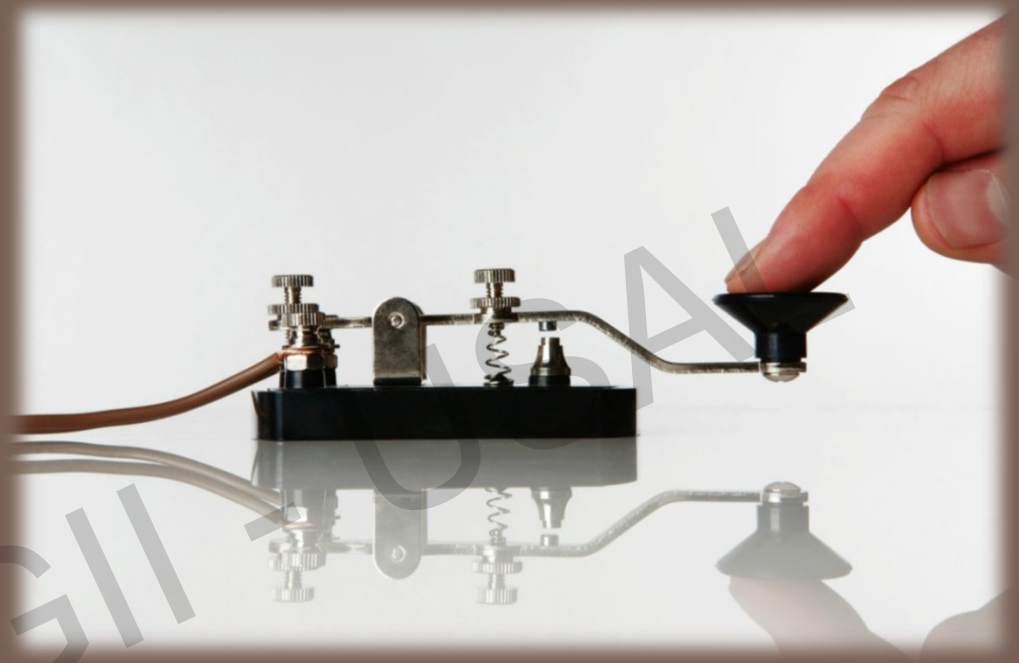


DESARROLLO DE APLICACIONES EN RED (SOCKETS)

(CC) Moreno, A. M. & Bravo, S. & Vázquez, A. (Redes I, 2025)

Contenido

- Objetivos y entorno de desarrollo
- Aplicación “Morse”
  - Especificaciones del protocolo
    - Mensajes
    - Ejemplo de dialogo
  - Requisitos

Objetivos y entorno de desarrollo

- El objetivo de esta práctica es implementar una aplicación en red como
  - usuario del nivel de transporte y
  - según la arquitectura o modelo de programación cliente-servidor
- Entorno de desarrollo
  - Estación de trabajo con S.O. Debian GNU/Linux 11 (*bullseye*) (nogal.usal.es)
  - Sockets de Berkeley
  - Lenguaje de programación C

Especificaciones del protocolo

- El servicio que vamos a implementar se denomina “Morse”
  - Convierte a código morse la frase recibida
- Emplea dos tipos de mensajes:
  - Peticiones de los clientes a los servidores
  - Respuestas de los servidores a los clientes
  - Los mensajes
    - Son siempre líneas de caracteres terminadas con los caracteres CR-LF (retorno de carro ‘\r’ (ASCII 13 (0x0D)), - avance de línea ‘\n’ (ASCII 10 (0x0A)))
    - La longitud máxima de las líneas no debe exceder 516 bytes, contando todos los caracteres (incluido el CR-LF final)
- **Peticiones del cliente al servidor**
  - El dialogo con el servidor comienza con la orden HOLA
  - A continuación, se envía la frase con la orden FRASE
  - Si no se desea enviar más frases se finaliza con la orden FIN

Mensajes MORSE (I)

- **Peticiones del cliente al servidor**
  - La orden HOLA se forma:
    - HOLA dominio-del-cliente[CR-LF]
    - Donde dominio-del-cliente es el dominio del originador y [CR-LF] los caracteres de retorno de carro y salto de línea. Por ejemplo: usal.es
  - La orden FRASE se forma:
    - FRASE *frase*[CR-LF]
    - Donde *frase* es la frase que se desea pasar a código morse. Los espacios se devolverán con el carácter '/'
  - Se pueden enviar las frases que se desee hasta que se finaliza con la orden FIN[CR-LF]

Mensajes MORSE (II)

- **Respuestas del servidor**

Núm.	Cadena	Descripción
220	Servicio preparado	Respuesta cuando el cliente realiza la conexión
240	OK	Respuesta a la orden HOLA
250	MORSE	Respuesta a la orden FRASE
221	Cerrando el servicio	Respuesta a la orden FIN
500	Error de sintaxis	Respuesta a errores de sintaxis en cualquier orden

Mensajes MORSE (III)

- Ejemplo de dialogo (S: Servidor, C: Cliente)
  - S:220 Servicio preparado
  - C: HOLA usal.es
  - S: 240 OK
  - C: PHRASE pepe
  - S: 500 Error de sintaxis
  - C: FRASE SOS
  - S: 250 MORSE .../---/...
  - C: FRASE Hoy es fiesta
  - S: 250 MORSE ..../---/-.--/./.../..-./.././.../-/.-
  - C: FIN
  - S: 221 Cerrando el servicio

NOTA:

- En la versión para UDP el cliente enviará un primer mensaje con una línea en blanco que sólo contenga los caracteres CR-LF puesto que en UDP no hay petición de conexión.
- Las líneas de los mensajes son siempre líneas de caracteres terminadas con los caracteres CR-LF (retorno de carro "\r" (ASCII 13 (0x0D)), - avance de línea "\n" (ASCII 10 (0x0A)))

Requisitos (I)

- **Programa Servidor**
  - Aceptará peticiones de sus clientes tanto en TCP como en UDP
  - Proporcionará al cliente el código Morse de las frases recibidas hasta que reciba la orden FIN
  - Registrará todas las peticiones en un fichero de "log" llamado peticiones.log en el que anotará:
    - Fecha y hora del evento
    - Descripción del evento:
      - Comunicación realizada: nombre del host, dirección IP, protocolo de transporte, nº de puerto efímero del cliente
      - Frase recibida: nombre del host, dirección IP, protocolo de transporte, el puerto del cliente y la frase.
      - Respuesta enviada: nombre del host, dirección IP, protocolo de transporte, puerto del cliente y la respuesta mandada.
      - Comunicación finalizada: nombre del host, dirección IP, protocolo de transporte, nº de puerto efímero del cliente
  - Se ejecutará como un "daemon"
- **Programa Cliente**
  - Se comunicará con el servidor bien con TCP o con UDP
  - Leerá por parámetros el nombre del servidor y si es TCP o UDP:
    - cliente nogal TCP
  - Realizará peticiones al servidor como se ha indicado anteriormente
  - Realizará las acciones oportunas para su correcta finalización

Requisitos (II): pruebas

- Durante la fase de pruebas el cliente podrá ejecutarse como se muestra en el ejemplo de diálogo anterior, pero en la versión para entregar el cliente
  - Leerá de un fichero las órdenes que ha de ejecutar. El nombre del fichero lo recibirá como parámetro
  - Escribirá las respuestas obtenidas del servidor y los mensajes de error y/o depuración en un fichero con nombre el número de puerto efímero del cliente y extensión .txt

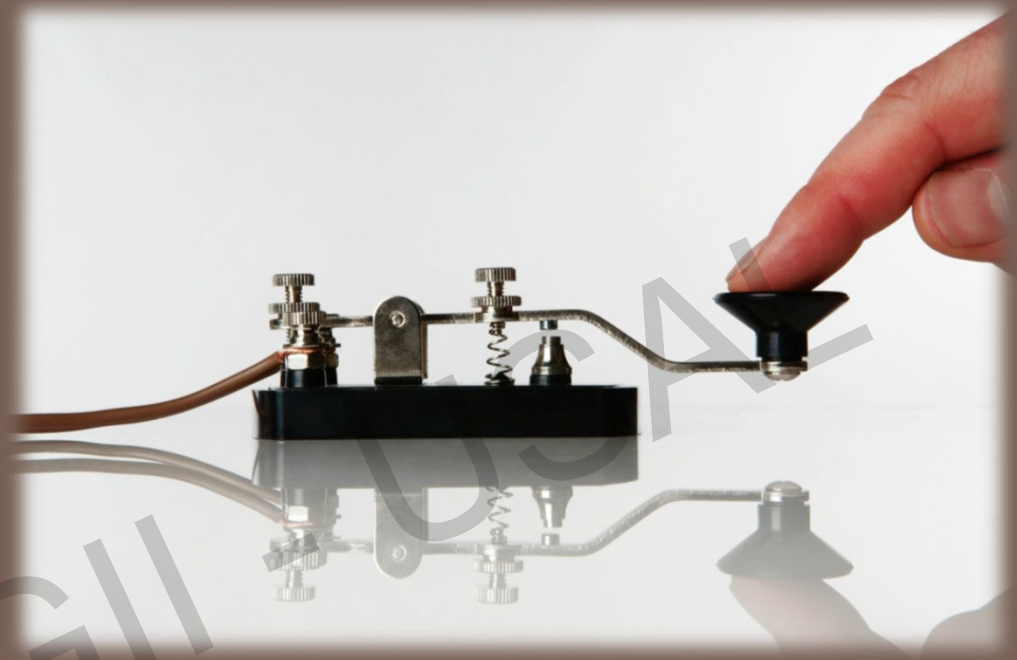
Requisitos (III): versión entregable

- Para verificar que esta práctica funciona correctamente y permite operar con varios clientes, se utilizará el *script* lanzaServidor.sh que ha de adjuntarse obligatoriamente en el fichero de entrega de esta práctica
- El contenido de lanzaServidor.sh es el siguiente:

```
# lanzaServidor.sh
# Lanza el servidor que es un daemon y varios clientes
# las ordenes estAn en un fichero que se pasa como tercer parAmetro
./servidor
./cliente nogal TCP ordenes.txt &
./cliente nogal TCP ordenes1.txt &
./cliente nogal TCP ordenes2.txt &
./cliente nogal UDP ordenes.txt &
./cliente nogal UDP ordenes1.txt &
./cliente nogal UDP ordenes2.txt &
```

Requisitos (IV): documentación

- Entregar un informe en formato PDF que contenga:
  - Detalles relevantes del desarrollo de la práctica
  - Documentación de las pruebas de funcionamiento realizadas



DESARROLLO DE APLICACIONES EN RED (SOCKETS)

(CC) Moreno, A. M. & Bravo, S. & Vázquez, A. (Redes I, 2025)