파일시스템이 없는 실시간 운영체제에서 Graphic User Interface 설계 및 구현*

강희성[0], 손필창, 이원용, 이철훈
충남대학교 컴퓨터공학과
{hskang[0], pcson, nissikr, clee}@cnu.ac.kr

The Design and Implementation of Graphic User Interface on Real-Time Operating System without File System

Hui-Sung Kang[0], Pil-Chang Son, Won-Yong Lee, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National University

요 약

실시간 운영체제 GUI 에서 가장 중점적인 것은 임베디드 시스템상에서 중요한 자원 중 하나인 메모리의 효율적인 관리이다. 특히 파일시스템이 없는 임베디드 시스템의 경우 GUI 를 사용하게 되면 폰트 이미지(Font Image)와 그림파일 등이 메모리상에 존재하여야 하기 때문에 메모리를 많이 차지하게 된다. 본 논문에서는 실시간 운영체제 UbiFOS™에서 파일시스템 모듈을 제거하고 GUI 를 설계 및 구현하였고 특히 폰트 이미지와 그림파일을 처리하는 방법과 폰트 이미지를 위한 메모리를 절약할 수 있는 메커니즘을 제안하였다.

1. 서 론

과거에는 흑백의 단순한 텍스트만을 보여주던 임베디드 시스템(Embedded System)들에 점차 화려하고 멋진 그래픽을 보여주는 디스플레이를 탑재하고 있다. 요즘 생산되는 핸드폰들은 모두 컬러 LCD 를 탑재하고 있고, 320X240 이상의 해상도를 가진 컬러 LCD 를 탑재한 PDA, PMP 의 사용이 늘어나고 있다.

임베디드 시스템에서는 대부분 제한된 크기의 디스플레이(Display)를 사용하고 있으며 이러한 장치에 텍스트, 그래픽 및 영상을 표시하기 위해서는 특별한 그래픽 사용자 인터페이스(Graphic User Interface:GUI)가 필요하다. 임베디드 시스템에서 사용되는 GUI 기술은 미려함과 편리함 이외에도 경량화, 최적화, 실시간성 지원, 플랫폼 다양성 지원 등이 필수적으로 요구되고, 데스크톱 분야에 비하여 임베디드 운영체제나 하드웨어 플랫폼에 따라 매우 다양하게 발전되어가고 있다.

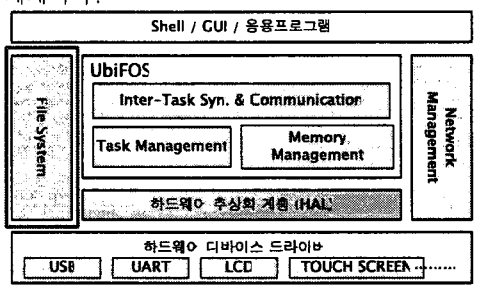
임베디드 운영체제로 사용되는 실시간 운영체제(Real-Time Operating System)에서는 마이크로 커널의 경량화뿐만 아니라, 실시간 운영체제와 함께 지원되는 기능들(GUI, 네트워크 등)에 대한 경량화가 이루어져야 한다. 특히 GUI 는 기능들의 특성상 메모리를 많이 차지하게 되고, 수행시간이 오래 걸리기 때문에 제한된 마이크로프로세서의 성능과 적은 용량의 메모리를 지닌 임베디드 시스템에서는 작고 가벼우면서 수행속도가 빠른 GUI 가 전체적인 시스템 성능향상에 큰 요소가 된다.

본 논문에서는 파일시스템이 없는 실시간 운영체제를 위한 경량화된 GUI 를 설계하고 구현한 내용을 기술한다. 2 장에서는 관련연구에 대해 살펴보고, 3 장에서는 파일시스템이 없는 실시간 운영체제를 위한 GUI 의 설계 및 구현을 다룬다. 4 장에서는 구현된 GUI 의 동작 및 실험결과를, 마지막으로 5 장에서는 결론 및 향후 연구과제에 대하여 기술한다.

2. 관련 연구

2.1 실시간 운영체제 UbiFOS™

본 논문에서 GUI 를 구현하기 위한 기반으로 사용한 UbiFOS™ 는 우선순위 기반 선점형 멀티 쓰레드 실시간 운영체제이다.



[그림 2-1] UbiFOS™의 기능 블록 다이어그램

[그림 2-1]은 UbiFOS™의 기능을 블록 다이어그램으로 나타낸 것이다. 운영체제에서 제공되는 기능은 태스크 관리, 태스크간 동기화 / 통신, 동적 메모리 관

* 본 연구는 정보통신부의 선도기반기술개발사업의 지원으로 수행되었습니다.

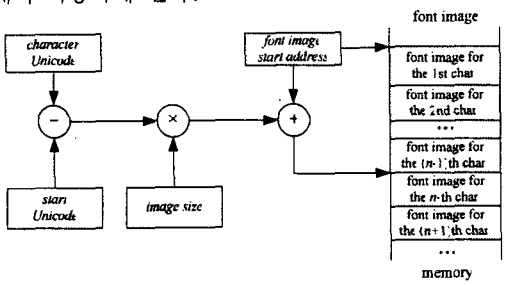
리 등이 있고, 본 논문에서는 파일시스템 모듈을 제거한 상태에서 GUI 를 구현하였다.

2.2 일반적인 폰트 출력 알고리즘

그래픽 환경에서 글자를 표현하기 위해서 사용되는 폰트(font)는 비트맵(Bitmap) 형식으로 되어있는 글자의 이미지를 ASCII code 를 이용하여 그에 할당된 비트맵 폰트 이미지를 찾아 내어 화면에 출력된다.

일반적인 실시간 운영체제용 GUI System 에서 문자를 출력할 때에는 [그림 2-2]와 같은 방식을 사용한다.

파일시스템이 없는 임베디드 시스템에서도 폰트를 사용할 수 있도록 설계하기 위해서는 ROM 혹은 Flash Memory 상에 존재하는 폰트 이미지들을 RAM 으로 복사해서 사용하게 된다.



[그림 2-2] 일반적인 폰트 출력 알고리즘

[그림 2-2]와 같이 연속된 특정 영역에 저장되어 있는 폰트 이미지들 가운데서 알맞은 폰트 이미지를 찾는 방식을 살펴보면, 우선 입력 받은 문자의 code 를 해당 문자 코드의 첫 번째 코드로 빼면 현재의 코드가 문자 set 중에서 몇 번째 위치인가를 나타내는 값이 리턴 된다. 이 값을 이용해 한 글자를 나타내는 폰트 이미지 사이즈를 곱해준 후 문자 set 이 저장된 메모리의 시작주소에 더해주면 원하는 문자의 폰트 이미지가 저장되어 있는 부분의 주소를 가리키게 된다. 이렇게 찾아진 폰트 이미지를 화면상에 출력하게 된다. 이 방식을 수식으로 나타내면 다음과 같다.

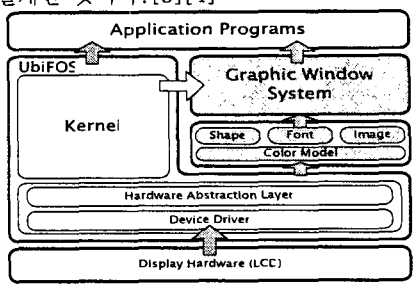
$$address = (character\ Unicode - start\ Unicode) \times 32 + font\ image\ start\ address$$

이 방식은 구현이 간단하나 사용되는 폰트 이미지를 찾기 위해서 사용여부에 관계 없이 모든 문자들의 이미지를 가지고 있어야 하기 때문에 한글, 한자와 같은 많은 문자들로 구성되어 있는 언어를 출력할 경우 폰트의 사이즈가 커져서 적은 메모리를 가진 임베디드 시스템에는 부적합하다.

3. 설계 및 구현

대부분의 실시간 운영체제를 사용하는 임베디드 시스템에서는 적은 메모리를 사용하여 원하는 기능을 수행하고 있기 때문에 메모리의 효율적인 사용이 임베디드 시스템의 성능을 좌우한다. 따라서 GUI 를 설계 구현하는데 있어서 메모리를 절약할 수 있는 메커니즘을 제공하여 메모리의 효율적인 사용을 도모해야 한다.

[그림 3-1]은 본 논문에서 구현한 GUI 의 구조로서, LCD 를 위한 디바이스 드라이버, 기본 그리기, 폰트, 이미지를 위한 그래픽 프리미티브와 이를 이용한 그래픽 윈도우 시스템을 Layered Architecture Design 방식으로 설계한 것이다.[3][4]



[그림 3-1] 구현한 실시간 운영체제용 GUI 구조

3.1 GUI 를 위한 기본 구조체

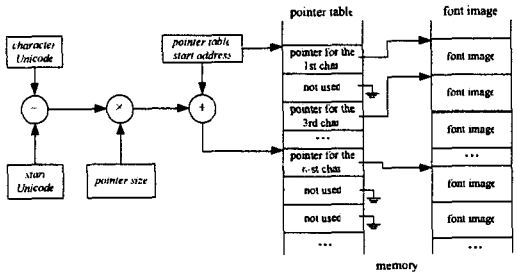
[그림 3-2]은 프로그램에서 LCD 상에 데이터를 출력할 때 사용하는 Graphic Controller 의 구조체이다.

```
struct GraphicController {
    COLORVAL foreground;  // 전경색상
    COLORVAL background;  // 배경색상
    XCOORD lastx;  // 마지막으로 출력한 LCD 상의 x 좌표
    YCOORD lasty;  // 마지막으로 출력한 LCD 상의 y 좌표
    struct fontImage *font;  // 폰트 이미지의 포인터
}
```

[그림 3-2] Graphic Controller 구조체

LCD 화면상에 점을 찍는 기능은 입력된 좌표상에 입력된 컬러를 찍어줌으로써 간단히 구현될 수 있고, 이는 LCD 드라이버의 역할이기도 하다. 이 함수를 이용하여 점, 선, 사각형, 다각형, 원, 타원 등과 같은 도형을 표현한다.

3.2 메모리를 절약하는 폰트 출력 알고리즘



[그림 3-3] 메모리를 절약하는 폰트 출력 알고리즘

[그림 3-3]은 메모리를 절약하는 폰트 이미지 검색 알고리즘을 그림으로 나타낸 것이다. 기존의 방식과 마찬가지로 입력된 문자의 코드를 해당 문자 set 의 맨 첫 글자의 코드로 뺀 값을 pointer 의 사이즈로 곱한다. 이렇게 얻어진 값을 pointer table 의 start address 를 더한 주소는 원하는 문자 이미지가 위치한 주소 값이 된다. 이와 같이 pointer table 을 indirect map 으로 사용하여 폰트 이미지를 구한다.

이때, 사용되지 않은 문자의 pointer table 의 slot 에는 0x00 의 주소를 가지고 있어 사용되지 않는 문자로 표시된다. 폰트의 이미지가 저장된 주소를 찾아내는 방식을 수식으로 나타내면 다음과 같다.

$$address = (character\ Unicode - start\ Unicode) \times 4 + pointer\ table\ start\ address$$

이 방식을 이용하면 메모리상에 pointer table 즉 indirect map 을 유지하기 위한 메모리가 1 글자당 4byte(32bit 머신의 경우)의 overhead 로 사용되지만 한글 혹은 한자에서 흔히 사용되는 16x16 의 폰트인 경우 1 글자당 32byte 의 메모리가 절약되기 때문에 메모리 효율을 기대할 수 있다.[2]

3.3 폰트 출력의 구현

파일시스템이 없는 실시간 운영체제에서 폰트를 사용하기 위해서는 폰트를 관리하는 구조체가 필요하다. [그림 3-4]는 폰트를 관리하는 구조체이다.

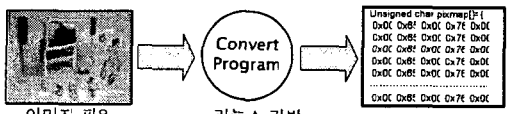
```
#define MAXCHAR 256
Struct FontData {
   unsigned char pont_name[16]; // 폰트의 이름
   unsigned char height;  // 폰트의 높이 (영어:8, 한글:16)
   struct { // 각 글자마다 가져야 할 값
      unsigned char width;  // 글자의 너비
      unsigned char offset_msb;  // 글자의 오프셋
      unsigned char offset_lsb;
   } per_char[MAXCHAR];
   Unsigned char data[1];  // 배열에 위치된 문자들
}
```

[그림 3-4] 폰트 관리 구조체

폰트 출력은 위의 알고리즘을 통해 얻은 폰트 이미지를 MK_DrawText() 함수를 사용하여 화면에 출력한다. 한글의 경우에는 완성형 폰트보다는 배열형태로 이루어진 조합형 한글을 사용하여 처리한다.

3.4 그림파일의 출력

[그림 3-5]와 같이 그림파일을 처리할 수 있는 프로그램을 이용해서 배열의 형태로 변환한다.



[그림 3-5] 그림파일 변환 과정

```
MK_DrawImage (XCOORD x, YCOORD y, struct Pixmap *pixmap,
              COLORVAL fgc, COLORVAL bgc)
```

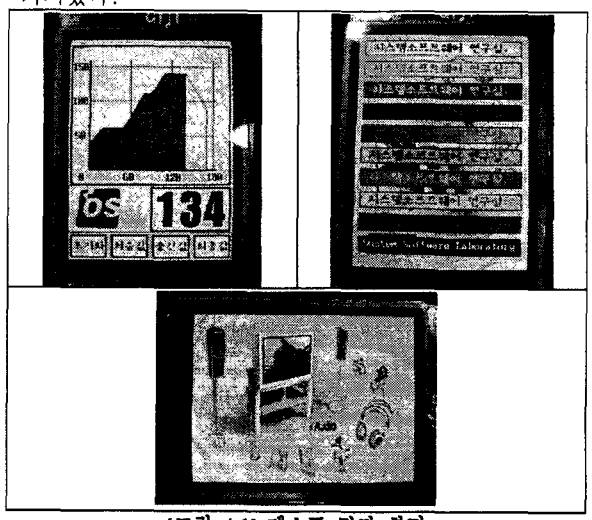
위의 MK_DrawImage() 함수를 이용하여 변환한 후 *pixmap 안의 데이터를 첫 배열부터 끝 배열까지 화면상에 출력하게 된다.

4. 테스트 환경 및 결과

테스트는 MBA2440 (CPU : ARM920T ㈜아이지시스템) 보드와 실시간 운영체제 UbiFOS™를 기반으로 하였다.

[그림 4-1]은 본 논문에서 구현된 GUI 의 실행결과이다. 기본적인 그래픽 호출과 한글폰트 출력, 그림파일 출력이 정상적으로 수행됨을 확인할 수 있다.

본 논문에서 구현된 GUI 는 메모리 크기를 많이 차지하는 그래픽 폰트의 크기를 줄임으로써 효율적인 메모리 사용이 가능하게 되었다. 제안된 알고리즘을 사용함으로써 일반적인 폰트 출력 알고리즘에 비해 한글의 경우 대략 66%, 한자의 경우 74%의 메모리 이익을 가져왔다.



[그림 4-1] 테스트 결과 화면

5. 결론 및 향후 과제

본 논문에서 구현된 실시간 운영체제용 GUI 는 GUI 의 근간이 되는 Color model 과 기본도형, 한글, 영문 문자출력기능, 그림파일 출력기능 등을 제공하고 있으며, 터치스크린과 키 입력 기능, 윈도우 매니지먼트 기능 또한 제공한다. 또한 설계 단계에서 하드웨어와 관련된 부분과 그렇지 않은 부분을 나누는 Layered Architecture Design 방식을 채택하여 다른 시스템으로의 이식성을 높였다. 향후 과제로는 미려한 외관을 위한 다양한 위젯의 구현과 파일 시스템이 존재하는 임베디드 시스템에서 다양한 트루타입 폰트(TrueType Font)를 로딩하는 기법을 설계하는 것이다.

참고문헌

[1] 윤기현, 김용희, 박희상, 이철훈, " Design of Graphic User Interface for the Real Time Operating System" , 한국정보과학회, Vol.29,No2(I),pp400-402,2002

[2] 윤기현, 성영락, 이철훈, " An Efficient Algorithm for Storing/Restoring Font Image in the Unicode for Small-Memory Embedded System" , ESA 03' UCMSS

[3] 강희성, 이정원, 조문행, 이철훈, " Design and Implementation of Graphic Window System for Real-Time Operation System" , 한국정보과학회, Vol, 32, No. 1(A), pp.856-858, 2005, 7

[4] http://microwindows.org