UNIVERSIDAD DE CANTABRIA

DEPARTAMENTO DE INGENIERÍA DE COMUNICACIONES

GRUPO DE INGENIERÍA TELEMÁTICA

PROTOCOLOS PARA LA INTERCONEXIÓN DE REDES

PRÁCTICA 4:
DESARROLLO DE APLICACIONES CLIENTE/SERVIDOR MEDIANTE LA PROGRAMACIÓN DE SOCKETS EN LINUX

1. OBJETIVOS DE LA PRÁCTICA

Los principales objetivos de esta práctica son:

- Apreciar la funcionalidad de la interfaz *socket* como herramienta básica para habilitar la comunicación entre procesos que se encuentran o no en la misma máquina.
- Realizar una aplicación sencilla en lenguaje C basada en el modelo cliente/servidor, sobre una conexión TCP o utilizando UDP.
- Analizar las diferentes fases de la comunicación, a través de la aplicación *netstat.*
- Estudiar la interacción entre los dos procesos mediante la monitorización del intercambio de paquetes y tramas con el analizador de protocolos Wireshark.

2. INTRODUCCIÓN

Cada alumno deberá desarrollar un servicio basados en la arquitectura Cliente-Servidor.

Los programas servidor, se ejecutarán en el PC local mientras que las aplicaciones cliente podrán ejecutarse desde cualquier PC del laboratorio.

El Sistema Operativo sobre el que se va a trabajar es *Linux Ubuntu 12.04 LTS*, y el compilador C es el de GNU. La compilación y *enlazado* de un fichero fuente se realizará, por tanto, mediante el comando *gcc*:

```
gcc <nombre_fichero_fuente> -o <nombre_fichero_ejecutable>
```

2.1 POSIBLES SERVICIOS

El alumno seleccionará el servicio que desarrollará de entre la siguiente lista:

- Calculadora en red: El usuario, a través del programa cliente, enviará al servidor peticiones para el cálculo de operaciones simples (suma, resta, producto, división) sobre dos números enteros. El programa servidor, una vez hecho el cálculo, devolverá el resultado correspondiente. El programa cliente mostrará por pantalla al usuario este resultado.
- Servicio de fecha y hora: El usuario, a través del programa cliente, enviará al servidor peticiones para conocer la fecha y hora. El programa servidor devolverá la fecha y hora. El programa cliente mostrará por pantalla al usuario este resultado.
- Generador de números aleatorios: El usuario, a través del programa cliente, enviará al servidor peticiones para que éste genere el conjunto de números primos entre cualesquiera dos valores indicados por el usuario. El programa servidor devolverá tantos números aleatorios como haya en el rango definido por el usuario hasta un máximo de 5. El programa cliente mostrará por pantalla al usuario todos los números.

El alumno decidirá si el servicio utilizará TCP o UDP como protocolo de transporte.

3. DESARROLLO DE LA PRÁCTICA

Para que el desarrollo de la práctica se asemeje más a cualquier protocolo de comunicaciones, el alumno deberá definir un protocolo especificando tanto que PDUs se

intercambian cliente y servidor en cada fase del servicio como el formato de las PDU intercambiadas.

Para la primera parte, el alumno deberá generar un diagrama temporal que ejemplifique el acceso, uso y cierre del servicio.

Para la segunda parte, el alumno deberá definir los campos de la PDU que usará en el intercambio de información entre cliente y servidor.

En todo caso, las PDU intercambiadas tendrán que tener una cabecera cuyo formato obedece al de la siguiente estructura:

```
struct header {
      unsigned char packetType;
      unsigned char id;
      unsigned char len;
};
```

- La variable *packetType* puede tomar diferentes valores (se recomienda utilizar constantes) en función de que tipo de PDU se trate (Ej: petición de servicio, confirmación de registro, petición de cierre, comando, respuesta, error, etc.).
- La variable *id* es un identificador único para la sesión de ese usuario que debe asignar el servidor al cliente en el momento que éste acceda al servicio.
- La variable *len* contiene el tamaño completo de la PDU incluyendo la cabecera.

3.1 DESARROLLO DEL SERVICIO

Los principales pasos para la implementación del servicio se describen a continuación.

Servidor

Las acciones que debe llevar a cabo el programa servidor son:

1. **Abrir** un *socket* a la capa de transporte y **asignarle** un puerto local, **informando** a la red tanto de este valor como de la disposición del servidor a aceptar peticiones de servicio.

2. **Esperar** que un cliente se comunique con él. Cuando reciba la petición de servicio se deberá presentar por pantalla la dirección IP del cliente con el siguiente formato: "**Atendiendo al cliente: [dirección IP del cliente]"**. El servidor atenderá la sesión del cliente.

3. Cuando el cliente haya finalizado su sesión se volverá al punto 2 para **esperar** nuevas peticiones de establecimiento de conexión por parte de nuevos clientes.

Cliente

Las acciones que debe llevar a cabo el programa cliente son:

1. **Abrir** un socket y **conectarse** a la dirección de red asignada al servidor. Esta dirección debe ser conocida por el cliente y responderá al esquema de generación de direcciones de la **familia de sockets** que se está utilizando. La dirección IP

del servidor se introducirá como argumento en la línea de comandos, al ejecutar la aplicación cliente.

2. Una vez establecida la comunicación, el cliente deberá primero registrarse y esperar la respuesta por parte del servidor antes de poder empezar a enviar comandos.

3. Cuando el usuario desee abandonar la sesión, el cliente deberá informar de esto al servidor antes de cerrarse.

4. MONITORIZACIÓN

Una vez realizados los programas y comprobado su correcto funcionamiento, ejecutarlos de nuevo monitorizando la comunicación entre los procesos mediante el analizador de protocolos. Analizar las capturas, explicando el intercambio de mensajes. La comunicación se monitorizará tanto de manera local como remota.
Por otro lado, se recomienda emplear el comando *netstat*, para comprobar, en cada momento, el estado de las conexiones.

ANEXO 1. Función calculadora

Esta función devuelve el resultado de la operación indicada a través del parámetro `operator` cuando se realiza utilizando `op1` como primer operando y `op2` como segundo operando.

Las operaciones quedan especificadas a través de las sentencias `#define`

```
#define SUMA 1
#define RESTA 2
#define PRODUCTO 3
#define DIVISION 4

float calculator(int op1, int op2, int operator)
{

float ret;

   switch(operator) {
        case SUMA:
                ret = op1 + op2;
                return ret;

        case RESTA:
                ret = op1 - op2;
                return ret;

        case PRODUCTO:
                ret = op1 * op2;
                return ret;

        case DIVISION:
                ret = (float)op1 / (float)op2;
                return ret;

        default:
                return -1;
   }
}
```

Un ejemplo de llamada a esta función podría ser el siguiente:

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

int min = 3;
int max = 4;
float calc;

calc = calculator(min, max, DIVISION);
printf("Resultado: %f\n", calc);

return 0;
}
```

Que mostraría por pantalla lo siguiente:

```
Resultado: 0.75
```

ANEXO 2. Función generadora de fecha y hora

Esta función no retorna ningún valor.

La función almacena en los parámetros de entrada los valores de la fecha y hora actuales.

```
void get_date_time(int *year, int *month, int *day, int *hour, int
*minute, int *second, char *timezone)
{
  struct tm *now = NULL;
  time_t time_value = 0;

  time_value = time(NULL);          /* Get time value              */
  now = localtime(&time_value);     /* Get time and date structure */

  *hour = now->tm_hour;              /* Save the hour value         */
  *minute = now->tm_min;             /* Save the minutes value      */
  *second = now->tm_sec;             /* Save the seconds value      */

  *year = now->tm_year + 1900;       /* Save the year value         */
  *month = now->tm_mon + 1;          /* Save the month value        */
  *day = now->tm_mday;               /* Save the day value          */

  strcpy(timezone, now->tm_zone);   /* Save the timezone value    */

}
```

Un ejemplo de llamada a esta función podría ser el siguiente:

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

int year, month, day, hour, minute, second;
char timezone[4];

get_date_time(&year, &month, &day, &hour, &minute, &second, timezone);

printf("%d-%d-%d ", year, month, day);
printf("%d:%d:%d %s",hour, minute, second, timezone);

return 0;
}
```

Que mostraría por pantalla lo siguiente:

```
2012-11-20 13:22:14 CET
```

ANEXO 3. Función generadora de números primos

Esta función devuelve el número de números primos, entre `numberMin` y `numbreMax`, que ha generado.

Como máximo generará 5 números primos.

El parámetro vector debe ser un array de enteros en el cual la función almacenará los números primos generados.

```
int prime_generator(int numberMin, int numberMax, int *vector) {
    unsigned int i, j, k;
    int pn;
    int index = 0;

    i = numberMin;
    j = 0;
    while (i <= numberMax) {
        j = i;
        k = 2;
        pn = 1;
        while (k < j) {
            if (j % k == 0) {
                pn = 0;
            }
            ++k;
        }
        if (pn > 0) {
            vector[index] = j;
            index++;
            if(index == 5)
                return index;
        }
        ++i;
    }
    return index;
}
```

Un ejemplo de llamada a esta función podría ser el siguiente:

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

int ret;
int vector[5];
int i, min = 2, max = 7;

ret = prime_generator(min, max, vector);

printf("%d numeros primos\n", ret);
for(i=0; i<ret; i++) {
   printf("%d. %d; ", i+1, vector[i]);
}
return 0;
}
```

Que mostraría por pantalla lo siguiente:

```
4 numeros primos
1. 2; 2. 3; 3. 5;4. 7;
```