Linux System Programming – Mini project

Q. Write a Linux System Program to read Kernel Masters bmp Logo and write in to the monitor.

Hint: In Linux, Monitor is called Frame Buffer is located in /dev/fb0. Choose logo resolution that matches to your monitor resolution.

struct fb_var_screeninfo:

fb_var_screeninfo is used to describe the features of a video card that are user defined. With fb_var_screeninfo, things such as depth and the resolution may be defined.

This structure contains fields such as the X-resolution, Y-resolution, bits required to hold a pixel

```
struct fb_var_screeninfo {
  __u32 xres;               /* Visible resolution in the X axis */
  __u32 yres;               /* Visible resolution in the Y axis */
  /* ... */
  __u32 bits_per_pixel;     /* Number of bits required to hold a
                               pixel */

  /* ... */
  __u32 pixclock;           /* Pixel clock in picoseconds */
  __u32 left_margin;        /* Time from sync to picture */
  __u32 right_margin;       /* Time from picture to sync */
  /* ... */
  __u32 hsync_len;          /* Length of horizontal sync */
  __u32 vsync_len;          /* Length of vertical sync */
  /* ... */
};
```

struct fb_fix_screeninfo:

Fixed information about the video hardware, such as the start address and size of frame buffer memory, is held in `struct fb_fix_screeninfo`. These values cannot be altered by the user:

This structure defines the properties of a card that are created when a mode is set and can't be changed otherwise.

```
struct fb_fix_screeninfo {
         char id[16];                    /* identification string eg "TT Builtin" */
         unsigned long smem_start;       /* Start of frame buffer mem */
                                         /* (physical address) */
         __u32 smem_len;                 /* Length of frame buffer mem */
         __u32 type;                     /* see FB_TYPE_*               */
         __u32 type_aux;                 /* Interleave for interleaved Planes */
         __u32 visual;                   /* see FB_VISUAL_*             */
         __u16 xpanstep;                 /* zero if no hardware panning  */
         __u16 ypanstep;                 /* zero if no hardware panning  */
         __u16 ywrapstep;                /* zero if no hardware ywrap    */
         __u32 line_length;              /* length of a line in bytes    */
         unsigned long mmio_start;       /* Start of Memory Mapped I/O   */
                                         /* (physical address) */
         __u32 mmio_len;                 /* Length of Memory Mapped I/O  */
         __u32 accel;                    /* Indicate to driver which     */
                                         /*  specific chip/card we have  */
         __u16 capabilities;             /* see FB_CAP_*                 */
         __u16 reserved[2];              /* Reserved for future compatibility */
};
```

Why 32 bits per pixel?

Almost all cases of 32 bits per pixel assigns 24 bits to the color, and the remaining 8 are the alpha channel or unused.
"32 bit" also usually means 32 bits total per pixel, and 8 bits per channel, with an additional 8 bit alpha channel that's used for transparency. 16,777,216 colors again.
This is sometimes referred to as 32 bit RGBA. 24 bit and 32 bit can mean the same thing, in terms of possible colors.

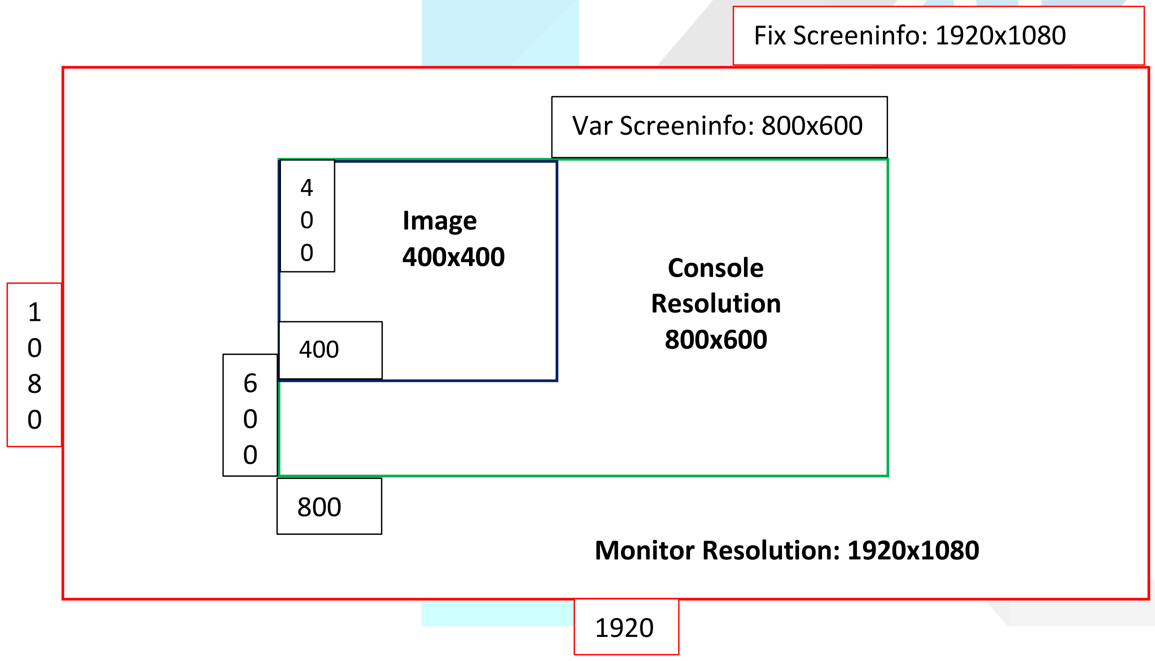
Assumption: Full HD monitor (1920x1080 resolution = 2K Resolution)
Monitor screen size = 1920*1080 = 2073600 pixels;
2073600*32bpp = 6,63,55,200 bits; 6,63,55,200/8 = 82,94,400 Bytes = 8.2M Bytes

Calculate screen size using fix info line width.
Line width is approx. 2K pixels and each pixel has 4 bytes (32bpp)
So, line width in bytes = 2K*4Bytes = 8K Bytes.

Screen size = line_length*var.yres

Your current console resolution is 800x600, so var.yres is 800.

So Screen size = fix_info.line_length * var_info.yres = 8K*600=4.8M Bytes;


Fix Screeninfo: 1920x1080
Var Screeninfo: 800x600
400
Image 400x400
400
Console Resolution 800x600
1080
600
800
Monitor Resolution: 1920x1080
1920

Please refer the below link to understand the bitmap file format

https://en.wikipedia.org/wiki/BMP_file_format#:~:text=The%20BMP%20file%20format%2C%20also,and%20OS%2F2%20operating%20systems.

Pseudo code

```
int fbFD, bmpFD;
struct fb_fix_screeninfo fix_info;          //fixed screen information
struct fb_var_screeninfo var_info;           //configurable screen info
struct stat finfo;
unsigned int *fb_ptr,temp;
char buff[80];
```

Step 1: Find out Variable Framebuffer resolution & fixed framebuffer length:

a. Open the frame buffer (/dev/fb0) in RDWR mode using open system call.
b. Collect the fix and var screen information (FBIOGET_FSCREENINFO & FBIOGET_VSCREENINFO) in the structure variables of struct fb_fix_screeninfo & struct fb_var_screeninfo using ioctl s/m call.
c. Collect line_length (line width) from fix_screeninfo and the screen resolutions (xres, yres & bpp) from var_screeninfo. Can calculate the screensize = xres * yres * (bpp / 8).

Step 2: Find out bmp image resolution:

a. Open the bitmap image in RDonly mode using open system call.
b. Read the header using read s/m call and collect the size of the image, **Offset** where the pixel array (bitmap data) can be found, screen resolution (**xres, yres & bpp**).

Step 3: Compare bmp resolution with framebuffer resolution:

a. If bmp resolution lesser than or equal to framebuffer resolution then got to next step otherwise show ERROR message “bmp resolution is more than framebuffer resolution”

Step 4: Framebuffer mapping with mmap() system call:

a. Map the frame buffer using mmap s/m call with size = fix_info.line_length * var_info.yres;
    a. fb_ptr = (unsigned int *) mmap (0, size, PROT_READ|PROT_WRITE, MAP_SHARED, fbFD, 0);

Step 5: Set image position:

a. Reposition read/write file offset to the offset provided in the header where the pixel array (bitmap data) can be found using lseek s/m call
    a. lseek(bmpFD, offset, SEEK_SET);
b. Divide the fix_info.line_length with 4, since we are not printing the image on the entire screen of our pc. int line_length = fix_info.line_length/4;

Step 6: Copy image pixel array in to the frame buffer:

a. Now copy the pixel **array** to the frame buffer.

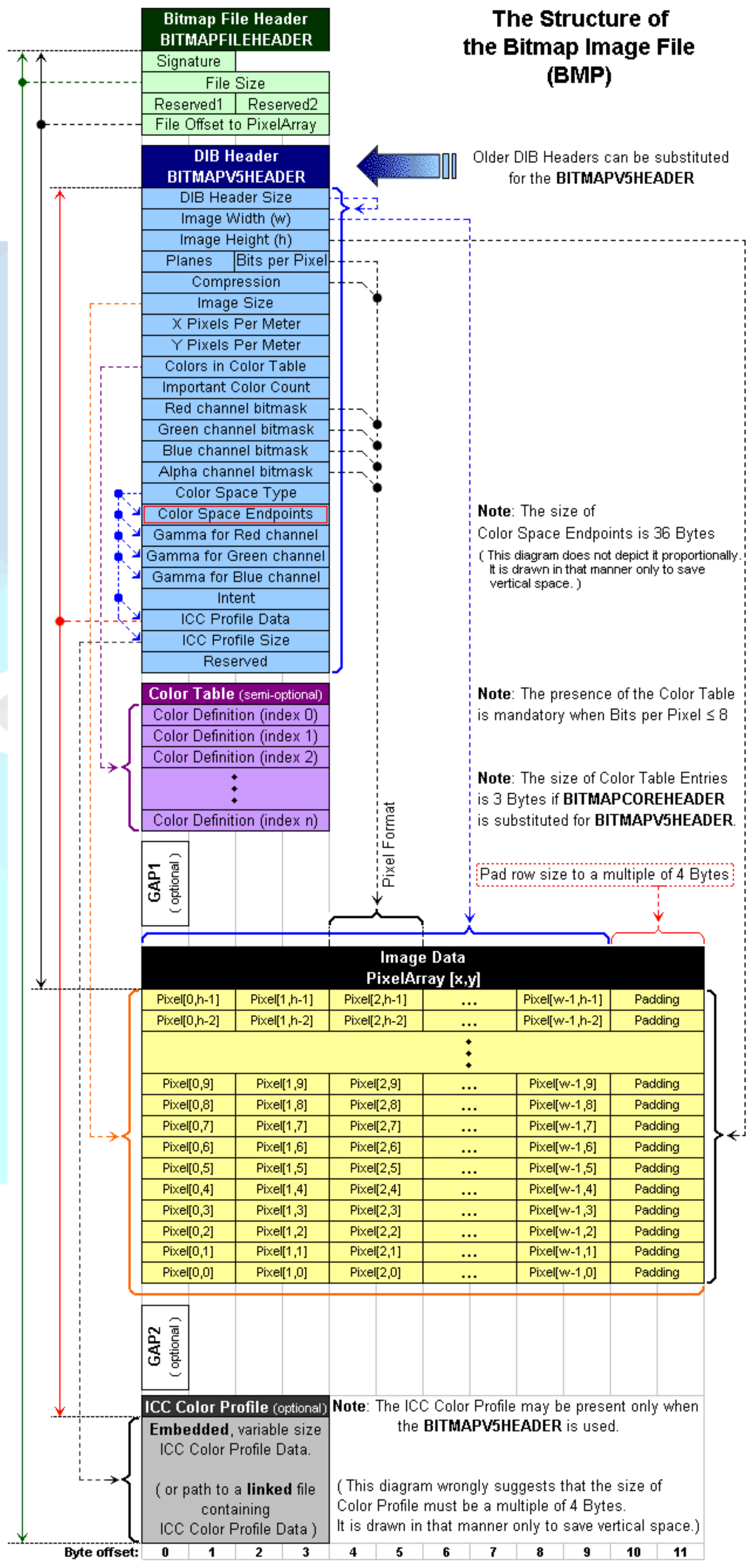
```
    a. for(i = bmp_yres - 1;i >= 0;i--)
           i. {
                  1. for(j = 0;j < bmp_xres;j++)
                  2. {
                         a. read(bmpFD, &temp, 4);
                         b. fb_ptr[i * line_length + j] = temp;
                  3. }
          ii. }
```

b. Unmap the image and frame buffer. And close both the files.

Example 1: KM_LOGO_800x600_32.bmp file content

800 x 600 pixel, 32bpp bitmap image				
Offset	**Size**	**Hex value**	**Value**	**Description**
BMP Header				
0h	2	42 4D	"BM"	ID field (42h, 4Dh)
2h	4	36 4C 1D 00	1920054 bytes (54+1920000)	Size of the BMP file (54 bytes header + 1920000 bytes data)
6h	2	00 00	Unused	Application specific
8h	2	00 00	Unused	Application specific
Ah	4	36 00 00 00	54 bytes (14+40)	Offset where the pixel array (bitmap data) can be found
DIB Header				
Eh	4	28 00 00 00	40 bytes	Number of bytes in the DIB header (from this point)
12h	4	20 03 00 00	800 pixels (left to right order)	Width of the bitmap in pixels
16h	4	58 02 00 00	600 pixels (bottom to top order)	Height of the bitmap in pixels. Positive for bottom to top pixel order.
1Ah	2	01 00	1 plane	Number of color planes being used
1Ch	2	20 00	32 bits	Number of bits per pixel
1Eh	4	00 00 00 00	0	BI_RGB, no pixel array compression used
22h	4	00 00 00 00	0 bytes	Size of the raw bitmap data (including padding)(No information. Must be 1920000 bytes)
26h	4	20 2E 00 00	11808 pixels/metre horizontal	X pixel per meter
2Ah	4	20 2E 00 00	11808 pixels/metre vertical	Y pixel per meter
2Eh	4	00 00 00 00	0 colors	Number of colors in the palette
32h	4	00 00 00 00	0 important colors	0 means all colors are important
Start of pixel array (bitmap data)				
36h	4	FF FF FF 00	255 255 255 00	White(1,1,1,0)
39h	4	FF FF FF 00	255 255 255 00	White(1,1,1,0)
3Ch	4	FF FF FF 00	255 255 255 00	White(1,1,1,0)
3Eh	4	FF FF FF 00	255 255 255 00	White(1,1,1,0)
41h	4	FF FF FF 00	255 255 255 00	White(1,1,1,0)
44h	4	FF FF FF 00	255 255 255 00	White(1,1,1,0)

Bmp file format structure


The Structure of the Bitmap Image File (BMP)
Bitmap File Header
BITMAPFILEHEADER
Signature
File Size
Reserved1
Reserved2
File Offset to PixelArray
DIB Header
BITMAPV5HEADER
Older DIB Headers can be substituted for the BITMAPV5HEADER
DIB Header Size
Image Width (w)
Image Height (h)
Planes
Bits per Pixel
Compression
Image Size
X Pixels Per Meter
Y Pixels Per Meter
Colors in Color Table
Important Color Count
Red channel bitmask
Green channel bitmask
Blue channel bitmask
Alpha channel bitmask
Color Space Type
Color Space Endpoints
Gamma for Red channel
Gamma for Green channel
Gamma for Blue channel
Intent
ICC Profile Data
ICC Profile Size
Reserved
Note: The size of Color Space Endpoints is 36 Bytes
(This diagram does not depict it proportionally. It is drawn in that manner only to save vertical space.)
Color Table (semi-optional)
Color Definition (index 0)
Color Definition (index 1)
Color Definition (index 2)
Color Definition (index n)
Note: The presence of the Color Table is mandatory when Bits per Pixel ≤ 8
Note: The size of Color Table Entries is 3 Bytes if BITMAPCOREHEADER is substituted for BITMAPV5HEADER.
GAP1 (optional)
Pixel Format
Pad row size to a multiple of 4 Bytes
Image Data
PixelArray [x,y]
Pixel[0,h-1]
Pixel[1,h-1]
Pixel[2,h-1]
Pixel[w-1,h-1]
Padding
Pixel[0,h-2]
Pixel[1,h-2]
Pixel[2,h-2]
Pixel[w-1,h-2]
Pixel[0,9] … Pixel[0,0]
Pixel[1,9] … Pixel[1,0]
Pixel[2,9] … Pixel[2,0]
Pixel[w-1,9] … Pixel[w-1,0]
GAP2 (optional)
ICC Color Profile (optional)
Embedded, variable size ICC Color Profile Data.
(or path to a linked file containing ICC Color Profile Data)
Note: The ICC Color Profile may be present only when the BITMAPV5HEADER is used.
(This diagram wrongly suggests that the size of Color Profile must be a multiple of 4 Bytes. It is drawn in that manner only to save vertical space.)
Byte offset: 0 1 2 3 4 5 6 7 8 9 10 11

Example 2: KM_LOGO_800x600_24.bmp file content

800 x 600 pixel, 24bpp bitmap image				
Offset	**Size**	**Hex value**	**Value**	**Description**
BMP Header				
0h	2	42 4D	"BM"	ID field (42h, 4Dh)
2h	4	38 F9 15 00	1440056 bytes (54+1440002)	Size of the BMP file (54 bytes header + 1440002 bytes data)
6h	2	00 00	Unused	Application specific
8h	2	00 00	Unused	Application specific
Ah	4	36 00 00 00	54 bytes (14+40)	Offset where the pixel array (bitmap data) can be found
DIB Header				
Eh	4	28 00 00 00	40 bytes	Number of bytes in the DIB header (from this point)
12h	4	20 03 00 00	800 pixels (left to right order)	Width of the bitmap in pixels
16h	4	58 02 00 00	600 pixels (bottom to top order)	Height of the bitmap in pixels. Positive for bottom to top pixel order.
1Ah	2	01 00	1 plane	Number of color planes being used
1Ch	2	18 00	24 bits	Number of bits per pixel
1Eh	4	00 00 00 00	0	BI_RGB, no pixel array compression used
22h	4	00 00 00 00	0 bytes	Size of the raw bitmap data (including padding)(No information. Must be 1440002 bytes)
26h	4	20 2E 00 00	11808 pixels/metre horizontal	X pixel per meter
2Ah	4	20 2E 00 00	11808 pixels/metre vertical	Y pixel per meter
2Eh	4	00 00 00 00	0 colors	Number of colors in the palette
32h	4	00 00 00 00	0 important colors	0 means all colors are important
Start of pixel array (bitmap data)				
36h	4	FF FF FF FF	255 255 255 255	White(1,1,1,1)
39h	4	FF FF FF FF	255 255 255 255	White(1,1,1,1)
3Ch	4	FF FF FF FF	255 255 255 255	White(1,1,1,1)
3Eh	4	FF FF FF FF	255 255 255 255	White(1,1,1,1)
41h	4	FF FF FF FF	255 255 255 255	White(1,1,1,1)
44h	4	FF FF FF FF	255 255 255 255	White(1,1,1,1)