



## **Video Lecture # 16**

# **Programming the Terminal Devices**

**Course: SYSTEM PROGRAMMING**

**Instructor: Arif Butt**

**Punjab University College of Information Technology (PUCIT)**  
**University of the Punjab**

Source Code files available at: <https://bitbucket.org/arifpucit/spvl-repo/src>  
Lecture Slides available at: <http://arifbutt.me>



# Agenda

---

- In UNIX every thing is a file
- Similarities and differences between regular & device Files
- Reading/Writing on some one's terminal
- Modes of terminal driver
- Attributes of terminal driver
- Modifying terminal attributes using `stty (1)`
- Where terminal attributes are saved?
- Modifying terminal attributes using
  - `system()` library call
  - `tcgetattr()` and `tcsetattr()` library calls
  - `ioctl()` system call



**Review OS with LinuxVideo Lec#25 on Device files**

**Review OS with LinuxVideo Lec#26 on Terminal attributes**

---



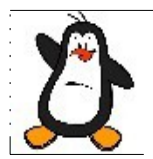
# In UNIX Everything is a File



# File Types in UNIX based Systems

---

- Regular files (**-**)
- Directories (**d**)
- Symbolic Links (**l**)
- Character special files (**c**)
- Block special files (**b**)
- Named pipes (**p**)
- Sockets (**s**)



# Device Files

---

## Character Special Files (c)

These files represents hardware devices that reads or writes a serial stream of data bytes. Devices connected via serial/parallel ports fall in this category. Examples of such devices are terminal devices, sound cards, and tape drives

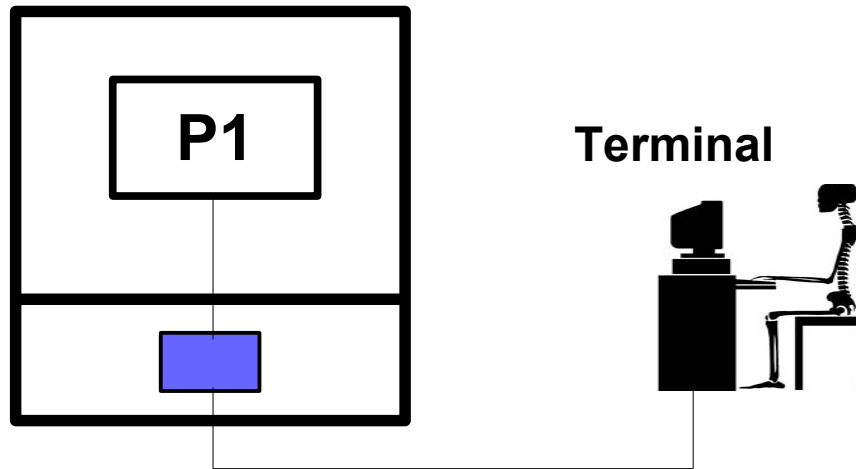
## Block special files (b)

These files represents hardware devices that reads or writes data in fixed size blocks. Unlike serial devices they provide random access to data stored on the device. Examples of such devices are HDD, SSD, and cdrom

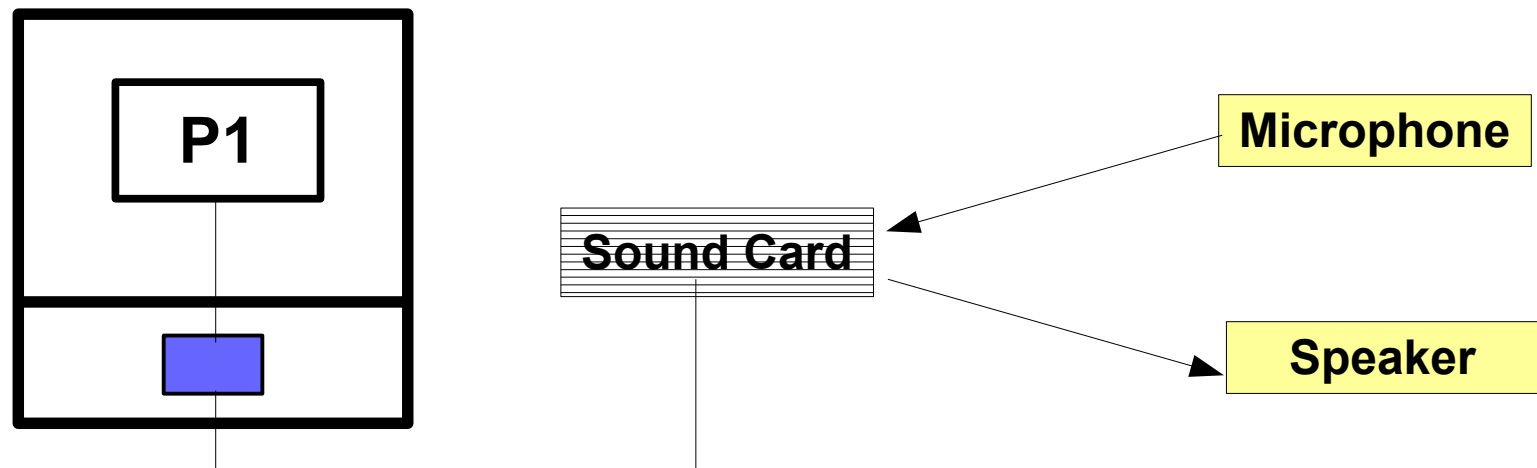


# Similarities Between Devices & Regular Files

For a process, a terminal is a source/destination of data

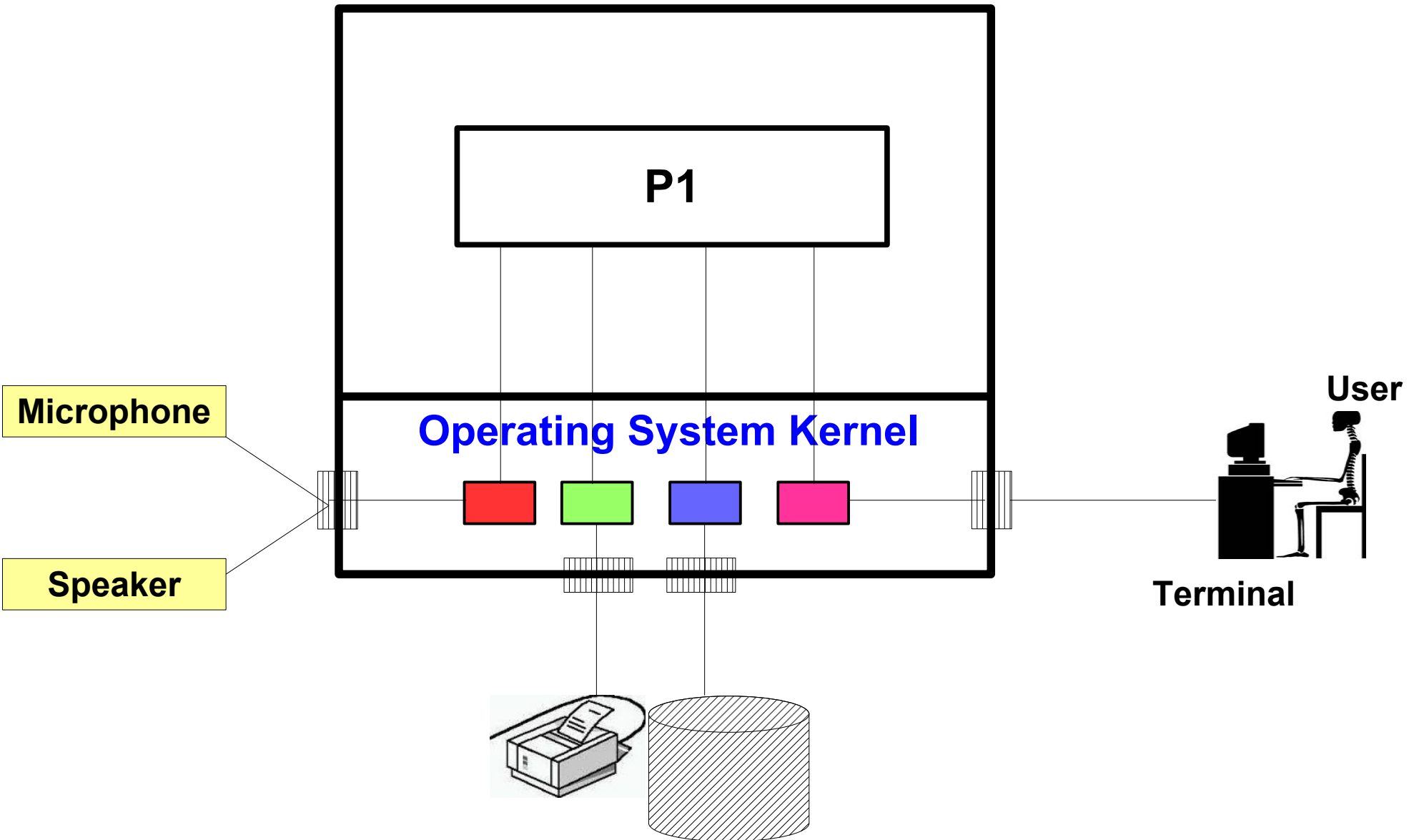


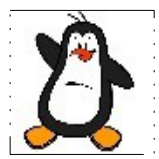
For a process, a sound card is a source/destination of data





# Overview of Device Files





# Differences Between Device & Regular Files

- A regular file is a container, while a device file is a connection
- The inode block of a regular file contains pointer that points to its data blocks, while the inode block of a device file contains pointer that points to a function inside the kernel called the device driver
- When you see the long listing, a regular file shows its size while a device file displays the major and minor number of the device driver at the place of size when you see its long listing

## Device Numbers:

Linux identify devices using a 16 bit number divided into two parts

- **Major number** (8 bits) that identifies the driver program
- **Minor number** (8 bits) that is used by the driver program to identify the instance





# Proof of concepts On Linux Terminal



# Modes of Terminal Driver

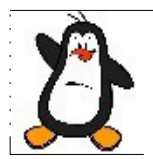
---

## Canonical Mode:

- Input is made available line by line, and the line goes to the process only after the user presses the Enter key (mean while it is buffered inside the `ttty` driver program)
- Line editing is enabled

## Non-Canonical Mode:

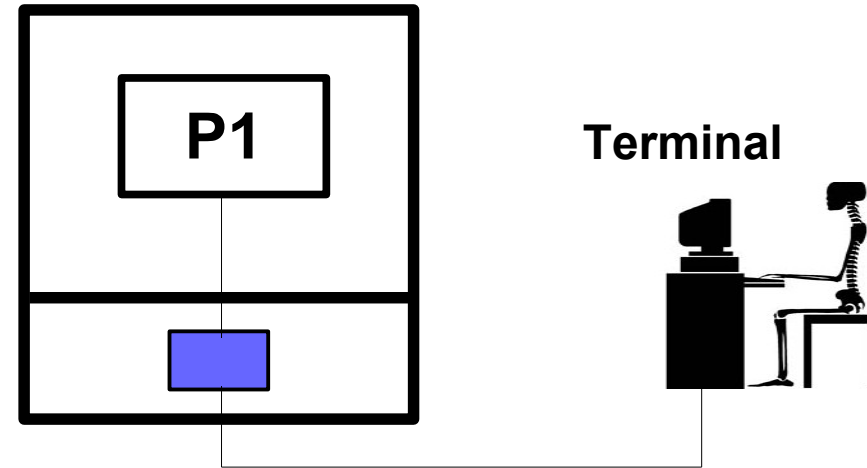
- Input is made available immediately as a key is pressed, without the need to press the Enter key (no buffering is done by the driver program)
- Line editing is disabled



# Attributes of Terminal Driver

Actions performed by `tty` driver on the data passing through it can be grouped into four main categories:

- Input Processing:** Processing performed by `tty` driver on the characters received via key board, before sending them to the process. Example: `icrnl`
- Output Processing:** Processing performed by `tty` driver on the characters received from process, before sending them to the display unit. Example: `onlcr`
- Control Processing:** How characters are represented? Example: `cs8`
- Local Processing:** What the driver do while the characters are inside the driver? Example: `icanon, echo`





# Attributes of Terminal Drivers

---

## 100\$ Question

How can we examine and modify the value of Terminal Attributes?



# Accessing & Modifying Terminal Attributes using `stty`

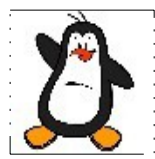


# Programming the Terminal Driver

---

There are three ways you can get/set the attributes of terminal driver inside your C program:

- Use `system()` library call
- Use `tcgetattr()` and `tcsetattr()` library calls
- Use `ioctl()` system call



# system() Library Call

---

```
int system(const char* command);
```

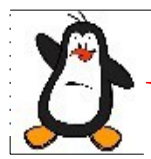
- It executes a command specified in **cmd** by calling **/bin/bash -c** command and returns after the command has been completed
- Return -1 on error and the return status of the cmd other wise
- Main cost of **system()** is inefficiency. Executing a command using **system()** requires the creation of at least two processes
  - One for the shell
  - Other for the command(s) it executes



# Accessing & Modifying Terminal Attributes using `system()`

`password_simple.c, password_system.c`

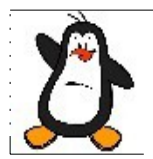




# Use `tcgetattr()` and `tcsetattr()` Library Calls

```
int tcgetattr(int fd, struct termios* info);  
int tcsetattr(int fd,int when,struct termios* info)
```

- The `tcgetattr()` copies current settings from `tty` driver associated to the open file `fd` into the struct pointed by `info`. Returns 0 on success and -1 on error
- The `tcsetattr()` sets the `tty` driver associated to the open file `fd` with the settings given in the struct pointed by `info`. The `when` argument tells when to update the driver settings. The `when` argument can take following values:
  - `TCSANOW`: update driver settings immediately
  - `TCSADRAIN`: wait until all o/p already queued in the driver has been transmitted to the terminal and then update the driver
  - `TCSAFLUSH`: wait for o/p queue to be emptied + flush all queued i/p data and then update the driver



# Structure `termios`

---

Some important members of the `termios` structure that of our interest right now are shown below:

```
struct termios{
    tcflag_t    c_iflag; //contains flags related to input processing
    tcflag_t    c_oflag; //contains flags related to output processing
    tcflag_t    c_cflag; //contains flags related to control processing
    tcflag_t    c_lflag; //contains flags relating to local processing
    . . .
    . . .
};
```



# Individual Bits of `termios` Flags

---

`c_iflag`

|    |       |         |       |       |      |       |       |       |       |        |       |        |        |        |        |
|----|-------|---------|-------|-------|------|-------|-------|-------|-------|--------|-------|--------|--------|--------|--------|
| 15 | 14    | 13      | 12    | 11    | 10   | 9     | 8     | 7     | 6     | 5      | 4     | 3      | 2      | 1      | 0      |
| -- | IUTF8 | IMAXBEL | IXOFF | IXANY | IXON | IUCLC | ICRNL | IGNCR | INLCR | ISTRIP | INPCK | PARMRK | IGNPAR | BRKINT | IGNBRK |

`c_oflag`

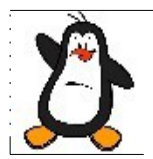
|    |    |       |       |       |        |       |       |       |       |        |       |       |       |       |       |
|----|----|-------|-------|-------|--------|-------|-------|-------|-------|--------|-------|-------|-------|-------|-------|
| 15 | 14 | 13    | 12    | 11    | 10     | 9     | 8     | 7     | 6     | 5      | 4     | 3     | 2     | 1     | 0     |
| -- | -- | FFDLY | VTDLY | BSDLY | TABDLY | CRDLY | NLDLY | OFDEL | OFILL | ONLRET | ONOCR | OCRNL | ONLCR | OLCUC | OPOST |

`c_cflag`

|    |    |    |         |        |        |       |       |       |        |        |       |        |       |         |       |
|----|----|----|---------|--------|--------|-------|-------|-------|--------|--------|-------|--------|-------|---------|-------|
| 15 | 14 | 13 | 12      | 11     | 10     | 9     | 8     | 7     | 6      | 5      | 4     | 3      | 2     | 1       | 0     |
| -- | -- | -- | CRTSTCS | CMSPAR | CIBAUD | LOBLK | CLOCL | HUPCL | PARODD | PARENB | CREAD | CSTOPB | CSIZE | CBAUDEX | CBAUD |

`c_lflag`

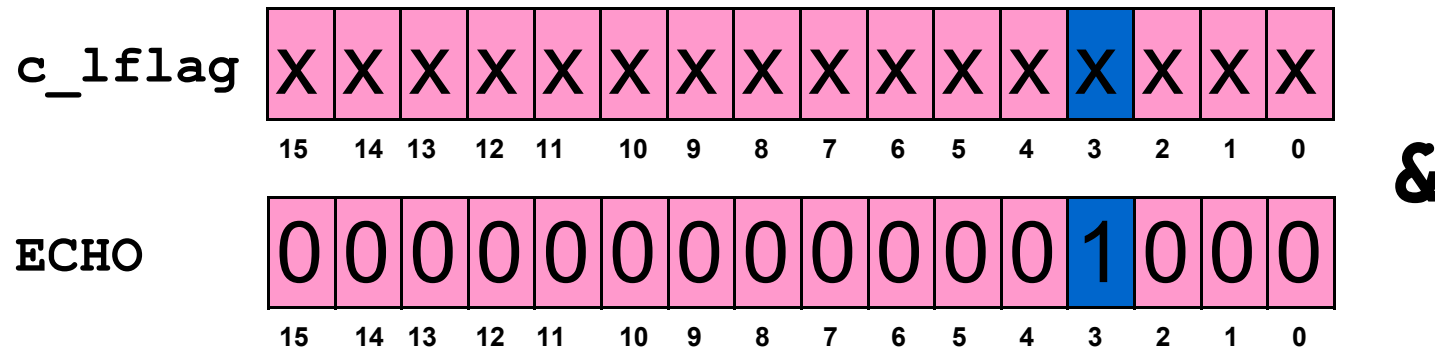
|        |        |        |        |        |         |        |         |         |        |       |       |      |       |        |      |
|--------|--------|--------|--------|--------|---------|--------|---------|---------|--------|-------|-------|------|-------|--------|------|
| 15     | 14     | 13     | 12     | 11     | 10      | 9      | 8       | 7       | 6      | 5     | 4     | 3    | 2     | 1      | 0    |
| IEXTEN | PENDIN | TOSTOP | NOFLSH | FLUSHO | DEFECHO | ECHOKE | ECHOPRT | ECHOCTL | ECHONL | ECHOK | ECHOE | ECHO | XCASE | ICANON | ISIG |

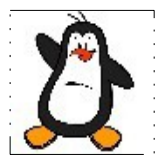


# Testing status of echo Flag

Code snippet to test the echo flag in the `c_lflag` member of `termios` structure:

```
struct termios info;
tcgetattr(0, &info);
if((info.c_lflag & ECHO) == 0)
    printf("echo is off, since its bit is 0");
else
    printf("echo is on, since its bit is 1");
```





# Changing Attributes of Terminal Driver

Three steps to change the attributes of a terminal driver:

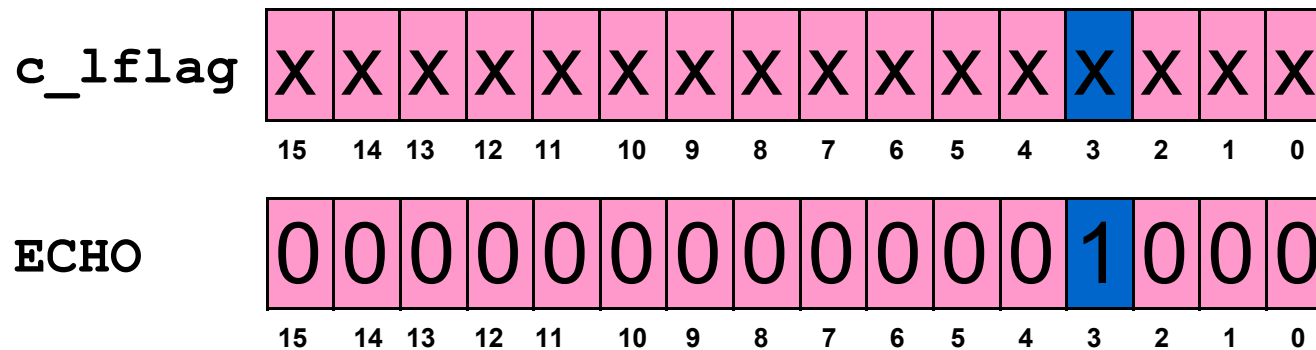
- Get the attributes from the driver
- Modify the attribute(s) you need to change
- Send these revised attributes back to the driver

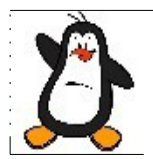


# Making the echo Flag ON

Code snippet to turn ON the echo flag in the `c_lflag` member of `termios` structure:

```
struct termios info;
tcgetattr(0, &info);
info.c_lflag = info.c_lflag | ECHO;
tcsetattr(0, TCSANOW, &info);
```

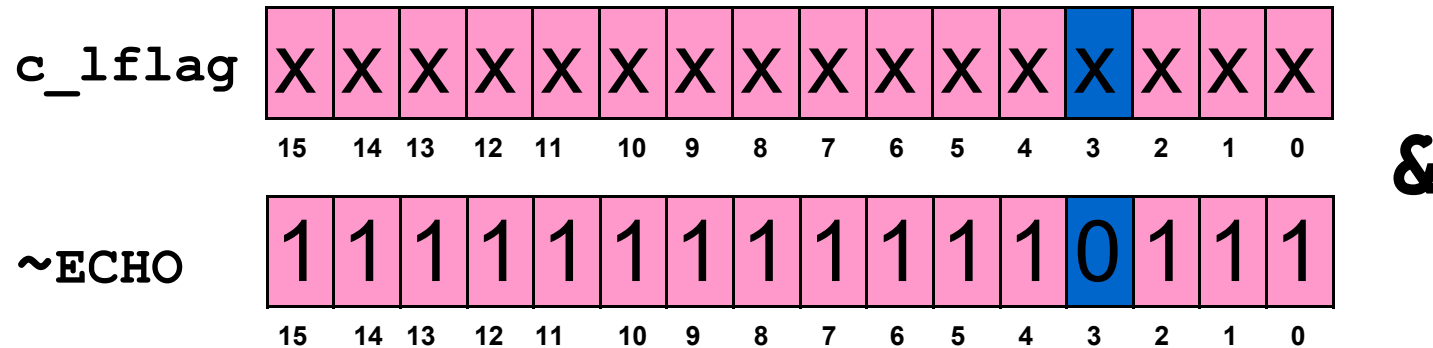




# Making the echo Flag OFF

Code snippet to turn OFF the echo flag in the `c_lflag` member of `termios` structure:

```
struct termios info;
tcgetattr(0, &info);
info.c_lflag = info.c_lflag & ~ECHO;
tcsetattr(0, TCSANOW, &info);
```





# Accessing & Modifying Terminal Attributes using `tcgetattr()` & `tcsetattr()`

`password_tcget.c`



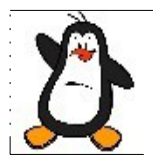


# **ioctl () System call**

---

```
int ioctl(int fd, int request[, arg, ...]);
```

- We have seen the use of `fcntl ()` system call to get/set the attributes of a disk file. To get/set the attributes of device files we can use the `ioctl ()` system call. Each type of device has its own set of properties and `ioctl` operations
- The first argument `fd` specifies an open file descriptor that refers to a device
- The second argument `request` specifies the control function to be performed based upon the device being addressed. Defined in `/usr/include/asm-generic/ioctls.h`
- Remaining optional arguments are request specific, defined in `/usr/include/x86_64-linux-gnu/bits/ioctl-types.h`



## Changing echo Flag Bit Using ioctl ()

Code snippet to turn ON the echo flag in the `c_lflag` member of `termios` structure using `ioctl ()`:

```
struct termios info;
ioctl(0, TCGETS, &info);
info.c_lflag = info.c_lflag | ECHO;
ioctl(0, TCSETS, &info);
```

Code snippet to turn OFF the echo flag in the `c_lflag` member of `termios` structure using `ioctl ()`:

```
struct termios info;
ioctl(0, TCGETS, &info);
info.c_lflag = info.c_lflag & ~ECHO;
ioctl(0, TCSETS, &info);
```



# Accessing & Modifying Terminal Attributes using `ioctl()`

`password_ioctl.c`



# Getting the Screen Size Using `ioctl()`

A video terminal screen, has size measured in rows and columns or pixels. The following code snippet displays the dimensions of the screen using `ioctl()` call:

```
struct winsize wbuf;
```

```
ioctl(0, TIOCGWINSZ, &wbuf);
```

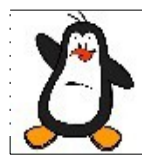
```
printf("%d rows x %d cols \n", wbuf.ws_row, wbuf.ws_col)
```

- Along with many other constants the second argument `TIOCGWINSZ` is defined in `/usr/include/asm-generic/ioctls.h`
- The `winsize` structure is defined in `/usr/include/x86_64-linux-gnu/bits/ioctl-types.h`



## `mystty.c`

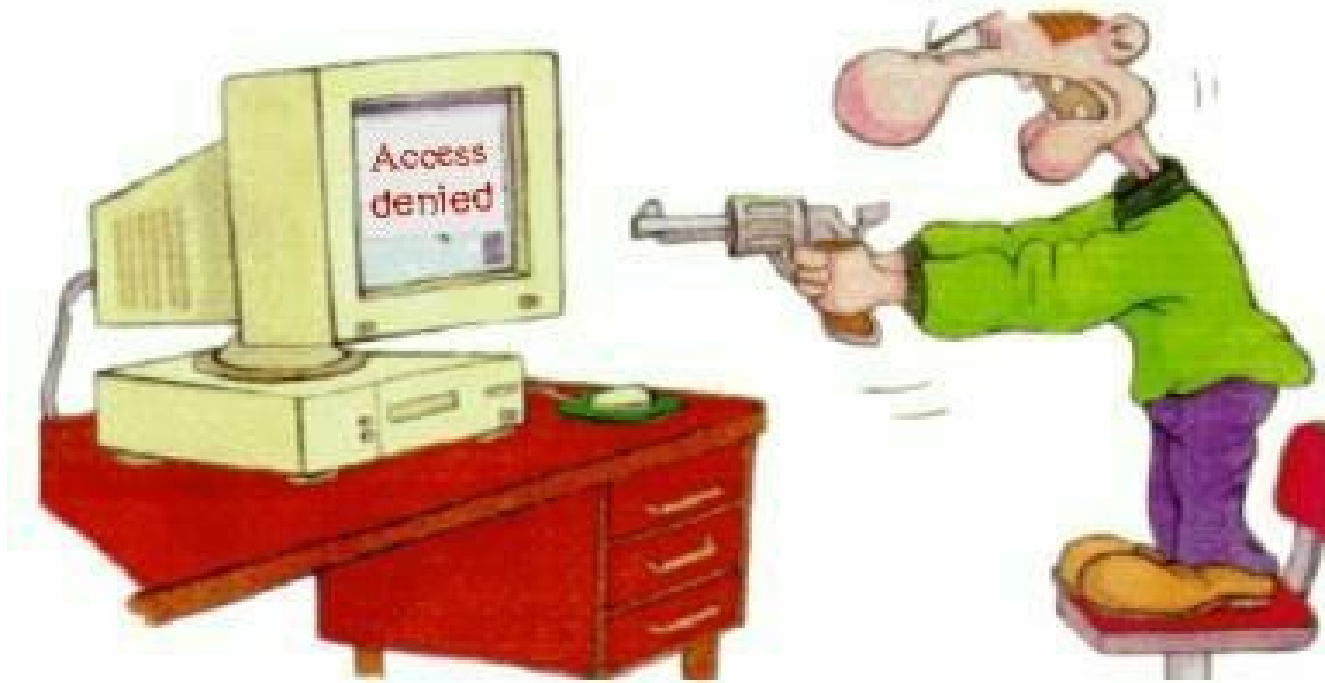
Try writing your own `stty` program, that mimic the behavior of `stty` command as close as possible



# Things To Do

---

O.k., and now you'll do exactly what I'm telling you !



If you have problems visit me in counseling hours. . . .

---