,MAXDATASIZE,process_output);
    pclose(process_output);
    /*When the process ends without output (ex, echo $i > a.txt)*/
    if (n == 0) {
        Write(connfd,"\n",2);
        continue;
    }
    Write(connfd,output,n);
    }
    timestamp(time_now);
    printf("%s: disconnected %s:%d\n",time_now,address,port);
    close(connfd);
    exit(0);
}

void chdhandler(int signal)
{
    pid_t pid;
    pid = wait(NULL);
    struct client c;
    if(remove_client(pid,&c)) {
    char localtime[25];
    timestamp(localtime);
    fprintf(logfd,"%s: disconnected %s:%d\n",
       localtime,c.address,c.port);
    }
}

void inthandler(int signal)
{
    struct client_list *next = clist;
    while (next != NULL) {
    next = clist->next;
    free(clist);
    clist = next;
    }
    fclose(logfd);
    close(listenfd);
    exit(0);
}

int main (int argc, char **argv)
{
```

```
signal(SIGCHLD,chdhandler);
signal(SIGINT,inthandler);
int connfd;
struct sockaddr_in servaddr;
int pnumber = 12344;

//Adicionar uma sentinela a lista de clientes
add_client(-1,"-1",-1);
if (argc == 2) {
sscanf(argv[1],"%d",&pnumber);
}
logfd = fopen("logfile.log","w");
/* No trecho abaixo um socket é inicializado */
listenfd = Socket(AF_INET, SOCK_STREAM, 0);

/* Estruturas são preparadas */
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family    = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port      = htons(pnumber);

/* A porta é reservada */
Bind(listenfd,(struct sockaddr *) &servaddr, sizeof(servaddr));

/* O servidor passa a receber conexões */
Listen(listenfd,LISTENQ);

for ( ; ; ) {
struct sockaddr_in client;
connfd = Accept(listenfd,&client);
char client_address[32];
int client_port = ntohs(client.sin_port);
inet_ntop(AF_INET,&client.sin_addr.s_addr,client_address,128);
/* This is a classic fork/branch server */
int chid = fork();
if (chid == 0) {
    deal_with_client(connfd,client_address,client_port);
} else {
    char localtime[25];
    timestamp(localtime);
    /* The parent is always resposable for logs */
    fprintf(logfd,"%s: connected %s:%d\n",
       localtime,client_address,client_port);
    add_client(chid,client_address,client_port);
    close(connfd);
    }
    }
    return(0);
}
```

```
//Netutils.c
#include "netutils.h"
int Socket(int family, int type, int flags)
{
  int sockfd;
  if ((sockfd = socket(family, type, flags)) < 0) {
    perror("socket");
    exit(1);
  } else
    return sockfd;
}
void Bind(int sockfd, struct sockaddr *servaddr, int size)
{
  if (bind(sockfd,
      servaddr, size) == -1) {
    perror("bind");
    exit(1);
  }
}

void Listen(int listenfd, int flags)
{
  if (listen(listenfd, flags) == -1) {
      perror("listen");
      exit(1);
   }
}

int Read(int connfd, char *command, int size) {
  int n;
  n = read(connfd,command,size);
  if (n < 0) {
    perror("read");
  }
  return n;
}

int Accept(int listenfd, struct sockaddr_in *client)
{
```

```c
  int connfd;
  socklen_t client_size = sizeof(*client);
  if ((connfd = accept(listenfd, (struct sockaddr *) client,
      &client_size)) == -1 ) {
        perror("accept");
        exit(1);
     }
  return connfd;
}
void Write(int connfd, char *command, int n)
{
  if(write(connfd, command, n) < 0) {
    perror("write");
    exit(1);
  }
}
void Connect(int sockfd, struct sockaddr *servaddr, int size)
{
  if (connect(sockfd, servaddr, size) < 0) {
      perror("connect");
      exit(1);
   }
}
void Inet_pton(int family, char *in, struct in_addr *servaddr)
{
  if (inet_pton(family, in, servaddr) <= 0) {
    perror("inet_pton error");
    exit(1);
  }
}
```

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

```
/* Poderia ter simplificado algumas chamadas, eliminando os int
    size */
/* fazendo os mesmos dentro dos wrappers, mas do jeito que fiz eles
    são */
/* compativeis o bastante para serem usados como drop-in
    replacements para */
/* as funções de baixo nivel*/

int Socket(int family, int type, int flags);
void Bind(int sockfd, struct sockaddr *servaddr, int size);
void Listen(int listenfd, int flags);
int Read(int connfd, char *command, int size);
int Accept(int listenfd, struct sockaddr_in *client);
void Write(int connfd, char *command, int n);
void Connect(int sockfd, struct sockaddr *servaddr, int size);
void Inet_pton(int type, char *in, struct in_addr *servaddr);
```