논문 2010-05-27

Non-OS 임베디드 시스템에서 개선된 알고리즘을 적용한 요구 페이징 기법

(Demand Paging Method Using Improved Algorithms on Non-OS Embedded System)

류 경 식*, 전 창 규, 김 용 득
(Kyeung Seek Lew, Chang Kyu Jeon, Yong Deak Kim)

Abstract : In this paper, we try to improve the performance of the demand paging loader suggested to use the demand paging way that is not based on operating system. The demand paging switching strategy used in the existing operating system can know the recently used pages by running multi-processing. Then, based on it, some page switching strategies have been made for the recently used pages or the frequently demanded pages. However, the strategies based on operating system cannot be applied in single processing that is not based on operating system because any context switching never occur on the single processing. So, this paper is trying to suggest the demand paging switching strategies that can be applied in paging loader running in single process. In the Return-Prediction-Algorithm, we saw the improved performance in the program that the function call occurred frequently in a long distance. And then, in the Most-Frequently-Used-Page-Remain-Algorithm, we saw the improved performance in the program that the references frequently occurred for the particular pages. Likewise, it had an enormous effect on keeping the memory reduction performance by the demand paging and reducing the running time delay at the same time.

Keywords : Embedded system, Demand paging, NAND flash, MMU

Ⅰ. 서 론

1. 연구배경

임베디드 시스템은 부팅용 코드 메모리로 NOR 플래시 메모리나 ROM을 사용해왔지만, 최근의 멀티미디어 장치들은 NAND 플래시 메모리를 이용하여 실행파일을 RAM으로 이동시켜 실행하는 NAND 부팅을 많이 사용하고 있다. 그러나 이 방법을 사용하게 되면 NOR나 ROM이 제거되어 경제적이지만 프로그램을 모두 RAM에 적재하여 실행하므로 기존의 방식에 비해 RAM의 소요량이 증가하게 된다.

이와 같이 한정된 메모리를 활용해야 하는 임베디드 시스템에서 NAND부팅으로 인하여 RAM의 사용량이 증가하게 되면 상대적으로 프로그램을 실행시 동적으로 사용하게 되는 스택이나 힙의 사용량이 줄어들게 된다. 따라서 본 연구에서는 운영체제를 사용하지 않는 NAND 부팅 기반 임베디드 시스템에서 프로그램의 DRAM 사용량을 절감하는 방안들에 대하여 연구를 수행하였다.

이러한 연구의 결과로 최초 제안된 방식이 eDPL(Embedded Demand Paging Loader)기법이다. 이 방식은 멀티프로세싱 운영체제에서 사용되는 요구 페이징기법을 운영체제가 없는 임베디드 시스템에서 이용할 수 있는 요구 페이징 기반 로더이다. 이를 이용할 경우 5%이내의 적은 시간 지연에서 코드 메모리가 40~80% 절감되는 효과가 있다. 하지만 이 보다 더 실행 시간 지연을 감소시켜 성능을 향상시키고자 하였다 [11].

그동안 요구 페이징의 시간 지연을 감소시키기

* 교신저자(Corresponding Author)
논문접수 : 2010. 10. 26., 수정일: 2010. 11. 11.,
채택확정 : 2010. 11. 25.
류경식, 전창규, 김용득 : 아주대학교 전자공학부

위한 많은 페이지 교환 전략들이 제안되었다. 하지만 이러한 기법들은 멀티프로세싱 기반 운영체제에서만 사용할 수 있다. 프로세스의 전환이 발생하면 최근에 사용한 페이지를 알 수 있기 때문에 이를 이용한 페이지 전환 전략이 만들어진다 [7][8]. 하지만 프로세스 전환이 발생하지 않는 운영체제가 없는 환경에서는 이러한 페이지 전환 전략을 채택할 수 없다. 따라서 본 논문은 단일 프로세스로 동작하는 eDPL에 적합한 요구 페이징 기법을 연구하였다.

2. 임베디드 시스템의 메모리 환경

고전적인 임베디드 시스템은 부팅시 필요한 코드를 ROM에 저장하고, 프로그램 동작 중 필요한 변수나, 스택, 힙 등은 RAM을 갖는 메모리 환경이었다. 이 구조에서 RAM은 속도는 빠르지만 용량이 크지 않은 SRAM을 사용하였다. 따라서 전원을 처음 인가하면 ROM에서 프로그램이 바로 실행되는 XIP(Execute In Place)방식을 사용해왔다.

그러나 점차 반도체 기술의 발전으로 프로세서의 속도가 빨라지고, 메모리의 용량이 증가하게 되면서 SRAM과 속도에 있어서 많은 차이가 있던 DRAM이 SDRAM으로 발전하게 되었다. 이 변화로 인해 ROM과 소용량의 RAM으로 이루어진 임베디드 시스템 환경이 점차 ROM의 실행파일을 RAM으로 이동시켜 프로그램을 구동하는 RAM 로딩 방식으로 대체되었다.

소형 가전이나 프로그램의 용량이 크지 않은 경우는 가격적인 면에서 저렴한 XIP를 사용하고 있지만, DRAM을 이용하는 시스템은 ROM의 프로그램을 RAM으로 이동시켜서 구동하기 때문에 초기 부팅은 느리지만 프로그램의 수행 속도가 빠른 장점을 갖고 있다.

이처럼 RAM 로딩 방식이 보편화된 상황에서 시스템들의 멀티미디어 지원을 위해서는 보조 메모리가 필요하게 된다. 이 때 주로 NAND 플래시 메모리를 사용하고 있다. NAND 플래시는 용량이 크고 가격이 저렴하지만 ROM과 같이 CPU 버스를 지원하지 않아 부팅용으로 사용할 수 없는 구조이다. 그래서 NAND로부터 프로그램을 RAM으로 적재하여 실행하는 구조를 사용하고 있다 [9].

3. NAND 부팅

앞서 말한바와 같이 NAND 메모리 자체는 부팅용으로 사용할 수 없기 때문에 별도의 ROM을 설치하거나 특수한 NAND를 사용해야 한다. 이러한 문제를 해결하기 위해 임베디드 프로세서 자체에 작은 부팅용 SRAM을 탑재하고 NAND 부팅이라는 기능을 지원하고 있다 [3][4].

NAND 부팅을 위해 임베디드 프로세서 내부에 탑재되는 SRAM은 가격 문제로 인하여 용량이 작은 편이다. 삼성전자의 S3C2440의 경우 4KB, 최근에 출시된 S5PC100 시리즈는 8KB를 갖고 있다. 따라서 이 작은 용량의 메모리에 모든 실행파일을 적재한다는 것은 불가능하다 [2].

따라서 초기 부팅시 가져오는 NAND의 0번 블록은 실제 응용프로그램을 담아두는 것이 아니라, 프로그램을 실행하게 해주는 로더를 기록하게 된다. 이 로더에 의해 NAND로부터 응용프로그램을 읽어주 메모리로 적재한 후 분기를 통해 실행한다 [10].

이 때 응용프로그램 전체가 DRAM에 적재되기 때문에 메모리 공간의 낭비가 발생하게 되고, 이 문제를 개선하기 위하여 요구 페이징 기법을 사용하게 된다.

4. 요구 페이징

요구 페이징은 프로그램을 실행할 때 필요한 부분만 일부분씩 메모리로 가져와 실행하는 기법을 말한다. 이 때 프로그램을 나누는 기준 크기를 페이지라고 부르며, 프로그램을 주 메모리(RAM)로 가져오는 과정을 스왑 인, 주 메모리에서 디스크로 저장하는 과정을 스왑 아웃이라고 한다. 이와 같이 실제 필요한 부분만 페이지로 가져옴으로써 시간 낭비와 메모리 공간의 낭비를 줄일 수 있다 [5].

주 메모리에 찾는 페이지가 존재하는지 판단하기 위해서 유효-무효 비트 기법이 사용되게 된다. 페이지 테이블에서 찾는 페이지가 유효 하다는 것은 해당 페이지가 메모리에 적재되어 있다는 것을 의미하고, 무효하다는 것은 해당 페이지가 존재하지 않거나 디스크에서 주 메모리로 적재되지 않았음을 의미한다.

이와 같이 제한된 페이지 테이블 공간에서 페이지를 관리하기 위해서 페이지 교환 전략을 사용하게 된다. 기존의 교환 전략에는 페이지의 적재 시점을 기준으로 가장 오래된 페이지를 교환하는 FIFO(First-In-First-Out), 사용 시점을 기준으로 가장 오랫동안 사용하지 않은 페이지를 교환하는 LRU(Least-Recently Used), 사용 빈도에 의해 페이지를 교환하는 MFU(Most Frequently Used), LFU(Least Frequently Used) 등이 있다.

5. eDPL의 요구 페이징

eDPL은 최근 저자가 개발한 운영체제가 없는 독립형 프로그램을 실행하기 위한 NAND 플래시 요구 페이징 로더이다.

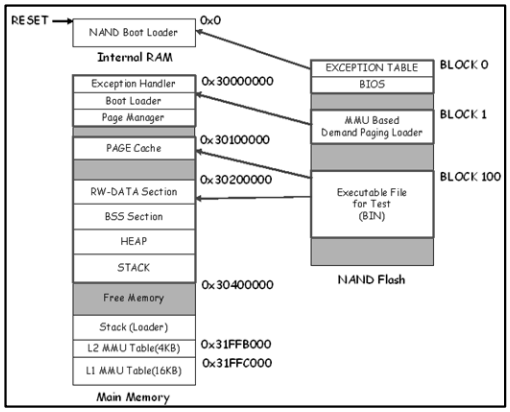

그림 1. eDPL의 동작과 메모리 구조
Fig. 1. Operation and memory organization of eDPL

운영체제 기반으로 동작하는 요구 페이징 기법을 대신 하는 eDPL은 그림1과 같은 메모리 구조를 이용하여 동작한다. NAND 플래시 블록 0번에 4KB의 NAND 부트 로더가 적재되며 eDPL은 NAND 플래시 1번 블럭에 최대 1MB 영역에 적재된다. eDPL이 사용하게 될 MMU 변환 데이블과 예외 처리를 위한 스택은 메모리의 하부에 위치한다. 응용 프로그램은 페이지 크기 단위로 분할되어 'PAGE Cache' 영역에 적재하게 된다. 페이지의 크기는 1KB와 4KB의 크기를 선택할 수 있도록 설계되어 있고, 각각 2~16개의 페이지 묶음을 선택할 수 있도록 하여 다양한 페이지 운영 전략에 따른 성능을 측정할 수 있도록 하였다 [4][11].

기존에 제안된 eDPL의 페이지 전환 전략은 순차 배정으로서 FIFO를 이용하여 제거할 페이지를 선정한다. 이 동작은 그림 2와 같다.

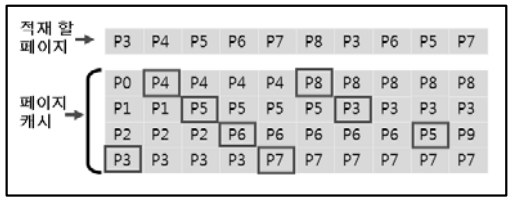

그림 2. FIFO 기반 페이지 교환 기법
Fig. 2. FIFO based page replacement method

초기에 페이지 캐시에 0, 1, 2, 3 페이지가 적재된 상태에서 4번 페이지가 적재 요구되면 가장 오래전에 적재된 0번 페이지를 제거하고 4번 페이지가 적재된다. 다시 5번 페이지가 적재 요구되면 1번 페이지가 제거된다.

따라서 본 연구에서는 이 eDPL의 요구 페이징 성능을 더 개선하는 페이지 전환 알고리즘을 제안하고 이의 성능 평가를 수행하였다.

II. 개선된 요구 페이징 기법

본 장에서는 기존 논문에서 연구한 eDPL의 페이지 전환 전략을 향상시키기 위해 개선된 페이지 전환 기법들을 제안한다.

1. 함수 복귀 예측 페이지 잔류

ARM 프로세서에서는 함수가 호출되면 함수의 복귀 주소를 LR 레지스터에 저장한다. 만약 그림 3와 같이 함수 A가 0x1000번지에서 0x2000번지의 함수 B를 호출하면 LR 레지스터는 함수 A의 복귀 주소인 0x1004가 저장된다.

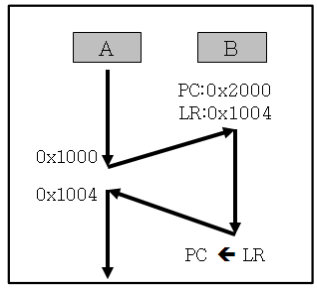

그림 3. ARM 프로세서의 함수 호출
Fig. 3. Function call of ARM processor

그러므로 함수 B는 함수 A로 복귀하기 위하여 LR을 PC에 대입하는 것만으로도 호출한 함수로 복귀하게 된다.

이 경우에 만약 함수 A와 함수 B가 다른 페이지에 존재한다면 초기에 함수 A를 수행할 때 함수 B는 페이지 캐시에 적재되지 않은 상태가 된다. 그러므로 B를 호출하는 순간 페이지 부재 오류로 인한 예외가 발생하게 되고, 함수 B가 속한 페이지를 페이지 캐시로 적재한다. 그런데 만약 페이지 교환 전략에 의해 교체될 페이지가 함수 A가 포함된 페이지라면 함수 B가 실행을 마치고 원래 함수로 복

귀할 때 역시 페이지 부재 오류가 발생하게 된다. 함수 A가 자주 불리는 함수라면 실행 될 때마다 페이지 부재 오류가 발생할 수 있다.

이러한 문제를 개선하기 위해 본 연구에서는 교체할 페이지를 계산할 때 현재 페이지 부재 오류가 발생된 페이지에서 복귀할 페이지는 페이지 캐시에서 제거되지 않도록 하는 전략을 제안하였다 [12].

ARM에서는 페이지 부재 예외가 발생하면 ABORT 모드로 진입하게 되며, 이 때 이전 모드(응용 프로그램이 구동되던 SYSTEM 모드)의 LR 레지스터를 통해 현재 호출된 함수에서 복귀할 함수의 페이지 위치를 확인할 수 있다 [1][2]. 따라서 본 논문에서는 페이지 부재 오류가 발생하면 예외 핸들러에서 원함수의 복귀 주소를 매개변수로 하여 페이지 관리기를 실행한다. 페이지 관리기의 동작은 그림 4에 순서도로 표현하였다. 함수에서 복귀할 페이지의 주소(R), 적재 요구페이지 주소(L) 그리고 제거 대상 페이지(D)를 선정한다. D는 Victim Pointer 변수에 의해 가리키고 있다. 이 때, 제거될 페이지(D)와 복귀주소를 포함하는 페이지(R)가 동일하면 선택된 제거 대상 페이지를 제거하지 않고 D를 '1' 증가하여 다음 페이지를 제거하도록 하였다.

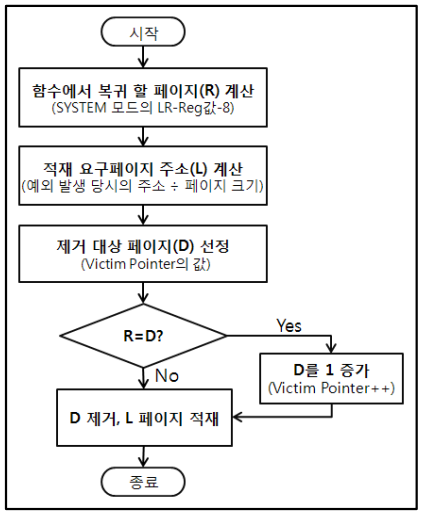

그림 4. 복귀 예측 알고리즘 순서도

Fig. 4. Return prediction algorithm flowchart

2. 잦은 전환 요구 페이지 잔류

기존 운영체제에서 사용하고 있는 페이지 전환 전략은 멀티 프로세싱 환경을 기반으로 하고 있기 때문에 프로세스의 전환이 발생할 때 최근에 참조한 페이지를 확인 할 수 있다. 하지만 본 논문에서 사용하는 단일 프로세스 기반의 요구 페이징 로더는 프로세스의 전환이 발생하지 않는다. 따라서 운영체제 기반에서 사용할 수 있는 LRU나 LFU와 같은 페이지 전환 전략은 적용 할 수 없다 [8].

이 절에서 제안하고자 하는 잦은 전환 요구 페이지 잔류 방식은 페이지 수만큼의 이력 버퍼를 두어서 최근에 적재가 일어난 페이지의 번호를 기록하여 적재가 많이 발생한 페이지는 잔류시키는 방식이다. 본 논문에서는 이 방식을 MFU-R(Most Frequently Used Page Remain)이라 부를 것이다.

이 방식을 구현하기 위해서 사용된 이력 버퍼는 환형 버퍼로 적용하여 버퍼 전체에서 가장 최근에 전환 되었던 페이지들의 페이지 번호가 저장된다. 페이지 관리기는 이력 버퍼를 이용하여 가장 빈도가 높게 전환이 발생한 페이지를 찾아내고 순차 배정 방식에 따라 교체할 페이지를 선택하게 된다. 선정하려는 페이지의 발생 빈도가 높을 경우 해당 페이지는 잔류시키고 그 다음 순차에 의한 페이지를 제거시킨다 [12].

앞에서 설명한 내용을 토대로 그림으로 나타낸 페이지 교환 기법은 그림 5와 같다. 10번째 주기에서 P3을 적재하려고 할 때 FIFO 동작에서는 P0을 제거하고 P3을 적재한다. 하지만 MFU-R에서는 이력 버퍼 내에서 P0의 적재 요구 횟수가 2회로 가장 많기 때문에 P0을 제거하지 않고 P6을 제거한다. 따라서 적재 요구가 많았던 P0은 페이지 캐시에 남아 있게 된다.

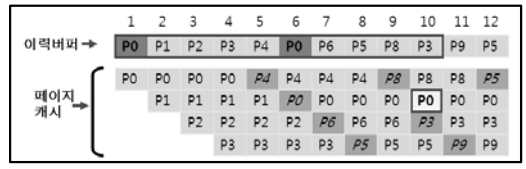

그림 5. MFU-R 기반 페이지 교환 기법

Fig. 5. MFU-R based page replacement method

이러한 전략을 사용하면 집중적인 연산 수행 시 특정 페이지에서 잦은 교체가 발생하는 경우에 페이지 잔류 확률을 증가시켜 페이지 전환 횟수를 절감시킬 수 있다. 반면에 활용이 완료된 페이지가 이전까지 전환 횟수가 많다는 이유로 캐시에서 계속 잔류할 경우 페이지 낭비가 발생할 수도 있다. 따라서 이 방식은 실행하는 응용 프로그램의 종류에 성능 향상 정도가 다른 부분이다.

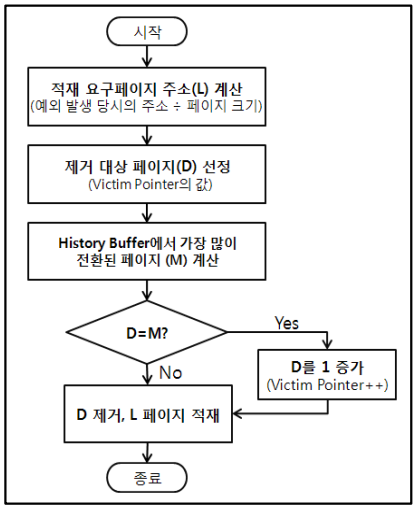

그림 6. MFU-R 알고리즘 순서도
Fig. 6. MFU-R algorithm flowchart

MFU-R 알고리즘의 동작을 순서도로 나타낸 것은 그림 6와 같다. 이력 버퍼를 통해 최근에 가장 많이 적재 요구 되어진 페이지(M)를 계산한다. 그리고 제거 대상 페이지(D)와 값을 비교하여 두 값이 같으면 D를 증가시켜서 M은 제거하지 않는다.

Ⅲ. 성능 평가

1. 실험 환경

MMU를 갖는 제품군 중에서 ARM v4T 아키텍쳐를 갖는 ARM920T 코어를 사용한 삼성전자의 S3C2440 프로세서를 이용하였다. 이 프로세서는 네비게이션, DMB 및 기타 모바일 시스템에 널리 사용되고 있으며, 테스트 환경을 위해서 (주)윌텍에서 개발한 ARM 교육용 보드인 GBOX를 사용하였다. 이 개발 교육용 보드는 32MB 메모리와 128MB NAND 플래시를 사용하고 있다.

본 논문에서 사용하는 모든 프로그램은 ADS v1.2 컴파일러를 기반으로 설계되었으며, 실제 동작하는 응용 프로그램은 NAND에 기록된다. 응용 프로그램의 동작 과정 중에는 필요에 따라 MicroSD Card를 통해 파일을 읽거나 기록하는 동작을 수행하기도 한다.

실험을 위한 응용 프로그램으로 Deflate Algorithm 압축 연산이 적용되는 ZIP 압축 프로그램, JPEG 라이브러리를 이용한 이미지 압축 프로그램 그리고 이미지를 LCD에 연속해서 출력하는 프로그램을 사용하였다. 사용한 프로그램에 대한 간략한 정보는 아래 표 1과 같다.

표 1. eDPL 성능 평가를 위한 응용 프로그램 정보
Table 1. Application program information for eDPL benchmark

	ZipUnZip	JPEG	Viewer
Bin 크기 (Byte)	91,604	91,520	85,320
RAM 사용 (Byte)	92,844	1,090,692	743,936
수행 시간 (msec)	3059	708	2133

표 1에서 Bin 크기는 컴파일된 응용 프로그램의 실제 크기를 말하며, RAM 사용은 프로그램이 메모리로 적재되어 ZI영역까지 초기화되었을 때의 크기를 말한다. 그리고 수행 시간은 요구 페이징 기능을 적용하지 않았을 때 프로그램이 수행되는데 걸린 시간이다. 요구 페이징 기법이 적용되면 페이지 전환이 발생하기 때문에 수행 시간이 증가하는 것을 볼 수 있다.

2. 실험 결과 및 분석

페이지의 크기는 모든 실험에서 1KB 고정된 값을 갖는다. 이는 동작시키는 프로그램의 크기를 고려한 것이고, 4KB를 이용할 경우 캐시의 크기가 실행하려는 프로그램 보다 커질 수도 있기 때문이다. 따라서 eDPL에서 사용하는 캐시의 전체크기는 페이지 수에 1KB를 곱한 값과 같다.

2.1 함수 복귀 예측 페이지 잔류

요구 페이징 알고리즘을 함수 복귀 예측 페이지 잔류형태로 수정된 상황에서 페이지 수의 변화에 따른 성능을 측정하였다. 측정한 결과 값은 다음 장의 표 2와 같다. 표에서 PA-전환은 현재 적재되어 있지 않은 페이지에 포함된 명령어로 분기를 할 때 Prefetch Abort 예외가 발생하여 페이지의 전환이 일어난 횟수를 의미한다. DA-전환은 프로그램을 실행하는 동안에 적재되어 있지 않은 데이터의 참조가 발생할 때 Data Abort 예외에 의한 페이지의 전환 횟수의 총합을 의미한다. ABORT가 발생할 때 마다 페이지의 전환이 일어나므로 두 값의 합으로 전체 페이지 전환 횟수를 계산할 수 있다.

수행 시간 지연율은 요구 페이징 기능을 전혀 사용하지 않았을 때를 기준으로 수행 시간의 증가량을 계산한 값이다. 페이지의 수가 12개를 넘어가는 시점에서 수행 시간의 지연이 1% 미만으로 낮아지는 것을 알 수 있다.

표 2. ZipUnZip에서 복귀 예측 알고리즘에 따른 성능 결과

Table 2. Demand paging performance result for return prediction algorithm in ZipUnZip

Page 수	4	6	8	10	12	14	16
PA-전환 횟수(회)	13277	6957	2700	569	416	353	248
DA-전환 횟수(회)	4354	1895	621	100	78	49	39
수행 시간 (msec)	4920	3965	3384	3254	3089	3087	3063
수행 시간 지연율(%)	60.8	29.6	10.6	6.3	0.9	0.9	0.1

제안하고자 하는 함수 복귀 예측 페이지 잔류 방식의 성능 평가를 위하여 단순히 순차 배정 방식을 사용하는 요구 페이징과 성능 비교 그래프는 그림 6과 같다. 그림 내에서 꺾은선 그래프는 동작 시간의 변화를 나타내고, 막대그래프는 페이지 전환 횟수를 나타낸다. 그리고 동작 시간 3초를 기준으로 있는 1점 파선은 요구페이징을 사용하지 않았을 때 동작시간이다.

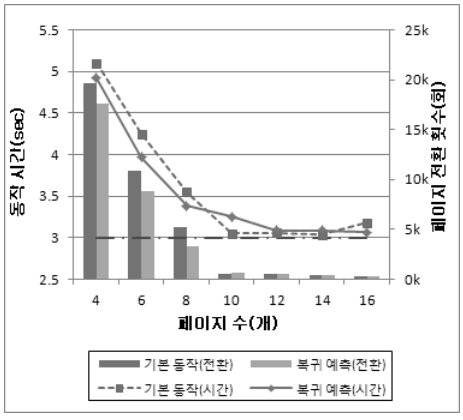

그림 7. ZipUnZip에서 복귀 예측 알고리즘에 따른 성능 비교

Fig. 7. Performance comparison for return prediction algorithm in ZipUnZip

그림 7에서 알 수 있듯이 페이지의 수가 10개 이하일 때는 복귀 예측 페이지를 잔류시키는 알고리즘의 성능이 좋다는 것을 알 수 있다. 페이지의 전환 횟수도 감소하고, 동작시간에 있어서도 더 빨라지는 것을 알 수 있다.

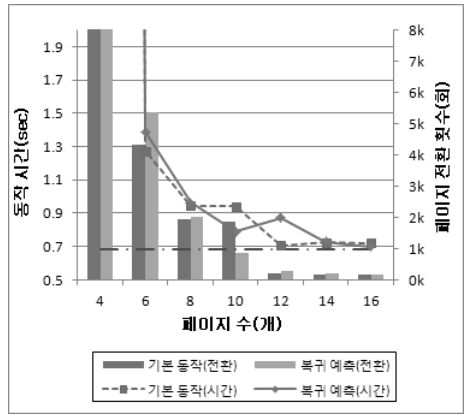

그림 8. JPEG Encoder에서 복귀 예측 알고리즘에 따른 성능 비교

Fig. 8. Performance comparison for return prediction algorithm in JPEG encoder

위와 동일한 실험을 JPEG Encoder에 대해서 수행한 후 성능 비교를 위한 그래프는 그림 8이다. 복귀 예측에 의해 잔류되는 페이지의 영향으로 페이지 수가 작을 때는 오히려 속도가 저하되는 모습을 보이고 있다. 이는 ZipUnZip으로 실험했을 때와 다른 양상을 갖는다. 그 이유는 JPEG의 함수의 크기가 커서 특정 페이지를 잔류시킨 상태에서 다른 페이지들의 교환이 많이 일어나기 때문이다. 이러한 결과는 프로그램의 크기에 따라 함수 복귀 예측 방식에 역효과를 가져올 수도 있다는 것을 의미한다.

다음 장에 있는 그림 9는 LCD Viewer에서 함수 복귀 예측 알고리즘의 성능을 기본 동작(순차 배정 방식)과 비교한 그래프이다. 페이지의 수가 14개보다 적을 때는 함수 복귀 예측 알고리즘의 성능보다 기본 동작의 성능이 좋았다. 하지만 페이지의 수가 14개 이상으로 커지는 구간에서는 성능의 차이가 작아지다가 페이지수가 16개가 되었을 때는 복귀 예측 알고리즘을 적용하였을 때 성능이 더 좋았다. 이는 LCD Viewer의 경우 호출하는 함수의 크기가 커서 복귀할 주소의 페이지를 잔류시키는 것 보다 여유 페이지를 크게 두는 것이 효율이 좋다는 것을 의미한다.

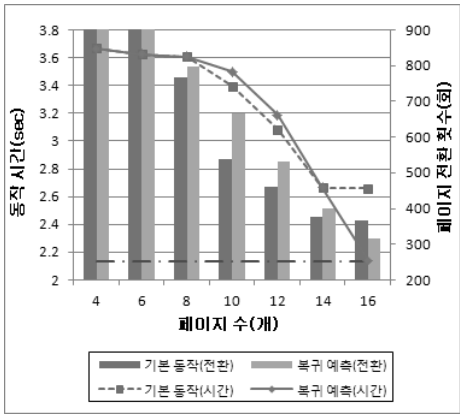

그림 9. LCD Viewer에서 복귀 예측 알고리즘에 따른 성능 비교
Fig. 9. Performance comparison for return prediction algorithm in LCD viewer

2.2 잦은 전환 요구 페이지 잔류

잦은 전환 요구가 발생하는 페이지를 잔류시키는 MFU-R알고리즘을 이용하여 ZipUnZip에 대해서 실험을 수행한 결과는 표 3과 같다. 페이지의 수가 12개 이상일 때의 결과는 요구 페이징을 사용하지 않을 때와 비교하여 지연 시간이 약 1% 차이임을 알 수 있다.

표 3. ZipUnZip에서 MFU-R 알고리즘에 따른 성능 결과
Table 3. Demand paging performance result for MFU-R algorithm in ZipUnZIP

Page 수	4	6	8	10	12	14	16
PA-전환 횟수(회)	16447	9051	3515	574	506	324	259
DA-전환 횟수(회)	3670	1738	651	84	81	47	33
수행 시간 (msec)	5328	4176	3462	3144	3099	3084	3212
수행 시간 지연율(%)	74.0	63.4	13.1	2.7	1.2	0.7	4.9

페이지의 수가 8개 이하일 경우는 잦은 전환 페이지를 잔류하는 알고리즘에서 지연 시간이 많이 증가하였다. 이는 앞에서 설명한 복귀 예측 알고리즘에 비해 성능이 낮은 결과이다. 단순히 전환 횟수만을 볼 때는 페이지 수가 14개 이상일 때는 MFU-R 알고리즘의 전환횟수가 더 적은 값을 갖는다. 즉, MFU-R 알고리즘의 경우 페이지의 수가 작은 상황에서 특정 시점에 잦은 전환이 일어났던 페이지가 다시 사용되지 않고 페이지 캐시에 보관하고 있기 때문에 효율이 낮다고 볼 수 있다.

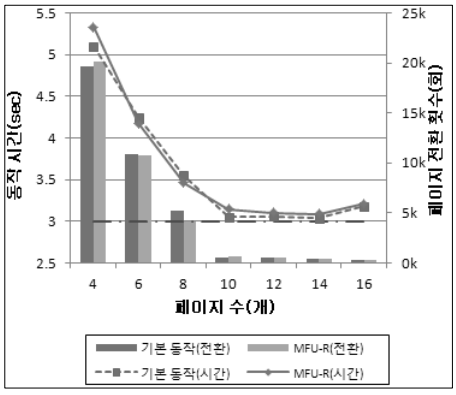

그림 10. ZipUnZip에서 MFU-R 알고리즘에 따른 성능 비교
Fig. 10. Performance comparison for MFU-R algorithm in ZipUnZip

그림 10은 FIFO 방식의 요구 페이징과 MFU-R 요구 페이징의 성능을 비교한 그래프이다. 페이지의 수가 6~8개 일 때 약간의 성능 향상이 있는 것을 볼 수 있다. 특히 페이지의 수가 8일 때 페이지 전환 횟수가 20% 정도 감소하였다. 최근 적재된 페이지의 번호를 담고 있는 이력 버퍼의 크기는 페이지의 수에 의해 결정되는데, ZipUnZip의 경우 최근 적재한 8개의 페이지 범위 내에서 잦은 요구가 이루어진다고 볼 수 있다.

MFU-R 알고리즘을 이용한 다른 두 실험에서도 순차 배정 요구 페이징 알고리즘에 비해 뚜렷한 성능 향상을 보기 어려웠다. 이러한 결과의 이유는 테스트로 사용한 프로그램들이 특정 시점에 참조가 여러 번 발생한 페이지가 다시 호출되는 경우가 적었기 때문이다. 이는 페이지 캐시에 참조되지 않을 페이지를 적재 고정하기 때문으로 페이지 크기가 작아지는 결과를 가져온다.

JPEG Encoder의 경우 페이지의 수가 8~10개일 때 MFU-R의 성능이 다른 경우에 비해 높다는 사실과, LCD Viewer과 같은 경우는 페이지의 수가 많을수록 성능이 높다는 점이다. 즉, 인코딩과 같이 행렬에 대해서 특정 연산을 반복수행하는 프로그램은 일정수준 이상의 페이지 수를 가져야 MFU-R 알고리즘이 유효하다는 것을 보여준다. 반면에 LCD Viewer와 같은 경우 그림파일을 불러와서 화면에 보

여주는 작업을 반복하기 때문에 캐시의 크기가 클수록 효율이 좋다고 말할 수 있다. 이러한 결과를 그래프로 나타낸 것은 아래와 같다. 그림 11은 JPEG Encoder의 MFU-R 성능을 FIFO 방식과 비교한 것이고, 그림 12는 LCD Viewer에 대해서 실험을 진행한 결과 그래프이다.

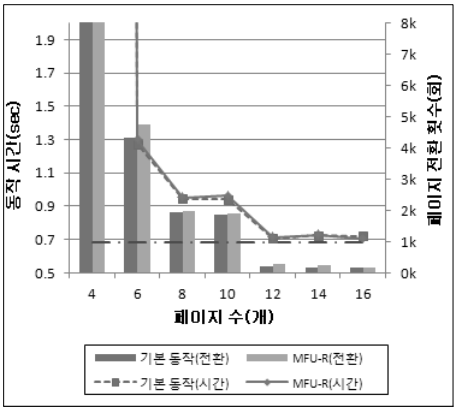

그림 11. JPEG Encoder에서 MFU-R 알고리즘에 따른 성능 비교
Fig. 11. Performance comparison for MFU-R algorithm in JPEG encoder

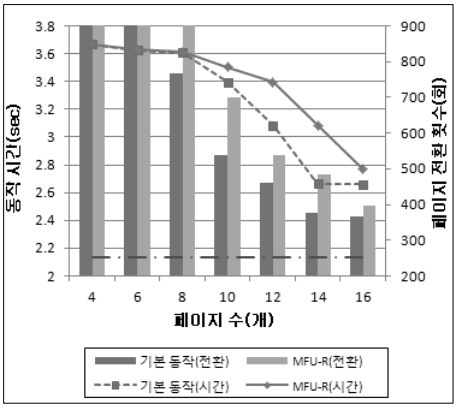

그림 12. LCD Viewer에서 MFU-R 알고리즘에 따른 성능 비교
Fig. 12. Performance comparison for MFU-R algorithm in LCD viewer

IV. 결 론

본 논문에서는 NAND 기반의 부팅 방식을 지원하는 운영체제가 없는 임베디드 시스템 환경에서의 RAM 사용량을 절감하기 위한 요구 페이징 기법의 개선된 알고리즘을 제안하였다. 이를 위해서 임베디드 기반의 요구 페이징 로더인 eDPL을 사용하였고, 성능 검증을 위하여 세 가지 응용 프로그램을 통해 결과를 도출하였다.

요구 페이징 성능 향상을 위해 두 가지 알고리즘을 제안하였다. 첫 번째로 제안한 함수 복귀 예측 페이지 잔류 알고리즘은 하나의 함수에서 다른 함수를 호출한 뒤 돌아오는 경우를 예측하여 페이지를 잔류시키는 알고리즘이었다. 이 방식의 경우 ZipUnZip 프로그램과 같은 압축연산에서 적은 페이지를 사용하는 상황에서 효율이 좋았다. 반면 LCD Viewer와 같은 이미지를 출력하는 프로그램의 경우 페이지의 수를 증가시키는 것이 효율을 높게 하였다.

두 번째로 제안한 MFU-R 알고리즘은 잦은 전환이 발생하는 페이지는 캐시에 남겨두고 다른 페이지들을 교체하는 방식이다. 실험 환경에서 사용한 프로그램의 크기가 작음에도 불구하고 캐시의 크기도 작았기 때문에 잦은 전환 페이지를 캐시에 남겨두는 방식은 효율이 낮은 결과를 가져왔다. 캐시의 크기와 수를 증가시키고 특정 함수를 자주 호출하는 프로그램에서는 보다 향상된 성능을 보여준다.

이처럼 사용하는 응용 프로그램의 크기나 연산의 특성에 따라서 요구 페이징의 성능 개선 정도는 차이를 갖고 있다. 향후에는 응용 프로그램의 특성 분석을 통해 적절한 요구 페이징 알고리즘을 적용할 수 있는 eDPL을 설계할 필요가 있다. 그러면 요구 페이징 기능이 갖는 RAM 절감 효과는 유지하면서도 성능의 저하를 최소화 할 수 있을 것으로 보인다.

참고문헌

[1] ARM Limited, ARM DUI 0056D: ARM Developer Suite Developer Guide, November 2001.
[2] ARM Limited, ARM DDI 0151C: ARM920T Technical Reference Manual, April 2001.
[3] C. Park et al., "A low-cost memory architecture with NAND XIP for mobile embedded systems", In CODES+ISSS, 2003, pp.138-143, 2003.

[4] C. Park et al., "Compiler-assisted demand paging for embedded systems with flash memory", EMSOFT'04, pp.114-124, Sep. 2004.
[5] J. L. Smith, "Multiprogramming under a page on demand strategy", Comm. ACM, Vol.10, pp. 636-646, Oct. 1967.
[6] K. Kaspersky, Code Optimization: Effective memory usage, A-I.ist Publishing, 2003.
[7] L.A. Belady, "A study of replacement algorithms for a virtual storage computer", IBM Systems Journal, Vol.5, Issue.2, pp. 78-101, June. 1996.
[8] Rik. van Riel. "Page replacement in Linux 2.4 memory management", USENIX Annual Technical Conference, FREENIX Track, pp. 165-172, 2001.
[9] Samsung Electronics Co., NAND Flash Memory and SmartMedia Data Book, 2002.
[10] Y.S. Joo et al., "Energy and performance optimization of demand paging with oneNAND flash", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.27, No.11, pp. 1969-1982, Nov. 2008.
[11] 전현재, 전창규, 류경식, 김용득, "ARM 프로세서에서 메모리 절감을 위한 요구페이징 기법의 구현", 대한임베디드공학회 추계학술대회, pp. 47-50, 2010.
[12] 전창규, 전현재, 류경식, 김용득, "ARM 기반 시스템에서 요구 페이징 기법의 성능 향상을 위한 알고리즘 연구", 대한임베디드공학회 추계학술대회, pp. 51-54, 2010.

저 자 소 개

류 경 식

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

전 창 규

2010년 : 아주대학교 전자공학부 학사.
현재, 아주대학교 대학원 전자공학부 석사과정.
관심분야 : 임베디드 시스템, 네트워크.
Email : jchangkyu@ajou.ac.kr

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