논문 2011-06-05

운영체제 없는 시스템의 메모리 절감을 위한 요구 페이징 기법

(A Demand Paging for Reducing The Memory Usage of OS-Less Embedded Systems)

류 경 식*, 전 현 재, 김 용 득
(Kyeung Seek Lew, Hyun Jae Jeon, Yong Deak Kim)

Abstract : For a NAND booting based embedded system, an application program on the NAND flash memory is downloaded to the RAM when the system is booted. In this case, the application program exists in both the RAM and the NAND flash so the RAM usage is increased. In this paper, we suggested the demand paging technique for the decreasing of the RAM usage for OS-less NAND booting based embedded systems. As a result of a benchmark test, 40~80% of the code memory usage was reduced with below 5% of execution time delay.

Keywords : Non-OS, Demand paging, Memory, RAM

Ⅰ. 서 론

본 연구에서는 NAND 플래시 부팅 기반의 운영체제 없이 독립형(Standalone)으로 실행파일이 구동되는 임베디드 시스템에서 주 메모리의 사용량을 절감할 수 있도록 멀티 프로세싱 기반의 운영체제에서 사용하는 요구 페이징(Demand Paging) 기법을 적용할 수 있는 플랫폼을 설계하였다. 그리하여 절감된 주 메모리의 사용량을 스택(Stack) 메모리의 증대에 사용하면 실행 중(Run time) 스택 넘침 오류를 줄일 수가 있어 시스템의 신뢰성 향상에 기여할 수 있다.

1. 임베디드 시스템의 메모리 환경

최근 휴대전화나 멀티미디어 기기의 출시가 많아지고 있다. 그때 채택되는 대용량 저가 보조 메모리로 NAND 플래시의 채택이 높아지고 있다.

과거에는 코드 메모리로 NOR 플래시, 데이터 메모리로 DRAM, 보조 기억장치로 NAND 플래시가 사용이 되었다. 그러나 NAND 플래시가 비휘발성 메모리이므로 코드도 NAND 플래시에 저장을 하게 되면 NOR 플래시를 제거 할 수 있게 된다.

그러나 NAND 플래시는 일반 메모리처럼 고전적인 CPU 버스에 직접 인터페이스 될 수 없으므로 부팅용으로 사용하기 위해서는 별도의 하드웨어 장치가 프로세서에 내장되어야 한다.

2. 연구 배경 및 연구 목적

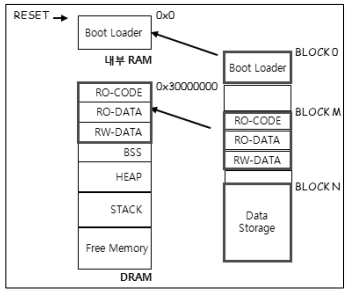

그림 1. NAND 부트로더의 동작
Fig. 1. Operation of a NAND bootloader

NAND 부팅은 그림 1과 같이 상위 수 KB(프로세서에 따라 다름)의 NAND 부트로더를 프로세서 내부 SRAM으로 적재한 후 이를 실행한다. 실행된

* 교신저자(Corresponding Author)
논문접수 : 2010. 12. 02., 수정일: 2010. 12. 20.,
채택확정 : 2010. 12. 28.
류경식, 전현재, 김용득 : 아주대학교 전자공학부

NAND 부트로더는 다시 지정된 NAND 메모리 위치에서 실제 실행할 프로그램을 DRAM으로 적재하고 실행하는 절차로 진행된다.

그러나 이러한 NAND 부팅을 이용하면 코드가 NAND 플래시와 RAM에 모두 존재하게 되므로 RAM의 소요량이 증대 되게 된다. 본 논문에서는 이러한 NAND 부팅의 단점을 보완하기 위해 운영체제에서 사용하는 요구페이징을 적용하기로 하고 먼저 기존 연구들을 분석하였다.

기존 연구들에서 NAND 플래시 부팅 기반 시스템에 멀티프로세싱 기반 운영체제를 사용하는 경우 요구페이징 효율을 증가시키기 위하여 NAND 플래시의 효율적인 억세스 기법들을 제안하고 있거나 [1] 운영체제 없는 환경에서는 요구페이징을 구현하기 위하여 컴파일러를 수정하는 것을 전제로 MMU도 없이 요구페이징을 구현하는 방법이 제안되었다 [2]. 그러나 본 연구에서는 기존 컴파일러를 그대로 이용하면서도 요구페이징이 가능하도록 하였다. 이를 통하여 NAND 플래시 부팅 기반 시스템의 DRAM에 응용프로그램이 실행될 때 메모리 사용량을 절감할 수가 있다. 이 때 절감한 메모리 사용량을 스택에 할당하여 응용프로그램 실행 시 스택 넘침으로 인한 시스템 오류를 방지하는데 활용하고자 하였다.

이를 위하여 본 논문은 운영체제 없이 요구페이징을 구현하기 위해 eDPL(embedded Demand Paging Loader)을 설계하였으며 벤치마크 프로그램을 대상으로 성능 평가를 수행하여 메모리 사용의 절감량과 실행 속도 저하율을 분석하였다.

II. 운영체제 없는 시스템의 요구페이징

1. C로 작성된 프로그램의 메모리 환경

일반적인 C로 작성된 프로그램은 그림 2와 같은 메모리 구조로 존재한다.

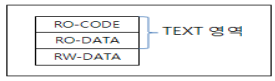

그림 2. 응용 프로그램 메모리 구조
Fig. 2. Memory structure of an application program

TEXT영역은 RO-CODE와 RO-DATA로 구성되며 RO-CODE는 프로그램 명령, 오퍼랜드, RO-DATA는 문자열 상수, const 지시자를 사용한 변수 및 상수, RW-DATA는 초기화된 전역변수, 초기화된 static 지시자를 사용한 지역변수가 존재하게 된다.

그림 2와 같은 메모리를 갖는 응용프로그램이 실제 실행 시는 그림 3과 같은 메모리를 갖게 된다. BSS 영역은 초기화 되지 않은 전역변수, 초기화 하지 않은 static 지시자를 사용한 지역변수가 존재하게 되고 동적으로 할당하기 위한 HEAP과 STACK이 존재하게 된다.

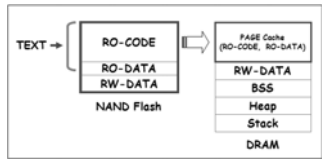

그림 3. 실행 시 응용 프로그램 메모리 구조
Fig. 3. Memory structure of an application program at run-time

2. PEP의 설계

eDPL이 NAND 플래시에 있는 실행 파일을 분할 적재하기 위하여 응용프로그램의 이진파일에서 RW-DATA의 위치와 크기, BSS 영역의 크기를 알아야 한다. 이를 위하여 본 논문에서는 ARM 컴파일러에서 생성한 ELF 파일을 분석하고 [3] 실제 실행파일인 이진파일을 추출하고 TEXT, RW-DATA, BSS 영역의 정보를 파악하여 이를 이진파일의 앞부분에 헤더로 추가하는 PEP(Post ELF Processor)를 PC 프로그램으로 설계하였다.

PEP가 생성하는 eDPL-BIN의 구조는 그림 4와 같고 추가되는 헤더의 구조는 그림 5와 같다.

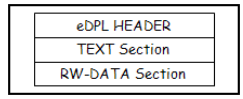

그림 4. eDPL-BIN 파일 구조
Fig. 4. File structure of an eDPL-BIN

```
struct eDPL_BIN_Headr_16
{
    unsigned int ROM_INIT_SIZE;
    unsigned int ROM_TEXT_SIZE;
    unsigned int RW_DATA_SIZE;
    unsigned int BSS_SIZE;
}
```

그림 5. eDPL-BIN의 헤더 구조
Fig. 5. Header structure of an eDPL-BIN

3. eDPL 메모리 환경

본 연구에서 실험을 위하여 설계되는 eDPL은 그림 6과 같은 메모리 맵을 이용하여 프로그램이 구동된다.

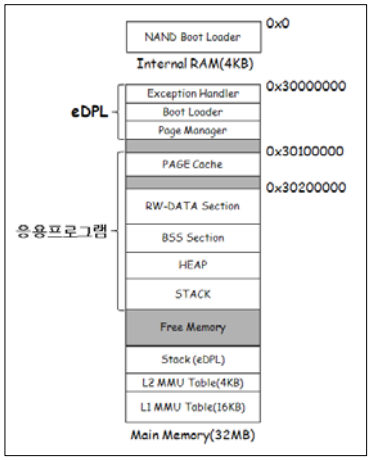

그림 6. eDPL 환경의 메모리 맵

Fig. 6. Memory map of an eDPL environment

응용 프로그램은 운영체제 없이 독립형으로 구동이 되지만 eDPL 기반에서는 요구 페이징 로더가 운영체제 역할을 대신하게 되므로 eDPL의 구동 환경과 맞물려서 설계되어야 한다.

0번지부터 4KB는 NAND 플래시 부트로더가 적재되며 eDPL은 최대 1MB 영역에 적재된다. eDPL이 사용하게 될 MMU 변환 테이블과 예외 처리를 위한 스택은 메모리 하부에 설치된다. 응용프로그램은 TEXT 영역인 RO-CODE와 RO-DATA 영역에서 페이지 교체가 이루어지게 되는데 페이지 캐시 영역에 적재하게 된다. 페이지 캐시는 1KB, 4KB의 크기를 선택할 수 있게 설계 하였고 각각의 크기에서 2~16개의 페이지 묶음을 선택할 수 있게 설계하여 다양한 페이지 크기에서 실험을 수행 할 수 있게 하였다.

4. eDPL의 역할과 응용프로그램의 구동

NAND 플래시에 eDPL과 실행파일(eDPL-BIN)을 탑재한 후 전원이 인가되면 NAND 플래시 부트로더가 eDPL을 DRAM적재하여 실행하게 되고 eDPL은 NAND 플래시에 있는 eDPL-BIN에서 페이지 캐시에 설정된 사이즈 크기의 실행파일을 적재 하게 되는데 eDPL-BIN의 가장 앞부분에서 불러오게 된다. 그 후 페이지 부재 요청이 있을 때 마다 설정된 페이지의 크기만큼 페이지 캐시에 적재와 제거를 수행하게 된다. 이러한 동작은 그림 7에 보이고 있다.

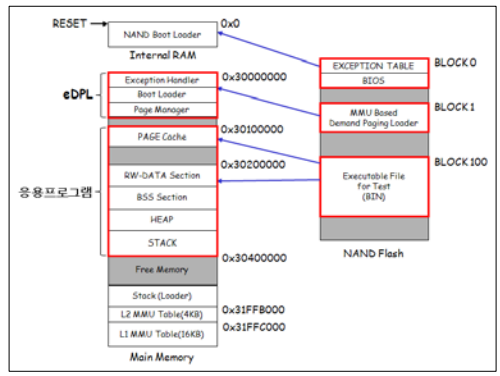

그림 7. eDPL의 동작과 메모리 구조

Fig. 7. Operation and memory organization of an eDPL

Ⅲ. 제안한 eDPL의 설계

1. ARM프로세서의 모드와 예외

표 1. ARM 프로세서의 예외 테이블

Table 1. Exception table for ARM processor

Exception	발생원인	예외 vector	전환모드
Reset	시스템 리셋	0x0	SVC
Undefined Instruction	정의 되지 않은 명령의 수행	0x4	UND
SWI	System Call	0x8	SVC
Pre-fetch Abort	명령어 fetch 시 메모리 violation	0xC	ABT
Data Abort	데이터 access시 메모리 violation	0x10	ABT
-	reserved	0x14	-
IRQ Request	Normal interrupt	0x18	IRQ
FIQ Request	Fast interrupt	0x1C	FIQ

ARM 프로세서는 프로그램 실행 중 프로그램을 중지시키고 다른 프로그램으로 전환하기 위한 예외가 표 1과 같이 마련되어 있다 [4].

eDPL이 초기에 페이지 캐시에 적재된 코드를 실행하다 적재되지 않은 프로그램 영역의 코드나 데이터 접근이 발생하면 각각 Prefetch-Abort(P-ABORT)와 Data-Abort(D-ABORT) 예외가 발생하게 된다. 따라서 본 연구에서는 ARM 프로세서의

P-ABORT와 D-ABORT 발생 시 처리를 위한 핸들러에서 페이지 전환을 관리 하도록 한다.

2. eDPL의 페이지 관리

초기에 eDPL의 페이지 캐시가 페이지 크기는 1KB 이고 페이지 수가 4개일 때의 MMU 테이블은 표 2와 같다. 아래의 주소는 표에서 사용된 주소를 표기화한 것이다.

⓪ : 0x30100000 ~ 0x301003FF
① : 0x30100400 ~ 0x301007FF
② : 0x30100800 ~ 0x30100BFF
③ : 0x30100C00 ~ 0x30100FFF
④ : 0x30101000 ~ 0x301013FF

표 2. 초기상태 MMU 테이블
Table 2. Initial status of an MMU table

Entry 번호	TEXT 메모리 주소 (VA)	적재된 페이지 캐시 주소 (PA)	Access 속성
0	⓪	⓪	가능
1	①	①	가능
2	②	②	가능
3	③	③	가능
4	④	-	불가능
	:	:	
1023	0x301FFC00 ~ 0x301FFFFF	-	불가능

표 3. 페이지 교체 후 MMU 테이블
Table 3. The MMU table status after pages replaced.

Entry 번호	TEXT 메모리 주소 (VA)	적재된 페이지 캐시 주소 (PA)	Access 속성
0	⓪	⓪	가능
1	①	-	불가능
2	②	②	가능
3	③	③	가능
4	④	①	가능
	:	:	
1023	0x301FFC00 ~ 0x301FFFFF	-	불가능

표 2는 응용 프로그램의 TEXT영역에 대응하는 테이블로서 전체 1024개 엔트리를 갖고 하나의 엔트리는 1KB 영역에 대한 가상주소와 물리주소의 변환을 담당한다. 따라서 처음 적재 시 0~3번 프레임에는 0~3번 페이지가 적재되어 있으므로 MMU 테이블의 0,1,2,3 엔트리만 접근속성을 가능으로 하고 나머지 엔트리는 불가능으로 한다. 그리고 변환하는 물리주소는 실제 페이지 캐시의 물리적 주소로 한다. 만약 적재된 영역의 응용 프로그램이 구동되다가 다른 페이지로 분기하거나 다른 페이지의 데이터를 접근하면 P-ABORT 또는 D-ABORT가 발생하게 되고 해당 핸들러의 페이지 관리기는 기존 페이지 캐시에서 제거할 프레임 번호와 새로 적재될 페이지를 계산한 후 새로운 페이지를 제거할 프레임에 적재한 다음 MMU 테이블의 엔트리 정보를 수정한다.

표 2와 표 3의 경우처럼 페이지의 크기가 1KB 이고 페이지 수가 4개일 때 4번 페이지가 새로이 적재 요구되고 순차배정 방식에 의한 교체 순서에 따라 1번 프레임이 제거될 경우 4번 페이지를 1번 프레임에 적재한 후 MMU의 1번 엔트리는 억세스 불가로 4번 엔트리는 억세스 가능으로 하고 4번 엔트리의 물리주소 영역을 1번 프레임의 메모리 주소 값으로 기록한다.

따라서 새로 적재된 4번 페이지의 가상주소는 표 2에서 보듯이 0x3010100 ~ 0x301013FF까지 1KB이지만 실제 MMU는 4KB(페이지 사이즈 1KB, 페이지 수 4개) 페이지 캐시의 1번 프레임에 적재되므로 물리주소는 0x30100400 ~ 0x301007FF로 설정 된다.

3. eDPL의 페이지 교체 알고리즘

eDPL은 페이지 교체를 위하여 기본적으로 순차배정(Round-Robin) 방식을 사용하며 순차배정을 위해 페이지 캐시의 수와 같은 크기의 FIFO를 이용한다[5].

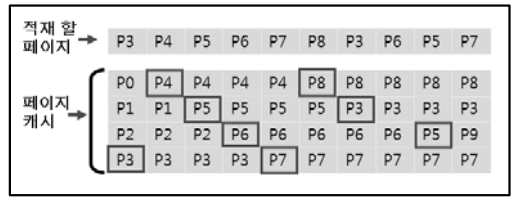

그림 8. FIFO 기반 페이지 교환 기법
Fig. 8. FIFO based page replacement method

FIFO를 이용한 동작은 그림 8과 같다. 초기에 페이지 캐시에 0, 1, 2, 3 페이지가 적재된 상태에서 4번 페이지가 적재 요구되면 가장 오래전에 적

재된 0번 페이지를 제거하고 4번 페이지가 적재된다. 다시 5번 페이지가 적재 요구되면 1번 페이지가 제거된다.

FIFO를 이용한 페이지 관리기의 페이지 교체 프로그램 동작은 그림 9와 같다 [1-5].

그림 9를 보면 P-ABORT나 D-ABORT 발생 시 발생 당시의 주소를 설정한 페이지 크기로 나누어 적재 요구페이지 주소(L)을 계산 하게 된다. 그 후 제거대상(Victim) 포인터의 값을 참고하여 제거 대상 페이지(D)를 선정 한다. D를 제거하고 L페이지를 적재한 후 MMU 테이블을 표 2와 표 3과 같은 순서에 의하여 수정하게 되면 페이지의 교체가 완료 되게 된다.

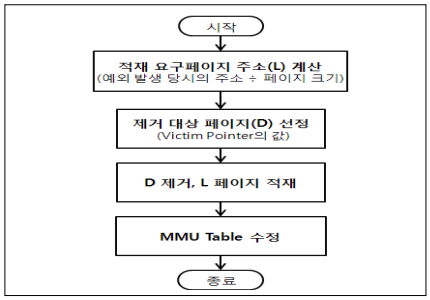

그림 9. 페이지 교체 알고리즘 순서도
Fig. 9. Flow chart of a page replacement algorithm

Ⅳ. 성능평가

1. 테스트를 위한 벤치마크 선정

실험용 보드로는 ARM920T 코어의 S3C2440 프로세서가 사용된 GBOX를 사용 하였다.

표 4. 사용된 벤치마크 프로그램
Table 4. List of used benchmark program

프로그램	프로그램 설명
WAV 음악 재생	실시간성이 중요하고 DMA를 사용, DMA에 미치는 영향 파악
ZIP-UNZIP	지역적 특성 강한 연속 데이터의 연산 수행, 공개 프로그램
동영상 재생	초당 15 프레임 동영상 재생, 인터럽트 사용, 실시간성 중요
LINPACK	공개된 선형대수 연산 수행 벤치마크 프로그램
DHRYSTONE	널리 알려진 프로세서 성능 측정용 벤치마크, 정수연산 수행

실험의 평가를 위한 프로그램으로 3개(WAV 음악 재생, ZIP-UNZIP, 동영상 재생)의 자체 개발 프로그램과 널리 사용되는 2개의 벤치마크(LINPACK, DHRYSTONE) 프로그램을 사용 하였다. 사용된 프로그램의 기능은 표 4와 같다.

2. ZIP-UNZIP 성능 평가 결과

성능 평가에 사용한 ZIP-UNZIP 프로그램의 TEXT 영역의 크기는 91,128B이며 요구 페이징을 적용하지 않았을 때 기본 동작시간은 2,987ms이다. 이 프로그램에 eDPL을 적용하여 실험한 결과는 표 5, 표6과 같다. 표5는 페이지 크기를 1KB로 하고 프레임 수를 4부터 16까지 변경하면서 실험한 결과이고 표 6은 페이지 크기를 4KB로 하고 프레임 수를 4부터 16까지 변경하면서 실험한 결과이다.

표 5. 1KB의 페이지 크기에서 ZIPUNZIP 분석
Table 5. Test results of a ZIPUNZIP program with a 1KB page size

프레임수	4	6	8	10	12	14	16
페이지 크기	1 KB	1 KB	1 KB	1 KB	1 KB	1 KB	1 KB
Text크기 (B)	91,128	91,128	91,128	91,128	91,128	91,128	91,128
기본동작 시간(ms)	2,987	2,987	2,987	2,987	2,987	2,987	2,987
RAM 감소량(B)	87,032	84,984	82,936	80,888	78,840	76,792	74,744
RAM 감소율(%)	95.51	93.26	91.01	88.76	86.52	84.27	82.02
동작시간 (ms)	49,750	7,070	3,552	3,089	3,053	3,044	3,057
시간 증가율(%)	1,565	136.69	18.92	3.21	2.00	1.91	2.34

표 6. 4KB의 페이지 크기에서 ZIPUNZIP 분석
Table 6. Test results of a ZIPUNZIP program with a 4KB page size

프레임수	4	6	8	10	12	14	16
페이지 크기	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB
Text크기 (B)	91,128	91,128	91,128	91,128	91,128	91,128	91,128
기본동작 시간(ms)	2,987	2,987	2,987	2,987	2,987	2,987	2,987
RAM 감소량(B)	74,744	66,552	58,360	50,168	41,976	33,784	25,592
RAM 감소율(%)	82.02	73.03	64.04	55.05	46.06	37.07	28.08
동작시간 (ms)	3,384	3,102	3,031	3,021	3,032	3,030	3,006
시간 증가율(%)	13.29	3.85	1.47	1.14	1.51	1.44	0.64

표5와 표6에서 페이지 크기는 요구 페이징으로 적재할 메모리 크기이며 프레임 수는 페이지 캐시

내의 프레임 수를 의미한다. TEXT 크기는 응용프로그램 중 요구 페이징으로 분할 적재될 TEXT 영역의 크기를 의미하고 RAM 감소량은 TEXT 크기에서 페이지 크기와 프레임 수를 곱한 페이지 캐시 메모리의 크기를 뺀 메모리의 크기를 의미한다. 기본동작시간은 요구 페이징을 적용하지 않은 벤치마크 프로그램의 실행시간이고 동작시간은 요구 페이징으로 실행시킨 실행시간 이며 시간증가율은 기본동작시간보다 지연된 동작시간의 비율을 의미한다.

표5을 보면 1KB의 페이지 크기에서는 페이지 수가 10개 일 때 프로그램의 실행 시간 지연율 3.21%에서 88.76%의 TEXT RAM 사용량의 감소율을 보이고 있다. 표6을 보면 4KB의 페이지 크기에서는 페이지 수가 6개일 때 3.85%의 실행 시간 증가율에서 73.03%의 RAM 사용량의 감소율을 보이고 있다. 두 경우 모두 5% 이하의 실행시간 지연을 보이지만 페이지 크기가 1KB, 10개 프레임 인 경우 페이지 캐시의 크기는 10KB이지만 페이지 크기가 4KB, 6개 프레임인 경우 페이지 캐시의 크기는 24KB이므로 이 프로그램에서는 1KB 페이지 크기를 사용하는 것이 더 효율적인 것으로 분석된다.

위의 표의 결과를 비교하기 편리하게 하기 위하여 그림 10에 1KB, 4KB 페이지 크기에 대하여 페이지의 수에 따른 각각의 동작시간과 메모리 사용량을 보이고 그림 11에 페이지 부재 오류에 의한 페이지 전환횟수를 그래프로 나타내고 있다.

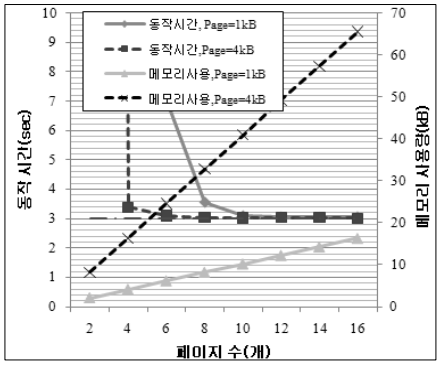

그림 10. ZIPUNZIP 결과 분석 그래프
Fig. 10. Graph results of a ZIPUNZIP program

그림 10의 그래프에서 3초 부근의 점선은 요구 페이징을 사용하지 않을 시의 기본 동작 실행시간을 의미한다.

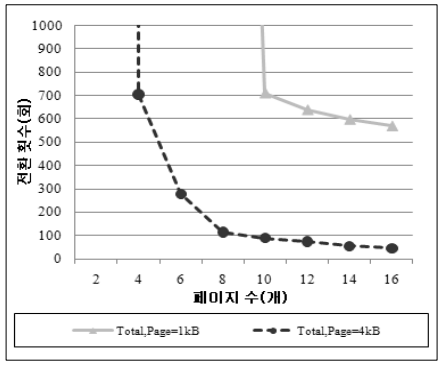

그림 11. ZIPUNZIP 페이지 전환 그래프
Fig. 11. The graph of the page replacement for the ZIPUNZIP program

그림 10, 11에서 보는 것처럼 페이지 크기가 커지고 페이지 수가 많아짐에 따라 동작시간 지연은 감소하고 전환횟수도 적어지는 것을 볼 수 있다.

3. 기타 벤치마크 성능 평가 결과

성능 평가를 위해 사용한 각각의 벤치마크 프로그램의 결과 중 5% 내외의 시간 지연을 보이며 메모리 사용의 감소량에 있어서도 가장 효율이 좋을 때의 페이지 크기와 페이지 수를 선정하여 표 7에 나타내고 있다.

표 7. 벤치마크별 프로그램 실험 결과
Table 7. Test results of each benchmark programs

대상 프로그램	음악재생	동영상	LINPACK	DHRYSTONE
페이지 수	6	8	14	6
페이지 크기	1 KB	4 KB	1 KB	1 KB
전체 RAM (B)	75,528	743,936	26,844	34,348
Text 크기 (B)	71,392	84,632	26,580	23,820
RAM 감소량 (B)	65,248	51,864	12,244	17,676
RAM 감소율 (%)	91.39	61.28	46.06	74.21
기본시간 (ms)	1,619	2,133	1,255	336
동작시간 (ms)	1,687	2,137	1,295	366
시간 증가율 (%)	4.20	0.19	3.19	6.71

전체적으로 5% 내외의 시간 지연에 RAM 사용량의 감소율이 40% ~ 80%를 보이고 있으며 동영상 재생만 4KB의 페이지 크기에서 효율적으로 나타나고 있는데 그 이유는 그래픽 데이터 처리의 특성 때문으로 보인다.

이상의 벤치마크 테스트를 통하여 제안한 eDPL이 실용적으로 사용이 가능함을 확인 하였고 절감한 20KB ~ 80KB의 메모리를 스택이나 힙에 배정하면 시스템의 신뢰성 제고가 가능하게 된다.

Ⅴ. RO-DATA 배제에 의한 성능 개선

1. RO-DATA 배제 전략

TEXT 영역 중 코드에서 호출되어 참조되는 상수를 페이지 교체에서 제외한다는 전략이다. 이를 위해 TEXT 영역 중 RO-DATA 영역을 그림 12와 같이 RW- DATA처럼 고정 메모리에 저장을 하였다.

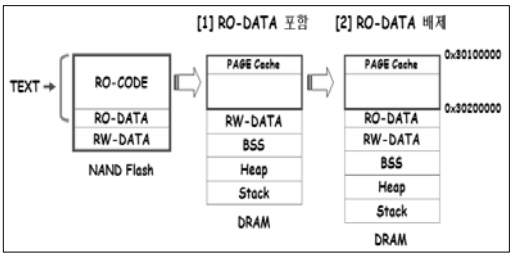

그림 12. RO-DATA 배제 전략
Fig. 12. RO-DATA exclusion strategy

그로 인해 상수 참조로 인한 D-ABORT의 발생 횟수가 줄어들 것이며 이는 응용 프로그램의 실행 시간 지연 감소로 이끌어 질 것이다. 그러나 상대적으로 RO-DATA영역을 고정 메모리에 적재하므로 TEXT영역의 메모리 사용량의 절감 효과는 감소 할 것이다. 이는 요구 페이징에 의한 속도 지연을 더 감소시키고자 하는 경우에 활용이 가능하다.

2. 성능 개선 결과 평가

표 8, 표 9, 표 10은 각각의 벤치마크 프로그램에서 성능개선을 위한 RO-DATA 배제 전략을 적용 했을 경우와 사용하지 않았을 때의 실험 결과를 보이고 있다. 예상한 바와 같이 RAM 사용량의 감소율은 RO-CODE영역의 고정적재로 인해 줄어들게 되었지만 거의 모든 영역에서 시간 증가율이 줄어든 것을 볼 수 있다. 다만 동영상, 음악재생에서는 동작시간의 감소가 적었는데 프로그램의 특성상 RO-DATA 영역의 호출이 빈번하지 않은 이유로 분석된다. DHRYSTONE에서는 시간 감소가 0으로 나왔는데 이 이유는 기본적으로 프로그램의 동작시간이 매우 짧은 프로그램이었기 때문에 실행 시간이 더욱 짧아지게 되어 msec단위까지 측정 가능한 타이머로는 측정이 안 되었기 때문이다.

표 8. 음악재생과 ZIP-UNZIP 실험 결과
Table 8. Test results of WAV play and ZIP-UNZIP programs

대상 프로그램	음악재생			ZIP-UNZIP		
RO-DATA 배제 여부	포함	배제	비교	포함	배제	비교
페이지 수	6		-	10		-
페이지 크기	1 KB		-	1 KB		-
전체 RAM	75,528 B			92,844 B		
Text 크기 (B)	71,392	37,188	-34,204	91,128	76,056	-15,072
RAM 감소량 (B)	65,248	31,044	-34,204	80,888	65,816	-15,072
RAM 감소율 (%)	91.39	83.48	-7.91	88.76	87	-1.76
기본시간	1,619 ms			2,987 ms		
동작시간 (ms)	1,687	1,686	-1	3,089	3,051	-38
시간 증가율 (%)	4.20	4.14	-0.06	3.21	2	-1.21

표 9. LINPACK과 DHRYSTONE의 실험 결과
Table 9. Test results of LINPACK and DHRYSTONE programs

대상 프로그램	LINPACK			DHRYSTONE		
RO-DATA 배제 여부	포함	배제	비교	포함	배제	비교
페이지 수	14		-	6		-
페이지 크기	1 KB		-	1 KB		-
전체 RAM	26,844 B			34,348 B		
Text 크기 (B)	26,580	25,832	-748	23,820	23,072	-748
RAM 감소량 (B)	12,244	11,496	-748	17,676	16,928	-748
RAM 감소율 (%)	46.06	44.50	-1.56	74.21	73.37	0.84
기본시간	1,255 ms			336 ms		
동작시간 (ms)	1,295	1,285	-10	366	366	0
시간 증가율 (%)	3.19	2.39	-0.8	6.71	6.71	0

표10은 동영상 재생 분석표이다. 동영상 재생은 다른 프로그램과는 다르게 4 KB의 페이지 크기를 갖는 경우 가장 성능이 좋게 나왔다. 이처럼 다양한 프로그램에서 각기 다른 결과를 보이고 있으므로 응용 프로그램에 따라 선별적으로 페이지 크기와 페이지 수를 채택하게 되면 적절한 동작시간 지연으로 코드 메모리의 사용량을 감소시키는 것이 가능하게 된다.

표 10. 동영상 재생 실험 결과
Table 10. Test results of a video play program

대상 프로그램	동영상 재생		
RO-DATA 배제 여부	포함	배제	비교
페이지 수	5		-
페이지 크기	4 KB		-
전체 RAM	743,936 B		
Text 크기 (B)	84,632	49,804	-34,828
RAM 감소량 (B)	51,864	17,036	-34,828
RAM 감소율 (%)	61.28	34.21	-27.07
기본시간	2,133 ms		
동작시간(ms)	2,137	2,136	-1
시간 증가율 (%)	0.19	0.14	-0.05

VI. 결론

본 논문에서는 NAND 플래시 메모리 또는 SD 카드 기반의 부팅 방식을 지원하는 운영체제가 없는 임베디드 시스템 환경에서 RAM의 사용량을 절감하기 위한 요구 페이징 기법인 eDPL에 대하여 제안하고 이를 설계하였다. 또한, 설계된 eDPL의 성능 평가를 위한 여러 가지의 벤치마크 프로그램을 사용하여 메모리의 사용량을 절감 하면서도 응용 프로그램을 적은 속도 지연으로 실행이 가능함을 확인하였다. 본 논문의 연구 결과를 임베디드 시스템의 개발에 적용할 경우 목표한 바와 같이 RAM의 사용량이 감소되어 이 감소량을 힙과 스택으로 배정이 가능하게 된다. 따라서 프로그램 실행 시에 시스템의 안정성을 저해하는 스택 넘침 문제나 힙의 부족으로 인한 프로그램 강제종료의 발생 가능성을 저하시킬 수가 있게 된다.

본 연구에서 제안한 eDPL을 실제 임베디드 시스템 개발에 적용 할 때 프로그램의 특성에 따라서 실시간성이 중요한 프로그램은 페이지 수와 크기를 크게 선택하여 메모리 사용의 절감량은 다소 줄더라도 실행 시간의 지연을 줄이는 방향으로 선택을 할 필요가 있으며 실시간 특성 보다 메모리 사용량의 절감에 의한 스택과 힙의 증가가 더 중요한 프로그램의 경우는 실행 속도 지연이 다소 커지더라도 메모리 사용의 절감량이 큰 조건을 선택하여야 한다.

이를 위하여 eDPL은 페이지 수를 2~16까지 페이지 크기를 1KB, 4KB 중에서 선택하여 실행시킬 수 있도록 하여 응용 프로그램을 eDPL을 이용, 실제 실험을 수행하여 eDPL이 보고해 주는 시간 지연율과 메모리 절감율에 따라 가장 적합한 페이지 수와 페이지 크기의 조건을 선택할 수가 있다. 이와 같이 eDPL을 이용하여 원하는 실행시간 지연을 갖는 조건을 선택하여 eDPL-BIN을 생성한다면 메모리 사용량의 절감 효과를 극대화 시키면서도 적은 시간 지연으로 프로그램을 수행시킬 수가 있다.

추후 본 연구 결과를 바탕으로 지속된 연구를 통하여 보다 더 큰 메모리 사용의 절감을 보이면서 응용 프로그램의 실행시간 지연은 최소화하는 새로운 기법들을 발굴하여 요구 페이징에 의한 성능 향상을 더욱 제고할 필요가 있다.

참고문헌

[1] Yongsoo Joo, Yongseok Choi, Chanik Park, Sung Woo Chung, Eui-Young Chung, Naehyuck Chang, "Demand paging for one-NANDTM flash eXecute-in-place", CODES+ISSS''06, pp. 229-234, Korea, Oct, 2006.

[2] Chanik Park, Junghee Lim, Kiwon Kwon, Jaejin Lee, Sang Lyul Min, "Compiler-assisted demand paging for embedded systems with flash memory", EMSOFT''04, pp. 114-124, Pisa, Italy, Sep, 2004.

[3] ARM Limited, SWS ESPC 0003 B-02: ARM ELF Development Systems Business Unit Engineering Software Group, June, 2001.

[4] ARM Limited, ARM DDI 0100D: ARM Architecture Reference Manual, Feb, 2000.

[5] ARBRAHAM SILBERSCHATZ, PETER BAER GALVIN, GREG GAGNE, Operating System Concepts, JOHN WILEY & SONS. INC, United States of America, 2005.

저 자 소 개

류 경 식

1991년 : 아주대학교 전자공학과 공학사.
1993년 : 아주대학교 전자공학과 공학석사.
2006년 : 아주대학교 전자공학과 박사 수료.
현재, (주)월텍 대표이사.
관심분야 : 임베디드 시스템, 자동화 시스템.
Email : keyseek@naver.com

전 현 재

2010년 : 아주대학교 전자공학부 학사.
현재, 아주대학교 전자공학부 석사과정.
관심분야 : 임베디드 시스템.
Email : hyunjaiyaa@naver.com

김 용 득

1971년 : 연세대학교 전자공학전공 학사.
1973년 : 연세대학교 전자공학전공 석사.
1978년 : 연세대학교 전자공학전공 박사.
현재, 아주대학교 전자공학부 정교수.
관심분야 : 통신, 컴퓨터, ITS.
Email : yongdkim@ajou.ac.kr