1. Direcciones IP

1.1. Direcciones IPv4

Todo ordenador se identifica en Internet con una dirección IP, en la actualidad de tipo IPv4. Veamos como almacenar y realizar cambios de formatos de direcciones IPv4 programando en C.

- *Formato textual*. Es decir, como cadenas de caracteres ASCII, por ejemplo, `char ip[]="193.110.128.200";` (16 caracteres).
- *Formato binario*. Como 4 bytes, que en C podemos considerar de dos formas:

  (1) Cuatro caracteres sin signo, por ejemplo, `uint8_t ip[]={193,110,128,200};` o `unsigned char ip[]={193,110,128,200};`

  (2) Un entero de 32 bits sin signo, por ejemplo, `uint32_t ip=3363860161u` ($193 + 110 \times 2^8 + 128 \times 2^{16} + 200 \times 2^{24}$).
- Es usual que este entero sin signo se encapsule en una estructura de tipo `struct in_addr` que contiene una única componente `uint32_t`.

`struct in_addr`	
`uint32_t s_addr`	dirección IPv4

  Por ejemplo, `struct in_addr ip; ip.s_addr=3363860161u;`
- El campo `uint32_t s_addr` puede contener una dirección IPv4 o bien las macros `INADDR_LOOPBACK` (lazo de vuelta), `INADDR_ANY` (cualquier dirección válida), `INADDR_BROADCAST` (para difusión), `INADDR_NONE` (indica un error).
- Estas estructuras se definen en `netinet/in.h` mientras que las funciones que vamos a ver a continuación están declaradas en `arpa/inet.h`. El tipo `uint32_t` está definido en `stdint.h`, que ya se incluye en `netinet/in.h`.

Funciones de conversión de formatos que sirven para IPv4 e IPv6:

- La función `int inet_pton (int af, const char *cp, void *buf)` convierte una dirección IPv4 o IPv6 desde el formato textual (`cp` es una dirección que apunta al texto de entrada) al binario (`buf` es una dirección que apunta al resultado, es decir, a la estructura de la dirección, siendo responsabilidad del programador asegurarse de que tiene el tamaño y tipo adecuado), `af` debe ser `AF_INET` (para IPv4) o `AF_INET6` (para IPv6). Devuelve 1 en caso de éxito. Un ejemplo de uso para IPv4 sería

  ```
  inet_pton(AF_INET, ip_text, &ip_addr)
  ```

  donde `ip_text` es una cadena de caracteres con el formato textual de entrada e `ip_addr` es una estructura del tipo `in_addr` con la IP de salida en formato binario.

- La función `const char * inet_ntop (int af, const void *cp, char *buf, size_t len)` convierte una dirección IPv4 o IPv6 del formato binario (`cp` es la dirección de memoria donde está la dirección que debe convertirse, es decir de la estructura) al textual (`buf` es una dirección que apunta al resultado, siendo responsabilidad del programador haberle reservado un tamaño `len`), según `af` sea `AF_INET` o `AF_INET6`. El valor devuelto por la función es un puntero al resultado. En caso de error devuelve un puntero NULL. Un ejemplo de uso para IPv4 sería

  ```
  inet_ntop(AF_INET, &ip_addr, ip_text, INET_ADDRSTRLEN)
  ```

  donde `ip_addr` es una estructura del tipo `in_addr` con la entrada en formato binario, `ip_text` es la cadena de caracteres con la salida en formato textual e `INET_ADDRSTRLEN` es una macro que indica la longitud en bytes (caracteres) del formato textual, 16 en IPv4.

Otras funciones de conversión de formatos:

- La función `uint32_t inet_addr (const char *name)` convierte la dirección IPv4 del formato textual (variable `name`) al formato de entero de 32 bits sin signo. Si la entrada no es válida, devuelve `INADDR_NONE`.

- La función `int inet_aton (const char *name, struct in_addr *addr)` convierte la dirección IPv4 del formato textual (variable `name`) al formato de entero de 32 bits sin signo encapsulado (estructura `addr`). La función devuelve un número distinto de cero si la dirección es válida y cero en caso contrario.

  Nota: esta función no existe en Windows, por lo que si queremos que los programas sean más fácilmente portables, es mejor usar la función anterior.

- La función `char * inet_ntoa (struct in_addr addr)` hace la operación inversa a la función anterior: convierte la dirección IPv4 del formato de entero de 32 bits sin signo encapsulado (estructura `addr`) al formato textual. El valor devuelto es un puntero a un buffer estático, por lo que sucesivas llamadas a la función reescriben este buffer.

NOTA: las funciones anteriores utilizan las direcciones IPv4 en el denominado orden de la red, que es el que debe usarse en Internet. Sin embargo, algunos ordenadores, como por ejemplo los PCs, almacenan los bytes en el orden contrario y entonces hay que preocuparse de reordenar los bytes antes de usarlos en los programas.

Las funciones anteriores dan los resultados en el orden correcto, excepto para las macros `INADDR_LOOPBACK`, `INADDR_ANY`, `INADDR_BROADCAST` e `INADDR_NONE`. Las funciones de reordenamiento de bytes las veremos en un apartado posterior.

- La función `uint32_t inet_network (const char *name)` convierte de formato textual a un entero en orden de host. En caso de una entrada no válida devuelve -1.

Partes de host y de red de una dirección IP.

- La parte de red de una dirección IP es aquella parte que es común para toda la red, mientras que la parte de host es la que numera cada uno de los hosts. Una dirección IP es de clase A cuando su primer byte empieza por el bit 0 (por ejemplo 12.231.34.5, 12=00001100), de clase B si empieza por 10 (por ejemplo 160.21.201.23, 160=10100000), y de clase C si empieza por 110 (por ejemplo 193.144.84.171, 193=11000001).

- La función `uint32_t inet_lnaof (struct in_addr addr)` devuelve la parte de host de la dirección IPv4.

- La función `uint32_t inet_netof (struct in_addr addr)` devuelve la parte de red de la dirección IP. Sólo funciona con IPv4 de clases A, B y C.

- La función `struct in_addr inet_makeaddr (uint32_t net, uint32_t local)` compone una dirección IPv4 combinando las partes de host y red.

- NOTA: en las tres funciones anteriores las direcciones IP se representan en orden de bytes de red (orden correcto), mientras que las partes de hosts y de red se representan en orden de bytes de hosts (orden inverso).

1.2. Direcciones IPv6

En un futuro próximo, se espera que los ordenadores se identifiquen en Internet con direcciones IPv6 (algunos ya lo hacen ahora). Veamos como trabajar en C con direcciones IPv6.

- Las direcciones IPv6 están compuestas por 128 bits que se suelen escribir como 8 números hexadecimales de 16 bits separados por : (dos puntos), por ejemplo, "1080:0:0:0:8:800:200C:417A". Las secuencias de ceros consecutivos puede abreviarse escribiendo ::, por ejemplo, la dirección IPv6 del lazo de vuelta (*loopback*), "0:0:0:0:0:0:0:1", puede abreviarse "::1".

- Las direcciones IPv6 se almacenan como estructuras `in6_addr` a las que se puede acceder (vía una union) en una gran variedad de formas. Una de ellas es como 16 bytes, es decir, 16 enteros sin signo de 8 bits del tipo `uint8_t`.

- Se definen dos constantes de tipo `struct in6_addr`, la primera es la constante `in6addr_loopback` que contiene la dirección del lazo de vuelta IPv6, "::1", mientras que la segunda es la constante `in6addr_any`, que contiene

"::", es decir una dirección no especificada. Se proporcionan también dos macros: `IN6ADDR_LOOPBACK_INIT` e `IN6ADDR_ANY_INIT` que se utilizan para iniciar las variables del usuario a los valores anteriores.

- Esta estructura se define en `netinet/in.h`, mientras que las siguientes funciones se definen en `arpa/inet.h`.
- En formato textual necesitan una cadena de caracteres de 46 bytes (mejor utilizar la macro `INET6_ADDRSTRLEN`). La explicación en el RFC 4291, sección 2.2.

1.3. Direcciones de las interfaces

Existen funciones en C que permiten obtener las direcciones IP (ya sea IPv4 o IPv6) asociadas a las interfaces del equipo.

- La función `int getifaddrs(struct ifaddrs **addrs)` crea en `addrs` una lista enlazada de estructuras (definidas en `sys/types.h` y `ifaddrs.h`) que describen las interfaces de red del sistema local. La función devuelve 0 en caso de éxito y -1 en caso de error. Algunos de sus componentes son:

`struct ifaddrs`	
`struct ifaddrs *ifa_next`	siguiente elemento de la lista
`char *ifa_name`	nombre de la interfaz
`unsigned int ifa_flags`	indicadores
`struct sockaddr *ifa_addr`	dirección de la interfaz
`struct sockaddr *ifa_netmask`	máscara de la interfaz

  Para recorrer la lista enlazada se puede usar un lazo del tipo

  `for (item = addrs; item != NULL; item = item->ifa_next).`

- La estructura `struct sockaddr` es una estructura genérica, que usaremos en las prácticas de sockets, que permite almacenar la dirección de la interfaz, ya sea IPv4 o IPv6. En realidad hay tres formatos distintos de direcciones, pero en la llamada a la función se convierten a esta estructura genérica (así no hay que usar funciones distintas para cada tipo de socket). El campo que necesitamos evaluar para saber si es IPv4 o IPv6 es:

`struct sockaddr`	
`sa_family_t sa_family`	aquí se almacena el valor `AF_INET` o `AF_INET6`

  A continuación se muestra para los formatos de IPv4 e IPv6:

  - Dirección de un socket IPv4.

`struct sockaddr_in`	
`sa_family_t sin_family`	aquí se almacena el valor `AF_INET`
`struct in_addr sin_addr`	la dirección IPv4 (ver el apartado)
`uint16_t sin_port`	el número de puerto (ver el apartado)

Su tamaño se computa con: `sizeof(struct sockaddr_in)`.

- Dirección de un socket IPv6.

`struct sockaddr_in6`	
`sa_family_t sin6_family`	aquí se almacena el valor `AF_INET6`
`struct in6_addr sin6_addr`	la dirección IPv6 (ver el apartado)
`uint32_t sin6_flowinfo`	esta componente de momento no se usa
`uint16_t sin6_port`	el número de puerto (ver el apartado)

Su tamaño se computa con: `sizeof(struct sockaddr_in6)`.

- Se puede usar un `switch` para comprobar si es IPv4 o IPv6. Si la dirección que se obtiene es IPv4 se puede hacer el *cast* a `struct sockaddr_in` y, si es IPv6, a `struct sockaddr_in6`.

```
switch (item->ifa_addr->sa_family) // IPv4 o IPv6
{
    case AF_INET:  // IPv4
    {
        struct sockaddr_in *s4 = (struct sockaddr_in *)item->ifa_addr;
        // Acceder a los campos de la estructura sockaddr_in
        break;
    }
    case AF_INET6: // IPv6
    {
        struct sockaddr_in6 *s6 = (struct sockaddr_in6 *)item->ifa_addr;
        // Acceder a los campos de la estructura sockaddr_in6
        break;
    }
}
```

  Usar la función `inet_ntop` para pasar a formato textual las direcciones IP y las máscaras obtenidas.

- La función `void freeifaddrs(struct ifaddrs *addrs)` libera la estructura `addrs`.

Ejercicios (se deben hacer todos en el mismo fichero C, opcionalmente excepto el 11):

1. Escribir un código C que tome como entrada los 4 bytes de una dirección IP por ejemplo, `uint8_t ip[]={193,110,128,200};` y que devuelva un entero de 32 bits sin signo, en este ejemplo, `uint32_t ip=3363860161u`. El código no debe realizar operaciones aritméticas de suma y producto sino que debe basarse exclusivamente en conversiones de punteros. Mostrar en pantalla un ejemplo.

2. En el mismo programa realizar la operación contraria a la anterior, pero partiendo de un `uint32_t` no de un `uint32_t *`. Mostrar en pantalla un ejemplo.

3. Convertir una dirección IP en formato textual a `uint32_t`. Hacerlo con las funciones `inet_pton`, `inet_aton` e `inet_addr`.

4. Definir cinco estructuras del tipo `in_addr` (o mejor, reutilizar una) e iniciar una de ellas con la dirección IPv4 de nuestro ordenador (copiarla de la etiqueta u obtenerla con el comando `ifconfig` en la línea *inet addr:* de alguna de las interfaces que contenga una válida) y las restantes a las 4 posibles macros. Pasar al formato textual las direcciones anteriores y mostrarlas en pantalla. Usar las funciones `inet_pton` y/o `inet_ntop`. Comprobar que la dirección del lazo de vuelta (127.0.0.1) se escribe al revés.

5. Comprobar que la función `inet_ntoa` devuelve un puntero a un buffer estático mediante un código que imprima la dirección del puntero (en hexadecimal) en sucesivas llamadas con distintas IPs.

6. Descomponer tres ejemplos de direcciones IPv4 de clases A, B y C en las partes de red y de host y mostrar los resultados en formato textual. ¿Cuántos bytes distintos de 0 hay en las partes de red y de host en cada una de las clases?

7. Componer una dirección IP a partir de las partes de host y de red en alguno de los casos del ejercicio anterior. Mostrar el resultado en pantalla en formato textual.

8. Comprobar que se puede usar la función `sprintf` para pasar una dirección IP del formato binario de 4 bytes al formato textual. Convertir el resultado a formato binario con la función `inet_pton` y comprobar que se realizó correctamente, imprimiendo el resultado y verificando la salida de la función (1 en caso de éxito).

9. Declarar tres direcciones IPv6 e iniciarlas respectivamente a: la dirección IP "1080:0:0:0:8:800:200C:417A", la macro IN6ADDR_LOOPBACK_INIT y la macro IN6ADDR_ANY_INIT. Imprimirlas en formato binario de 16 bytes en hexadecimal. Tener en cuenta que, dada la forma especial de estas dos macros, hay que hacer la asignación en la sentencia de declaración de las variables *struct in6_addr*. Para más detalles, consultar el archivo `/usr/include/netinet/in.h`.

10. Convertir al formato textual el resultado del punto anterior y comprobar que se obtienen las direcciones IPv6 de partida.

11. Escribir una función para obtener el nombre, dirección IP y máscara de todas las inferfaces de vuestro equipo, usando la función `getifaddrs`. Tener en cuenta que pueden ser direcciones IPv4 o IPv6. Comprobar la salida con el comando `ifconfig`.

2. Servicio de Nombres de Dominio (DNS)

Los nombres de host se obtienen consultando la base de datos del DNS. Veamos como acceder a esta base de datos programando en C.

- Las estructuras y funciones del DNS se definen en `netdb.h`. En caso de error se puede determinar lo que ocurrió con `herror("frase")`.
- La estructura predefinida `struct hostent` se utiliza para almacenar una entrada del DNS y contiene las siguientes componentes:

`struct hostent`	
`char *h_name`	el nombre "oficial" del host.
`char **h_aliases`	lista de alias del host, acabada con un puntero nulo.
`int h_addrtype`	el tipo de dirección, `AF_INET` (o `AF_INET6` para IPv6).
`int h_length`	la longitud en bytes de cada dirección.
`char **h_addr_list`	lista de IPs del host, acabada con un puntero nulo.
`char *h_addr`	sinónimo para `h_addr_list[0]`, la primera IP del host.

- Las direcciones IP almacenadas en los campos `*h_addr` y `**h_addr_list` se dan en formato binario de array de caracteres de longitud `h_length` (4 *unsigned char* en el caso de IPv4). Es responsabilidad del programador pasarlas al formato requerido (textual o binario) usando las funciones adecuadas o mediante conversiones de punteros. Por ejemplo,

  ```
  struct hostent *host1;
  struct in_addr IP1;
  ...
  IP1.s_addr=*((uint32_t *)host1->h_addr)
  ```

  Los punteros dobles pueden tomarse como arrays de *char*. Por ejemplo,

  ```
  for(i=0;host1->h_aliases[i]!=NULL;i++)
  { ... host1->h_aliases[i] ... }
  ```

  Las direcciones IP se dan en el orden de red (el usado en Internet).

Funciones:

- La función `struct hostent * gethostbyname (const char *name)` devuelve la información del DNS sobre el host denominado *name*. Si la búsqueda falla, la función devuelve un puntero nulo.
- La función `struct hostent * gethostbyaddr (const char *addr, size_t length, int format)` devuelve la information del DNS sobre el host de dirección IP `addr`. El parámetro `addr` es un puntero a una dirección IPv4 o IPv6 en formato binario (*uint32_t* o *unsigned char* de 4 bytes), `length` es el tamaño en bytes de la dirección y `format` es `AF_INET` o `AF_INET6`. Si la búsqueda falla, la función devuelve un puntero nulo.

- NOTA: las dos funciones anteriores devuelven un puntero a un buffer estático, y que por lo tanto, se sobreescribe en sucesivas llamadas a la función. Además, las funciones no son reentrantes, es decir, que el buffer se puede sobreescribir si varios hilos de un mismo programa llaman a la función.

- La base del datos del DNS puede escanearse usando las funciones `sethostent`, `gethostent` y `endhostent`.

  La función `void sethostent (int stayopen)` abre la base de datos del DNS para comenzar el escaneo. Si el parámetro `stayopen` es distinto de cero, las sucesivas llamadas de `gethostent` al DNS no cerrarán la base de datos (como usualmente hacen). Esto aumenta la eficiencia al no tener que reabrir la base de datos de cada vez.

  La función `struct hostent * gethostent (void)` devuelve la siguiente entrada de la base de datos. Si no hay más entradas, entonces devuelve un puntero nulo.

  La función `void endhostent (void)` cierra la base de datos.

Ejercicios (asumir que todas las direcciones son IPv4):

1. Escribir una función que tome como entrada un nombre de host y que imprima en pantalla todos los datos que devuelve el DNS, mostrando las IPs en formato textual. La función deberá chequear el caso de fallo (esto siempre debe hacerse con cualquier llamada al sistema). Hacer una función que dado un puntero a un `struct hostent` imprima toda la información, ya que os será útil para los siguientes ejercicios. Probar con *www.elpais.es*. Comprobar el resultado con el comando `dig nombre` en la ANSWER SECTION.

2. Lo mismo pero con entrada una dirección IP (en formato binario de 4 bytes). Probarla para una IP del ejercicio anterior. Comprobar el resultado con `dig -x IP` en la ANSWER SECTION.

3. ¿Puede usarse la función `memcpy` para sacar una copia del buffer apuntado por el valor devuelto de la función `gethostbyname` a un lugar seguro y así evitar que en dos llamadas sucesivas a la función se sobreescriba el resultado? ¿Por qué? Para comprobarlo hacer un código que llame a la función `gethostbyname`, que realice a continuación la copia e imprima la información de la estructura copiada y que vuelva a llamar a la función `gethostbyname` e imprima de nuevo la información salvada.

4. Obtener los nombres de host de las IPs asociadas a un nombre. Para ello probaremos con un host que devuelva varias IPs y que para cada IP nos devuelva un nombre de host, por ejemplo, *www.xataka.com*. Para evitar que la información devuelta por la función que obtiene un nombre de host

a partir de la IP sobreescriba la información de devuelve la función que obtiene las IPs a partir del nombre, se debe seguir los siguientes pasos:

   a) Obtener la lista de direcciones IP.

   b) Sacar una copia de la lista de direcciones IP a un lugar seguro por el método que se estime más adecuado (puede aprovecharse para convertir la lista a un array de `uint32_t` o de `struct in_addr` unidimensional).

   c) En un lazo aparte, para cada una de las direcciones IP copiadas, consultar el DNS y obtener sus nombres de host.

5. Escribir un programa C que abra la base de datos local del DNS y obtenga su contenido completo mediante sucesivas llamadas a la función `gethostent` (hasta que devuelva un puntero nulo), mostrando en pantalla la información obtenida; por último, cerrar la base de datos.

3. Conversión del orden de los bytes

Diferentes clases de ordenadores utilizan diferentes convenciones en el orden de los bytes que componen las palabras (de los enteros, por ejemplo). Algunos ordenadores comienzan por el byte más significativo (orden "big-endian") mientras que otros comienzan por el byte menos significativo (orden "little-endian").

Esto podría ser un problema cuando se comunican máquinas de diferente tipo, sin embargo, los protocolos de Internet especifican una convención de ordenamiento para la transmisión de datos. Este orden se denomina *orden de la red*. Desafortunadamente, en la arquitectura Intel el orden es justo al revés.

- Cuando se establece una conexión mediante sockets, hay que asegurarse que los números IP y los puertos estén escritos en el orden de la red. Adicionalmente prodríamos querer que todos los datos transmitidos como enteros se conviertan también al orden de la red, aunque esto no es estrictamente necesario si las máquinas origen y destino son del mismo tipo.
- Si usamos las funciones `getservbyname`, `gethostbyname` o `inet_addr` para obtener las direcciones de puerto e IP, los valores obtenidos están ya en el orden de la red y pueden utilizarse inmediatamente en el socket.
- En otro caso, debemos convertir los valores explícitamente. Para ello tenemos las funciones `htons` y `ntohs` para convertir los puertos y las funciones `htonl` y `ntohl` pata convertir las direcciones IPv4. Estas funciones están declaradas en `netinet/in.h`.

Funciones:

- La función `uint16_t htons (uint16_t hostshort)` convierte los enteros de 16 bits `hostshort` del orden de host al orden de red.
- La función `uint16_t ntohs (uint16_t netshort)` convierte los enteros de 16 bits `netshort` del orden de la red al orden de host.
- La función `uint32_t htonl (uint32_t hostlong)` convierte los enteros de 32 bits `hostlong` del orden de host al orden de red.
- La función `uint32_t ntohl (uint32_t netlong)` convierte los enteros de 32 bits `netshort` del orden de red al orden de host.

Ejercicios:

1. Convertir un puerto (un entero de 16 bits) de un orden a otro, hacer la conversión inversa y mostrar en pantalla en formato hexadecimal los valores inicial, convertido y convertido inverso.
2. Lo mismo para una dirección IPv4. Los valores inicial, convertido y convertido inverso mostrarlos en formato textual y hexadecimal.

4. Puertos y protocolos

4.1. Puertos

- Los números de puerto son enteros de 16 bits (rango de 0 a 65.535). Dos programas, a menos que se modifiquen las opciones de socket, no pueden acceder al mismo puerto en el mismo host. Los puertos menores que la macro `int IPPORT_RESERVED` están reservados para uso del supervisor y para servicios estándar, tales como *finger* y *telnet*. Los puertos mayores o iguales a la macro `int IPPORT_USERRESERVED` están reservados para el uso de los usuarios, puesto que nunca se usarán automáticamente. Por último, cuando se usa un socket sin especificar su puerto, el sistema le genera automáticamente un puerto comprendido entre `IPPORT_RESERVED` e `IPPORT_USERRESERVED`.

- La estructura predefinida `struct servent` almacena información sobre cada uno de los servicios. Contiene las siguientes componentes:

`struct servent`	
`char *s_name`	el nombre "oficial" del servicio.
`char **s_aliases`	lista de alias del servicio, acabada con un puntero nulo.
`int s_port`	el número de puerto (en el orden *big-endian*).
`char *s_proto`	el nombre del protocolo que usa este servicio.

  Existe una base de datos que contiene las direcciones de puerto de los servicios "bien conocidos", usualmente es el fichero */etc/services*.

- NOTA: algunos ordenadores (entre ellos los PCs) trabajan con bytes en orden contrario al usado en Internet. Por ello, cada vez que introduzcamos un número en las funciones siguientes hay que llamar a la función de reordenamiento *htons*, mientras que para imprimir un número devuelto por estas funciones hay que llamar a la función *ntohs* (ver el apartado anterior).

  De igual forma, cuando se escriba un número de puerto a un socket hay que usar la función *htons*.

- Las estructuras están declaradas en el fichero de cabeceras `netinet/in.h`, mientras que las funciones se declaran en `netdb.h`.

Funciones:

- La función `struct servent * getservbyname (const char *name, const char *proto)` devuelve un puntero a la información del servicio tomando como entrada el nombre del servicio `name` y el protocolo `proto` (por ejemplo, "*tcp*" o "*udp*"). Si el servicio no existe, devuelve NULL.

- La función `struct servent * getservbyport (int port, const char *proto)` también devuelve la información del servicio pero toma como entradas el puerto y el protocolo. Si el servicio no existe, devuelve NULL.

- La base de datos de servicios puede escanearse usando las funciones `setservent`, `getservent` y `endservent` (estas funciones no son reentrantes).

  La función `void setservent (int stayopen)` abre la base de datos para iniciar el escaneo. Si el argumento `stayopen` es distinto de cero, entonces las sucesivas llamadas de la función `getservent` no cierran la base de datos. Esto aumenta la eficiencia.

  La función `struct servent * getservent (void)` devuelve la siguiente entrada de la base de datos; si no hay más, devuelve un puntero nulo.

  La función `void endservent (void)` cierra la base de datos.

Ejercicios:

1. Obtener el valor de las dos variables que definen los rangos de puertos.
2. Obtener la información de los servicios *http*, *telnet* y *smtp*, y mostrar en pantalla la información obtenida. Hacer una función que dado un puntero a un `struct servent` imprima toda la información.
3. Obtener la información de los puertos 7, 21 y 110.
4. Consultar la base de datos de puertos en llamadas sucesivas hasta que se devuelva un puntero nulo y mostrar para cada entrada el nombre del servicio, los alias, el puerto y el protocolo. Puesto que la lista es muy grande, ejecutarlo con `./ejecutable | less`. Comparar la salida con el archivo `/etc/services`.

4.2. Protocolos

Los protocolos de Internet (TCP, UDP, etc.) también se pueden especificar por números, almacenados, normalmente, en el fichero */etc/protocols*. Este tema no es demasiado interesante porque habitualmente puede ponerse como protocolo el número 0, lo que le indica al sistema que tome el defecto: *tcp* para servicios orientados a conexión y *udp* para servicios sin conexión.

- La estructura `struct protoent` representa cada una de las entradas de la base de datos de protocolos y consta de las siguientes componentes:

  `char *p_name`: el nombre "oficial" del protocolo.
  `char **p_aliases`: lista de alias del protocolo acabada en puntero nulo.
  `int p_proto`: el número del protocolo (en el orden de host).

- La función `struct protoent * getprotobyname (const char *name)` devuelve la información sobre el protocolo denominado `name`. Si no existe tal protocolo, la función devuelve un puntero nulo.

- La función `struct protoent * getprotobynumber (int protocol)` devuelve la información sobre el protocolo de número `protocol`. Si no existe tal protocolo, la función devuelve un puntero nulo.