Instituto Tecnológico de Costa Rica

Lenguajes de programación

Grupo 2

Proyecto #1: Sistema de mensajería

Integrantes:

Jose Fabio Hidalgo Rodríguez (2017100950)

Joshua Jimenes Leitón (200823980)

Jose Paulo Zúñiga Valverde (2016113355)

22 de mar. de 2019

Tabla de contenidos:

Introducción:

C es un Lenguaje de programación multipropósito, creado por Dennis Ritchie en 1973, en Bell Laboratories, para uso con sistema operativo Unix, compiladores y programas. Posee archivos de encabezado y archivos fuente. Este lenguaje se considera de bajo nivel, por lo que se ponen en práctica los conocimientos mediante la realización de un sistema de servidor y clientes con sockets, en donde múltiples clientes deben poder conectarse y enviar mensajes entre estos pasando por el servidor.

Se realiza el programa implementando dos funciones importantes: fork y pipe. La función fork, que se utiliza para crear nuevos procesos, que se convierten en procesos hijos de el proceso padre u original. [4].

Descripción del proyecto:

El programa de mensajería va a tener un servidor central que almacene toda la información de los usuarios. Luego, cuando un usuario A quiera enviar un mensaje a un usuario B, ese mensaje pasará primero por el servidor central, antes de ser enviado al usuario B.

Las siguientes son las funcionalidades del programa:

- Registrar usuario: los usuarios deben poder registrarse en el servidor central. A la horade registrarse, se debe obtener la IP del usuario de forma automática, sin necesidad de que el usuario la especifique de forma manual. Con respecto al puerto, siempre se usará un valor predeterminado que estará especificado en un archivo de configuración, pero ese valor podrá ser modificado por los usuarios, sin necesidad de recompilar el código.
- Enviar mensajes: el programa permitirá a los usuarios enviarle mensajes a alguno de sus contactos, para lo cual se especificará el nombre de usuario, y el mensaje, y el programa deberá enviar el mensaje de texto a dicho usuario, a través del servidor. El mensaje podrá ser de texto. Los mensajes no serán enviados de un usuario directamente a otro usuario, sino que tendrán que pasar por medio del servidor.

- Recibir mensajes: Para permitir el envío y recepción de mensajes de forma simultánea, deberán manejar dos sockets. La idea es usar fork() para bifurcar el programa en dos procesos, uno que se encargue de enviar y otro que se encargue de recibir. Deben usar diferentes colores para poder diferenciar los mensajes enviados de los recibidos.

Para desarrollar algunas de las funcionalidades del programa, deberán investigar el uso de los sockets de red en Linux. Además, deberán investigar cómo manejar diferentes procesos mediante el uso de fork.

Función fork():

Esta función es la que se encarga de crear un nuevo proceso dentro de un proceso. El nuevo proceso creado es una copia exacta del original, con la única diferencia que cada uno de ellos tiene su propio identificador de proceso (pid o process identification). [1].

En el sistema de mensajería, el fork se utiliza para atender a 2 o más clientes de forma simultánea, así como en el manejo de envío y recepción de mensajes en el servidor de forma que se puedan atender todos los mensajes que se envían.

Pipe():

Para comunicar los procesos padre e hijo del fork, se emplea un pipe, el cual se encarga de enviar los mensajes de un cliente en el proceso padre a otro cliente en el proceso hijo.

El pipe “p” se hereda al hacer el fork() que da lugar al proceso hijo, pero es necesario que el padre haga un close() de p[0] (el lado de lectura de la tubería), y el hijo haga un close() de p[1] (el lado de escritura de la tubería). Una vez hecho esto, los dos procesos pueden emplear la tubería para comunicarse (siempre unidireccionalmente), haciendo write() en p[1] y read() en p[0], respectivamente. [2].

Sockets:

Socket designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada.

Un socket es, como su propio nombre indica, un conector o enchufe. Con él podremos conectarnos a ordenadores remotos o permitir que éstos se conecten al nuestro a través de la red. En realidad, un socket no es más que un descriptor de fichero un tanto especial.

Implementación:

Se inicia con un sistema sencillo de sockets cliente/servidor por el cual basarse para añadir el resto de las funcionalidades que se requieren para el proyecto.

Seguido, a esta base se le añaden forks para permitir la conexión de varios clientes al servidor. Antes de los forks se declaran los pipes por los cuales los procesos van a acceder a información vital para el correcto flujo de los datos.

Funciones:

Cliente

```c
int main(int argc, char* argv[]){
  pid_t pid;

  configuration config;
  if (ini_parse("test.ini", handler, &config) < 0) {
        printf("No se puede cargar 'test.ini'\n");
        return 1;
  }
  printf("Configuracion cargada de 'test.ini': puerto=%d\n",
        config.Usport);

  //PORT = config.Usport;

        int clientSocket, ret;
        struct sockaddr_in serverAddr;
        char buffer[1024];
    char user[15];

    //Obtencion y guardado de nombre de usuario en struct User
    //struct User user;
    printf("Ingrese el nombre de usuario: ");
    //scanf("%s", &user.name[0]);
    scanf("%s", &user[0]);
    //printf("El nombre de usuario guardado es:%s\n", user.name);
    printf("El nombre de usuario guardado es:%s\n", user);

    //Obtencion y guardado de nombre de direccion ip en struct User
    //Extraido de: https://www.geeksforgeeks.org/c-program-display-hostname-ip-address/

    char hostbuffer[256];
    char *IPbuffer;
    struct hostent *host_entry;
    int hostname;

    hostname = gethostname(hostbuffer, sizeof(hostbuffer));
    host_entry = gethostbyname(hostbuffer);
    IPbuffer = inet_ntoa(*((struct in_addr*) host_entry->h_addr_list[0]));
    //strcpy(user.ipAddress, IPbuffer);
    //printf("%s\n", user.ipAddress);

        clientSocket = socket(AF_INET, SOCK_STREAM, 0);
        if(clientSocket < 0){
                printf("[-]Error en la conexion.\n");
                exit(1);
        }
```

```
93          printf("[+]Socket del cliente se ha creado.\n");
94
95          memset(&serverAddr, '\0', sizeof(serverAddr));
96          serverAddr.sin_family = AF_INET;
97          serverAddr.sin_port = htons(config.Usport);
98          serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
99
100         ret = connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
101         if(ret < 0){
102                 printf("[-]Error en la conexion.\n");
103                 exit(1);
104         }
105         printf("[+]Conectado al servidor.\n");
106
107
108   //una vez conectado, se envia el username como primer mensaje
109   send(clientSocket, user, 1024, 0);
110   //fork
111   if((pid = fork()) != 0){//hijo espera que el usuario digite un mensaje y enviarlo al servidor
112     while(1){
113       //el formato del mensaje es el siguiente:
114       //buffer [1024]
115       //0-14 > Origen
116       //15-29 > destino
117       //30-1024 > mensaje
118       strcpy(buffer, user);
119       scanf("%s", buffer+30);
120       //revisa si el  mensaje es para salir dek programa
121       if(strcmp(buffer+30, ":exit") == 0){
122                         close(clientSocket);
123                         printf(RED"[-]Disconnected from server."RESET"\n");
124         send(clientSocket, buffer, 1024, 0);
125                         exit(1);
126                 }
127       else{// si no es par salir, pide destinatario
128         printf(GRN"Para: \t"RESET"");
129         scanf("%s", (buffer+15));
130         send(clientSocket, buffer, 1024, 0);
131       }
132     }
133   }
134
135   else{ //padre espera recibir un mensaje
136     while(1){
137       if(recv(clientSocket, buffer, 1024, 0) < 0){
138         printf("[-]Error in receiving data.\n");
```

```
139         }
140         else{
141           if(strcmp(buffer,"\0") > 0){
142             printf(YEL"%s : %s "RESET"\n", buffer, buffer+30);
143           }
144         }
145         bzero(buffer, sizeof(buffer));
146       }
147 
148     }
149           return 0;
150 }
```

Servidor:

```c
int main(){
        //variables del socket
        int sockfd, ret;
        struct sockaddr_in serverAddr;
        int newSocket;
        struct sockaddr_in newAddr;
        socklen_t addr_size;
        //buffers para enviar y recibir strings
        char buffer[1024];
        char inbuf[1024];
        //inicializador del fork()
        pid_t childpid;
        pid_t childpid2;

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if(sockfd < 0){
                printf("[-]Error in connection.\n");
                exit(1);
        }
        printf("[+]Server Socket is created.\n");

        memset(&serverAddr, '\0', sizeof(serverAddr));
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(PORT);
        serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

        ret = bind(sockfd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
        if(ret < 0){
                printf("[-]Error in binding.\n");
                exit(1);
        }
        printf("[+]Bind to port %d\n", 4444);

        if(listen(sockfd, 10) == 0){
                printf("[+]Listening....\n");
        }else{
                printf("[-]Error in binding.\n");
        }
        //Pipes (usuarios y mensajes)
        int usersPipe[2];
        int p[2];
        if(pipe(usersPipe)<0)
                exit(1);
        //se le agrega el contador de usuarios al pipe
        write(usersPipe[1],"0",4);
```

```
58          while(1){
59                  //esperar nueva conexion
60                  newSocket = accept(sockfd, (struct sockaddr*)&newAddr, &addr_size);
61                  if(newSocket < 0){
62                          exit(1);
63                  }
64                  //NUEVO USUARIO
65      //se guarda un username para el cliente
66                  char username[15];
67                  recv(newSocket, buffer, 1024, 0);//primer mensaje es el nombres
68                  strcpy(username, buffer);
69                  printf("Nueva conexion de %s desde %s:%d\n",username, inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port));
70
71                  char newUser[20];
72                  strcpy(newUser, username);
73                  sprintf(newUser+15, "%d", newSocket);
74                  //Revisar cuantos usuarios hay con la primera salida del pipe, que seria la cantidad de usuarios
75                  char cantUs[5];
76                  read(usersPipe[0],cantUs,4);
77                  int cant = atoi(cantUs);
78                  sprintf(cantUs,"%d", cant+1);
79                  //envia al pipe la cantidad anterior +1
80                  write(usersPipe[1],cantUs,4);
81                  //se recorre el pipe la cantidad de usuarios que habian
82                  //queda el valor de la nueva cantidad de usuarios como primera salida
83                  //solo se leen y se vuelven a meter para acomodar el pipe
84                  for(int x = 0; x < cant; x++){
85                          read(usersPipe[0],buffer,20);
86                          write(usersPipe[1],buffer,20);
87                  }
88                  write(usersPipe[1], newUser, 20);
89                  printf("%s : %s : %s\n", newUser, newUser+15, cantUs);
90                  //==============
91
92                  if(pipe(p)<0)
93                          exit(1);
94                          //primer fork
95                          if((childpid2 = fork())!= 0){
96                                  //el padre del primer fork, ejecuta otro forks
97                                  if((childpid = fork()) != 0){
98                                          close(sockfd);
99                                          while(1){
100                                                 recv(newSocket, buffer, 1024, 0);
101
102                                                 if(strcmp(buffer+30, ":exit") == 0){
```

```
103                                             printf("%s has disconnected from %s:%d\n",username, inet_ntoa(newAddr.sin_a
104                                             exit(1);
105                                     }else{
106                                             //printf("recibiendo: %s : %s : %s \n", buffer, buffer+15, buffer+30);
107                     write(p[1],buffer,1024);
108                                     }
109                                     bzero(buffer, sizeof(buffer));
110                             }
111                             break;
112                     }
113                     //hijo de segundo fork
114                     //revisa el pipe por mensajes nuevos
115                     else{
116                             while(1){
117                                     read(p[0],buffer, 1024);//lee mensaje
118                                     char sendUser[15];
119                                     strcpy(sendUser, buffer+15);
120                                     read(usersPipe[0],cantUs,4);
121                                     int canti = atoi(cantUs);
122                                     write(usersPipe[1],cantUs,4);
123                                     //recorre todo el pipe de usuarios y revisa el destinatario
124                                     for(int x = 0; x < canti; x++){
125                                             read(usersPipe[0],inbuf,20);
126                                             if((strcmp(inbuf,sendUser)) == 0){// si lo encuentra envia el mensaje
127                                                     int sendSocket = atoi(inbuf+15);
128                                                     printf("enviando: %s : %s : %s : %d \n", buffer, buffer+15, buffer+
129                                                     send(sendSocket, buffer,1024,0);
130                                             }
131                                             write(usersPipe[1],inbuf,20);
132                                     }
133                                     bzero(buffer, sizeof(buffer));
134                                     bzero(inbuf, sizeof(inbuf));
135                             }//-------------------------------------------
136                             break;
137                     }
138             }
139     }
140     close(newSocket);
141     return 0;
142 }
```

Conclusiones:

- Registrar usuario: se logró implementar un sistema en el cual mediante el servidor se guarden todos los usuarios registrados en el sistema por medio de la aplicación cliente.
- Enviar mensajes: se logró realizar el envió de mensajes por medio de sockets, en el cual, a partir del cliente, se pasa un mensaje codificado de tal forma que el cliente pueda enviárselo al cliente correspondiente.
- Recibir mensajes: se logró realizar un sistema en el cual se implementa el fork en el servidor, junto con el sistema de comunicación de procesos pipe. El servidor es capaz de estar atento a una nueva conexión de un cliente, revisar si le llegó un mensaje o si más bien debe enviar un mensaje. El mensaje entra codificado de tal forma que el servidor escoge al cliente al que mandárselo correctamente.

Fuentes:

[1]- https://nideaderedes.urlansoft.com/2009/10/26/procesos-en-c-crear-un-nuevo-proceso-con-fork/

[2]- https://www.programacion.com.py/escritorio/c/pipes-en-c-linux

[3]- https://www.programacion.com.py/noticias/sockets-en-c-parte-i-linux

[4]- http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html

[5] - https://hernandis.me/2017/03/20/como-hacer-un-makefile.html