Cátedra de Sistemas Operativos II

FACULTAD DE CIENCIAS EXACTAS,
FÍSICAS Y NATURALES

Trabajo Práctico N°1

Alumno: Oliva Arias, Carlos Agustín
Fecha: 10 de Abril de 2019

Índice

1. Introducción

1.1. Propósito

El propósito del trabajo, es la simulación de un sistema cuyo objetivo es posibilitar la conexión del satélite con su estación terrena para la transmisión de información, así como también actualizaciones del software.

1.2. Ámbito del sistema

El sistema esta compuesto por:

- Un satélite, representado por una raspberry pi 4, que genera imágenes de la superficie terrestre. El mismo posee un SO raspbian.
- Su estación terrena, representada por una computadora de propósito general.

Se conectan entre sí en una red local.

1.3. Definiciones, Acrónimos y Abreviaturas

Stack Tcp/Ip: La pila TCP/IP es un conjunto de protocolos que posibilitan las comunicaciones de Internet.
MTU (Maximum Transfer Unit): La unidad máxima de transferencia refiere al tamaño en bytes de la unidad de datos más grande que puede enviarse a nivel de capa de enlace.
MSS (Maximum Segment Size): La unidad máxima de segmento refiere al tamaño en bytes de la unidad de datos más grande que puede enviarse a nivel de capa de aplicación. Usualmente ->MTU - 40 Bytes.
CLI (Command-Line Interface): Un programa con interfaz por línea de comandos solo tiene una interfaz de usuario donde el mismo ejecuta comandos mediante la introducción de caracteres.
mDNS (multicast Domain Name Server): Protocolo de resolución de nombres a direcciones ip para redes pequeñas que no cuentan con servidores DNS.

1.4. Referencias

[1] `https://en.wikipedia.org/wiki/Multicast_DNS`

[2] `https://github.com/raspberrypi`

[3] `https://medium.com/@au42/the-useful-raspberrypi-cross-compile-guide-ea56054de`

[4] https://www.raspberrypi.org/documentation/linux/usage/systemd.md

1.5. Descripción general del documento

Las siguientes secciones describen el proceso de realización del trabajo, las herramientas y archivos auxiliares utilizados.

2. Descripción general

El producto a desarrollar es un software (desarrollado en lenguaje C), que permite realizar la conexión, control y transferencia de un programa servidor con un único programa cliente, que simula un satélite geoestacionario con su correspondiente estación terrena.

2.1. Funciones del producto

El sistema debe cumplir 3 funciones una vez desplegado completamente, todas disponibles desde la estación terrena donde un usuario puede requerirlas. estas son:

- update firmware.bin envía un archivo binario al satélite con una actualización del software ("firmware") cliente, subido el archivo, se debe reiniciar el satélite con la nueva actualización.
- start scanning inicia un escaneo de toda la cara de la Tierra una sección del planeta. Cada escaneo se debe enviar a la estación terrena, que va a estar midiendo el tiempo que le lleva el escaneo de todo el disco (desde que envía el comando, hasta que recibe el último datagrama). Este proceso debe ser lo más óptimo posible. El tamaño de cada escaneo está determinado por el tamaño máximo de un datagrama.
- obtener telemetría cada satélite le envía a la estación terrena la siguiente información:
  - Id del satélite
  - Uptime del satélite
  - Versión del software
  - Consumo de memoria y CPU

2.2. Restricciones

Se requiere el desarrollo en lenguaje C.

2.3. Suposiciones y dependencias

Dado que la comunicación entre el satélite y su estación terrena requiere de una dirección ip conocida y para evitar configurar cualquiera de ambas con una dirección fija, las comunicaciones se apoyan sobre mDNS[1]. En la estación terrena, al necesitarse la compilación cruzada del binario para enviar al satélite, se resolvió la utilización de cmake y utilidades provistas en el github de la fundación Raspberry.[2]

3. Diseño de solución

La solución consta de los siguientes archivos fuente:

- sockets.c
- test_terrena.c
- test_satelite.c

Como de varios archivos de configuración: un makefile, un proyecto de cmake, y un archivo de configuración para la unidad de servicio de systemd.

En el archivo *sockets.c* se escribieron todas las funciones que permiten la conexión mediante sockets, la escritura/lectura de los mismos de diversas formas, comunes a ambos programas implementados. Los archivos *test_satelite.c* y *test_terrena.c* responden a la arquitectura cliente-servidor en esos roles respectivamente.

Se optó por la utilización de un modelo en banda para la transmisión utilizando un único puerto tcp(8080), y también el puerto udp (8080) para el requerimiento en udp.

Los archivos de configuración pueden consultarse en el apéndice.

4. Implementación y resultados

Se presentan los archivos de código fuente utilizados. Los mismos están basados en el código fuente proporcionado por la cátedra con las modificaciones necesarias al caso.

```c
#include <errno.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#define MAX 1460

struct sockaddr_in serv_addr_udp;

int open_tcp_server(char *port_str)
{
  int sockfd, newsockfd, puerto, pid;
  struct sockaddr_in serv_addr, cli_addr;
  unsigned int clilen;
  sockfd = socket( AF_INET, SOCK_STREAM, 0);
  if ( sockfd < 0 ) {
    perror( "Fallo socket.\n" );
    exit( 1 );
  }
  setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){1},
     sizeof(int));
  memset( (char *) &serv_addr, 0, sizeof(serv_addr) );
  puerto = atoi( port_str );
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons( puerto );
  if ( bind(sockfd, ( struct sockaddr *) &serv_addr, sizeof(
     serv_addr ) ) < 0 )
  {
    perror( "Fallo bind " );
    exit( 1 );
  }
  listen( sockfd, 5 );
 clilen = sizeof( cli_addr );

  while( 1 )
 {
  newsockfd = accept( sockfd, (struct sockaddr *) &cli_addr,
     &clilen );
  if ( newsockfd < 0 ) {
   perror( "Fallo accept " );
   exit( 1 );
  }
```

```
46    pid = fork();
47    if ( pid < 0 )
48      {
49     perror( "Fallo fork " );
50     exit( 1 );
51    }
52    if ( pid == 0 )  // Proceso hijo
53      {
54     close( sockfd );
55        //printf( "SERVIDOR [%s]: newsockfd: %d\n",port_str,
              newsockfd );
56        return newsockfd;
57    }
58    else
59      {
60     //printf( "SERVIDOR [%s]: New client,proceso hijo: %d\n",
          port_str, pid );
61     close( newsockfd );
62    }
63   }
64  }
65
66  int open_udp_server(char * port_str)
67  {
68    int sockfd, puerto;
69   sockfd = socket( AF_INET, SOCK_DGRAM, 0 );
70   if (sockfd < 0) {
71      return -3;
72   }
73   memset( &serv_addr_udp, 0, sizeof(serv_addr_udp) );
74   puerto = atoi( port_str );
75   serv_addr_udp.sin_family = AF_INET;
76   serv_addr_udp.sin_addr.s_addr = INADDR_ANY;
77   serv_addr_udp.sin_port = htons( puerto );
78   memset( &(serv_addr_udp.sin_zero), '\0', 8 );
79
80   if( bind( sockfd, (struct sockaddr *) &serv_addr_udp, sizeof
        (serv_addr_udp) ) < 0 ) {
81      return -4;
82   }
83    return sockfd;
84  }
85
86  int open_tcp_client(char *ip_str,char *port_str)
87  {
88    int sockfd, puerto;
89    struct sockaddr_in serv_addr;
90    struct hostent *server;
91
```

```
92    puerto = atoi( port_str );
93    sockfd = socket( AF_INET, SOCK_STREAM, 0 );
94    if ( sockfd < 0 ) {
95      return -1;
96    }
97
98    server = gethostbyname( ip_str );
99    if (server == NULL) {
100     return -2;
101   }
102   memset( (char *) &serv_addr, '0', sizeof(serv_addr) );
103   serv_addr.sin_family = AF_INET;
104   bcopy((char *)server->h_addr,(char *)&serv_addr.sin_addr.
        s_addr, server->h_length );
105   serv_addr.sin_port = htons( puerto );
106   if ( connect( sockfd, (struct sockaddr *)&serv_addr, sizeof
        (serv_addr ) ) < 0 ) {
107     return -3;
108   }
109   return sockfd;
110 }
111
112 int write_cmd(int sockfd,void *cmd_ptr)
113 {
114   char buffer[MAX];
115   strcpy(buffer,cmd_ptr);
116   size_t n;
117     n = write( sockfd, buffer, strlen(buffer) );
118     if ( n < 0 ) {
119       return -1;
120     }
121     return 0;
122 }
123
124 void read_cmd(int sockfd, void *cmd_ptr)
125 {
126   char buffer[MAX];
127   int n;
128   memset( buffer, 0, MAX );
129   n = recv( sockfd, buffer, MAX-1 ,0);
130   if ( n < 0 ) {
131     return;
132   }
133   strcpy(cmd_ptr,buffer);
134 }
135
136 int send_file(int fd, char *filename_str)
137 {
138   long int lsize;
```

```
139   FILE * pFile;
140   pFile = fopen(filename_str,"rb");
141   if( pFile == NULL ){
142    return -1;
143   }
144   fseek (pFile , 0 , SEEK_END);
145   lsize = ftell (pFile);
146   rewind (pFile);
147   size_t num = lsize;
148   char snum[6];
149   sprintf(snum,"%zu",num);
150   write_cmd(fd,snum);
151   size_t n, m;
152   unsigned char buff[MAX];
153    //copy the file into the socket:
154    //printf("Voy a enviar %s bytes\n",snum);
155   do
156   {
157     n = fread((void *)buff, 1, sizeof(buff), pFile);
158     if (n)
159     {
160       m = write(fd, buff, n);
161     }
162     else
163     {
164       m=0;
165     }
166   } while ((n > 0) && (n == m));
167   // close the file
168   fclose (pFile);
169   return 0;
170
171 }
172 int receive_file (int sockfd,char *filename_str)
173 {
174   char snum[MAX];
175   read_cmd(sockfd,snum);
176   int num = atoi(snum);
177   FILE *fp;
178   size_t n,total;
179   total=0;
180   unsigned char buff[MAX];  // to store message from client
181   fp=fopen(filename_str,"wb"); // stores the file content in
        filename_str in the program directory
182   if( fp == NULL ){
183    return -1;
184   }
185   while(total < num){
186     //printf("%zu/%d\n",total,num);
```

```
187     n = recv(sockfd,buff, sizeof(buff), 0);
188     total+=n;
189     fwrite(buff,1,n,fp);
190   }
191   fclose(fp);
192   return 0;
193 }
194
195 int read_upd_socket(int sockfd,char *ptr)
196 {
197  int n;
198   unsigned int tamano_direccion = sizeof( struct sockaddr );
199   char buffer[ MAX ];
200   memset( buffer, 0, MAX );
201   n = recvfrom( sockfd, ptr, MAX-1, MSG_DONTWAIT, (struct
        sockaddr *)&serv_addr_udp,&tamano_direccion );
202   return n;
203 }
204
205 int write_udp(char *ip_str,char *port_str,char *str)
206 {
207   int sockfd, n;
208   unsigned int tamano_direccion;
209   struct sockaddr_in dest_addr;
210   struct hostent *server;
211
212   server = gethostbyname( ip_str );
213   if ( server == NULL )
214   {
215       return -1;
216   }
217   sockfd = socket( AF_INET, SOCK_DGRAM, 0 );
218   if (sockfd < 0)
219   {
220       return -2;
221   }
222
223   dest_addr.sin_family = AF_INET;
224   dest_addr.sin_port = htons( atoi( port_str ) );
225   dest_addr.sin_addr = *( (struct in_addr *)server->h_addr );
226   memset( &(dest_addr.sin_zero), '\0', 8 );
227
228   tamano_direccion = sizeof( dest_addr );
229   n = sendto( sockfd, (void *)str, MAX, 0, (struct sockaddr
        *)&dest_addr, tamano_direccion );
230   if ( n < 0 )
231   {
232       return -3;
233   }
```

```
234   return 0;
235 }
```

sockets.c

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #include "sockets.h"
7  #define MAX 1460
8
9  void user_login();
10
11 int main(int argc, char *argv[] )
12 {
13   user_login();
14   printf("\033[5;22;36m");
15   printf("Esperando óconexin...");
16   printf("\033[m"); /* stop blinking attribute from further
       text */
17   printf("\n");
18
19   int fd = open_tcp_server("8080");
20   int udp_fd = open_udp_server("8080");
21   char *user_input;
22   if ((user_input = (char *) malloc(MAX * sizeof(char))) ==
       NULL)
23   {
24     exit(1);
25   }
26   char *telemetria_ptr;
27   if ((telemetria_ptr = (char *) calloc(MAX , sizeof(char)))
       == NULL)
28   {
29     exit(1);
30   }
31   printf("\033[1A"); // Move up a line
32   printf("Servidor conectado   \n");
33   sleep(1);
34   do
35   {
36   printf("#");
37   fgets(user_input,MAX,stdin);
38   strtok(user_input, "\n");
39   if (strcmp(user_input,"uf") == 0)
40   {
41       /*Envio comando*/
```

```
42        write_cmd(fd,"uf");
43        sleep(1);
44        send_file(fd,"crosscompile/sat.o");
45        printf("Sale el proceso %d\n",getpid());
46        sleep(10);
47        exit(1);
48    }
49    else if (strcmp(user_input,"ss") == 0)
50    {
51        time_t start, end;
52        double elapsed_time;
53        write_cmd(fd,"ss");
54        start = time(&start);
55        receive_file(fd,"recibido.jpg");
56        end = time(&end);
57        elapsed_time = difftime(end,start);
58        printf("El proceso de transferencia llevo %.0f segundos
              .\n", elapsed_time);
59    }
60    else if (strcmp(user_input,"ot") == 0)
61    {
62        write_cmd(fd,"ot");
63        memset(telemetria_ptr,' ',MAX);
64        sleep(3);
65        read_upd_socket(udp_fd,telemetria_ptr);
66        printf("%s\n",telemetria_ptr);
67    }
68    else if (strcmp(user_input, "exit") == 0)
69    {
70        write_cmd(fd,"exit");
71    }
72    else
73    {
74        printf("Solo se admiten 4 comandos: obtener telemetria,
              "
75         "start scanning, update firmware o exit\n");
76    }
77    }
78    while(strcmp(user_input,"exit") != 0);
79    close(fd);
80    return 1;
81  }
82
83
84  void user_login()
85  {
86    char *u;
87    if ((u = (char *) malloc(MAX * sizeof(char))) == NULL) {
88    printf( "malloc failed: \n");
```

```
89    exit(1);
90    }
91
92    char *p;
93    if ((p = (char *) malloc(MAX * sizeof(char))) == NULL) {
94    printf( "malloc failed: \n");
95    exit(1);
96    }
97
98    int tries_left=3;
99    _Bool auth=0;
100   char passwd[]="admin";
101   char user[]="admin";
102
103   do
104   {
105     printf("$ Ingrese usuario: ");
106     fgets(u,20,stdin);
107     strtok(u, "\n");
108     p=getpass("$ Ingrese ñcontrasea: ");
109     if(strcmp(user,u)==0 && strcmp(passwd,p)==0 )
110     {
111       auth=1;
112     }else{
113       printf("Mala combinacion de usuario y ñcontrasea.\n");
114       tries_left--;
115     }
116   }
117   while(auth ==0 && tries_left!=0);
118
119   if(tries_left==0 || auth == 0)
120   {
121     perror( "mala autenticacion");
122     free(u);
123     free(p);
124     exit(1);
125   }
126 }
```

test_terrena.c

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #include "sockets.h"
7  #define MAX 1460
8
```

```
9  char uf[]= "uf";
10 char ss[]= "ss";
11 char ot[]= "ot";
12 char version[]="2.21\n";
13 void telemetria(char *str);
14 void print_error(char * str);
15
16 int main(int argc, char *argv[] )
17 {
18   // print_error("Inicia el programa");
19   int fd = open_tcp_client(argv[1],"8080");
20   if(fd <0 )
21   {
22     char msg[MAX];
23     sprintf(msg,"Ocurrio error %d fd.",fd);
24     print_error(msg);
25     exit(-1);
26   }
27   char buffer[MAX];
28   // print_error("entro al while principal");
29   do
30   {
31     read_cmd(fd,buffer);
32     strtok(buffer, "\n");
33     if (strcmp(buffer,uf) == 0)
34   {
35       receive_file(fd,"b.out");
36       /*Leo el checksum que me envian*/
37       read_cmd(fd,buffer);
38       popen("cp -f b.out sat.o","w");
39       popen("sudo chmod 777 sat.o","w");
40       close(fd);
41       popen("sudo reboot","w");
42   }
43   else if (strcmp(buffer,ss) == 0)
44   {
45       sleep(1);
46       send_file(fd,"file.jpg");
47   }
48   else if (strcmp(buffer,ot) == 0)
49   {
50       char *tel_ptr;
51       if ((tel_ptr = (char *) malloc(MAX * sizeof(char))) ==
             NULL) {
52         exit(-1);
53       }
54       telemetria(tel_ptr);
55       write_udp(argv[1],"8080",tel_ptr);
56   }
```

```
57    }
58    while(strcmp(buffer,"exit") != 0);
59    close(fd);
60    return 1;
61  }
62
63  void telemetria(char *str)
64  {
65    memset(str,0,MAX);
66    strcat(str,"Id del satelite: 1\n");
67    strcat(str,"Uptime: ");
68    char aux[MAX];
69    FILE * uptime = popen("uptime","r");
70    fgets(aux,MAX,uptime);
71   pclose(uptime);
72    strcat(str,aux);
73    strcat(str,"Version del software: ");
74    strcat(str,version);
75    FILE * memoria = popen("free -h | grep -i \"Mem\"","r");
76    fgets(aux,MAX,memoria);
77    pclose(memoria);
78    strcat(str,aux);
79    FILE * cpu = popen("top -b -n1 | grep -i \"Cpu(s)\"" ,"r");
80    fgets(aux,MAX,cpu);
81    pclose(cpu);
82    strcat(str,aux);
83  }
84
85  void print_error(char * str)
86  {
87    time_t now;
88    time(&now);
89    printf("%s: %s\n",strtok(ctime(&now), "\n"),str);
90  }
```

test_satelite.c

5. Conclusiones

A pesar de la relativa experiencia/confianza en C del alumno, este trabajo se torno complejo para la finalización del mismo, en cuanto el diseño general fue claro desde un comienzo, algunos detalles de implementación respecto a los sockets dilataron el tiempo de entrega esperado. Sin embargo se pudo completar con todos los requerimientos esperados.

6. Apéndices

6.1. Compilación cruzada

La compilación cruzada se logro basandose en una guía online [3]. La misma consta de clonar el repositorio que provee el toolchain necesario y la posterior creación de un proyecto de cmake. Se crean dos archivos para este propósito, los cuales se integran aquí.

```
cmake_minimum_required (VERSION 3.0)
# Name our project
project (sat)
# Add all the *.c files in our source directory to our
    executable output
FILE(GLOB SRC_FILES *.c)
add_executable(sat.o ${SRC_FILES})
```

CMakeLists

```
# Define our host system
SET(CMAKE_SYSTEM_NAME Linux)
SET(CMAKE_SYSTEM_VERSION 1)

# Define the cross compiler locations
SET(CMAKE_C_COMPILER   /home/agustinoli/Documents/SO2/TP1_v3/
    tools/arm-bcm2708/arm-rpi-4.9.3-linux-gnueabihf/bin/arm-
    linux-gnueabihf-gcc)
SET(CMAKE_CXX_COMPILER /home/agustinoli/Documents/SO2/TP1_v3/
    tools/arm-bcm2708/arm-rpi-4.9.3-linux-gnueabihf/bin/arm-
    linux-gnueabihf-gcc)

# Define the sysroot path for the RaspberryPi distribution in
     our tools folder
SET(CMAKE_FIND_ROOT_PATH /home/agustinoli/Documents/SO2/
    TP1_v3/tools/arm-bcm2708/arm-rpi-4.9.3-linux-gnueabihf/arm
    -linux-gnueabihf/sysroot/)

# Use our definitions for compiler tools
```

```
SET(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)

# Search for libraries and headers in the target directories
   only
SET(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
SET(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)

add_definitions(-Wall -pedantic -Werror -std=gnu99)
```

Toolchain-rpi

6.2. Unidad de systemd

Se creó una unidad de systemd como servicio para cumplir el requerimiento de actualización del firmware.[4]

```
[Unit]
Description=Servicio para TP1-SO2
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=5
User=pi
StandardOutput=file:/home/pi/Documents/SO2/TP1_v3/service.log
WorkingDirectory=/home/pi/Documents/SO2/TP1_v3
ExecStart=/home/pi/Documents/SO2/TP1_v3/sat.o tolimanDell.
   local

[Install]
WantedBy=multi-user.target
```

so2-tp1.service