在Linux操作系统下FrameBuffer直接写屏 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/252/2021_2022__E5_9C_A8Linux_E6_93_c103_252888.htm 因为Linux是工作在保护模式下，所以用户态进程是无法象DOS那样使用显卡BIOS里提供的中断调用来实现直接写屏，故Linux抽象出FrameBuffer这个设备来供用户态进程实现直接写屏。 在继续下面的之前，先说明几个背景知识： FrameBuffer主要是根据VESA标准的实现的，所以只能实现最简单的功能。 由于涉及内核的问题，FrameBuffer是不允许在系统起来后修改显示模式等一系列操作。（好象很多人都想要这样干，这是不被允许的，当然如果你自己与驱动的话，是可以实现的） 对FrameBuffer的操作，会直接影响到本机的所有控制台的输出，包括XWIN的图形界面。 好，现在可以让我们开始实现直接写屏： 打开一个FrameBuffer设备 通过mmap调用把显卡的物理内存空间映射到用户空间 直接写内存。 fbtools.h#ifndef _FBTOOLS_H_#define _FBTOOLS_H_#include /* a framebuffer device structure */typedef struct fbdev{int fb.unsigned long fb_mem_offset.unsigned long fb_mem.struct fb_fix_screeninfo fb_fix.struct fb_var_screeninfo fb_var.char dev[20].} FBDEV, *PFBDEV./* open amp.(pFbdev->fb_var))){printf("ioctl FBIOGET_VSCREENINFO\n").return FALSE.}if (-1 == ioctl(pFbdev->fb,FBIOGET_FSCREENINFO,amp. (~PAGE_MASK).pFbdev->fb_mem = (unsigned long int)mmap(NULL, pFbdev->fb_fix.smem_len pFbdev->fb_mem_offset,PROT_READ | PROT_WRITE,

MAP_SHARED, pFbdev->fb, 0).if (-1L == (long) pFbdev->fb_mem){printf("mmap error! mem:%d offset:%d\n", pFbdev->fb_mem,pFbdev->fb_mem_offset).return FALSE.}return TRUE.}/* close frame buffer */int fb_close(PFBDEV pFbdev){close(pFbdev->fb).pFbdev->fb=-1.}/* get display depth */int get_display_depth(PFBDEV pFbdev).{if(pFbdev->fbfb_var.bits_per_pixel.}/* full screen clear */void fb_memset (void *addr, int c, size_t len){memset(addr, c, len).}/* use by test */#define DEBUG#ifdef DEBUGmain(){FBDEV fbdev.memset(amp.fbdev)==FALSE){printf("open frame buffer error\n").return.}fb_memset(fbdev.fb_mem fbdev.fb_mem_offset, 0, fbdev.fb_fix.smem_len).fb_close(&amp.fbdev).}