

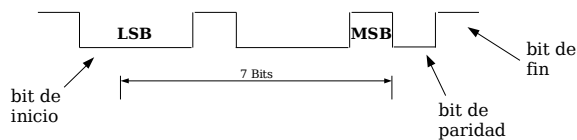
Unix: Entrada-Salida en un terminal Programación en Redes

Terminal I/O: Conceptos

- Un terminal clásico es un teclado y una pantalla (o impresora) conectados a un computador usando un puerto asíncrono
- Para el sistema operativo lo importante es el puerto de comunicaciones que conecta el terminal con el host
 - No interesa lo que esté al otro lado del cable
- Un puerto de comunicaciones asíncronas puede ser utilizado para hablar con otros computadores, impresoras o equipo especial de entrada-salida

Terminal I/O: Conceptos

- Un dispositivo serial, envía un carácter de un bit a la vez:
 - Comienza con el bit menos significativo y finaliza con el bit más significativo
 - El último bit a ser enviado es el bit de paro
- Enviar el carácter "D": (100 0100)



Terminal I/O: Conceptos

- El bit de paridad indica si viene un número par o impar de unos
- El número de bits por segundo que son transmitidos contando el bit de inicio y fin es conocido como 'baud rate'
- Si el bit de fin no es 1, se generará un error
- Si todos los bits son ceros, incluso el bit de fin, se generará una condición de quiebre

Terminal I/O: Conceptos

- La mayoría de los terminales operan en modo full-duplex:
 - Cuando se presiona una tecla, se envía un carácter desde el terminal al computador
 - Cuando el computador recibe el carácter lo envía de vuelta al terminal para que sea mostrado (echo)
- Haciendo echo, el computador tiene el control sobre los caracteres desplegados por el terminal
 - Un programa podría “esconder” los caracteres que se tipean, ejemplo: entrada de passwords, comandos en el editor vi

Terminal I/O: Conceptos

- Un programa podría utilizar “echo” de una secuencia especial de caracteres:
 - Al presionar la tecla 'backspace', el computador podría generar el despliegue de la secuencia: backspace-space-backspace, para borrar el último carácter

Parámetros del Terminal

- Los parámetros de un terminal pueden ser manipulados por medio de la estructura “termios”
- Para copiar los parámetros desde el sistema operativo a una estructura “termios” se utiliza la función:

```
int tcgetattr(int fd, struct termios *ptermios);
```

- El parámetro fd, debe ser un file descriptor válido asociado a un terminal

Parámetros del Terminal

- Para copiar los parámetros desde la estructura “termios” al sistema operativo, se utiliza:

```
int tcsetattr(int fd, int option,
             struct termios *ptermios);
```

- Con una llamada asigna todos los parámetros del terminal, el argumento “option” debe ser:
 - TCSANOW: refleja los cambios inmediatamente
 - TCSADRAIN: refleja los cambios después de transmitir toda la salida a “fd”, se utiliza para cambios que afectan la salida
 - TCSAFLUSH: igual que el anterior, pero además descarta toda la entrada recibida que no ha sido leída

Parámetros del Terminal

- La función "tcsetattr()" no detecta errores en la estructura termios
- Para saber si un atributo de un terminal en particular fue asignado correctamente, se debe hacer: "tcsetattr()" seguido de "tcgetattr()"
- Para cambiar los parámetros de un terminal, bastaría con leer sus atributos, cambiar la estructura recibida y escribirla
- La estructura "termios" y las funciones que operan con ella están definidas en <termios.h>

Parámetros del Terminal

- La estructura termios tiene 5 parámetros, los cuales tienen FLAGS definidos por el sistema:

```
tcflag_t c_iflag;      /* modos de entrada */
tcflag_t c_oflag;      /* modos de salida */
tcflag_t c_cflag;      /* modos de control */
tcflag_t c_lflag;      /* modos locales */
cc_t      c_cc[NCCS];  /* caracts. de control */
```

- c_iflag: Hay 11 flags distintos. El único portable es ISTRIP (borrar bit de paridad)

Parámetros del Terminal

- c_oflag: No hay flags portables. Controla postproceso
- c_cflag: Controla información relacionada con el hardware
- c_lflag: controla el echo y el procesamiento de caracteres:
 - ECHO: activa el eco
 - ICANON: activa proceso del input
 - ISIG: permite signals
 - TOSTOP: suspende procesos del background que escriban en el terminal.
- c_cc: Arreglo de caracteres de control, de tamaño NCCS. No deben ser cambiados.

Parámetros del Terminal

- Ejemplo: lectura de password, algoritmo:
 - Escribir "Password: "
 - Descartar caracteres tipeados antes del prompt
 - Apagar el eco
 - Leer password
 - Restaurar el eco
- password.c

Terminal I/O

- **Procesamiento de la entrada:**
 - Cuando una aplicación lee de un disco, los datos son transferidos de manera simple al programa, sin ningún procesamiento particular
 - Cuando leemos desde un terminal, el sistema puede realizar algún procesamiento antes de enviar los datos al programa de usuario. Este procesamiento consiste:
 - Hacer “echo” de los datos
 - Buscar algún carácter especial

Terminal I/O

- **Procesamiento de la Salida:**
 - Varios tipos de procesamiento en la salida pueden ser requeridos
 - Por ejemplo, un terminal puede necesitar cierta demora después de un salto de línea, para dar el tiempo suficiente para realizar “scroll”
 - No existe un estándar para el procesamiento de la salida, cada sistema implementa lo que necesita
- **Control de un Modem:**
 - Cuando un terminal es conectado a un host por medio de un modem y una línea telefónica, el programa debería tener cierto control sobre la conexión telefónica

Terminal I/O

- **El estándar POSIX provee un control mínimo:**
 - Señal SIGHUP, se puede enviar al programa si la conexión que controla el terminal se pierde inesperadamente
 - El host puede “colgar” la llamada vía tcsetattr()
- **Velocidad del terminal:**
 - La estructura “termios” contiene información sobre la velocidad de salida y entrada del terminal (baud rate)
 - Funciones que permiten acceder a estos valores:

```
speed_t cfgetispeed ( struct termios *p );
```

- devuelve la velocidad de entrada guardada en la estructura termios

Terminal I/O

- ```
speed_t cfgetospeed (struct termios *p);
```
- devuelve la velocidad de salida guardada en la estructura termios apuntada
- **Para asignar velocidades existen las funciones:**

```
int cfsetispeed(struct termios *p, speed_t sp);
int cfsetospeed(struct termios *p, speed_t sp);
```

    - Copian el valor “sp” en la estructura apuntada por “p”
    - Retornan 0 en caso de éxito y -1 en caso de error
    - Los cambios no se verán reflejados hasta que se llame la función: tcsetattr()

## Terminal I/O

- El tipo "tipe\_t" está definido en <termios.h> y es un valor sin signo
- Símbolos de la forma BXXXX están definidos para cada valor legal de "baud rate", por ejemplo:

| Símbolo | Baud Rate |
|---------|-----------|
| B0      | 0         |
| B50     | 50        |
| B75     | 75        |
| ...     | ...       |
| B19200  | 19200     |
| B38400  | 38400     |

- B0 deshabilita la comunicación

## Terminal I/O

- Funciones de control:
  - Espera a que todos los datos escritos vía "write" hayan salido:
 

```
int tcdrain (int fd);
```
  - Descarta datos en la entrada o salida:
 

```
int tcflush (int fd, int option);
```
  - "option" puede ser:
    - TCIFLUSH: Descarta datos recibidos y aún no leídos
    - TCOFLUSH: Descarta datos escritos, pero aún no transmitidos
    - TCIOFLUSH: Los dos anteriores

## Programación en Redes

- ¿Cómo se comunican 2 programas que están funcionando en máquinas distintas?
  - En Unix BSD se introdujo un mecanismo llamado "sockets"
- Existen 2 modelos de tipos de comunicación:
- Tipo "connection-oriented": se establece un enlace entre los dos programas, el cual existe durante toda la "conversación".
  - El enlace permite el envío de un flujo continuo de datos, preserva el orden de los mensajes y asegura la entrega confiable de los mensajes.
  - Se realiza control de flujo (Protocolo TCP, Transfer Control Protocol)

## Programación en Redes

- Tipo Datagramas:
  - envía paquetes de datos, sin que haya garantía sobre la confiabilidad de su entrega ni sobre el orden en que llegan
  - Se debe especificar el destinatario en cada paquete
  - Protocolo UDP (User Datagram Protocol)
- Para enviar un mensaje a un programa que corre en otra máquina es necesario especificar:
  - Dirección de esa máquina (192.80.24.83) o bien su nombre (anakena.dcc.uchile.cl)
  - Número del puerto en el cual el programa está escuchando
- Los puertos 1 al 1024 están reservados para los servicios "bien conocidos"

## Programación en Redes

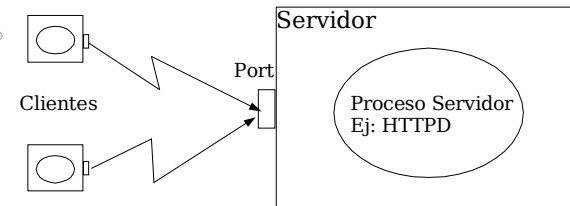
- Ejemplo de servicios “bien conocidos”:

| Servicio | Puerto |
|----------|--------|
| HTTP     | 80     |
| SMTP     | 25     |
| FTP      | 21     |
| SSH      | 22     |
| TELNET   | 23     |

- La dirección IP: 127.0.0.1 identifica a la máquina en donde se está corriendo el programa (localhost)
- Los sockets proveen una interfaz común para diversos protocolos de comunicación

## Programación en Redes

- Las familias de protocolos se definen en <sys/socket> y las más usadas son:
  - AF\_UNIX: para comunicaciones locales, procesos en una misma máquina
  - AF\_INET: comunicaciones en Internet
- Esquema típico de cliente-servidor:



## Programación en Redes

- Los procesos que funcionan en el servidor son conocidos como demonios:
  - Funcionan en un ciclo infinito
- Comunicación entre procesos:
  - En el lado del servidor primero se debe crear el socket para recibir conexiones con la función:

```
int socket(int dominio, int tipo, int protocol);
```

- Retorna un entero que será el “descriptor” del socket, será -1 en caso de error

## Programación en Redes

- Socket proceso servidor:
  - El dominio, será AF\_INET o AF\_UNIX
  - El tipo puede ser (entre otros):
    - SOCK\_STREAM: Conexión bidireccional, secuencial y confiable
    - SOCK\_DGRAM: Para el caso de datagramas, sin conexión y de una longitud máxima
  - El protocolo, es un entero que identifica el número de protocolo a utilizar:
    - El archivo /etc/protocols muestra los diferentes protocolos
    - Para la mayoría de nuestros casos utilizaremos 0 (IP, internet protocol)

## Programación en Redes

- Una vez que se tiene el socket, es necesario dar una dirección donde escuchar:

```
int bind(int sockfd, struct sockaddr *my_addr,
 socklen_t addrlen);
```

- En la estructura "sockaddr" se especifican los datos de la dirección para el socket "sockfd"
- Retornará 0 en caso de éxito y -1 en caso de error

## Programación en Redes

- La estructura sockaddr es la unión de diversos tipos dependiendo del protocolo
- Para el caso de Internet se utiliza sockaddr\_in, que tiene los siguientes elementos:
  - sin\_family: AF\_INET (address family)
  - sin\_port: puerto, debe estar en "network byte order"
  - sin\_addr: estructura in\_addr
- La estructura in\_addr tiene el siguiente elemento:
  - s\_addr: Dirección en network byte order

## Programación en Redes

- Para rellenar la estructura:

```
struct sockaddr_in sa;
struct hostent *hp;
char myname[MAXHOSTNAME+1];

bzero(&sa, sizeof(struct sockaddr_in));
gethostname(myname, MAXHOSTNAME);
hp = gethostbyname(myname);
if (hp == NULL)
 return -1;
sa.sin_family = AF_INET;
sa.sin_port = htons(portnum);
bcopy(hp->h_addr_list[0], &sa.sin_addr.s_addr,
 hp->h_length);
```