실시간 운영체제를 위한 Graphic User Interface의 설계

윤기현[0] 김용희 박희상 이철훈
충남대학교 컴퓨터공학과
(khyoon[0], yonghee, hspark)@ce.cnu.ac.kr, chlee@ce.cnu.ac.kr

Design of Graphic User Interfaces for the Real Time Operating System

Ki-Hyun Yoon [0], Yong-Hee Kim, Hee-Sang Park, and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약

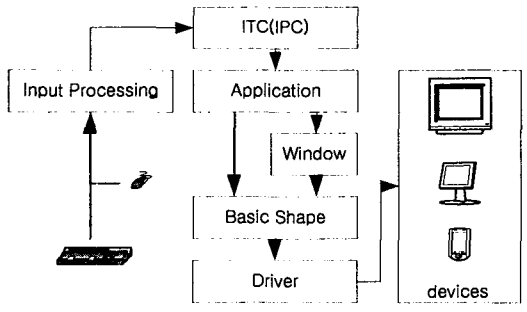
실시간 운영체제를 이용하는데 있어서 GUI(Graphic User Interface)는 실시간 운영체제를 이용한 제품을 사용하는 사람들에게 제품에 대한 편한 환경을 제공하는데 목적이 있다. 이에 본 논문에서는 실시간 운영체제(Real Time Operating System)상에서 GUI를 구현하는데 있어서 중요한 부분들을 고려하고, 실시간 운영체제에 알맞은 환경을 구축하기 위한 Graphic User Interface를 설계하였다.

1. 서 론

최근 몇 년사이 그래픽 하드웨어의 발전 및 가격하락과 함께 PDA, cellular phone등의 사용과 최근 상용화 되기 시작한 IMT 2000 의 등장으로 실시간 운영체제에서의 GUI의 필요성이 증대되고 있으나, 국내에서의 연구는 아직 미미한 실정이다.

본 논문은 GUI를 설계하는데 있어서 고려해야 할 부분과 그에 대한 실제 설계에 대한 내용을 기술하고 있다. 2 장에서는 그래픽 하드웨어와 GUI 사이의 Interface를 어떤 형태로 할 것인가를, 3 장에서는 Bitmap을 어떻게 보여줄 것인가, 4 장에서는 font의 처리를 어떻게 할 것인가, 5 장에서는 Container 객체의 설계를 어떻게 할 것인가, 6 장에서는 Update된 화면의 처리를 어떻게 할 것인가를 기술 하였다. 마지막으로, 7 장에서는 결론 및 향후 연구과제를 기술하였다.

2. 하드웨어 인터페이스 설계



[그림 1] Graphic User Interface의 구성

GUI의 목적은 그래픽 하드웨어에 어플리케이션이 원하는 화면을 보여줄 수 있도록 하는데 있다. [그림 1]에서 보는 바와 같이, 어플리케이션은 입력 장치(키보드,마우스)로부터 입력을 받아서 처리하고, 화면은 어플리케이션에서 표현해야 할 화면을 처리하여 그래픽 하드웨어로 출력한다. 이때, 그래픽 하드웨어와 어플리케이션 사이에서 출력 환경을 제공하는 부분이 바로 GUI이고 그중에서도 하드웨어와의 통신을 담당하는 부분이 드라이버 즉, 하드웨어 인터페이스이다.

```
struct screendevice {
    int  xres;
    int  yres;
    long ncolors;
    void *addr;  /* hardware address space */
    /* ..... */

    /* primitive functions */
    /* open & close */
    /* drawpixel , drawhorzline, drawvertline */
    /* ..... */
};
```

[표 1] 하드웨어 인터페이스의 구성

[표 1]은 하드웨어 인터페이스의 하드웨어에 관련된 정보를 보여주는 것으로서, xres, yres는 하드웨어가 제공하는 화면의 크기, ncolors는 하드웨어에서 표현할 수 있는 색상, addr은 하드웨어가 사용하고 있는 메모리 스페이스의 포인터를 나타낸다. 또한, 화면에 출력하기 위해서 하드웨어 초기화와 종료, 점그리기와 같은 primitive function들이 정의되어 있다.

3. Bitmap 처리

GUI를 이용할 때 그림, 도표등을 표현하여야 할 필요가 있는 경우가 생긴다. 이런 경우를 위하여 그림, 도표등을 표현할 수 있는 그림 파일의 형식인 Bitmap을 표현할 수 있도록 GUI를 설계하였다. Bitmap을 실시간 운영체제에서 이용하기 위해서는 Bitmap을 저장할 수 있는 저장공간이 필요한데, 파일시스템이 갖추어진 하드웨어의 경우에는 Bitmap을 파일시스템에서 가져다가 사용하면 된다. 그러나 그렇지 않는 경우(하드웨어에 파일시스템이 갖추어져 있지 않은 경우)에는 실시간 운영체제와 함께 ROM에 포함되거나, 하드웨어 구동중에 외부 포트로부터 이미지를 다운로드하여 일정한 메모리영역에 이미지를 보관하였다가 보여주어야 한다. 본 논문에서는 파일시스템이 갖추어져 있지 않은 환경에 포커스를 맞추어 Bitmap의 출력을 설계 하였다.

```
typedef struct {
  int width;
  int height;
  const unsigned char *raw;
} BitmapData;
```

[표 2] Bitmap 의 처리를 위한 데이터구조

[표 2]는 bitmap을 메모리에 저장하여 사용할 수 있도록 해주는 구조로서, width, height는 이미지의 크기를 저장하는 변수이고, raw는 저장된 비트맵을 포인트 하는 변수이다.

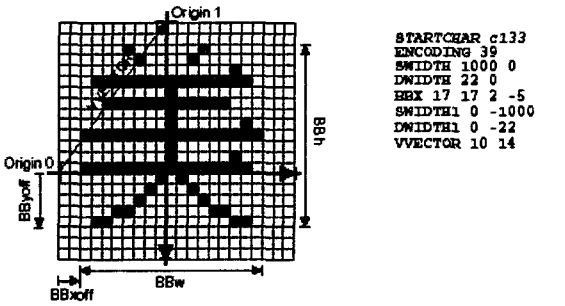
```
void imgRender (const BitmapData *img, int left, int top)
{
  unsigned x,y;
  int index=0;

  /* left와 top이 화면내에 있는지를 검사하는루틴 필요 */

  for (y=0;y<img->height;y++) {
    for (x=0;x>img->width;x++) {
      drawPixel(x+left, y+top, img->raw[index]);
      index++;
    }
  }
}
```

[표 3] Bitmap 처리 알고리즘 (pseudo code)

[표 3]은 BitmapData 구조에 저장되어 있는 Bitmap을 화면에 출력하는 알고리즘을 보여준다. Left와 top의 위치부터 시작해서 bitmap을 pixel단위로 left+img->width, top+img->height의 위치까지 화면에 출력하게 된다.

4. Font

그래픽 환경에서 글자를 표현하기 위해서 사용되는 Font는 bitmap형식으로 되어 있는 글자의 이미지를 ASCII code를 이용하여 찾아내어 화면에 출력하는 과정을 이용한다. Windows™ 또는 LINUX™에서는 다양한 폰트를 파일의 형태로 저장하여 제공하고 있다. 그러나, Bitmap에서와 마찬가지로 실시간 운영체제에서 파일을 사용할 수 없는 경우가 있을 수 있으므로, 운영체제에 포함시켜 사용하도록 설계 하였다.

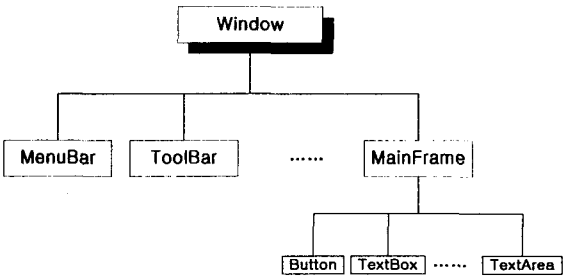


[그림 2] Font 의 구조

[그림 2]는 Bitmap형태의 Font 파일의 구조를 보여주는 예이다. 이러한 Bitmap형태의 Font의 대표적인 것으로는 Bitmap Distribution Font(BDF)가 있다. 이러한 BDF형식의 Font 파일을 툴을 이용하여 array형태로 변환할 수 있다. 이렇게 변환된 파일을 운영체제 컴파일 때에 포함시켜 컴파일 하면 array형태로 저장된 폰트들을 이용할 수 있다. 그러나, BDF형식의 한글 폰트들은 용량이 큰 완성형 한글을 사용하기 때문에 실시간 운영체제에 사용하기 어렵게 된다. 이를 해결하기 위하여 array형태로 이루어진 조합형 한글을 사용하여 text를 처리하도록 설계 하였다.

5. Container 객체의 처리

GUI를 구성하는 객체 중에는 다른 객체를 포함할 수 있는 객체를 필요로 하게 되는데 이러한 객체를 Container객체라고 한다. Container 객체들은 객체내에 포함하고 있는 다른 객체들을 통합하여 묶어주는 역할을 하므로, Container 객체가 화면에서 이동하게 되면 따라서 객체내에 포함된 다른 객체들도 이동할 수 있도록 처리된다.



[그림 3] MS Window 의 Container 객체 구조

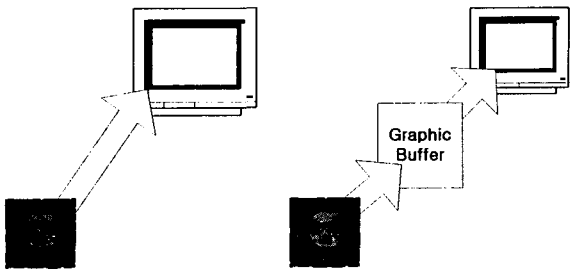
[그림 3]에서와 같이 윈도우의 경우 윈도우 위에 메뉴바, 툴바, 메인 프레임 등을 가지고 있고, 메인프레임 안에는 버튼, 텍스트박스등의 객체들이 포함되어 윈도우 프로그램을 만들게 되는데 Window객체 혹은 MainFrame객체들은 Container객체의 좋은 예라 할 수 있겠다.

```
typedef enum {CONTAINER, LINE, BOX, ……, TEXT}
DrawableType;

typedef struct {
    int left, top;
    int right, bottom;
}Area;

typedef struct  {
    Area area;
    Int color;
    DrawableType type;
}Drawable;

typedef struct {
    Drawable drawable;
    Drawable *containerListPtr;

    /* …… */
}Container;
```

[표 4] Container 객체를 위한 데이터구조

[표 4]는 Container객체를 구현하기 위한 데이터 구조로 Container 데이터 구조 중에서 drawable변수는 현재의 컨테이너에 대한 정보를 담고 있는 변수이고 containerListPtr은 컨테이너에 포함되어 있는 객체들을 포인트 하는 Linked List를 포인트 하는 변수이다.

6. 변경된 화면의 처리



[그림 3] 변경된 화면 처리의 두가지 방법

[그림 3]에서 보는 것처럼 변경된 화면을 처리하는 방법은 그래픽 하드웨어에 직접 그리는 방법과 그래픽 버퍼를 두어서 그래픽 하드웨어에 화면을 출력하기 전에 변경된 부분을 모두 그래픽 버퍼에 그린 후 화면을 변경하는 두가지 방법이 있다. 그래픽 하드웨어에 직접 그리는 방법은 별도의 처리과정 없이 직접 하드웨어에 그리기 때문에 빠르게 그릴 수 있는 반면에 변경되는 객체가 많아지면 많은 작업을 한꺼번에 해 주어야 하기 때문에 화면 반짝거림 현상이 나타나게 되고, 반대로 그래픽 버퍼를 두어서 처리하는 방법은 화면 반짝거림 현상은 최소화 할 수 있지만 그리는 작업을 하드웨어에 직접 해 주지 않기 때문에 시간적, 공간적인 오버헤드가 나타난다. 최근에 개발되어 시판되는 그래픽 하드웨어의 속도가 크게 향상되었고, 현재 개발되어 있는 GUI Library들이 후자의 방법을 택하는 추세에 있기 때문에 그래픽 버퍼를 사용하여 변경된 화면을 처리하는 방식으로 설계하였다.

7. 결론 및 향후 연구 과제

본 논문에서는 실시간 운영체제에 알맞도록 하드웨어 인터페이스, 비트맵, 폰트처리 등에 대한 설계를 하였다. 실제 태스크에서의 사용과정에서 발생할 수 있는 그래픽 하드웨어 사용시 발생할 수 있는 Priority Inversion문제와 같은 Resource Management부분을 실시간 운영체제의 ITC (Inter-Task Communication)과 연계하여 설계에 수정을 가하고, 최종 설계를 바탕으로한 구현이 이루어져야 하겠다. 또한, Graphic User Interface의 최종 목표인 윈도우의 처리에 대한 연구도 진행되어야 하겠다.

8. 참고문헌

[1] G-Windows and G View, GESPAC Inc. Website : http://www.gespac.com/

[2] Photon, QNX Software Systems Ltd. Website : http://www.qnx.com/

[3] Microwindows, http://www.microwindows.org/

[4] Niall Murphy, " Graphic Libraries for Embedded Systems " , Embedded Systems Programming, Aug. 1997.

[5] Niall Murphy, " GUI Development : Embedding Graphics, Part I " , Embedded Systems Programming, Jul. 1999.

[6] Niall Murphy, " GUI Development : Embedding Graphics, Part II " , Embedded Systems Programming, Aug. 1999.

[7] Adobe Developer Support " Glyph Bitmap Distribution Format (BDF) Specification" http://partner.adobe.com/ Mar. 1993