LZSS 압축 알고리즘을 적용한 PDA용 Embedded Linux 파일 시스템 설계

장 승 주†

요 약

본 논문은 LZSS 압축 알고리즘을 변형하여 임베디드 리눅스 운영체제 파일 시스템으로 사용할 수 있도록 설계한다. 본 논문에서는 PDA 임베디드 리눅스 파일 시스템에 변형된 LZSS 압축 알고리즘을 적용하여 저장 공간의 효율적인 관리를 할수 있도록 하였다. 임베디드 리눅스 기반의 PDA 시스템에 대한 압축 파일 시스템 설계는 메모리 사용을 극대화시킬 수 있다. 이와같이 임베디드 시스템에 압축 파일 시스템을 제공함으로써 임베디드 시스템의 한계였던 소량의 저장 공간 문제를 해결하도록 해준다. 본 논문에서 제안하는 압축 파일 시스템을 사용할 경우 저장 공간을 늘리지 않고 저장 공간을 늘리는 효과를 낼 수 있도록 해준다.

키워드 : 임베디드 시스템, 압축 파일 시스템, 변형된 LZSS 압축 알고리즘, 저장공간

Design of an Embedded Linux File System with LZSS Algorithm for the PDA System

Jang, Seung Ju†

ABSTRACT

I design an Embedded File System in Linux Operating System by applying modified LZSS compressed algorithm. This suggested Compressed File System which is modified file system of the Linux O.S saves the storage space. The compressed file system supports efficient use of storage space. The suggesting file system solves the small space of embedded system. The suggesting file system of this paper gives effect of the large storage space without extending the storage space.

Key Words : Embedded System, Compressed File System, Modified Lzss Compressed Algorithm, Storage Space

1. 서 론

Embedded라는 단어를 살펴보면 "내장된", "고정된" 등의 의미를 가지고 있다. 여기서 임베디드 시스템(Embedded System)의 정확한 정의를 내린다면 '미리 정해진 특정 기능을 수행하기 위해 컴퓨터의 하드웨어와 소프트웨어가 조합된 전자제어 시스템'이라고 할수 있다[1]. 임베디드 시스템의 종류로는 TV, 냉장고, 세탁기, 휴대폰, PDA 등 소형 전자 제품으로써 헤아릴 수도 없이 많다.

이들 중 임베디드 시스템을 탑제한 PDA(Personal Digital Assistant) 즉, 휴대형 정보 단말기는 현재 많은 분야에서 널리 사용되고 있으며, 크기는 노트북보다 훨씬 작은 소형 컴퓨터이면서 기능은 전자수첩보다 강력한 컴퓨팅 파워를 갖고 있다. 현대적인 추세는 임베디드 시스템에서의 멀티미디어 매체 구현 및 데이터베이스 운용 등 점차적으로 한정된 저장 매체에서 상대적으로 더욱 커진 데이터 저장 및 처리를 필요로 하고 있다. 따라서, 효율적인 저장매체의 관리는 그 중요성이 더욱더 커지고 있다.

임베디드 리눅스에서 주로 사용하는 파일 시스템으로는 Ram Disk, cramfs, jffs, jffs2 파일시스템이 있다. 이중 jffs2 파일시스템은 리눅스 커널 2.4버전에서 동작하고 갑작스러운 전원공급 차단에 대비한 저널링 기능을 가지고 있다. 단, 저널링을 하기위한 약간의 공간 낭비와 압축으로 인해 성능이 약간 느리다는 단점이 있다.

본 논문은 임베디드 리눅스 기반에서 PDA에 압축 파일 시스템을 구축하여 메모리 사용을 극대화하는 것이다. 사용된 압축 알고리즘은 LZSS 압축 알고리즘을 임베디드 시스템에 적합하도록 변형하여 사용한다. 본 논문은 JFFS2 파일시스템에 새로운 압축 알고리즘을 적용하여 PDA 시스템의 메모리를 보다 더 효율적으로 이용하도록 함으로써 메모리 공간을 늘리는 효과를 만들 수 있다. LZSS 압축 알고리즘을 변형하여 임베디드 리눅스 운영체제의 파일 시스템으로 사용할 수 있도록 설계한다. 변형된 압축 알고리즘을 사용하여 임베디드 시스템에 실제 적용을 한다. 본 논문에서는 PDA용

† 정 회 원 : 동의대학교 컴퓨터공학과 부교수
논문접수 : 2005년 2월 24일, 심사완료 : 2006년 3월 14일

임베디드 리눅스 파일 시스템에 압축 알고리즘을 적용하여 저장 공간의 효율적인 관리를 할 수 있도록 하였다. 임베디드 리눅스 기반의 PDA에 대한 압축 파일 시스템 설계는 메모리 사용을 극대화시킬 수 있다.

본 논문의 구성은 2장에서 관련연구를 살펴보고, 3장에서 파일시스템 설계 내용에 대해 살펴보고, 4장 결론 순으로 되어있다.

2. 관련 연구

JFFS2(Journalling Flash File System version 2)[1, 2, 3]는 JFFS의 다음 버전으로 스웨덴의 Axis Communications에서 개발된 JFFS 파일 시스템을 레드헷에서 임베디드 시스템을 위해 만든 새로운 저널링 파일 시스템이다. 현재 JFFS2 파일 시스템에서는 자체적으로 메모리 사용 효율을 위한 압축 알고리즘을 지원하고 있다. 그러나 이 압축 알고리즘은 사용자가 필요할 경우 사용할 수 있도록 하는 기능적인 측면이 강하다. 최근에는 JFFS2 파일 시스템에서 메모리 사용의 극대화 및 안정화를 위한 압축 알고리즘 및 파일 시스템 구축에 대한 연구가 활발히 이루어지고 있다.

JFFS2 파일 시스템에서 사용된 RLC(Run Length Compression) 압축 알고리즘[4, 5, 6]에 비해서 보다 더 압축률이 좋은 사전 기반의 변형된 LZSS 압축 알고리즘으로 대체함으로써 임베디드 시스템의 문제점인 부족한 저장 공간을 효율적으로 사용 할 수 있을 것이다. 기존에 사용 된 RLC 압축 알고리즘은 압축 시 반복성이 많은 데이터 패턴이 존재할 경우에 사용 할 수 있는 압축 알고리즘이었다. 그래서 반복성이 많지 않은 데이터 패턴이 많이 존재 할 경우에 생기는 압축의 비효율성 문제점을 해결하기 위해 사전기반의 LZSS 압축 알고리즘이 제안되었다.

고배율 Lossless 압축 알고리즘으로는 MIPS 기반 3.5:1 의 압축 알고리즘이 개발되어 사용되고 있다[7, 8]. 기존의 압축 파일 시스템의 경우에 압축 파일 시스템에 고 배율 압축 알고리듬과 연동하여 필요시 필요한 부분만을 압축 해제하여 사용하는 on-the-fly을 사용하여 연구, 개발되었다[9, 10]. 스위칭 동작 최소화를 위한 데이터 압축 알고리즘은 문장을 인위적으로 만들어서 아스키 코드를 사용했을 경우가 있다. 또 다른 경우는 변화된 코드, 허프만 코드를 계산한 뒤 허프만 코드 상태에서의 테이블 값과 변환된 테이블 값을 사용해서 각 문자들의 부호화 과정을 거치면서 레지스터에 적재되는 테이블 값들의 비트 변화 횟수를 측정하여 사용하는 경우이다. 저 전력 통신을 위한 에너지 효율적인 기존의 압축 알고리즘에서 한글의 경우에 한글 표기 단위인 2 바이트 단위로 데이터를 압축하며 한글의 표기상의 특성을 활용하는 장점이 있다[11, 12].

3. 압축 파일 시스템 설계

3.1 JFSS2 파일시스템에 적용할 압축 알고리즘

JFFS2 파일시스템에 적용할 압축 알고리즘으로써 LZSS 압축 알고리즘을 사용한다. 이 알고리즘은 1982년에 Storer와 Szymanski가 개발한 알고리즘으로 LZ77에서 조금 더 개선이 이루어진 알고리즘이다. 알고리즘의 메인 아이디어는 이미 한번 이상 사용된 문자열은 기존의 문자열의 상대적 위치와 일치하는 길이만을 저장한다는 것입니다. 이 알고리즘은 LZ77과 유사하면서도 약간 향상된 압축 효율을 가지고 있다.

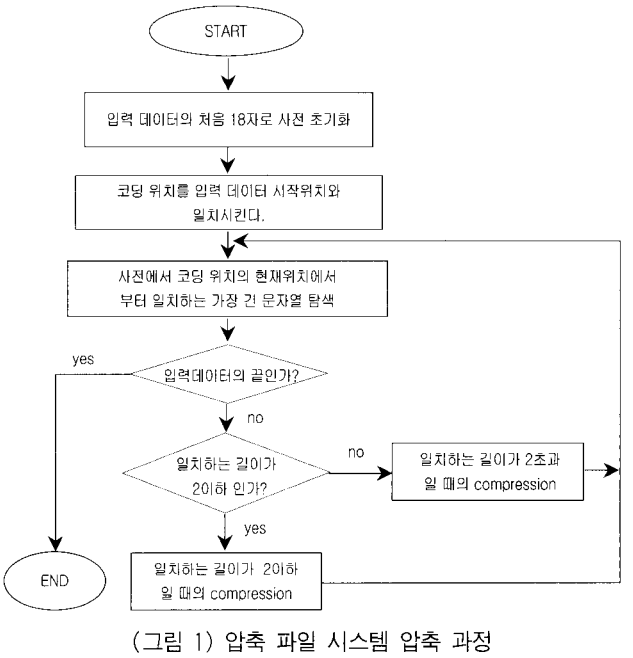
LZSS 압축알고리즘은 처음에 공백문자로 초기화 된 링 버퍼를 유지한다. 그 후 사전용으로 쓸 데이터를 입력 버퍼로부터 읽어 온다. 그 후 방금 읽은 사전의 내용과 비교하여 같은 데이터 패턴 중 가장 긴 데이터 패턴을 찾기 위해 버퍼를 검색한다. 그리고 찾은 데이터 패턴의 길이와 시작 위치를 출력 버퍼에 보낸다.

버퍼의 크기가 4096 바이트이면, 위치를 나타내는 부분은 12비트 크기로 인코딩 된다. 같은 데이터 패턴이 나타나는 길이를 4비트 크기로 나타낸다면[위치, 길이]는 2바이트로 표현된다. 만약 같은 데이터 패턴 중 가장 긴 데이터 패턴이 2바이트 이하이면, 인코딩없이 출력 버퍼에 시작 데이터를 보내고, 바로 다음 데이터부터 위 과정을 다시 시작한다. 매번의 과정 마다 임시의 비트를 사용해 인코딩되었는지, 아닌지를 알린다. 또한 같은 데이터 패턴 중 가장 긴 패턴을 찾는데 걸리는 시간을 줄이기 위해 이진 트리 구조를 사용한다.

3.2 압축 단계

(그림 1)은 임베디드 리눅스 커널의 파일 시스템에서 동작하는 압축 알고리즘을 적용한 단계를 보여주고 있다.

(그림 1)의 압축 과정은 입력되는 데이터로부터 18바이트를 받아 사전을 초기화 한다. 압축을 하기 위해서 읽어들이는 데이터 크기는 18바이트 단위로 읽어들인다. 사전은 각 파일별로 구성을 한다. 각 파일의 처음부터 18바이트까지의 내용이 사전압축을 할 때 사용되고 압축을 해제 할 때도 들어오는 데이터 중 압축을 할 때 사용되는 사전의 내용으로



(그림 1) 압축 파일 시스템 압축 과정

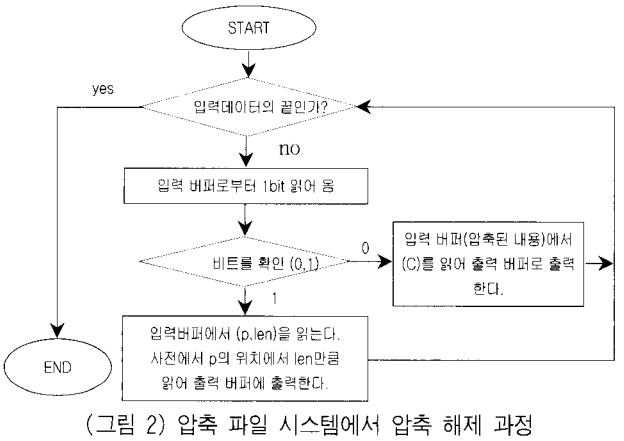
사용되는 부분은 인코딩하지 않았기 때문에 그 값을 이용하여 사전으로 사용하면 압축 해제할 때 문제가 없다. 이렇게 파일마다 사전을 두는 이유는 압축 성능을 보다 더 중요하게 보기 때문이다.

코딩 위치(입력 버퍼 내에서 현재 압축을 진행하고 있는 처리지점의 오프셋)를 입력 버퍼의 시작부분에 위치시킨다. 사전 내에서 입력 버퍼의 앞부분과 일치하는 가장 긴 문자열을 찾는다(p = 찾은 문자열의 사전 내 위치, len = 일치한 문자열의 길이, C = 입력 버퍼의 character). len이 2보다 큰지 비교 한다. 만약 2보다 크다면 (p,len)을 출력 값으로 출력을 한고 len만큼 코딩 위치를 진행시킨다. 2이하이면, 입력버퍼를 출력하고 코딩 위치를 1만큼 진행시킨다. 입력 버퍼에서 읽을 데이터가 없을 때 까지 사전에서 코딩 위치의 현 위치에서부터 일치하는 가장 긴 문자열 탐색 부분부터 반복을 실행한다. 최종 압축 파일에는 압축 1비트와 압축된 데이터가 저장되게 된다.

(그림 1)은 압축 파일 시스템에서 압축이 시작되면서 끝날 때까지 과정을 나타낸다. (그림 1)의 과정에서 구체적인 함수의 구현 내용은 다음과 같다. 먼저 write.c 파일에서 jffs2_compress()함수를 호출한다. 이 함수는 압축하는 타입을 결정하고 그 타입에 맞추어 압축알고리즘이 구현되어 있는 함수를 호출하여 인자 값으로 입력데이터와 사이즈를 넘겨주는 역할을 한다. jffs2_compress() 함수에서 압축할 타입에 대해서 결정을 하고 jffs2_rtime_compress() 함수를 호출하여 실제적인 압축을 실행 하는 것이다. 즉 LZSS 압축 알고리즘이 jff2_rtime_compress() 함수에 구현되어 있다는 것이다. 이 함수는 write.c 파일에 구현되어있는 jffs2_compress() 함수에서 호출되는 함수로써 입력 데이터와 사이즈를 jffs2_compress() 함수에서 넘겨받아 압축을 담당하는 함수이다. jffs2_rtime_compress() 함수를 통하여 압축된 파일은 PDA의 user 부분에 저장이 된다. 그리고 jffs2_rtime_compress() 함수는 compr_rtime.c 파일에 구현되어 있다.

3.3 압축 해제 단계

(그림 2)는 임베디드 리눅스 커널의 압축 파일 시스템에서 동작하는 압축 해제 과정을 설명하고 있다.



(그림 2) 압축 파일 시스템에서 압축 해제 과정

(그림 2)의 압축 해제 과정은 입력버퍼에서 1bit를 읽는다. 만약 더 이상 읽을 데이터가 없다면 종료한다. 입력 버퍼에서 읽어온 1비트를 확인한다.(0,1) (jffs2_rtime_compress()에서 사전의 내용이 된 데이터는 인코딩을 하지 않고 저장되었고 상태 bit는 1로 설정했었다. 상태 bit가 1이면 인코딩되어 있지 않았다는 것을 뜻하므로 data_in으로부터 읽은 데이터를 바로 cpage_out에 출력한다.) data_in은 압축된 데이터의 입력부분이고 cpage_out은 출력으로 사용된다. 상태 비트가 0 이면 입력버퍼 (압축된 내용)에서 데이터를 읽어 출력 버퍼(cpage_out)로 출력한다. 1이면 입력 버퍼에서 (p,len)을 읽는다. 사전에서 p의 위치에서 len만큼 읽어 출력 버퍼(cpage_out)에 출력한다. 이 과정은 입력(data_in)된 데이터가 없을 때까지 반복을 한다.

(그림 2)는 압축 파일 시스템에서 압축된 파일의 디코드되는 과정을 나타낸 그림이다. (그림 2)의 압축 해제 과정에 대한 구현 내용을 실제 함수를 통해서 설명하면 다음과 같다. 먼저 read.c 파일에서 jffs2_decompress() 함수를 불러온다. 이 함수는 압축되어 있는 타입에 맞게 그와 상응하는 압축해제 알고리즘이 구현되어 있는 함수를 호출하여 인자 값으로 압축 해제를 시킬 데이터와 데이터의 사이즈를 인자 값으로 넘겨주는 역할을 담당하고 있다. jffs2_decompress() 함수에서 jffs2_rtime_decompress() 함수를 호출하여 압축해제를 한다. jffs2_rtime_decompress() 함수는 compr_rtime.c 파일에 구현되어 있다. 본 논문에서 jffs2_rtime_decompress() 함수를 사용하여 LZSS의 decode 하는 부분을 적용 시켜다. 이 함수는 jffs2_decompress() 함수에서 호출되는 함수로써 압축 해제할 데이터와 사이즈를 jffs2_decompress() 함수에서 넘겨받아 압축해제를 담당하는 함수로써 LZSS의 압축해제 알고리즘이 구현되어 있다. jffs2_rtime_decompresss() 함수를 통하여 user 부분에 저장되어 있던 파일을 압축해제 시킨다.

4. 구 현

4.1 개발 환경

본 논문의 PDA 포팅 및 파일 시스템 설계를 위한 호스트 PC와 타겟 보드의 셋팅 환경은 아래 <표 1>과 같다.

<표 1>에서 호스트 PC는 인텔 CPU를 사용하고, 운영체제는 리눅스 2.4.12 버전을 사용한다. 컴파일러는 gcc-2.96 버

〈표 1〉 기본 시스템 환경

	호스트 PC	타켓 보드
호스트	Intel Pentium III Celeron 933MHz 128MB ram	IPAQ H3630 (StrongArm-1110) 32 MB ram
운영체재	Linux-2.4.13-1hl (Hancom 2.2)	Linux-2.4.18-rmk3-hh 6-j3
컴파일러	Gcc-2.96	Cross-2.95.3
MTD 패치	mtd-snapshop patch JFFS2 (Journaling Flash File System version 2)	
타켓 보드 ETC	Boot loader : linuette Architecture : BusyBox	

전을 이용한다. 타켓 시스템은 IPAQ H3630을 사용하고 운영체제는 리눅스 2.4.18을 사용한다. 컴파일러는 cross-2.95.2 버전을 사용한다.

4.2 구현 내용

4.2.1 PDA용 압축 파일 시스템 설계

이 부분에서는 PDA용 압축 파일 시스템 설계 내용을 설명한다. 설계된 압축 파일 시스템은 변형된 LZSS 압축 알고리즘을 이용하고 기존의 JFFS2 파일 시스템을 이용하여 설계를 한다. PDA 시스템에 리눅스 커널의 포팅, mtd 패치, 파일 시스템을 확인하는 단계는 다음 (그림 3)과 같다.

설 계 단 계	설 계 내 용
1) 크로스 컴파일러 설치	Arm용 GCC설치
↓	↓
2) 파일시스템영역 패치	mtd 패치
↓	↓
3) 컴파일 환경 수정	Arm용 컴파일
↓	↓
4) 커널 구조체 정보 수정	메모리블록 설정
↓	↓
5) JFFS2 파일시스템 수정	입출력부분 수정
↓	↓
6) menuconfig	커널 기능 설정
↓	↓
7) zImage & modules 생성	PDA용
↓	↓
8) minicom 환경 셋팅	커널과 동기화
↓	↓
9) root.cramfs, usr.jffs2 제작	PDA용 ramdisk
↓	↓
10) 이미지들의 upload	커널, ramdisk
↓	↓
11) 파일의 저장과 출력	입출력 확인

(그림 3) PDA용 리눅스 포팅 및 파일시스템 설계 과정

(그림 3)에서는 PDA용 리눅스 포팅 및 파일 시스템 설계 과정을 도표로 나타낸 것이다. 설계 단계에서는 일반적인 리눅스 포팅에 대해 해야 할 일을 나열한 것이고 설계 내용에서는 실제로 구현한 내용을 순서대로 나타낸 것이다.

4.3 임베디드 리눅스를 PDA 시스템에 포팅하는 과정

여기서는 본 논문에서 제안한 압축 파일 시스템을 실제 시스템에 이식하는 과정을 통해서 구현된 내용을 설명한다. 여기서 설명하는 내용은 구현을 하기 위한 시스템 환경 구축의 일반적인 내용을 중심으로 설명한다.

4.3.1 컴파일 환경 수정

커널 소스의root에 있는 Makefile을 (그림 4)와 같이 아키텍쳐 및 크로스 컴파일러 변수를 수정한다.

```
.....
ARCH := arm
.....
CROSS_COMPILE = /usr/local/arm/2.95.3/bin./arm-linux-
.....
```

(그림 4) 수정된 Makefile

(그림 4)와 같이 CROSS_COMPILE 경로를 수정하는 것은 크로스 컴파일러가 설치되어 있는 디렉토리를 찾아 컴파일할 때 사용하기 위함이다. 또한, include의 파일에서 (그림 4)의 makefile의 아키텍쳐와 동일하도록 (그림 5)와 같이 수정한다.

```
ln -s include/asm-arm include/asm
ln -s include/asm/arch-sa1100 include/asm/arch
ln -s include/asm/proc-armv include/asm/proc
```

(그림 5) include의 아키텍쳐의 수정

(그림 5)에서는 include 부분을 ln 명령어를 사용하여 심볼릭 링크를 시키는 과정이다. 커널 컴파일을 시작할 때 사용되는 부분으로써 심볼릭 링크 이름이 정확해야 한다.

4.3.2 mtd 패치 및 커널 구조체 정보 수정

MTD는 임베디드 디바이스에서 고형체 파일시스템을 구성하는데 사용하는 플래시 메모리, RAM, 그리고 비슷한 다른 칩셋 등 메모리 장치이다. 일반적인 StrongArm용 커널의 포팅 단계와 다르게 본 논문에서는 파일시스템의 컨트롤에 대한 내용을 다루기 때문에 mtd 패치를 하는 부분이 추가되었고 여기서 그 방법을 설명한다.

mtd 패치를 하는 방법은 크게 두 가지로 나뉘는데 커널 소스 트리 안에서 패치하는 방법과 커널 트리 밖에서 패치하는 방법이 있다. 본 논문에서는 커널 트리 안에 패치를 하는 방법을 택하였다. (그림 6)은 mtd 패치 단계를 보여준다.

```
tar vfxz mtd-snopshot.tar.gz
cd mtd/patches
sh patches.sh /커널이 위치한 절대경로
```

(그림 6) mtd 패치 단계

(그림 6)에서는 mtd 패치를 시키는 과정을 나타낸 것이다. 이부분은 새로이 jffs2 파일시스템을 사용하기 때문에 추가해줘야 할 부분으로서 (그림 6)과 같이 추가하면 mtd 패치가 이루어진다. mtd 패치를 하고 drivers/mtd/maps/ 디렉토리 아래에서 타켓 보드인 PDA의 커널 구조체 부분을 결정하는 sa1100-flash.c 파일의 MTD 구조체 부분을 (그림 7)과 같이 수정한다.

```
static struct mtd_partition h3600_partitions[] =
{
{name: "H3600 boot firmware",size: 0x00040000, offset: 0,
        mask_flags: MTD_WRITEABLE},
{name: "H3600 kernel",size: 0x000c0000, offset: 0x0008000},
{name: "H3600 params", size: 0x00040000, offset: 0x00040000},
{name: "H3600 root cramfs", size: 0x140000, offset: 0x140000},
{name: "H3600 usr jffs2",offset: 0x280000, size: 0xd80000}
}
```

(그림 7) 커널 구조체를 수정한 부분

(그림 7)은 부트 로더와 관련하여 mtd 수정 구조체를 보여주고 있다. 부트 로더의 크기와 커널크기 등을 감안해 본다면 (그림 7) 같이 수정을 해주어야 커널과 다른 이미지를 upload 하는데 에러가 발생하지 않을 것이다.

5. 실 험

본 논문의 실험은 두가지 측면으로 이루어졌다. 하나는 압축률을 어느정도 높일 수 있는지에 대한 부분과 다른 하나는 기존의 다른 압축 알고리즘과의 비교 실험이다. 그리고 압축에 따른 속도 저하가 어느 정도인지에 대한 실험을 수행하였다.

5.1 압축 효율

본 논문에서는 LZSS 압축 알고리즘을 구현하고 이를 jffs2 파일시스템에 적용하여 실험을 하였다. 실험을 위한 데이터는 임베디드 리눅스 시스템 환경에 많이 사용하는 데이터 형태를 중심으로 실험하였다. 따라서 실험 데이터는 적은 크기의 데이터를 중심으로 이루어졌다. 압축 파일 시스템에서 압축 효율을 실험하기 위하여 입력된 바이트는 49, 50, 223 바이트 등으로 한다. 그리고 이 입력된 바이트를 압축 파일 시스템을 이용하여 압축한 결과 파일의 크기가 어떻게 되는지를 실험한다. 파일 시스템에서 압축 효율을 실험한 결과는 <표 2>와 같다.

<표 2>의 결과 값을 얻기 위해 본 논문에서는 입력되는 사이즈와 압축이 되어 파일로 저장되어질 부분의 값을 (그림 8)의 소스를 사용하여 출력하였다.

PDA 도스 쉘상에서 확인할수 있도록 디버그 메시지를 (그림 8)의 소스를 사용하여 출력하였다. (그림 8)의 소스코드는 printk를 통하여 입력된 바이트 수와 출력된 바이트 수를 확인할 수 있도록 하여 압축 효율을 좀더 정확한 수치를 통하여 도출해 낼수 있었다.

<표 2>의 실험 결과에서 jfss2를 이용하고 LZSS 압축 알고리즘을 이용한 경우에 압축 효율은 입력 바이트 크기가 작을수록 높은 효율을 보이고 입력 바이트 크기가 클수록 낮은 효율을 보인다. 즉 입력 바이트 크기가 증가할 수록 본 논문

〈표 2〉 압축 알고리즘 적용 결과

입력된 바이트수	출력된 바이트수	압축효율
49	15	0.306
50	24	0.48
41	17	0.414
223	167	0.748
51	55	1.078
117	133	1.136

```
printk("In : %ld bytes\n",textsize);
printk("out : %ld bytes\n",codesize);
```

(그림 8) 실험을 위한 프로그램 코드

〈표 3〉 RLC 압축 알고리즘 적용 결과

입력된 바이트수	출력된 바이트수	압축효율
49	21	0.429
50	35	0.7
41	26	0.63
223	210	0.941

〈표 4〉 용량이 큰 파일에 대한 압축율 비교 실험 결과

파일 크기(KByte)	변형된 LZSS 압축 알고리즘	RLC 압축 알고리즘
5	0.763	0.752
30	0.827	0.768
50	0.819	0.795
200	0.786	0.764

에서 변형한 LZSS 압축 알고리즘을 적용한 경우 압축 효율이 떨어짐을 알 수 있다. 이것은 본 논문에서 변형한 LZSS 압축 알고리즘이 적은 크기의 데이터를 사용하는 임베디드 시스템 환경에 적합하게 되었다는 것을 의미한다. 따라서 이 실험 결과 본 논문에서 제안하는 변형된 LZSS 압축 알고리즘이 데이터 양이 적은 임베디드 시스템 환경에 적합한 압축 파일 시스템으로 사용하기에 적합함을 알 수 있다. 그러나 <표 2>의 결과에서 특정한 경우에 압축율이 오히려 원래 데이터보다 늘어나는 경우가 발생한다. 이것은 예외적인 경우로 압축 알고리즘이 모든 경우에 좋은 압축율을 보일 수 있는 것은 아니라는 사실을 보여준다. 그러나 일반적으로 대부분의 경우에 데이터 크기가 적을 경우 본 논문에서 제안한 변형된 LZSS 압축 알고리즘이 좋은 성능을 보임을 알 수 있다.

5.2 압축 효율 비교 실험

RLC알고리즘과 본 논문에서 제안한 변형된 LZSS 압축 알고리즘에 대한 성능 측정을 수행하였다. 성능 측정 환경은 동일한 환경에서 이루어졌다. 다음 <표 3>은 RLC 알고리즘에 대한 압축 효율 측정 결과이다.

<표 3>의 결과와 <표 2>의 결과를 비교해 보면 본 논문에서 제안하는 변형된 LZSS 압축 알고리즘의 압축률이 기존의 RLC 압축 알고리즘보다 20% 정도 우수함을 알 수 있다. 용량이 큰 경우의 압축 효율 비교 실험 결과는 다음 <표 4>와 같다.

<표 4>의 결과에서 파일 크기가 클 경우에 본 논문에서 제안하는 변형된 LZSS 압축 알고리즘이 기존의 RLC 압축 알고리즘보다 성능이 좋지 않음을 알 수 있다. 이것은 변형된 압축 알고리즘이 소량의 데이터에 적합하게 압축 알고리즘이 동작함을 알 수 있다. 앞으로 연구에서는 용량이 큰 파일에 대해서 압축율이 현재보다 좋아질 수 있도록 연구가 진행되어야 할 것이다.

5.3 압축 알고리즘의 성능

본 논문에서 제안하는 변형된 LZSS 압축 알고리즘을 임

〈표 5〉 LZSS 압축 알고리즘의 성능 측정 결과(시간 단위 : ns)

파일 크기 (KByte)	압축 알고리즘을 적용하지 않은 경우 시간	LZSS 압축 알고리즘을 적용한 경우 시간
5	0.002	0.002
30	0.05	0.0056
50	0.07	0.82
200	0.15	0.23

베디드 리눅스 시스템에 적용했을 경우에 어느 정도 성능의 부담이 되는지를 실험하였다. 실험 결과는 다음 <표 5>와 같다.

<표 5>는 임베디드 리눅스 시스템에서 파일 시스템 내에 동일한 크기의 파일에 대해서 압축을 하지 않은 경우에 대한 데이터 읽기 시간과 LZSS 압축 알고리즘을 적용한 경우에 데이터 읽기 시간을 측정한 결과이다. 본 논문에서 제안한 변형된 LZSS 압축 알고리즘을 적용한 경우가 압축 알고리즘을 적용하지 않은 경우보다 성능이 약 20%~40% 정도 떨어짐을 알 수 있다.

6. 결 론

본 논문에서는 PDA용 mtd 패치를 한 임베디드 리눅스의 파일시스템(JFFS2)의 LZSS 압축 알고리즘을 적용하여 기억공간을 보다 효율적으로 사용 임베디드 리눅스 기반의 PDA에 대한 압축 파일 시스템 설계는 메모리 사용을 극대화시킬 수 있다. 이로써 PDA 압축 파일시스템은 보다 나은 저장 공간의 관리가 이루어질 수 것이고 본 연구 결과를 바탕으로 임베디드 시스템을 사용하는 PDA 단말기 등 정보 가전기기들에 적용하여 사용할 경우 보조기억장치의 저장 공간 효율을 기대할 수 있다. 본 논문에서 구현한 LZSS 압축 알고리즘을 jffs2 파일시스템에 적용하여 실험을 하였다. 본 논문의 실험은 두가지 측면으로 실험이 이루어졌다. 하나는 압축률을 어느정도 높일 수 있는지에 대한 부분과 다른 하나는 기존의 다른 압축 알고리즘과의 비교 실험이다. 그리고 압축에 따른 속도 저하가 어느 정도인지에 대한 실험을 수행하였다. 일반적으로 대부분의 경우에 데이터 크기가 적을 경우 본 논문에서 제안한 변형된 LZSS 압축 알고리즘이 좋은 성능을 보임을 알 수 있다. 또한 본 논문에서 제안한 변형된 LZSS 압축 알고리즘을 적용한 경우가 압축 알고리즘을 적용하지 않은 경우보다 성능이 약 20% ~40% 정도 떨어짐을 알 수 있다. 본 논문에서 제안한 변형된 LZSS 압축 알고리즘은 소량의 데이터를 많이 사용하는 임베디드 시스템에 적합하다.

본 논문에서 제시하는 알고리즘은 소량의 메모리를 갖는 임베디드 시스템에 적합하다. 본 논문에서 제시하는 알고리즘이 성능의 부담을 주지만 압축률을 높여서 기억공간의 활용을 높일 수 있는 임베디드 시스템 환경에 적합하다.

참 고 문 헌

[1] 이연조, 임베디드 리눅스 프로그래밍, pcbook, 2002.

[2] 박장수, Linux hacker들을 위한 UNIX KERNEL 완전분석으로 가는 길, pp.110-140, 1995.

[3] 이석재, 이재권, 송석일, 유재수 "분산 파일시스템을 위한 효율적인 협력캐시 알고리즘의 설계 및 구현", 2004.05.

[4] 임근수, 양훈모, 차호정 "선택적 압축방식에 기반한 확장된 플래시 메모리 스와핑 시스템", 2002.10.

[5] KARIM TAGH MOUR, Building Embedded LINUX SYSTEMS, O'REILLY, 2003.

[6] Ross N. Williams, "An Extremely Fast Ziv-Lempel Data Compression Algorithm", Data Compression Conference, 1991.

[7] MOSHE BAR, Linux File Systems, Osborne, pp.219-232, 2001.

[8] Khalid Sayood and Moragn Kaufmann, Introduction to Data Compression Second Edition, 2000.

[9] David A Rusling "The Linux Kernel," January, 1999.

[10] Nathan K. Edel, Deepa Tuteja, Ethan L. Miller, and Scott A. Brandt, "A compressing file system for non-volatile RAM," 2004.

[11] Craig Hollabaugh, Embedded Linux Hardware, Software, and Interfacing,Crag Hollabaugh, Addison-Wesley, 2002.

[12] Gary Nutt, kernel project for Linux, Addison Wesley, 2000.

장 승 주

e-mail : sjjang@deu.ac.kr

1985년 부산대학교 계산통계학과(전산학학사)

1991년 부산대학교 계산통계학과(전산학석사)

1996년 부산대학교 컴퓨터공학과(박사)

1987년~1996년 한국전자통신연구원 시스템 S/W연구실

1993년~1996년 부산대학교 시간강사

2000년~2002년 University of Missouri at Kansas City, visiting professor

1996년~현재 동의대학교 컴퓨터공학과 부교수

관심분야 : 운영체제, 임베디드 시스템 운영체제, 분산시스템, 시스템 보안