NAND 플래시 메모리 기반의 실시간 내장형 시스템을 위한 요구 페이징 및 피닝 혼합 기법

Demand paging and pinning techniques for NAND flash memory-based embedded real-time systems

하 건 수, 임 성 수, 김 지 홍*
Keonsoo Ha, Sung-Soo Lim, Jihong Kim

Abstract : Demand paging has not been used for real-time embedded systems due to its unpredictable page fault behavior. However, as applications for real-time embedded systems are complicated, demand paging is desirable to reduce the memory requirement. In this paper, we propose demand paging and pinning techniques suitable for real-time embedded systems. By pinning code segments with a large number of page faults, the proposed techniques can better bound the worst case execution times of the applications while keeping memory requirement lower. Our experimental results show that the proposed techniques reduce the main memory requirement up to 47% over the shadowing technique without violating the timing constraints.

Keywords : 내장형 시스템, 실시간 시스템, NAND 플래시 메모리, 요구 페이징

1. 서론

NAND 플래시 메모리가 보조 기억 장치로써 사용되는 환경에서 응용 프로그램의 코드를 실행하기 위한 방법으로 쉐도잉 기법과 요구 페이징 기법이 널리 쓰인다. 쉐도잉 기법은 시스템의 부팅 시간에 수행될 프로그램의 모든 코드를 메모리에 로딩하고, 요구 페이징 기법은 수행될 코드의 요구가 있을 때마다 해당 코드를 로딩해서 수행하는 기법이다.

최근 내장형 시스템용 응용프로그램의 코드 사이즈가 점차적으로 증가함에 따라, 많은 메모리를 요구하는 쉐도잉 기법은 사용하기 힘들어 질 것이다. 반면, 요구 페이징의 경우 메모리 요구량은 적지만, 예측 불가능한 페이지 폴트 때문에 실시간 시스템의 시간 제약이 있는 임베디드 시스템에 적용하기 어렵다.

본 논문에서는 적은 메모리를 쓰면서도 실시간 시스템의 시간 제약을 만족시킬 수 있는 요구 페이징과 피닝 기법을 함께 사용하는 기법을 제안한다. 피닝 기법이란 메모리의 특정 영역에 코드를 로딩한 후, 프로그램이 종료될 때까지 메모리에 상주시키는 기법을 말한다. 페이지 폴트를 많이 일으키는 코드에 대해서는 피닝을 적용하고, 나머지에 대해서는 요구 페이징 기법을 사용함으로써 메모리 사용량을 절약하면서도 실시간 시스템의 제약을 만족시킬 수 있다. 또한 요구 페이징 기법과 피닝 기법간의 최적의 설정을 찾는 휴리스틱 알고리즘을 제안한다. 실험 결과로부터 제안한 기법들이 실시간 시스템의 시간적 제약을 만족시키면서도 쉐도잉 기법만을 적용한 시스템 대비 최대 47% 까지 메모리를 절약할 수 있음을 확인했다.

* 교신 저자
논문접수 : 2007. 10. 11., 채택확정 : 2007. 10.19.
하건수 : 서울대학교 컴퓨터 공학부
임성수 : 국민대학교 컴퓨터 공학부
김지홍 : 서울대학교 컴퓨터 공학부
※ 이 연구를 위해 연구 장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.
이 논문은 2007 년도 두뇌한국21 사업과 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구입니다
(No. R0A-2007-000-20116-0).

II. 요구 페이징 기법과 피닝 기법의 설정

1. 관련 연구들

[1] 에서는 요구 페이징 기법을 사용하는 NAND 플래시 메모리 기반의 내장형 시스템을 위한 페이지 케쉬 관리 기법이 제안됐다. [2] 에서는 페이지 폴트 에 의한 지연 시간을 최소화하기 위해서 병렬화된 페이지 폴트 핸들러가 제안됐다. 이런 연구들이 각각 에너지 및 성능에 초점을 맞춘 반면, 본 논문에서는 실시간 시스템의 시간 제약 만족 및 메모리 사용량의 최소화를 목적으로 한다.

2. 요구 페이징 기법과 피닝 기법의 설정

본 논문에서는 요구 페이징 기법과 피닝 기법을 이용해서 적은 메모리로 실시간 시스템의 시간적 제약을 만족 시킬 수 있도록 하는 시스템 설정을 찾는 것을 목적으로 한다. 제안하는 기법의 개관은 그림 1 과 같다. 먼저 실시간 시스템 스케쥴 대상이 되는 태스크를 입력받은 후, 피닝 결정 알고리즘을 통해서 피닝 될 코드를 결정하고 피닝 정보 테이블에 저장한다. 메모리 할당 알고리즘에서는 이를 이용해서 태스크들이 스케쥴 될 때, 최소의 메모리로 실시간 시스템의 시간 제약을 만족 시킬 수 있는 요구 페이징과 피닝 간의 설정을 찾아낸다.

2.1 가정

모든 태스크는 실시간 시스템용 태스크이고, 쉐도잉 만을 적용했을 때 실시간 시스템의 시간 제약을 만족한다. 각 태스크는 일정한 양의 메모리를 할당받고, 메모리 영역은 요구 페이징을 위한 영역과 피닝을 위한 영역으로 나뉜다. 제안한 기법에 의해서 피닝될 코드로 정해진 영역은 부팅 시간에 피닝을 위한 메모리 영역에 로딩된다. 요구 페이징을 위한 영역은 태스크들과 공유된다. 또한 태스크들은 Rate monotonic 기법[3]에 의해서 스케쥴 된다.

2.2 피닝 결정 알고리즘

이 알고리즘에서는 입력된 각 태스크에 할당된 메모리에서 최악 실행 시간이 가장 짧은 피닝과 요구 페이징의 조합을 결정한다. 이를 위해 피닝의 양과 피닝 될 코드 영역을 변경하면서 최악 실행 시간을 평가하고, 주어진 메모리 크기에서의 가장 짧은 최악 실행 시간을 갖는 설정을 저장한다.

페이지 폴트를 고려한 최악 실행 시간을 구하는 기본 아이디어는 컨트롤 플로우 그래프에서 베이직 블록의 수행 시간들을 더하는 기법[4]에 페이지 폴트 비용을 더하는 것이다. 이 계산 기법은 본 논문의 범위를 넘어가는 주제이므로 향후 연구 주제로 남겨두고 얻을 수 있다고 가정한다.

이 알고리즘의 작동은 다음과 같다. 어떤 태스크를 쉐도잉하기 위해 M 개의 메모리가 필요하다고

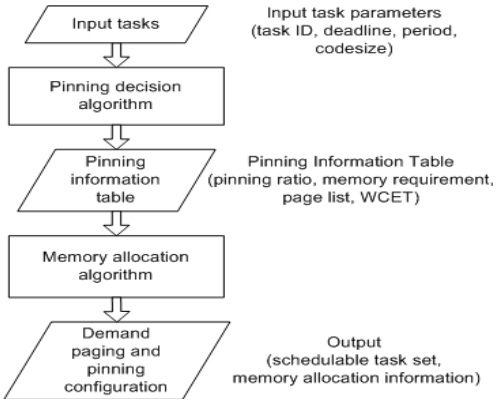

그림 1. 제안된 기법의 개관
Fig. 1. Overview of the proposed techniques

하면, (M-1) 개의 페이지만큼 그 태스크에 할당한다. 그리고 이 중 하나의 페이지만을 피닝을 위해 사용하고, 나머지 (M-2) 개의 페이지에 대해서는 요구 페이징을 사용한다. 하나의 피닝을 위한 공간에 어떤 코드 영역이 피닝되어야 하는지 결정하기 위해서 코드를 페이지 단위로 나눈 후, 한 페이지씩 피닝을 위한 영역에 로딩 하고, 최악 실행 시간을 구한다. 이렇게 하면 각 페이지가 피닝 되었을 때의 최악 실행시간을 알 수 있게 된다. 그리고 이중에 가장 최악 실행 시간을 크게 줄인 코드 영역을 피닝하기로 결정한다. 그리고 이번에는 피닝을 위한 메모리 영역을 한 페이지 더 늘리고, 요구 페이징을 위한 영역을 한 페이지 줄인 후, 추가로 피닝 할 코드 영역을 동일한 방식으로 찾아낸다. 이런 과정을 최악 실행 시간이 줄어들지 않을 때 까지 반복한다. 그리고 최종 최악 실행 시간이 데드라인보다 짧으면, 피닝 정보 테이블에 저장한다. 저장되는 내용은 (피닝이 적용되는 메모리 영역 비율, 메모리 요구량, 피닝 될 코드 영역 리스트, 최악 실행 시간) 등이 된다. 이 과정을 할당 메모리 크기를 줄이면서 실시간 시스템 제약을 만족시키지 못 할 때까지 반복하면, 각 메모리 크기별로 최소의 최악 실행 시간을 갖는 요구 페이징 기법과 피닝 기법 간의 설정이 테이블에 저장된다.

2.3 메모리 할당 알고리즘

메모리 할당 알고리즘은 피닝 정보 테이블을 이용해서 최소의 메모리로 여러 개의 태스크들이 실시간 시스템에서 스케쥴링 가능하도록 한다.

알고리즘은 먼저 현재 메모리 크기에서 태스크 집합 내의 각 태스크들의 최악 응답 시간이 실시간 시스템의 시간 제약을 만족시키는지 검사한다. 여기

서 현재 메모리 크기란 각 태스크들이 할당받은 메모리 크기의 합을 의미하는데, 초기값은 각 테스크의 피닝 정보 테이블에서 가장 나중에 저장된 메모리 크기의 합이다. 만약 모든 태스크들이 조건을 만족 시키면 추가 메모리 할당 없이 종료된다. 이것은 현재의 메모리로 실시간 시스템 시간 제약을 만족함을 의미한다. 만약 그렇지 않으면 메모리를 추가로 할당해서 최악 실행 시간을 낮춘다. 추가로 메모리를 계속 할당하다보면 쉐도잉하는데 필요한 메모리 크기만큼 할당한 메모리가 커질 수도 있는데, 이 경우는 제안한 기법으로는 메모리 요구량을 줄일 수 없음을 의미한다.

현재 메모리 상태에서의 스케쥴 가능 여부를 판단하기 위해서는 각 태스크들의 최악 반응 시간을 구할 수 있어야 한다. 제안된 기법에서는 요구 페이징 적용에 의한 컨텍스트 스위칭이 일어날 때 발생하는 페이지 폴트를 고려해야 한다. 이를 고려한 최악 반응 시간은 아래와 같다.

$$r_i^{k+1} = C_i + \sum_{j \in hp} \lceil \frac{r_i^k}{p_j} \rceil * (\mathrm{C_j} + \mathrm{MD_i}) \qquad (1)$$
$$r_i^0 = \sum_{j \in hp} C_j$$

위 식에서 hp 는 태스크 i 보다 우선 순위가 높은 태스크 번호의 집합을 의미하고, p_i, C_i 는 각각 테스크 i 의 주기, 실행 시간을 의미한다. 또한 MD_i 는 태스크 i 에 요구 페이징을 위해 할당된 메모리 영역의 크기를 의미한다. 즉, 컨텍스트 스위칭이 발생할 때마다 제거 되었던 낮은 우선 순위의 태스크 페이지들이 예전 상태로 복구되기 위해 발생하는 비용이다. r_i^k 를 k 가 0 일 때부터 계산하기 시작해서 r_i^k 의 값이 변하지 않을 때까지 재귀적으로 계산한 후, 그 값을 데드라인과 비교하게 된다.

만약 r_i^k 가 데드라인 보다 크면, 추가로 하나의 메모리 영역을 할당한다. 태스크의 주기와 피닝 정보 테이블의 메모리별 최악 실행 시간을 고려해서 추가 할당시 가장 이득이 큰 것을 고르게 된다. 아래 식의 결과가 가장 큰 태스크 i 에 추가로 메모리를 할당한다.

$$g_i = \frac{L(p_1,\ldots,p_n)}{p_i} * (C_i(M_c) - C_i(M_c+1)) \quad (2)$$

여기서 $L(p_1,\ldots,p_n)$ 는 모든 태스크 주기들의 최소 공배수를 의미하고, $C_i(M_c)$ 은 메모리 크기가 M_c 일 때의 최악 실행 시간을 의미한다. 즉, 메모리 페이지 하나를 추가로 할당했을 때, 최악 실행 시간 상의 이득과 발생 빈도를 곱해서 가장 큰 값을 가지는 태스크에 메모리를 할당하는 것이다.

Ⅲ. 실험 결과

제안한 기법은 대상 응용 프로그램의 특성에 효과가 의존한다. 다양한 특성을 가진 태스크들에 대해서 제안한 기법이 얼마나 메모리 요구량을 절약할 수 있는 지를 알아보기 위해서 여러 가지 특성을 가진 태스크들을 정의한 후, 이용률의 변동에 따른 메모리 요구량의 변화를 시뮬레이션 했다. 임의로 정의한 태스크 입력 파라미터는 표 1 과 같다. 먼저 10개의 태스크를 임의로 정의한 후, 피닝의 효과가 보통인 태스크, 큰 태스크, 작은 태스크 를 각각 TS1, TS2, TS3 나눴다.

이를 입력으로 해서 메모리 할당 알고리즘을 적용한 결과는 그림 2 와 같다. 그림의 X 축은 태스크 집합의 이용률을 의미하고 Y 축은 태스크 집합 내의 태스크들에 쉐도잉만을 적용했을 때의 메모리 요구량으로 정규화된 실시간 시스템의 시간 제약을 만족시킬 수 있는 각 태스크별 메모리의 요구량이다. 이용률이 커짐에 따라 제안한 기법은 실시간 시스템 시간 제약을 맞추기 위해서 페이지 폴트를 최대한 줄일 수 있도록 메모리를 추가 할당하게 되므로 메모리 사용량이 증가한다. 또한 태스크 집합을 구성하는 태스크들이 피닝에 의해 최악 실행 시간의 절감이 클수록, 메모리를 추가로 할당할 때, 이용률의 감소가 크므로, 적은 메모리로도 실시간 시스템의 시간 제약을 잘 만족 시킬 수 있다. TS2 의 경우 이용률이 가장 낮은 10% 일 때, 최대 47% 까지 메모리를 절약할 수 있었다.

표 **1.** 입력 태스크 파라미터

Table 1. Input task parameters

태스크 집합	태스크 ID	쉐도잉		제안기법
		WCET (ms)	메모리 (4KB)	메모리 (4KB)
TS1	a	2000	50	30
	b	600	50	45
	c	1000	30	21
	d	600	10	4
TS2	e	900	36	18
	d	600	10	4
	f	700	15	9
	g	300	12	7
TS3	h	55	30	28
	b	600	50	45
	i	50	20	18
	j	500	100	91

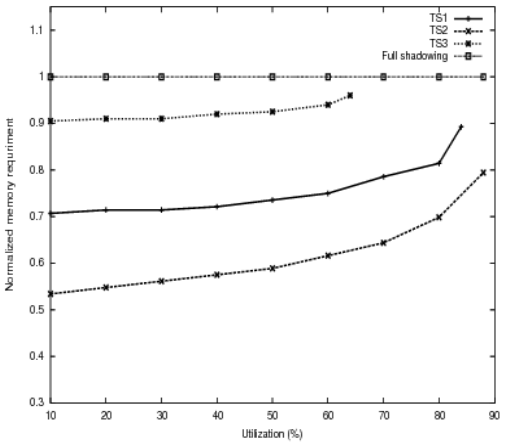

그림 2. 태스크들의 이용률 변화에 따른 정규화된 메모리 요구량

Fig. 2. Normalized memory requirements under varying utilizations

V. 결 론

이 논문에서는 적은 메모리를 사용하면서 실시간 시스템의 시간적 제약을 만족시킬 수 있는 요구 페이징 기법과 피닝 기법 간의 최적의 설정을 찾는 알고리즘을 제안했다. 해당 알고리즘을 통해서 허용할 수준의 시간 복잡도로 요구 페이징 기법과 피닝 기법간의 설정을 찾아 낼 수 있었다. 또한 실험 결과를 통해서 제안한 기법을 사용하는 것이 실시간 시스템 성격을 갖는 NAND플래시 메모리 기반의 내장형 시스템에서 메모리 사용량을 크게 줄일 수 있음을 확인했다.

향후에는 제안한 기법을 적용했을 때의 최악 실행 시간을 구하는 기법과 개선된 메모리 사용량 최적화를 위한 알고리즘 등에 관해서 연구할 예정이다.

참고문헌

[1] C. Park, J.-U. Kang, S.-Y. Park and J.-S. Kim, "Energy-aware demand paging on NAND flash-based embedded storages," Proc.of the international symposium on low power electronics and design (ISLPED), 2004.

[2] J.In, I. Shin and H. Kim, "SWL: a search-while-load demand paging scheme with NAND flash memory," Proc. of the ACM SIGPLAN/SIGBED conference on languages, compilers, and tools for embedded systems(LCTES), 2007.

[3] C. Liu and J. LAYLAND. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the Association for Computing Machinery, 20(1): 46 - 61, January 1973.

[4] A.C.Shaw. Reasoning about time in higher language software. IEEE Transaction on Software Engineering, 15(7):875-889, July 1989.

[5] SAMSUNG, NAND Flash Memory, K9F1G08R0A

저 자 소 개

하 건 수

2005년 성균관대학교 정보통신공학 학사. 2005 년 ~ 현재 서울대학교 컴퓨터공학부 석박사 통합과정

관심분야: 임베디드 소프트웨어, 운영 체제

Email: air21c@davinci.snu.ac.kr

임 성 수

1993년 서울대학교 컴퓨터 공학부 학사. 1995년 서울대학교 컴퓨터 공학부 석사. 2002년 서울대학교 컴퓨터 공학부 박사. 현재, 국민대학교 컴퓨터 공학부 조교수.

관심분야: 임베디드 소프트웨어, 실시간 시스템

Email: sslim@kookmin.ac.kr

김 지 홍

1986년 서울대학교 계산통계학과 학사. 1988년 University of Washington 컴퓨터과학과 석사. 1995년 University of Washington 컴퓨터과학 및 공학과 박사. 현재, 서울대학교 컴퓨터 공학부 교수

관심분야: 임베디드 시스템, 컴퓨터 구조, 실시간 시스템, 저전력 시스템

Email: jihong@davinci.snu.ac.kr