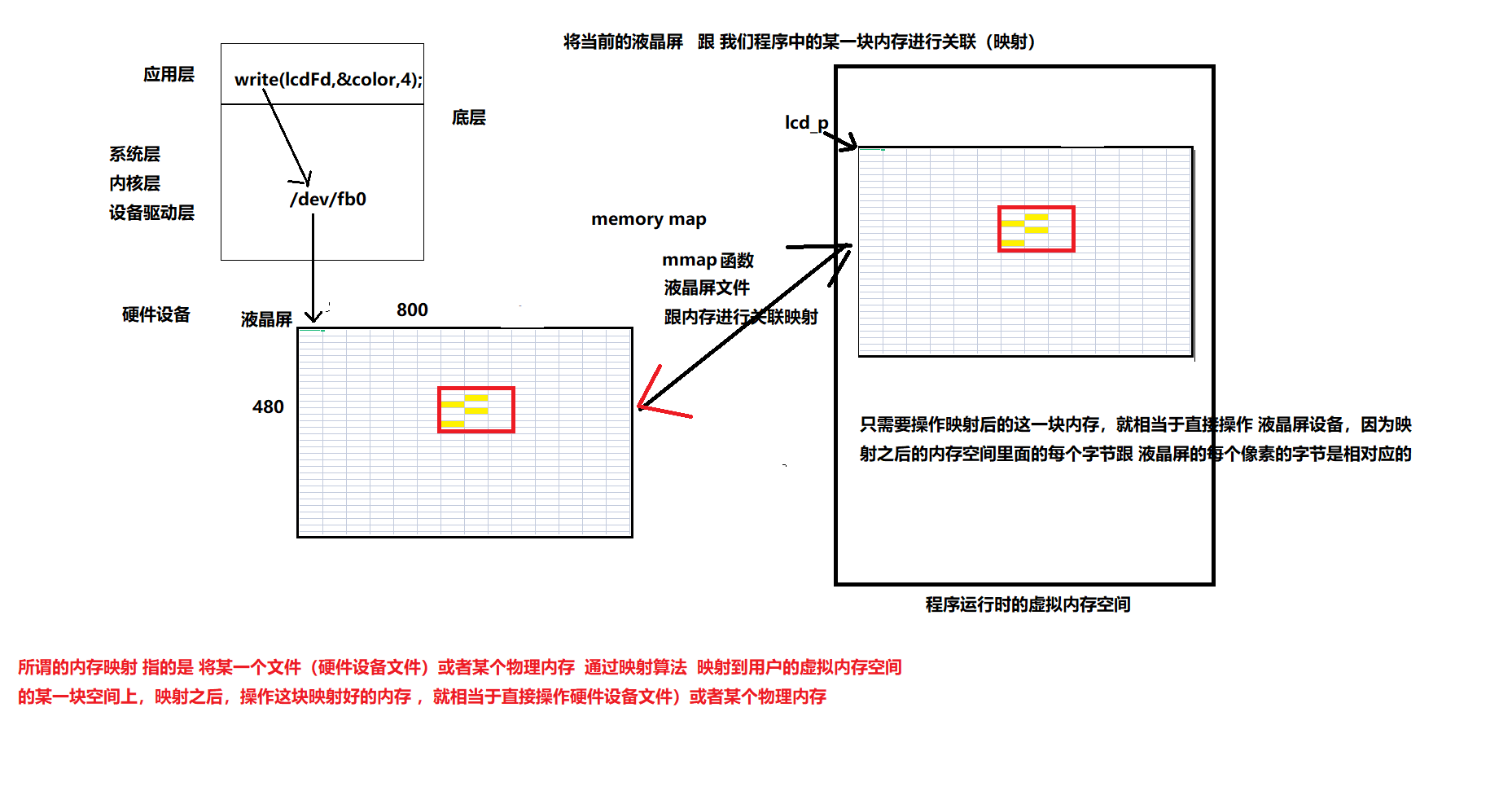
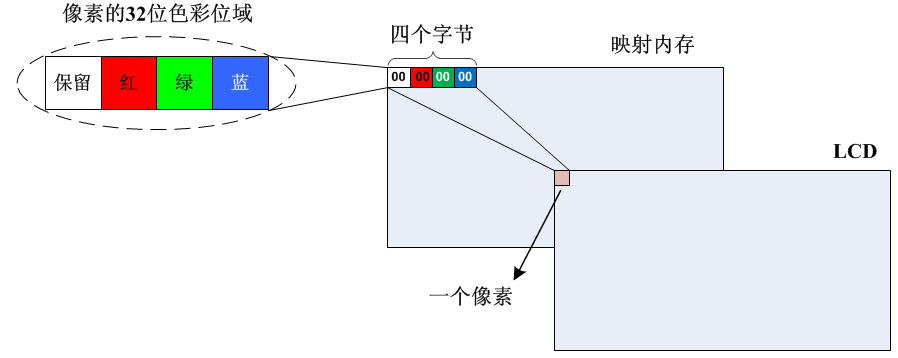
**内存映射**

虽然LCD设备本质上也可以看作是一个文件，在文件系统中有其对应的设备节点，可以像普通文件一样对其进行读写操作（read/write），但由于对字符设备的读写操作是以字节流（逐个字节往后填充）的方式进行的，所以如果我们想要液晶屏的指定位置写入颜色数据很不方便，需要进行文件光标的偏移。所以我们操作液晶屏还有另外一种更好的方式，那就是将液晶屏与某块内存进行关联，也就是为LCD做内存映射，将屏幕的每一个像素点跟映射内存一一对应，而映射内存可以是二维数组，因此就可以非常方便地通过操作二维数组中的任意元素，来操作屏幕中的任意像素点了。这里的映射内存，有时被称为显存。





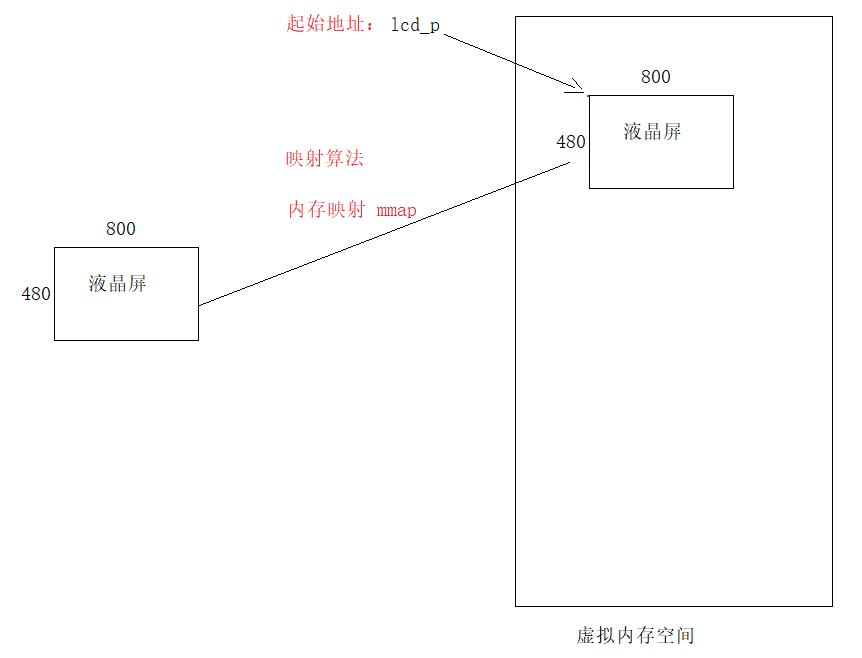
**1、内存映射与系统IO操作的区别**

普通IO流程：

open()访问文件 -> 得到一个文件描述符fd -> 直接往文件描述符fd中写入数据就可以 -> 关闭文件描述符fd。

内存映射流程：

open()访问文件 -> 得到一个文件描述符fd -> 根据文件描述符fd去内存空间中映射一片空间，得到一个地址lcd\_p -> 直接对这块内存进行读写 -> 对应文件就会有相应的变化。 -> 撤销映射 -> 关闭文件。



本质：将一个文件或者是一段物理内存直接映射到进程的虚拟空间中的某块内存空间，将来我们对文件进行数据处理，只需要处理这块内存即可。

**2、相关接口**

**内存映射**

#include <sys/mman.h>

void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset);

参数：

addr： 不为NULL -> 用户选择内存空间的起始地址 0.00001%

NULL -> 系统为用户去自动分配某一块空间 99.99999%

length： 映射的内存大小，例如：lcd液晶 800\*480\*4

prot：

PROT\_EXEC Pages may be executed.

PROT\_READ Pages may be read.

PROT\_WRITE Pages may be written.

PROT\_NONE Pages may not be accessed.

如果需要多个权限，则使用"|"连接在一起，例如： PROT\_READ|PROT\_WRITE

flags：

当有多个进程同时映射这块内存时，该参数可以决定在某一个进程内使映射内存的数据发生变更时是否影响其他进程，也决定是否影响其对应的文件数据。

MAP\_SHARED：1、进程共享内存 2、内存数据的变更同步到对应的文件

MAP\_PRIVATE：1、内存私有 2、内存数据的变更不会影响到对应的内存

fd：文件描述符

offset：文件偏移量（从文件的哪个地方开始进行映射）0

返回值：

成功：指向那片内存空间的区域的地址

失败：(void \*)-1

**解除内存映射**

#include <sys/mman.h>

int munmap(void \*addr, size\_t length);

参数：

addr： 需要撤销映射的内存的起始地址 -> mmap()函数的返回值

length：撤销的长度

返回值：

成功：0

失败：-1

**3、示例**

例子1： 使用内存映射的方式显示颜色到lcd液晶上。

int main()

{

int colorbuf[800\*480];

int i;

//1、打开液晶屏文件/dev/fb0，

int lcdfd = open("/dev/fb0",O\_RDWR);

if(lcdfd == -1)

{

printf("open lcd error\n");

return -1;

}

//将液晶屏文件 通过内存映射mmap的方式 映射到虚拟内存空间的某一块空间上，得到这一块内存的起始地址，后续我们操作这片内存空间，就相当于直接操作液晶屏文件

int \*lcd\_p = mmap(NULL, //你要映射的内存空间的起始地址，为NULL系统自动给你分配

800\*480\*4,//你要映射的内存空间的大小

PROT\_READ|PROT\_WRITE,//映射的权限

MAP\_SHARED,//1、进程共享 2、对应的文件会同步发生变化

lcdfd,//映射液晶屏文件

0//偏移量 ，默认为0

);

//解除映射

munmap(lcd\_p, 800\*480\*4);

//3、关闭文件

close(lcdfd);

}