가상 메모리 기반의 실시간 임베디드 시스템의 페이지 교체 정책에 대한 연구

김종찬[O], 이창건[*], 하은용
서울대학교 컴퓨터공학부, 안양대학교 컴퓨터공학과

Page Replacement Policy for Virtual-memory based Real-time Embedded Systems

Jong-Chan Kim[O], Chang-Gun Lee[*], Eun-Yong Ha
{jongchank, cglee}@snu.ac.kr, eyha@anyang.ac.kr
The School of Computer Science and Engineering, Seoul National University
Dept. of Computer Science and Engineering, Anyang University

요 약

실시간 요건을 필요로 하는 임베디드 시스템의 경우 예측가능성(predictability)이 매우 중요하다. 그렇기 때문에 이러한 시스템들은 가상 메모리를 사용하지 않는 단순한 실시간 운영체제(RTOS) 를 사용하는 경우가 일반적이다. 하지만, 임베디드 시스템에 요구되는 기능 요건들이 복잡해짐에 따라 Linux와 같은 가상 메모리 기반의 범용 운영체제를 채택하는 경우가 늘고 있으며, 이런 경향은 앞으로 더욱 심해질 전망이다. 가상메모리 시스템은 필요한 메모리 사용량을 줄일 수 있을 뿐만 아니라 응용 프로그램 개발과 디버깅을 용이하게 하기 때문에 기존의 복잡하고 어려운 실시간 운영체제의 개발환경을 사용하는 경우에 비해 높은 개발 생산성을 기대할 수 있다. 하지만, 가상 메모리 시스템의 요구 페이징 기법은 시스템의 예측가능성을 떨어뜨리기 때문에 일반적으로 실시간 요건을 필요로 하는 시스템에 적용되지 못하고 있다. 본 논문은 요구 페이징 기법의 사용을 전제로 한 임베디드 시스템의 실시간 요건을 만족시키기 위한 페이지 교체 기법을 제안한다.

1. 서 론

전통적으로 임베디드 시스템 분야와 범용 시스템 분야를 구분하는 가장 중요한 기준은 시스템 구성에 필요한 컴퓨팅 파워, 실시간성의 필요 여부이다. 임베디드 시스템은 부피를 작게 차지해야 하기 때문에 작고 그에 따라 컴퓨팅 파워가 낮은 프로세서를 사용하고, 실시간성이 요구된다. 그리고 전력 소모량이 적어야 하기 때문에 일반적인 고성능 범용 프로세서를 사용할 수 없었다.

최근의 하드웨어, 소프트웨어의 기술 진보는 이러한 패러다임을 바꾸고 있다. 컴퓨팅 파워가 높은 저전력 임베디드 프로세서가 개발되고[1], 이에 Windows CE, Embedded Linux 와 같은 범용 운영체제가 탑재되는 경우가 늘어나고 있다[2]. 높은 컴퓨팅 파워를 가진 프로세서에 가상 메모리 기능을 갖추고 있는 범용 운영체제가 탑재될 경우 응용 프로그램 개발이 쉬워지고 그에 따라 임베디드 시스템 개발자가 되기 위한 초기 진입 장벽이 낮아져서 임베디드 SW 산업 전체의 규모가 커지게 된다.

하지만, 미리 정해지지 않은 불특정 다수의 응용 프로그램을 수행하는데 적합하도록 설계된 가상 메모리 시스템의 경우 임베디드 시스템의 중요한 기능 요건인 실시간성을 만족시키기 어렵다. 이는 가상 메모리 시스템이 요구 페이징 기법을 사용하기 때문이다. 요구 페이징 기법에서는 프로그램의 코드, 데이터 페이지들이 실제로 처음 접근되는 시점에 운영체제의 페이지 폴트 처리기에 의해서 메모리에 적재된다. 이 페이지들은 이후 불특정 시점에 운영체제의 페이지 교체 정책에 의해서 메모리에서 삭제된다. 만약 이 페이지가 다시 접근되면 운영체제는 다시 페이지 폴트 처리기를 사용해서 해당하는 페이지를 플래시 메모리와 같은

* : 교신 저자

보조 기억장치-(secondary storage)에서 메모리로 읽어 들이게 되고 그동안 프로그램 수행은 지연되게 된다. 불특정 다수의 응용 프로그램이 수행되는 범용 시스템에서 페이지 교체와 페이지 폴트의 패턴을 예측하는 것은 현실적으로 불가능하기 때문에 범용 시스템의 실시간성 보장은 현실적으로 어렵다.

임베디드 시스템은 범용 시스템과는 달리 시스템 설계 단계에서 응용 프로그램의 종류와 수가 고정되기 때문에 시스템의 메모리 접근 패턴을 예측할 수 있고 이를 이용하여 최적의 페이징 전략(Optimal Paging Strategy)을 제품 출시 이전에 결정할 수 있다.

본 논문에서는 요구 페이징 기법 하에서 실시간성을 보장하기 위한 기존의 범용 운영체제의 접근 방법을 살펴보고 각 방식의 문제점들을 짚어본 뒤 고정된 응용 프로그램만을 수행하는 임베디드 시스템에 적합한 페이지 교체 정책을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 전통적인 페이지 교체 정책에 대해서 살펴보고, 3장에서는 범용 운영체제의 경우 실시간 보장을 위해 어떤 기법을 사용했는지 알아본다. 4장에서는 임베디드 시스템에 적합한 새로운 페이지 교체 정책을 제안하고, 5장에서는 결론을 내린다.

그림 1 산업용 기계에서 엔터테인먼트 기기까지 다양한 임베디드 시스템의 적용 분야

2. 전통적인 페이지 교체 정책

전통적인 페이지 교체 정책은 일정한 크기의 메모리에서 다양한 페이지 참조 패턴을 수행하여 페이지 폴트가 가장 적게 일어나는 알고리즘을 찾는 것을 목적으로 하였다. 그렇기 때문에 다양한 워크로드들에 공통적인 특성을 찾는 것이 중요했으며, 대표적인 알고리즘인 LRU(Least Recently Used)의 경우 대부분의 워크로드에 참조 지역성(Locality of Reference)이 있다는 사실을 이용하고 있다.

LRU 기법은 메모리에 상주하고 있는 페이지중 가장 오래 전에 참조된 페이지를 교체한다. 즉, 최근 접근한 시간만을 이용하는 것인데, 그렇기 때문에 LRU는 작업 집합(Working Set)의 변화에 잘 대응할 수 있다. 하지만, LRU는 지속적으로 자주 접근되는 중요한 페이지보다 최근 처음으로 접근된 페이지를 선호하기 때문에 정작 중요한 페이지가 교체되는 경우가 생기게 된다. 즉, 과거 정보의 활용도가 떨어진다. 순차 접근을 수행하는 워크로드의 경우 이런 부작용이 더욱 심하다.

LFU(Least Frequently Used) 기법은 페이지 참조 회수를 기반으로 참조 회수가 가장 작은 페이지를 교체한다. 하지만, 이 기법은 LRU와는 반대로 과거 정보에 많이 의지하기 때문에 과거에 자주 접근되었으나 현재는 접근되지 않는 페이지가 계속 메모리에 상주하게 되는 현상이 발생한다. 즉, 작업 집합의 변화에 대한 적응력이 떨어진다.

LRU, LFU 양쪽의 단점을 극복하기 위한 연구로는 LRU-K[5] 기법과 2Q[6] 기법이 있다. LRU-K 기법은 각 페이지 별로 마지막에서 K 번째 참조된 시간을 이용하여 페이지에 대한 선호도를 결정한다. 이 기법은 최근 접근 시간과 일정 기간 동안의 접근 빈도를 동시에 고려하기 때문에 LRU 기법보다 좋은 성능을 보인다.

2Q 기법은 LRU 기법의 순차 접근에 대한 단점을 극복하기 위해서 처음 접근된 페이지들은 임시 큐에 보관하고, 이 임시 큐 안에서 또다시 접근되는 페이지들만을 두 번째 큐로 옮겨서 관리하는 방식이다. 임시 큐는 FIFO 방식을 따르기 때문에 순차 접근되는 페이지들은 임시 큐에서 잠시 머물다가 FIFO 알고리즘에 의해서 바로 교체되게 된다.

LRU-K, 2Q 모두 디스크 블록 버퍼에 대한 교체 정책으로 연구되었으나 현재는 메모리 페이지에 대한 교체 정책으로도 쓰이고 있다. 리눅스 커널[7]의 경우 2Q 방식을 사용하고 있다.

3. 운영 체제 제공 기능

범용 운영 체제는 실시간성이 요구되는 프로세스가 페이지 폴트로 인한 지연을 겪지 않도록 하기 위해서 몇 가지 기능을 제공한다.

mmap() 시스템콜의 MAP_POPULATE 옵션을 이용하면 매핑되는 주소 공간의 모든 페이지들이 시스템콜 수행 시점에 물리 메모리에 매핑되게 된다. 이 방식은 영역 전체를 고정시키기 때문에 메모리 영역을 세세하게 제어할 수 없다는 단점이 있다. posix_madvise() 시스템콜의 POSIX_MADV_WILLNEED 옵션을 이용하면 메모리 영역을 세세하게 제어할 수 있으나 커널에 힌트를 주는 방식이기 때문에 경우에 따라서 커널이 힌트를 무시할 수도 있다. 실제로 많이 사용되는 방식은 mlockall() 시스템콜을 이용하는 방식인데 이 시스템콜을 호출하는 프로세스의 주소 공간은 페이지 교체 대상에서 제외된다. 이 방식은 또한

mmap() 방식과는 달리 프로그램의 코드 영역에도 쉽게 적용할 수 있다.

4. 임베디드 시스템의 페이지 교체 정책

4.1. 응용 프로그램들의 특성에 따른 분류

임베디드 시스템의 응용 프로그램들은 아래와 같이 특성에 따라 분류할 수 있다.

표 1 임베디드 시스템의 응용 프로그램들의 특성에 따른 분류

Case Number	Application	Characteristics
(1)	Periodic Task	Hard Real-time without Human Interaction
(2)	GUI Application	Soft Real-time withHuman interaction
(3)	Aperiodic Task	Soft Real-time without Human interaction
(4)	Others	No Real-time requirement

경성 실시간성(Hard Real-time)을 요구하는 응용 프로그램의 경우 사람과의 상호작용(Human Interaction)이 없으며 주기적인 작업(Periodic Task)이다(1). 대표적인 예로는 로봇의 모터를 제어하는 프로그램을 들 수 있다.

연성 실시간성(Soft Real-time)을 요구하는 응용 프로그램의 경우 사람과의 상호작용을 수반하는가, 수반하지 않는가에 따라 크게 두 가지로 구분할 수 있다. 전자(2)의 경우 동영상 플레이어와 같은 GUI 응용 프로그램이 대표적이고 후자(3)의 경우 일반적인 관리 목적의 응용 프로그램들이 해당한다.

(1), (2), (3) 어디에도 해당하지 않는 응용 프로그램은 실시간 요건이 없는 경우에 해당한다(4).

4.2. 각 분류별 페이지 교체 정책

임베디드 시스템의 경우 수행될 응용 프로그램들이 설계 시점에 고정되기 때문에 이 응용 프로그램들에 대한 메모리 참조 패턴을 미리 분석할 수 있다. 이를 위해서 OProfile[8] 과 같은 메모리 프로파일링 툴을 사용하거나 운영체제의 페이지 폴트 처리기를 수정하여 사용할 수 있다. 이렇게 분석된 응용 프로그램별 메모리 참조 패턴을 이용하여 4.1 의 4가지 분류에 의거해 각각 다음과 같은 페이지 교체 정책을 적용한다.

(1)에 해당하는 응용 프로그램은 페이지 폴트에 의한 지연을 겪지 않도록 해야 한다. 이를 위해 프로그램 구동시 전체 페이지를 모두 메모리에 고정시키고 나서 프로그램 수행을 시작하도록 한다. 이 경우 초기 구동 속도가 요구 페이징에 비해서 느려지게 되지만 일단 프로그램이 구동되면 페이지 폴트에 의한 지연을 겪지 않게 된다.

(2)에 해당하는 경우는 사람과의 상호작용을 수반하는 동영상 플레이어 등에 해당한다. 이 분류에 속하는 응용 프로그램들은 초기 구동속도가 중요하다. 초기 구동이 느릴 경우 사용자는 시스템의 응답 시간이 느리다고 인식하게 된다. 이러한 응용 프로그램들은 사용자의 선택에 의해서 초기 시작 코드 페이지를 시스템 부팅 시점에 메모리에 고정시키도록 한다. 초기 시작 코드 페이지 외의 페이지에 대해서는 (3)의 정책을 따른다.

(3)에 해당하는 경우는 요구 페이징과 메모리 고정 방식을 병행해서 사용한다. MFU(Most Frequently Used) 페이지를 메모리에 고정하여 페이지 폴트의 수를 줄인다.

(4)에 해당하는 경우는 (1), (2), (3) 에서 예약된 메모리 페이지를 제외한 나머지 메모리 영역을 이용하여 요구 페이징을 수행한다.

4.3. 제안된 페이지 교체 정책의 구현

4.2에서 제안된 페이징 정책을 구현하기 위해서는 프로그램 초기 구동 시점에 모든 페이지 혹은 일부 페이지를 메모리에 적재한 후 프로그램을 수행하는 기술이 필요하다. 리눅스 커널을 수정하여 이 기술을 구현하였다.

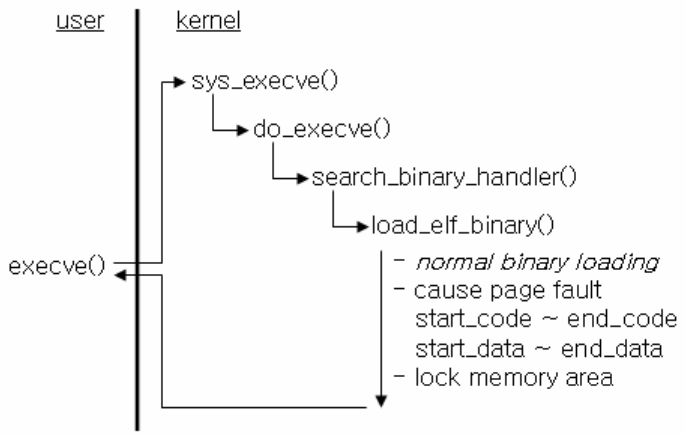

그림 2 리눅스를 이용한 초기 구동시 메모리 적재 기술의 구현

ELF 바이너리 프로그램을 실행하면 쉘 프로그램에서 execve() 시스템콜을 호출한다. 이 시스템콜은 커널 내부의 sys_execve() 함수를 호출하고 최종적으로 ELF 바이너리 로더인 load_elf_binary() 함수가 호출된다. 이 함수에서 일반적인 로딩 작업을 마친 후 가상 메모리 영역의 코드 영역과 데이터 영역에 대해서 강제로

페이지 폴트를 내고 해당 메모리 영역을 고정시킨다. 위와 같은 방법으로 필요한 메모리 영역을 시스템 구동시점에 메모리에 고정시킬 수 있다.

5. 결 론

본 논문은 가상 메모리 기반의 실시간 임베디드 시스템에서 실시간 요건을 만족시키기 위한 페이지 교체 정책을 제안하고 이를 구현하였다.

이를 위해 설계 단계에서 시스템에 탑재될 응용 프로그램의 종류가 결정되는 실시간 임베디드 시스템의 특징을 이용하여 응용 프로그램을 실시간 요건과 사람과의 상호작용 여부를 기준으로 4가지로 분류한 후 각각에 알맞은 페이지 교체 정책을 제안하였다.

그리고 이를 구현하기 위해 시스템 초기 시작 시점에 필요한 메모리 페이지를 미리 메모리에 고정시키는 기술을 리눅스 커널을 수정하여 구현, 테스트 하였다.

위와 같은 과정을 통해 임베디드 시스템의 실시간 요건과 체감 응답시간을 개선할 수 있음을 보였다.

컴퓨팅 시스템의 가장 중요한 두 가지 자원은 CPU와 메모리이다. 운영체제는 각 응용 프로그램들에 대해서 마치 각자가 CPU와 메모리를 독점적으로 사용하고 있는것과 같은 환상을 주기 위해서 프로세스 스케쥴링과 가상 메모리를 지원한다. 이 중에서 실시간성을 만족시키기 위한 스케쥴링에 대한 연구는 많이 진행되었으나 실시간성을 만족시키기 위한 메모리 예약과 페이지 교체 기법에 대한 연구는 많이 진행되지 않았다. 앞으로 가상 메모리를 사용하는 임베디드 시스템의 경우 실시간 요건 만족을 위한 메모리 관리 기법이 중요한 기술로 사용될 것이다.

참고 문헌

[1] G. Gerosa, “A Sub-1W to 2W Low-Power IA Processor for Mobile Internet Devices and Ultra-Mobile PCs in 45mm High-k Metal-Gate CMOS,” *Proceedings of the 2008 International Solid-State Circuits Conference*, 2008

[2] Snapshot of the embedded Linux market - April, 2007, http://www.linuxdevices.com/articles/AT7065740528.html, 2007

[3] Daniel P. Bovet, Marco Cesati, *Understanding the Linux Kernel*, O’Reilly, 2005.

[4] Mel Gorman, *Understanding The Linux Virtual Memory Manager*, Prentice Hall, 2004.

[5] E. J. O’Neil, P. E. O’Neil, and G. Weikum, “The LRU-K Page Replacement Algorithm for Database Disk Buffering,” *Proceedings of the 1993 ACM SIGMOD Conference*, pp. 297-306, 1993.

[6] T. Johnson, and D.Shasha, “2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm,” *Proceedings of the 20th International Conference on Very Large Data Bases*, pp. 439-450, 1994.

[7] The Linux Kernel Archives, http://www.kernel.org

[8] OProfile, http://oprofile.sourceforge.net

[9] Rebert Love, *Linux Kernel Development*, Sams Publishing, 2004.

[10] Nathan C. Burnett, John Bent, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, “Exploiting Gray-Box Knowledge of Buffer-Cache Management,” *Proceedings of the USENIX 2002 Annual Technical Conference Paper*.

[11] Dawson R. Engler, M.Frans Kaashoek, and James O’Toole, “Exokernel:An Operating System Architecture for Application-Level Resource Management,” *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pp. 251-266, December, 1995.

[12] H. Tokuda, T. Nakajima, and P. Rao, "Real-Time Mach: Towards a Predictable Real-Time System", *Proceedings of USENIX Mach Workshop*, October, 1990.

[13] S. F. Kaplan, “Compressed Caching and Modern Virtual Memory Simulation,” Ph.D. Thesis, University of Texas at Austin, December, 1999.