(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.
G06F 12/00 (2006.01)

(45) 공고일자 2007년08월13일
(11) 등록번호 10-0748498
(24) 등록일자 2007년08월06일

(21) 출원번호 10-2001-0028447
(22) 출원일자 2001년05월23일
심사청구일자 2006년04월04일

(65) 공개번호 10-2002-0089642
(43) 공개일자 2002년11월30일

(73) 특허권자 엘지전자 주식회사
서울특별시 영등포구 여의도동 20번지

(72) 발명자 송동섭
서울특별시동작구신대방동590-13

(74) 대리인 박장원

(56) 선행기술조사문헌
10-1994-0006034
10-1996-0025212

심사관 : 권오성

전체 청구항 수 : 총 2 항

(54) 임베디드 시스템의 파일 읽기 방법

(57) 요약

본 발명은 파일 시스템이 없는 임베디드 시스템상에서 구동되는 프로그램의 파일 읽기방법에 관한 것으로, 일반적으로 파일을 메모리에 로드(load)하고, 파일 입력 기능들을 메모리에서 읽도록 프로그램 소스 코드를 전반적으로 수정해 줘야 한다. 하지만, 파일 포인터(File pointer, fp)를 파라메터로 받아오거나 리턴(return)해야 하는 API(Application program interface)의 경우, 파일 포인터(fp)의 고려없이 수정하게 되면 프로그램의 정상적인 동작이 수행되지 않는 문제점이 있었다. 따라서, 본 발명은 파일의 시작 포인터를 가리키는 필드(_base)와, 그 파일의 전체 사이즈(size)를 나타내는 필드(_cnt)와, 그 파일의 다음 시작 포인터를 가리키는 필드(_ptr)를 메모리상의 주소로 설정하여 파일을 오픈(open)하고, 포인터(_ptr)가 가리키는 곳(data)에 소정 크기(len=size1*size2)의 블록을 읽어와서 저장하고, 그 크기(len) 만큼씩 포인터(_ptr)를 증가시키며 파일을 리드(fread)하도록 소스코드를 수정하고, 파일을 클로즈(Close) 시키도록 이루어짐으로써, 파일 읽기 코드가 많은 소프트웨어를 파일 시스템이 없는 임베디드 시스템에 포팅할 경우, 전반적인 소프트웨어의 동작에 지장을 주지 않으면서 파일을 읽는것과 같은 효과가 있다.

대표도

도 4

특허청구의 범위

청구항 1.

파일 시스템이 없는 임베디드 시스템에서 메모리에 저장된 임의의 파일을 읽음에 있어서, 그 파일의 시작 포인터를 가리키는 필드(_base)와, 그 파일의 전체 사이즈(size)를 나타내는 필드(_cnt)와, 그 파일의 다음 시작 포인터를 가리키는 필드(_ptr)를 메모리상의 주소로 설정하여 파일을 오픈(open)하고, 포인터(_ptr)가 가리키는 곳(data)에 소정 크기(len=size1*size2)의 블록을 읽어와서 저장하고, 그 크기(len) 만큼씩 포인터(_ptr)를 증가시키며 파일을 리드(fread)하도록 소스코드를 수정하고, 파일을 클로즈(Close) 시키도록 이루어진 것을 특징으로 하는 임베디드 시스템의 파일 읽기 방법.

청구항 2.

제1항에 있어서, 상기 파일 리드(fread)시의 파일 크기(_cnt)는, 블록 크기(len)가 파일 사이즈(_cnt)보다 작을 경우, 그 블록 크기(len)를 임의의 값(count)으로 설정하여 그 설정값(count) 만큼씩 파일 사이즈(_cnt)를 감소시키고, 파일 사이즈(_cnt)가 블록 크기(len)보다 작을 경우에는, 그 파일 사이즈(_cnt)를 임의의 값(count)으로 설정하여 그 설정값(count) 만큼씩 파일 사이즈(_cnt)를 감소시키는 것을 특징으로 하는 임베디드 시스템의 파일 읽기 방법.

명세서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 임베디드 시스템의 파일 읽기 방법에 관한 것으로, 특히 파일 시스템이 없는 임베디드 시스템(Embedded system)에서 파일 포인터를 조작하여 메모리에 로드된 파일을 읽을 수 있도록 하는 임베디드 시스템의 파일 읽기 방법에 관한 것이다.

파일 시스템(File system)이란, 전체적인 기억장치 관리 시스템의 한 부분으로서, 주로 보조 기억 장치 상의 파일을 관리하는 시스템을 의미하며, 여기에는 MS-Dos 파일 시스템, 유닉스 파일 시스템, 네트워크 파일 시스템 등 컴퓨터 시스템에 따라 여러 가지가 있다.

보다 구체적으로 파일시스템(file system)이란, 운영체제가 파티션이나 디스크에 파일들이 연속되게 하기 위해 사용하는 방법 또는 자료 구조를 의미하는 것으로, 파일을 저장하는 데 사용되는 파티션이나 디스크를 가리킬 때나, 파일시스템의 형식을 가리킬 때 사용되기도 한다.

디스크나 파티션이 포함하고 있는 파일시스템의 차이는 중요한데, 약간의 프로그램들(합리적으로 충분히 파일시스템을 만드는 프로그램을 포함해서)은 디스크나 파티션의 원시 섹터를 직접 조정한다.

대부분의 프로그램들은 파일시스템 위에서 작동하며, 파일시스템이 없는(혹은 다른 형식의 파일시스템이 있는) 파티션에서는 작동하지 않는다.

파티션이나 디스크가 파일시스템으로서 사용될 수 있게 하기 위해서는 초기화되어야 하며, 파일정보 기록을 위한 자료구조를 디스크에 만들어야 한다.

한편, 어떤 소프트웨어를 임베디드 시스템에 포팅(porting) 하려는 경우, 소프트웨어 내용에는 파일을 리드(read)하는 부분이 포함될 경우가 있다.

그런데, 임베디드 시스템에는 파일 시스템(일종의 하드디스크와 같은 보조 기억장치)이 있는 경우도 있고 그렇지 않은 경우도 있다.

따라서, 만약 도1에 도시된 바와 같이, 파일 시스템이 있는 경우에는 소프트웨어 내용에 파일을 리드하는 부분이 있더라도 별다른 처리를 필요로 하지 않지만, 파일 시스템이 없는 경우에는 파일을 메모리에 로드(load)하고, 파일 입력 기능들을 메모리에서 읽도록 프로그램 소스코드를 전반적으로 수정해 줘야 한다.

그러나, 프로그램 내에 파일 포인터(File pointer, fp)를 파라메터로 받아오거나 리턴(return)해야 하는 API가 있는 경우, 파일 포인터(fp)의 고려없이 수정하게 되면 프로그램의 정상적인 동작을 기대할 수 없게 된다.

왜냐하면, 해당 API가 프로그램 내에서 언제, 어떻게 호출될지 모르기 때문이다. 이런 연유로 파일을 리드하는 소스코드를 어떻게 바꾸는가 하는 문제는 매우 중요하다.

발명이 이루고자 하는 기술적 과제

따라서, 본 발명은 상기와 같은 종래의 문제점을 해결하기 위하여 창출한 것으로, 파일 시스템이 없는 임베디드 시스템(Embedded system)에서 파일 포인터를 사용해 메모리에 로드된 파일을 읽을 수 있도록 하는 임베디드 시스템의 파일 읽기 방법을 제공함에 그 목적이 있다.

발명의 구성

이와 같은 목적을 달성하기 위한 본 발명은, 파일읽는 내용이 포함된 API를 호출하는 함수호출자(caller)에서는, 이 API를 호출함으로서 실제로 파일시스템상의 파일을 읽는것과 똑같은 효과를 얻을 수 있는 것을 특징으로 한다.

또한, 본 발명은 이 API에 파라메터 형식으로 넘어가는 파일 포인터 때문에 같은 파일 포인터를 쓰는 다른 API에게 오동작을 일으키지 않도록 하는 것을 특징으로 한다

이하, 본 발명에 따른 일실시예를 첨부한 도면을 참조하여 상세히 설명하면 다음과 같다.

본 발명에서는 파일 시스템이 구비되지 않은 임베디드 시스템에서 파일이 16진수 형태로 메모리상에 로드되는 경우, 파일 포인터를 조작하여 마치 파일 시스템에서 파일을 읽는 것과 같은 효과를 나타낸다.

도2는 파일 입/출력 기능에 공통적으로 쓰이는 파일 구조를 보인 예시도로서, 로드(load)하려는 파일이 'Status_bar.gif'라고 할 경우를 예로 들어 살펴보기로 한다.

도3과 같이 메모리에는 부호없는 8비트 문자형(unsigned char type)으로 16진수(hexadecimal) 형태로 저장하며, 전체 길이는 32비트 정수형(long type)으로 저장한다.

상기와 같이 메모리에 로드할 파일의 형태는 동종 업계에 종사하는 엔지니어의 경우, 간단한 변환 프로그램을 작성하여 쉽게 변환이 가능하므로 포맷 변환에 관한 구체적인 설명은 생략하기로 한다.

상기와 같이 파일(Status_bar.gif)이 파일 시스템이 아닌 메모리 상에 로드되면, 각 파일 입/출력 기능들은 다음 표1과 같이 파일에 관련된 소스 코드를 약간 수정하는 것에 의해, 메모리를 파일 시스템으로 인식하게 하여 프로그램 소스의 전반적인 수정없이도 동작이 가능하게 한다.

[표 1]

용도	원래 함수	발명 코드

File 열기	fp=fopen(status_bar.gif,"r");	fp->_ptr=status_bar;
		fp->_cnt=status_bar_size;
		fp->_base=status_bar;

File 닫기	fclose(fp);	처리 안함

File 읽기	fread(data,size1,size2,fp);	if(status_bar_size!=0){
		len=size1*size2;
		memcpy(data,fp->_ptr,len);
		fp->_ptr+=len;
		if(len<fp->_cnt)
		count=len;
		else
		count=fp->_cnt;
		fp->_cnt-=count;
		}

File 크기		status_bar_size;

여기서, '_ptr'은 파일에서 다음 시작 포인터를 나타내고, '_cnt'는 전체 파일 크기를 나타내고, '_base'는 그 파일의 시작 포인터를 나타내는 것으로 '_ptr'과는 달리 시작 포인터가 변하지 않는다는 것이다.

가령, 다음과 같은 함수가 있다고 가정한다.

File_read(FILE*fp)

{

fread(data,size1,size2,fp);

}

상기 함수에서 파일을 읽는 'fread' 대신에 메모리에 로드된 파일을 읽는 일반적인 코드를 적용한다면, 파라메터 'fp'에 아무런 변화가 없으므로 소프트웨어 기능 전반에 영향을 미치게 된다.

그러나, 본 발명에 의한 방식을 이용하여 파일 입/출력 함수의 소스 코드를 수정하면, 파일 포인터 'fp'에 조작이 가해지기 때문에 소프트웨어는 정상으로 동작하여 원하는 결과를 출력하게 되는 것이다.

즉, 전체적인 임베디드 시스템은 파일 입/출력 API(Application Program Interface)를 이용하여 파일 시스템에서 파일을 읽어오지만, 도4에 도시된 바와 같이 API를 본 발명에 의한 코드로 수정해 줌으로써, 전반적인 시스템의 동작을 저해하지 않으면서 파일 시스템이 있는 것과 같은 동작을 수행하게 되는 것이다.

발명의 효과

이상에서 설명한 바와 같이 본 발명 임베디드 시스템의 파일 읽기 방법은 상기와 같은 파일 읽기 코드가 많은 소프트웨어를 파일 시스템이 없는 임베디드 시스템에 포팅할 경우, 전반적인 소프트웨어의 동작에 지장을 주지 않으면서 파일을 읽는 것과 같은 효과가 있다.

도면의 간단한 설명

도 1은 파일 시스템이 구비되어 있는 임베디드 시스템의 개념적 구성을 보인 예시도.

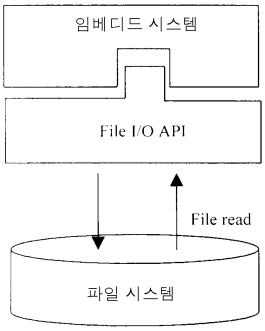
도 2는 파일 입/출력 기능에 공통적으로 쓰이는 파일 구조를 보인 예시도.

도 3은 본 발명을 적용하기 위해 변환된 파일의 포맷을 보인 예시도.

도 4는 파일 시스템이 구비되어 있지 않은 임베디드 시스템의 개념적 구성을 보인 예시도.

도면

도면1



도면2

```
struct _iobuf {
        char *_ptr;
        int   _cnt;
        char *_base;
        int   _flag;
        int   _file;
        int   _charbuf;
        int   _bufsiz;
        char *_tmpfname;
        };
typedef struct _iobuf FILE;
```

도면3

```
unsigned char status_bar[3398]={0x3c,0x21,0x2d,0x2d,0x20,0x43,0x6f, ... , 0x3e};
long status_bar_size=3398;
```

도면4