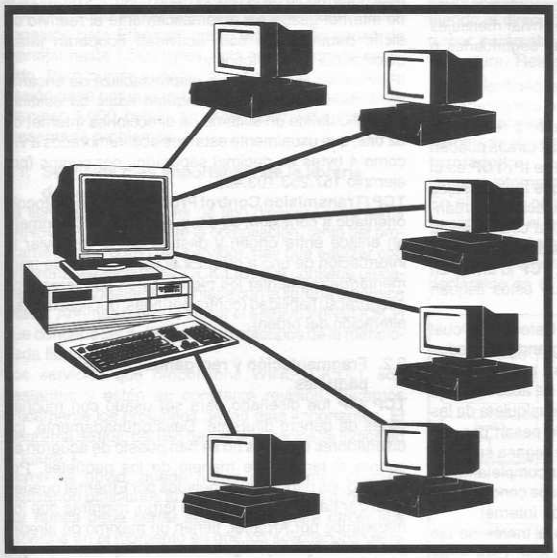
Comunicación entre programas a través de Internet



- **Apolinar González Potes, M.Sc.**
  Ingeniero Electrónico,
  Profesor Auxiliar - Universidad del Valle
- **Héctor Angulo Bulla, Esp.**
  Ingeniero Electricista
  Profesor Auxiliar - Universidad del Valle
- **Angel García Baños**
  Ingeniero de Telecomunicación,
  Profesor Asistente - Universidad del Valle

* Este texto se recibió, para ser publicado, en julio de 1996.

RESUMEN

Se hace una introducción al protocolo TCP/IP de comunicación entre computadores vía Internet y se muestran tres aplicaciones desarrolladas por los autores.

ABSTRACT

An introduction about computer communication TCP/IP protocol is made and then three applications developed by the authors are shown.

PALABRAS CLAVES

TCP/IP, Internet, comunicación, WINSOCK.DLL, C, C++, Windows

1. INTRODUCCIÓN

Hay una nueva puerta abierta para los programadores: la creación de aplicaciones distribuidas que se comunican a través de una librería de funciones (**WINSOCK.DLL**). Ya hay ejemplos muy conocidos de estas aplicaciones como el ftp, el telnet y el mismo netscape. Pero el programador no está limitado a ellas sino que puede crear nuevos servicios, según le guíe su imaginación.

Presentaremos una breve introducción al protocolo TCP/IP de Internet y a la librería de sockets **WINSOCK.DLL**, que posibilita acceder a los servicios de la red con nuestras aplicaciones escritas en lenguaje C o C++. Y mostraremos tres aplicaciones concretas implementadas usando estos servicios.

2. EL PROTOCOLO TCP/IP

TCP / IP es un protocolo desarrollado para permitir la cooperación entre los proveedores de hardware y software, para gestionar las redes. Este protocolo fue desarrollado por una comunidad de investigadores centrados alrededor de Arpanet y es más conocido como **«TCP / IP Network»**.

Aunque **TCP / IP** parece ser el término más común en nuestros días, muchos documentos se refieren a él como «Internet protocols». Internet es una colección mundial de redes que incluye Arpanet, NSFnet, redes regionales tales como Nysernet, redes locales que se encuentran generalmente en universidades e instituciones de investigación y algunas redes de aplicación militar. Todas estas redes están interconectadas, entre ellas, de modo que un usuario puede enviar mensajes de unas a otras, excepto donde hay seguridades o restricciones en el control de acceso.

Muchos protocolos realizan tareas especificas, tales como transferencia de archivos entre computadores, envío de correos o tratamiento de datos o resultados ubicados en otros computadores. Estas tareas pueden ser realizadas por los protocolos **TCP** e **IP**. **TCP** es el protocolo responsable de asegurar que los mensajes han llegado al otro lado de una comunicación, guardándolos para retransmitirlos si por alguna circunstancia no llegaron a su destino. Si un mensaje es muy largo para ser enviado en un solo paquete, **TCP** lo divide en diversos paquetes y se asegura que estos lleguen correctamente a su destino.

TCP / IP está basado en el modelo «Catenet», el cual asume que hay un número muy grande de redes independientes conectadas a la vez por diferentes entradas. El usuario debe ser capaz de acceder a los computadores o a otros recursos en cualquiera de las redes. Como los paquetes a menudo pasan por una docena de redes diferentes, antes de llegar a su destino, la asignación de la ruta debe ser completamente invisible al usuario. Todo lo que éste debe conocer para acceder a otro sistema es su dirección Internet.

El proceso de transmisión de datos a través de las redes es una tarea bastante complicada. Esta tarea generalmente se fragmenta en componentes simples que construyen otras para realizar toda la tarea. Para manejar la diversidad de tareas requeridas por las redes, muchos software de redes dividen los requerimientos en series de niveles. Las reglas y convenciones implementadas para cada nivel son conocidas colectivamente como niveles de «protocolos». Existe un estándar conocido como OSI constituido por 7 niveles jerárquicos, en cada uno de los cuales se ofrece un servicio. TCP/IP no es compatible con OSI, pero es un protocolo más antiguo y más desarrollado. El protocolo IP es una mezcla de los niveles 2 y 3 de OSI, mientras que el protocolo TCP es similar al nivel 4.

2.1. TCP/IP y transferencia de información

La información es transferida en paquetes y cada uno de estos es enviado, individualmente, a través de la red, que gestiona cada uno como entidades separadas. Por ejemplo, para transferir un archivo que tiene una longitud de 10.000 bytes, el protocolo divide el archivo, por ejemplo en 20 paquetes de 500 bytes y los envía individualmente, para luego ser reubicados en el destino y formar nuevamente el archivo de 10.000 bytes. Durante el tránsito de los paquetes, la red no reconoce ninguna conexión entre ellos. El paquete número 10 puede, por ejemplo, llegar antes que el número 13 o incluso puede llegar duplicado. También es posible que algún paquete llegue con errores a algún nodo intermedio de la red. En este caso, los protocolos de Internet gestionan automáticamente el reenvío de dicho paquete. En esta actividad cooperan jerárquicamente dos protocolos:

IP (Internet Protocol): se responsabiliza de encaminar individualmente cada paquete hacia su destino. Para ello define un sistema de direcciones Internet de 32 bits, que usualmente estamos acostumbrados a ver como 4 bytes en decimal separados por puntos (por ejemplo 157.253.103.42).

TCP (Transmision Control Protocol): es un protocolo orientado a conexión, es decir, que establece primero un enlace entre origen y destino antes de enviar la información de uno a otro. Es el responsable de fragmentar/defragmentar los mensajes en paquetes y de asegurar su fiabilidad (contra pérdidas, duplicaciones o alteración del orden).

2.2. Fragmentación y reorganización de paquetes

TCP / IP fue diseñado para ser usado con muchas redes de género diferente. Desafortunadamente, los diseñadores de redes no se han puesto de acuerdo en cuanto al tamaño de manejo de los paquetes. Por ejemplo, los paquetes manejados por Ethernet pueden alcanzar hasta 1.500 bytes de largo, mientras que los manejados por Arpanet, tienen un máximo de alrededor 1.000 bytes y algunas redes rápidas manejan tamaños de paquetes mucho mas grandes. El protocolo **IP** tendría problemas de realización bastante serios si no se le asignan tamaños adecuados para el manejo de paquetes. Como el manejo de paquetes grandes es mucho más eficiente que el de los pequeños, la posibilidad de adaptarse cada vez a la mejor posibilidad sería lo más deseado. Por otro lado, el protocolo **TCP** tiene la capacidad de «negociar» el tamaño del paquete. Así, cuando se abre una conexión con **TCP**, ambos terminales envían el tamaño del máximo paquete que pue-

den manejar y escogen el más pequeño entre las dos respuestas para continuar la conexión.

3. SOCKETS

Si se quiere enviar un archivo de un computador a otro, necesariamente debemos conocer la dirección Internet del destino y el servicio especializado en hacer dicha transferencia, por ejemplo, en este caso el **ftp** (*file transfer protocol*). Otro servicio muy usado es el **telnet** para comunicación con terminales remotos.

Los sockets son dispositivos de comunicación sobre los cuales se puede enviar o recibir datos. Asociado a cada socket hay una tripleta: la dirección Internet, el servicio que ofrece y el protocolo (para el caso de Internet se trata del TCP, pero hay otros).

Esta situación no está completamente resuelta. Por ejemplo, suponiendo que dos computadores hacen conexión bajo Ethernet, como esta red es capaz de manejar hasta 1.500 bytes, éste será el tamaño escogido. Pero puede suceder que en algún punto la conexión pase por una red Arpanet capaz solamente de manejar 1.000 bytes, en cuyo caso ésta conexión presentaría problemas.

3.1. Servicios mas importantes de la librería de sockets

La librería de sockets es de tipo dinámica y se llama WINSOCK.DLL. Durante la compilación, tanto del programa cliente como del servidor, hay que enlazar con la librería estática WINSOCK.LIB, que contiene únicamente los esqueletos de enlace con la librería dinámica citada. Además hay que incluir el archivo WINSOCK.H que contiene las definiciones y prototipos de la mencionada librería.

Los servicios que proporciona WINSOCK.DLL son bastantes y están en constante revisión. Veremos únicamente los prototipos de las más importantes. El diagrama de flujo del uso de estos servicios se indica en la figura 1.

Como es obvio, puede haber uno o más clientes realizando peticiones al servidor. En principio, el servidor nunca debe cerrar sus sockets, para ofrcer sus servicios en el momento en que los soliciten los clientes.

Las funciones indicadas en la figura son:

int PASCAL FAR WSAStártup (WORD wVersionRequested, LPWSADATA lp WSAData);

Es obligatoriamente la primera llamada que hay que hacer. Se especifica, en el primer parámetro, la versión requerida de la librería. Como segundo parámetro se pasa un puntero a una estructura que rellenará la función, con información de detalles acerca de la implementación. Retorna un código de error (0 si no hay errores).

SOCKET PASCAL FAR socket(int af, int type, int protocol);

Abre un nuevo socket.Como primer parámetro de entrada recibe el tipo de formato de la dirección; actualmente sólo está soportado el AF_INET, que es el formato Internet. El segundo parámetro de entrada especifica el tipo de socket, que puede ser SOCK_STREAM (para TCP) o SOCK_DGRAM (para UDP). El tercer parámetro de entrada usualmente se especifica a 0 para utilizar el protocolo por defecto. Esta función abre un socket y retorna un identificador a dicho socket.

int PASCAL FAR bind(SOCKET s, const struct sockaddr far *name, int namelen);

Asocia un socket con la dirección local del computador. En el primer parámetro se especifica el socket. En el segundo parámetro se especifica una estructura conteniendo la dirección IP asignada al computador. En el tercer parámetro se especifica el tamaño de dicha estructura. Retorna un código de error (0 si no hay errores).

struct servent FAR *PASCAL FAR getservbyname(const char FAR *service, const char FAR *protocol);

Obtiene información del servicio solicitado u ofrecido. En el primer parámetro se especifica el nombre del servicio. En el segundo parámetro se especifica el protocolo. Retorna un puntero a una estructura conteniendo el número del servicio. Los servicios deben declararse en el fichero SERVICES que usualmente

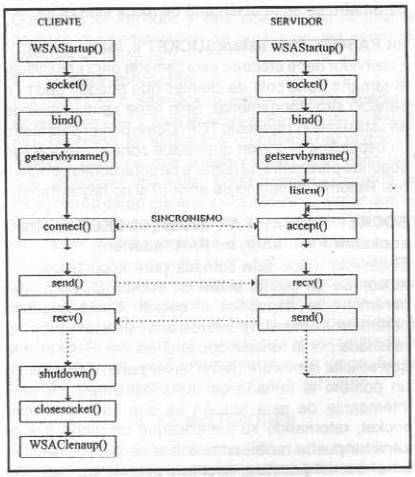


Figura 1.

está en el directorio de WINDOWS (o en el SLIP, si se está conectado a Internet vía modem). Nombres de servicio comunes que pueden encontrarse allí, con su número de port y protocolo asociados son, entre otros:

```
ftp             21/tcp
telnet          23/tcp
time            37/tcp  timserver
time            37/udp  timserver
nameserver      42/tcp  name        # IEN 116
whois           43/tcp  nicname
domain          53/tcp  nameserver #name-domain
serverdomain    53/udp  nameserver
finger          79/tcp
ns              105/tcp             # ph name server
```

Cada programador puede hacer uso de estos servicios. Pero la verdadera potencialidad de este fichero es que se pueden añadir nuevos servicios asociándolos a números de port que no estén usados. Para las aplicaciones que mostraremos más adelante, nosotros añadimos:

```
controlador     121/tcp
comelon         122/tcp
```

int PASCAL FAR connect(SOCKET s, const struct sockaddr FAR *name, int namelen);
Esta llamada la realiza cada cliente para establecer conexión con el servidor. Como primer parámetro se especifica el socket que se desea conectar. Como segundo parámetro se especifica una estructura que contiene la dirección IP del servidor al que se quiere contactar, así como el servicio solicitado. Como tercer parámetro se pasa el tamaño de dicha estructura.

int PASCAL FAR listen(SOCKET s, int backlog);
El servidor debe efectuar esta llamada para especificar el tamaño de la cola de clientes que pueden pedir un servicio simultáneamente. Sólo tiene sentido cuando se está usando protocolo TCP. Como primer parámetro se especifica el socket que recibe conexiones y como segundo parámetro, el número de conexiones pendientes. Retorna un código de error (0 si no hay errores).

SOCKET PASCAL FAR accept(SOCKET s, struct sockaddr FAR *addr, int FAR *addrlen);
El servidor hace esta llamada para aceptar una conexión de un cliente sobre un socket. Como primer parámetro se especifica el socket. Como segundo parámetro se pasa un puntero a una estructura que será rellenada por la función con la dirección IP del cliente que solicita conexión. Como tercer parámetro se pasa un puntero al tamaño de dicha estructura. Lo más interesante de esta función es que crea un nuevo socket, retornando su identificador; de modo que el servidor puede recibir conexiones de nuevos clientes en el socket primitivo, mientras atiende los servicios solicitados en cada nuevo socket que va creando. Con técnicas de programación adecuadas (multi-threads en Windows-NT o bien usando el propio bucle de dispatch de mensajes de Windows 3.11) se puede crear un servidor múltiple, es decir, que tenga la capacidad de atender simultáneamente a varios clientes.

int PASCAL FAR send(SOCKET s, const char FAR *buffer, int len, int flags);

Sirve para enviar bytes a través de un socket previamente conectado. El primer parámetro especifica el socket. El segundo, el buffer donde están los bytes a enviar. El tercero, el tamaño del buffer. Y, el cuarto, especifica el modo en que se hace el envío, que usualmente se pone a 0. Si no hay errores, esta función retorna el número de bytes que se enviaron efectivamente, que no tiene por qué coincidir con el solicitado. El programador debe incluir esta función en un bucle, de modo que reintente la transmisión de los datos que no se lograron enviar en la anterior iteración.

int PASCAL FAR recv(SOCKET s, char FAR *buffer, int len, int flags);
Sirve para recibir el número de bytes especificados (como tercer parámetro), en el buffer (segundo parámetro) y recibidos en el socket indicado (primer parámetro). El cuarto parámetro usualmente se pone a 0. Al igual que la función send(), retorna el número de bytes que efectivamente se recibieron, por lo cua es recomendable meter esta función en un bucle del cual se salga cuando no queden más bytes por llegar.

int PASCAL FAR shutdown(SOCKET s, int how);
Sirve para deshabilitar la recepción (si how es 0), el envío (si how es 1) o ambos (si how es 2), del socket especificado como primer parámetro.

int PASCAL FAR closesocket(SOCKET s);
Cierra el socket especificado.

int PASCAL FAR WSACleanup(void);
Con esta función se termina el uso de la librería de sockets WINSOCK.DLL. Debe ser, pues, la última función que se llame en nuestra aplicación.

int PASCAL FAR WSAGetLastError();
En cualquier momento, a lo largo del programa cliente o servidor, se puede invocar esta función para obtener el código del último error que se haya producido. En general, el manejo (y especialmente la recuperación) de errores es bastante complejo en esta librería.

4. APLICACIONES

4.1. Primera aplicación

Se desarrolló una aplicación escrita en lenguaje C, en la cual puede haber uno o más computadores clientes,

con baja potencia de cómputo, cuya misión es tomar datos de sensores, periódicamente, de un proceso a controlar. Estos datos se envían a un computador servidor, de mayor potencia, que calcula una ley de control y devuelve a cada respectivo cliente el valor de actuación necesario para mantener el proceso dentro de la consigna que se desee.
Esta aplicación corría (tanto el programa cliente como el servidor) en Windows pero haciendo uso de la ventana por defecto. Es decir, realmente era una aplicación al estilo D.O.S.

4.2. Segunda aplicación

Posteriormente, se adaptó la misma aplicación para que corriera realmente en Windows. Esto introdujo ciertos problemas debido a que los **sockets** no son reentrantes mientras que Windows si lo es.
Concretando, el problema principal es que casi todas las funciones de sockets son bloqueantes, es decir, la función no retorna hasta que se ha realizado en su totalidad. Esto es especialmente crítico en el servidor, ya que si, por ejemplo, entra a un **accept()** en espera de una conexión de un cliente, todo el sistema operativo de Windows se bloquea hasta que efectivamente algún cliente establezca conexión. Esto es contrario a la filosofía de programación de Windows que exige que no haya bucles «infinitos» y que las respuestas a los mensajes sean lo más rápidas posible.
La propia librería **WINSOCK.DLL** nos ofrece una función para solventar esta dificultad. Se trata de:

int PASCAL FAR WSAAsyncSelect(SOCKET s, HWND hWnd, unsigned int wMsg, long lEvent);

por medio de la cual se puede añadir a Windows un nuevo mensaje, que se emitirá automáticamente cada vez que ocurra alguno de los siguientes eventos:

- Petición de un cliente de nueva conexión
- Listo para escribir (se enviaron todos los datos pendientes)
- Listo para leer (llegaron nuevos datos).

Entonces, cada vez que llegue, a través de Windows, un mensaje de este tipo, el servidor chequeará cual de estos eventos fue el causante y entrará respectivamente a las funciones siguientes, con la seguridad de que se completarán inmediatamente, evitándose bloqueos:

- **accept( )**
- **send( )**
- **recv( )**

4.2.1. Características de la apli cación cliente-servidor de control para windows

Se puede seleccionar en el cliente la ley de control y la función de transferencia de la planta (ambos como numerador y denominador en z, al estilo Matlab). Cuando se seleccione la opción de **comienzo**, todos estos parámetros se enviarán al **servidor**.
En la ventana del **servidor** aparecerá la dirección IP del **cliente** que lo ha contactado.
Puede haber varios **clientes** ejecutándose simultáneamente en varios (o en el mismo) computadores.
Los parámetros de planta y ley de control los almacena el **servidor** internamente, para cada **cliente** activo.
El set-point de cada **cliente** se puede cambiar en tiempo real, accionando en su ventana las flechas de *scroll vertical* con el *mouse*.
En cada instante de muestreo (seleccionable por el usuario en cada **cliente**) se envía al **servidor** el **set-point** y la variable medida de **salida**. El **servidor** aplica la ley de control y devuelve la señal de **actuacion**. Todo ello se gráfica en tiempo real en la ventana del **cliente**:

- **set point = verde**
- **salida = rojo**
- **actuacion = azul**

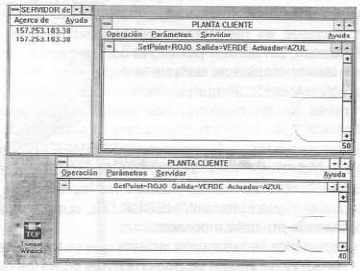


Figura 2

Corriendo el servidor y dos clientes, en la misma pantalla del computador, se verá algo así como lo indicado en la figura 2.

4.3. Tercera aplicación

Por último, se desarrolló una aplicación más divertida, en lenguaje C++ y para Windows. Se trata de un juego muy conocido, en el que hay un tablero por el que se mueven gusanos que pueden agrandarse cada vez que se comen una fruta. Cada jugador comanda el movimiento de un gusano y el objetivo es lograr que el gusano propio se agrande lo más posible, pero sin chocarse con las paredes o con otros gusanos.
Cada jugador acciona las flechas para mover su gusano desde una aplicación cliente. El servidor viene a ser como el árbitro del juego: recibe peticiones de movimiento de cada gusano, evalúa si ellas dan lugar a choques o a comer frutas y devuelve como resultado la nueva posición resultante del tablero de juego. En la figura 3 puede verse una instantánea del juego.

5. CONCLUSIONES

La experiencia que obtuvimos con este trabajo fue bastante positiva ya que se abre la posibilidad de crear aplicaciones distribuidas en una red de computadores.

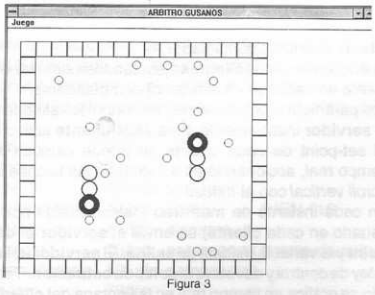


Figura 3

En cuanto a lo negativo, está la dificultad para depurar estas aplicaciones, por cuanto los compiladores de C tradicionales no proporcionan una herramienta adecuada para ello. Y lo complejo que es hacer el tratamiento adecuado de errores, especialmente si lo que se persigue es recuperarse en el caso de que se produzcan, sin abortar por ello la comunicación.
Una táctica empleada, durante la depuración, fue utilizar el protocolo SLIP (que permite la conexión a Internet, a través de un modem), por medio del programa TRUMPET, y corriendo el cliente y el servidor en el mismo computador. Este programa TRUMPET proporciona varias opciones de depuración, abriendo una ventana sobre la cual se pueden ver las llamadas a funciones de la librería WINSOCK.DLL que se están efectuando en cada momento.
Es importante señalar que los datos que se transmiten con send() y recv() son bloques de bytes sobre los que no se hace ningún chequeo de tipo; por tanto, es muy importante que el cliente y servidor envíen-reciban el mismo número de bytes y que correspondan al mismo tipo de variables (float, int, etc.). Si esto no se cumple el caos que se organiza es fenomenal. Este es uno de los principales puntos débiles (en cuanto a la metodología de programación) de la librería de sockets presentada. Para solventarlo se está en el proceso de definición de un nuevo estándar internacional de comunicación cliente-servidor, orientado a objetos, y cuyo nombre es CORBA (Common Object Request Broker Architecture).

6. BIBLIOGRAFÍA

[1]. —,SCO TCP/IP and SCO NFS: Configuration and Administration Student Guide. SCO Open Systems Software.
[2]. BENNET, S. Real-Time Computer Control. An Introduction. Prentice Hall International Series in Systemes and Control Engineering, 1994.
[3]. VILA,J. Notas del Curso de Sistemas Operativos. Doctorado en Automática e Informática Industrial, 1995.
[4]. TANENBAUM, A. S., Sistemas Operativos, diseño e implementación. Prentice - Hall Hispanoamericana, S. A, México, 1988.
[5]. ALCALDE, MORERA, J. PEREZ J.A. CAMPANERO, Introducción a los Sistemas Operativos. Mc Graw Hill, 1991.
[6]. MILENKOVIC M., Sistemas operativos, conceptos y diseño. Mc Graw Hill, Madrid., 1989.
[7]. DEILTEL H. M. An Introduction to Operating Systems. Addison Wesley, Reading Massachusetts, 1984.
[8]. RUEDA F. Sistemas Operativos. Mc Graw Hill, Bogotá, 1989.
[9]. COFFMAN E. G. JR AND DENNING P. J. Operating Systems Theory. Prentice-Hall, Englewood Cliffs-NJ, 1973.
[10]. ALLARD, KEITH MOORE, DAVID TREADWELL, Programación avanzada de redes empleando el API de Windows Sockets. Revista Microsoft para programadores, Noviembre 1993

LOS AUTORES

Angel García Baños
Ingeniero de Telecomunicación de la Universidad Politécnica de Madrid. Profesor Asistente del Departamento de Electricidad en la Universidad del Valle. Actualmente cursa estudios de doctorado con la Universidad Politécnica de Valencia. Dirige la Especialización de Computadores y Sistemas Digitales del PPIEEC y la Cátedra de Arquitecturas Digitales. Desarrolla actualmente su trabajo de investigación en la línea «Lógica digital reprogramable basada en FPGAs».
email: angarcia@maxwell.univalle.edu.co

Héctor Angulo Bulla
Ingeniero Electricista. Especialista en Redes de Comunicación de la Universidad del Valle. Actualmente cursa estudios de doctorado con la Universidad Politécnica de Valencia. Profesor Auxiliar del Departamento de Ciencias de la Computación de la Universidad del Valle. Areas de Investigación: Sistemas Operativos de Tiempo Real. Redes Neuronales.
email: hangulo@borabora.univalle.edu.co

Apolinar González Potes
Estudios de pregrado en Ciencias del Ingeniero, Area Electrónica. Maestría en Automática en la Universidad Claude Bernard de Lyon Francia. Profesor Auxiliar del Departamento de Electricidad en la Universidad del Valle. Actualmente cursa estudios de doctorado en Automática e Informática Industrial en la Universidad Politécnica de Valencia, España. Area de investigación «Validación de sistemas en tiempo real con redes de petri coloreadas».
email: apogon@maxwell.univalle.edu.co