디바이스 드라이버 응용 어플리케이션

- Chapter 11 -

Contents

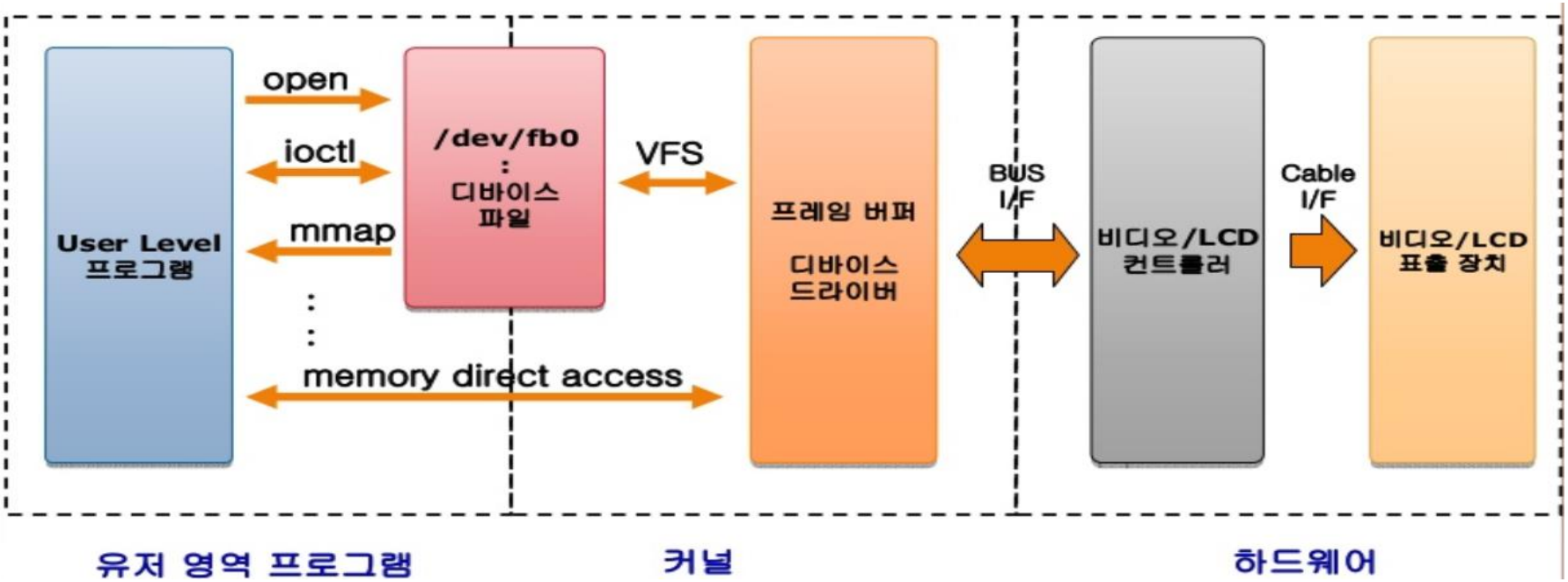
FrameBuffer의 개념

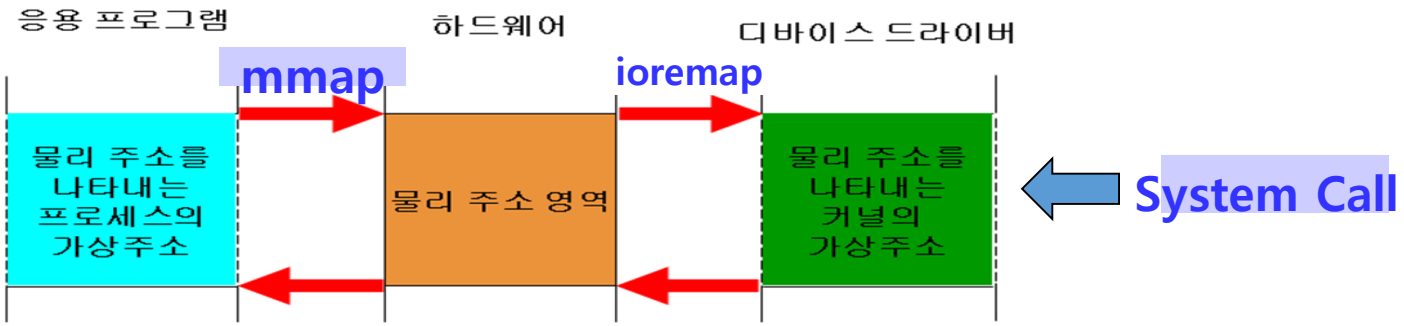
- FrameBuffer(FB)
  - Frame buffer는 임베디드 리눅스에서 그래픽을 표현하는 역할 (안드로이드도 이용)
  - 커널에서 확보되는 비디오 메모리 내의 정보를 실제 FrameBuffer로 전달할 메모리 공간
  - FrameBuffer(FB)는 그래픽 하드웨어를 추상화(abstraction) 해 줌
    - 표시장치의 종류가 많고 환경도 다양하기 때문에 디바이스 드라이버로써 이식성을 높이고 표준화된 인터페이스를 제공하기 위해 필요
    - FB 이용 시 임베디드 장비에서 여러 장비에 대한 이식성이 향상되는 장점
  - FrameBuffer 디바이스 드라이버
    - 그래픽 H/W를 사용자 프로그램 레벨에서 제어하는 기능을 지원하는 디바이스 드라이버
    - 리눅스에서는 일반적으로 해당 하드웨어 장치 드라이버를 제공(/dev/fb[n])
    - /dev/fb0에 데이터를 쓰는 것 만으로 접근이 가능 (system call 또는 mmap 활용)
    - Frame Buffer의 지원 및 노드 확인 방법

```
# ls -l /dev/f*
```

FrameBuffer의 개념

- FrameBuffer(FB) 의 제어 구조





FrameBuffer의 개념

FrameBuffer의 기능

1) 리눅스 표준 그래픽 디바이스 드라이버

   - 응용 프로그램이 그래픽 장치를 사용할 수 있도록 커널에서 제공해주는 장치 드라이버

2) 그래픽 장치 초기화

   - 하드웨어를 초기화 하고 기본 해상도를 제어

3) 그래픽 장치 정보 제공

   - 설정된 해상도 등의 그래픽 장치의 정보를 어플리케이션에서 읽을 수 있도록 정보를 제공

4) 그래픽 장치 제어

   - 해상도 조정, 모니터 주파수 조정 등을 응용 프로그램이 제어할 수 있게 지원

5) 그래픽 메모리 매핑

   - 응용 프로그램에서 직접적으로 비디오 메모리에 접근하여 처리할 수 있도록 메모리 매핑을 제공

FrameBuffer의 개념

- FrameBuffer 구조체
  - Framebuffer와 관련된 헤더파일은 linux/fb.h에 정의되어 있음
  - 응용 프로그램에서는 정의된 구조체를 이용해 IOCTL 함수로 FB 디바이스 드라이버로 접근
  - 프레임버퍼의 정보는 다음의 두 가지의 구조체에 담겨진다
    - fb_fix_screeninfo
      - 프레임버퍼 관련 고정된 정보를 가지고 있는 구조체
      - 프레임 버퍼의 메모리 크기 등과 같은 고정된 하드웨어의 정보를 얻기
    - fb_var_screeninfo
      - 프레임버퍼의 가변 정보를 가지고 있는 구조체
      - 비디오 모드, 모니터 주파수, 해상도(Resolution), bpp(bit per pixel) ,가상 화면 등등 변경이 가능한 프레임버퍼 정보를 얻거나 설정
    - VESA(Video Electronics Standards Association) ➔ 베사
      - 비디오와 멀티미디어 장치의 표준화를 추진하는 단체인 비디오 전자 공학 협회
      - 해상도, 버스 및 모니터/TV 등의 고정방식 규격 등의 매우 다양한 표준화 활동을 한다.

FrameBuffer의 개념

- FrameBuffer 구조체
  - fb_fix_screeninfo → 고정된 정보를 가지고 있는 구조체

```
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;                    /* Type of acceleration available */
        __u16 reserved[3];              /* Reserved for future compatibility */
};
```

FrameBuffer의 개념

- FrameBuffer 구조체
  - fb_var_screeninfo → 가변 정보를 가지고 있는 구조체

```
Struct fb_var_screeninfo{
        __u32 xres;                     /* visible resolution*/
        __u32 yres;
        __u32 xres_virtual;             /* virtual resolution*/
        __u32 yres_virtual;
        __u32 xoffset;                  /* offset from virtual to visible */
        __u32 yoffset;                  /* resolution*/
        __u32 bits_per_pixel;           /* guess what*/
        __u32 grayscale;                /* != 0 Graylevelsinstead of colors */
        structfb_bitfieldred;           /* bitfieldin fbmemif true color, */
        structfb_bitfieldgreen;         /* else only length is significant */
        structfb_bitfieldblue;
        structfb_bitfieldtransp;        /* transparency*/
```

FrameBuffer의 개념

- FrameBuffer 구조체의 접근 방법
  - ioctl() 함수를 이용하여 시스템콜을 통해 커널이 관리하고 있는 프레임버퍼 정보를 가지고 오거나 프레임버퍼 설정을 변경하는 내용을 적용하는 예

```
ioctl_ret = ioctl(fb_fd, FBIOGET_VSCREENINFO, &fbvar);
ioctl_ret = ioctl(fb_fd, FBIOGET_FSCREENINFO, &fbfix);

fbvar.bits_per_pixel= 8;
ioctl_ret= ioctl(fb_fd, FBIOPUT_VSCREENINFO, &fbvar);
```

프레임버퍼 실습 공통 사항

▶ 실습 종류

1) Framebuffer - fbinfo
2) Framebuffer - fbset → 잘못 설정하면 화면해상도가 바뀔 수 있으니 실습하지 말것
3) Framebuffer - fbpixel → 잘 안보임
4) Framebuffer - fbrect
5) Framebuffer - fbranrect → 일반적인 I/O 호출(시스템 콜)을 통한 랜덤 사각형 그리기
6) Framebuffer - fbmranrect → mmap() 호출을 통한 랜덤 사각형 그리기

프레임버퍼 실습 공통 사항

▶ 실습 방법

1) 프레임 버퍼 예제 가져와서 압축 풀기 (배포된 CD에서 /work/achro5250 폴더로 가져옴)

```
# cd ~/ACHRO-5250-1.5.0.2/examples/linux/application
# cp -a frame_buffer.tar.gz /work/achro5250
# cd /work/achro5250
# tar xvfz frame_buffer.tar.gz
# cd /work/achro5250/framebuffer
```

2) 실습 소스프로그램 컴파일하고 타겟 보드로 옮기는 작업 (모든 소스들을 동일한 방법)

```
# arm-linux-gcc -o fbXXX mmap_fbXXX.c
```

- arm-linux-gcc가 동작되지 않으면 심볼릭 링크 설정이 안된 것임 ➔ arm-none-linux-gnueabi-gcc 명령을 사용할 것

① nfs를 이용해 타겟으로 전송하여 실행 ➔ # cp fbXXX /nfsroot

② 우분투에서 SD 카드의 “/media/Achro5250_System”으로 복사하고 타겟에서 실행

→ 이 경우 모든 소스를 컴파일 한 후 한번에 옮기는 것이 편리. 이번 실습에서는 이 방법을 이용

프레임버퍼 실습 정보 수집

프레임버퍼 정보 수집 - fbinfo.c

1) 프레임 버퍼의 정보는 두 가지의 구조체에 담겨진다

- fb_var_screeninfo: 프레임버퍼의 가변 정보를 가지고 있는 구조체
- fb_fix_screeninfo: 프레임버퍼의 고정 정보

2) 프레임 버퍼 정보 수집 프로그램 - fbinfo.c

- 프레임 버퍼를 사용하기 위한 헤더

```
/* Filename : fbinfo.c */
// …생략…
 #include <linux/fb.h>   // Frame Buffer API
```

- 프레임 버퍼 구조체 생성

```
int main(int argc, char** argv)  {
    int check;
    int frame_fd;
    struct fb_var_screeninfo st_fvs; // 프레임버퍼의 가변 정보
    struct fb_fix_screeninfo st_ffs; // 프레임버퍼의 고정 정보
```

프레임버퍼 실습 정보 수집

프레임버퍼 정보 수집 - fbinfo.c - 계속

프레임 버퍼 장치 열기

```
frame_fd = open("/dev/fb0",O_RDWR);
if(frame_fd < 0)  {
      perror("Frame Buffer Open Error!");
      exit(1);
}
```

프레임 버퍼 가변 정보 수집

```
check = ioctl(frame_fd, FBIOGET_VSCREENINFO,&st_fvs);
if(check < 0)   {
        perror("Get Information Error - VSCREENINFO!");
        exit(1);
 }
```

프레임 버퍼 고정 정보 수집

```
check = ioctl(frame_fd,FBIOGET_FSCREENINFO,&st_ffs);
if(check < 0)   {
       perror("Get Information Error - FSCREENINFO!");
       exit(1);
}
```

프레임버퍼 실습 정보 수집

프레임버퍼 정보 수집 - fbinfo.c - 계속

수집된 정보의 출력

```
printf("==================₩n");
printf("Frame Buffer Info₩n");
printf("-----------------₩n");
printf("X - res    : %d₩n",st_fvs.xres);
printf("Y - res    : %d₩n",st_fvs.yres);
printf("X - v_res : %d₩n",st_fvs.xres_virtual);
printf("Y - v_res : %d₩n",st_fvs.yres_virtual);
printf("Bit/Pixel : %d₩n",st_fvs.bits_per_pixel); //BPP
printf("-----------------₩n");
printf("Buff Size : %d₩n",st_ffs.smem_len);
printf("==================₩n");
```

열었던 장치 닫기

```
close(frame_fd);
```

프레임버퍼 실습 정보 수집

▶ 프레임버퍼 정보 수집 - fbinfo.c - 컴파일 및 실행

① SD카드를 이용 ➔ 반드시 이 방법을 이용할 것!

▷ 다음 명령을 실행한 후 컴파일 된 바이너리를 우분투에서 SD 카드의 "/media/Achro5250_System" 디렉토리에 복사한 후 타겟에 넣고 실행(./fbinfo)

```
# arm-none-linux-gnueabi-gcc -o fbinfo fbinfo.c
# cp -a fbinfo /media/Achro5250_System/
# sync
```

① NFS를 이용

▷ 컴파일 된 바이너리를 NFS 디렉토리에 복사한다.

```
# arm-none-linux-gnueabi-gcc -o fbinfo fbinfo.c
# cp -a fbinfo /nfsroot
# sync
```

▷ Mounf nfs directory. changing nfs directory and execute. ➔ 〈타겟 보드〉

```
# mount -t nfs [우분투 IP]:/nfsroot /mnt/nfs -o rw, rsize=1024,nolock // mount nfs
# cd /mnt/nfs                    // move to nfs directory
# ./fbinfo                       // run binary
```

프레임버퍼 정보 변경

프레임버퍼 정보 변경 - fbset.c (실행하지 말것)

프레임 버퍼는 기기가 지원하는 범위 내에서 해당 출력을 설정에 맞게 바꿀 수 있다.

프레임 버퍼 정보 변경 프로그램 - fbset.c

- 프레임 버퍼를 사용하기 위한 헤더

```
/* FILENAME : fbset.c  */
// ... 생략 ...
#include <linux/fb.h>
```

- 프레임 버퍼 사용자 구조체 생성

```
typedef struct _user_fb {  // 사용자 지정 정보를 담는 구조체 함수 타입 선언
    int xres;
    int yres;
    int bpps;
} user_fb;
```

- 프레임 버퍼 장치 열기

```
int main(int argc, char** argv)  {
    // ... 생략 ...
    struct fb_var_screeninfo st_fvs;
    user_fb my_fb = {400,240,24};    // my_fb 생성과 동시에 초기화
    frame_fd = open("/dev/fb0",O_RDWR);
```

프레임버퍼 정보 변경

프레임버퍼 정보 변경

프레임 버퍼 정보 변경 프로그램 - fbset.c - 계속

프레임 버퍼를 사용하기 위한 헤더

```
/* FILENAME : fbset.c  */
// ... 생략 ...
#include <linux/fb.h>
```

기존 프레임버퍼 정보 가져오기

```
if(check=ioctl(frame_fd, FBIOGET_VSCREENINFO,&st_fvs))
```

프레임 버퍼 변경 정보 설정

```
    st_fvs.xres = my_fb.xres;  //사용자 지정 값을 프레임버퍼 구조체 값으로 변경
    st_fvs.yres = my_fb.yres;
    st_fvs.bits_per_pixel = my_fb.bpps;

    check=ioctl(frame_fd, FBIOPUT_VSCREENINFO,&st_fvs)
```

프레임버퍼 닫기

```
close(frame_fd);
```

프레임버퍼 정보 사용

- 프레임버퍼 정보 사용 - fbpixel.c 프로그램
  - 프레임 버퍼에 점(Pixel) 그리기 프로그램
    - 프레임 버퍼의 픽셀 데이터
    - LCD 속성에 따라서 한 점을 표시하는데 8비트, 16비트, 24비트 등으로 구성된다.
      - bpp(bit per pixel)
      - 8비트는 일반적으로 그레이스케일과 같은(농도를 가지는 흑백) 타입의 영상 출력에 이용
    - 16비트인 경우 LCD의 데이터 라인이 16개이며 RGB 5, 6, 5등을 통해 영상을 출력.
      - RGB565는 점 하나를 표현하는데 Red 5비트, Green 6비트, Blue 5비트를 이용
    - 24비트인 경우 LCD의 데이터 라인이 24개이며, RGB 8, 8, 8등을 통해 영상을 출력
      - RGB888은 점 하나를 표현하는데 Red 8비트, Green 8비트, Blue 8비트를 이용
    - Achro-5250은 32비트로 설정되어 있지만 실제는 24비트 RGB 8, 8, 8을 이용하여 영상을 출력

프레임버퍼 정보 사용

프레임버퍼 정보 사용 - fbpixel.c 프로그램

프레임버퍼 헤더 선언

```
#include <linux/fb.h>
```

픽셀 생성 함수 작성

```
unsigned int makepixel(U32  r, U32 g, U32 b)  {
    return (U32)((r<<16)|(g<<8)|b);
}
```

프레임 버퍼 열기

```
frame_fd = open("/dev/fb0",O_RDWR)
```

프레임 버퍼 정보 수집

```
check=ioctl(frame_fd,FBIOGET_VSCREENINFO,&fvs)

if(fvs.bits_per_pixel != 32) {
        perror("Unsupport Mode. 32Bpp Only.");
        exit(1);
}
```

프레임버퍼 정보 사용

- 프레임 버퍼 정보 수집

```
check=ioctl(frame_fd,FBIOGET_VSCREENINFO,&fvs)
if(fvs.bits_per_pixel != 32) {
        perror("Unsupport Mode. 32Bpp Only.");
        exit(1);
}
```

- 프레임 버퍼의 시작 위치 설정과 이동 위치 설정

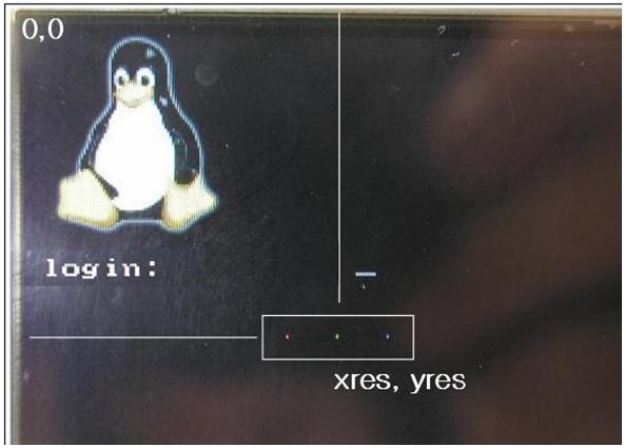
```
lseek(frame_fd, 0, SEEK_SET)
offset = 120*fvs.xres*sizeof(pixel)+100*sizeof(pixel);
lseek(frame_fd,offset,SEEK_SET);
```

- 표시할 점의 색상 설정

```
// 0~255의 값을 이용, 255,0,0 은 적색 – RGB 순으로 기록
pixel = makepixel(255,0,0);
```

- 데이터 기록

```
// write의 기록 단위는 Byte 카운트이므로 나누기 8
write(frame_fd, &pixel, fvs.bits_per_pixel/8);
```



프레임버퍼 정보 사용

프레임버퍼 정보 사용 - fbrect.c 프로그램

프레임 버퍼에 사각형(Rectangle) 그리기 프로그램

선언부 및 기본 소스 구현

- 기존 점그리기와 동일하게 헤더를 선언하고, 동일한 장치를 연다.

사각형을 그리기 위한 좌표

- posx1(X축 시작점), posx2(X축 끝점), posy1(Y축 시작점), posy2(Y축 끝점)

```
int posx1, posy1, posx2, posy2;
```

사각형의 크기 설정

```
int main(int argc, char** argv)  {
    // ... 생략 ...
    posx1 = 100;  // 사각형의 크기를 결정한다.
    posx2 = 150;  // rect(100,150,120,170)을 구현한 것이다.
    posy1 = 120;
    posy2 = 170;
```

프레임버퍼 정보 사용

프레임버퍼 정보 사용 - fbrect.c

점 그리기

```
    for(repy=posy1; repy < posy2; repy++) {
        offset = repy * fvs.xres * (32/8) + posx1 * (32/8);
        if(lseek(frame_fd, offset, SEEK_SET) < 0) {
            perror("LSeek Error!");
            exit(1);
        }
        for(repx = posx1; repx <= posx2; repx++)
            write(frame_fd, &pixel,(32/8));
    }
```

I/O 방식과 memory map 방식의 비교

- I/O 방식의 프로그램 - 랜덤 사각형 그리기
  - 프레임 버퍼를 사용하기 위한 헤더
    - 앞서 사각형 그리기(fbrect.c)와 동일한 소스이며 다만, 사각형을 그리기 위한 색상과 위치를 난수를 이용하여 일정 수만큼 반복하도록 한 다음 두 가지 기록 방식에 대한 결과를 얻도록 한다
  - 랜덤 색상 만드는 함수
    - 색상을 좀더 다양하게 생성하고 싶다면, srand와 time을 적절이 이용하면 된다.

```
unsigned int random_pixel(void)  {
    return rand();
}
```

  - 좌표 스와핑
    - 시작점보다 끝점이 값이 작은 경우 스와핑

```
void swap(int *swapa, int *swapb) {
    int temp;
    if(*swapa > *swapb) {
        temp = *swapb;
        *swapb = *swapa;
        *swapa = temp;
    }
}
```

I/O 방식과 memory map 방식의 비교

I/O 방식의 프로그램 - 랜덤 사각형 그리기

I/O 방식

I/O 방식의 반복

```
while(1<count--) {
        pixel = random_pixel(); // 랜덤 색상 값 구함
        posx1 = (int)((fvs.xres*1.0*rand())/(RAND_MAX+1.0));   // 랜덤좌표 x1
        posx2 = (int)((fvs.xres*1.0*rand())/(RAND_MAX+1.0));   // 랜덤좌표 x2
        posy1 = (int)((fvs.yres*1.0*rand())/(RAND_MAX+1.0));   // 랜덤좌표 y1
        posy2 = (int)((fvs.yres*1.0*rand())/(RAND_MAX+1.0));   // 랜덤좌표 y2
        swap(&posx1, &posx2); // 항목당 초기위치 설정
        swap(&posy1, &posy2); // 가독성상 표현, 조건문 사용이 우선되어야 함.

        for(repy=posy1; repy < posy2; repy++) {
            offset = repy * fvs.xres * (32/8) + posx1 * (32/8);
            if(lseek(frame_fd, offset, SEEK_SET) < 0) {
                perror("LSeek Error!");
                exit(1);
            }
            for(repx = posx1; repx <= posx2; repx++)
                write(frame_fd, &pixel,(32/8));
        } // End of For
    } // End of While
```

I/O 방식과 memory map 방식의 비교

- I/O 방식의 프로그램 - 랜덤 사각형 그리기
  - 메모리 매핑 방식
    - 메모리 매핑 방식의 경우 매핑 영역지정 및 할당

```
unsigned int* pfbdata;
pfbdata = (unsigned short *) mmap(0, fvs.xres*fvs.yres*32/8, PROT_READ|
      PROT_WRITE, MAP_SHARED, frame_fd, 0);        // 메모리 매핑
if((unsigned)pfbdata == (unsigned)-1) {
          perror("Error Mapping!₩n");
}
```

    - 반복구문 - mmap 방식의 반복

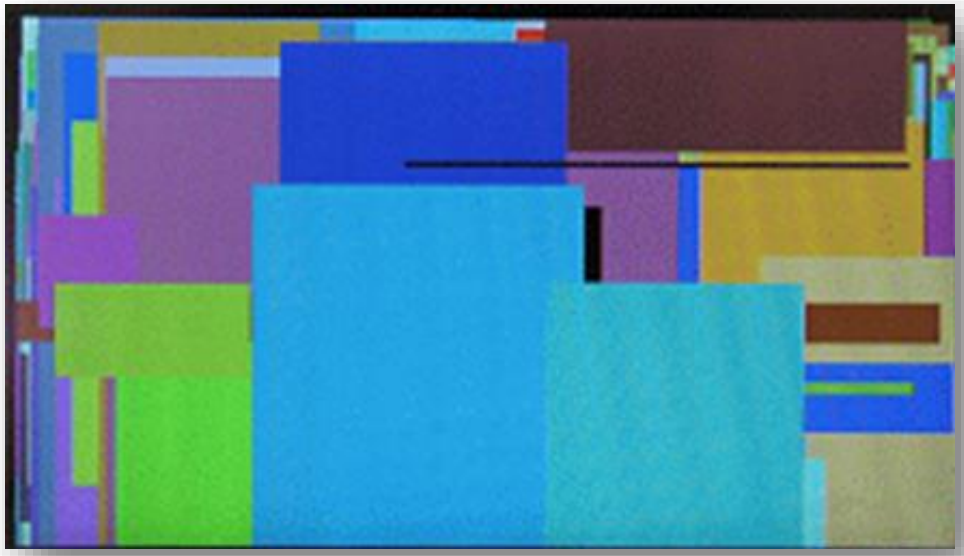
```
while(1<count--)  {
        pixel = random_pixel();
        posx1 = (int)((fvs.xres*1.0*rand())/(RAND_MAX+1.0));
        posx2 = (int)((fvs.xres*1.0*rand())/(RAND_MAX+1.0));
        posy1 = (int)((fvs.yres*1.0*rand())/(RAND_MAX+1.0));
        posy2 = (int)((fvs.yres*1.0*rand())/(RAND_MAX+1.0));
        swap(&posx1, &posx2);
        swap(&posy1, &posy2);

        for(repy=posy1; repy <= posy2; repy++) {
            offset = repy * fvs.xres;

            for(repx=posx1;repx<=posx2;repx++)
              *(pfbdata+offset+repx) = pixel;
        } // End of For
    } // End of While
```

I/O 방식과 memory map 방식의 비교

- I/O 방식과 memory map의 결과 비교
  - 랜덤 사각형 결과
    - 랜덤이므로 사각형의 크기나 위치는 달라질 수 있다
    - 메모리 매핑 방식의 속도가 월등히 빠르다
      - I/O 방식은 시스템 콜을 이용하므로 커널 영역에서의 수행 시간이 크다
      - 메모리 맵 방식은 응용 프로그램의 수행 시간이 커널 영역의 수행시간에 비해서 크지만, 커널 내부에서의 불필요한 복사가 줄어들기 때문에 전체 수행시간은 매우 단축된다



Q & A