NAND 플래시 메모리 기반의 실시간 임베디드 시스템에서의 demand paging 비용 분석

이영호[O] 임성수
국민대학교 전산과학과
{buriburi[O], sslim}@kookmin.ac.kr

Analysis of demand paging Cost for Flash Memory-based Real-Time Embedded Systems

Young-Ho Lee[O] Sung-Soo Lim
Computer Science, Kookmin University

요 약

NAND 플래시 메모리 기반의 실시간 임베디드 시스템에서는 일반적으로 shadowing 기법을 통해 프로그램을 수행한다. 그러나 shadowing 기법은 시스템의 부팅 시간을 증가시키고 불필요한 DRAM 영역을 차지한다는 단점 때문에 자원 제약이 심한 실시간 임베디드 시스템에는 적합하지 않다. 이에 대한 대안 중 하나는 demand paging 기법을 활용하는 것이다. 단, demand paging 환경에서는 page fault에 의한 시간 지연 때문에 태스크의 최악 실행 성능을 예측하기 어렵다. 따라서 본 논문에서는 NAND 플래시 메모리 기반의 실시간 임베디드 시스템에서 demand paging 비용을 고려한 태스크 최악 성능 분석 기법을 제안한다. 제안하는 기법은 각 태스크에 대해 demand paging 비용을 계산하고, 이를 전통적인 WCRT 분석 기법과 결합하는 방법을 사용한다. 또한 demand paging 비용과 WCET 분석을 독립적으로 고려함으로써, 최악의 경우에도 분석 결과의 안정성을 보장하고 기존의 방법에 비해 분석 복잡도를 줄였다.

1. 서 론

최근 NAND 플래시 메모리를 데이터뿐만 아니라 프로그램 코드를 저장하기 위한 목적으로 사용하는 임베디드 시스템이 증가하고 있다. 임의 접근이 가능한 NOR 플래시 메모리와는 달리 NAND 플래시 메모리에서는 순차 접근만을 허용하기 때문에, 기존의 NAND 플래시 메모리 기반 실시간 임베디드 시스템에서는 프로그램을 수행하기 위한 주로 shadowing 기법[1]을 사용해왔다. 그러나 shadowing 기법은 실행하지 않는 페이지 영역도 DRAM에 적재하기 때문에, DRAM 영역을 지나치게 낭비하고 부팅 시간을 높일 수 있다는 단점이 있다. 이러한 특성 때문에 shadowing 기법은 자원 제약이 심한 실시간 임베디드 시스템에 적합한 기법이라고 할 수 없다.

이에 대한 대안 중 하나로 최근 실시간 임베디드 시스템을 위한 demand paging 기법이 주목을 받고 있다. 최근까지 demand paging 시스템은 실시간 임베디드 시스템보다는 서버나 데스크톱 환경에서 사용되었으나, 시스템이 복잡해지고 다기능을 요구함에 따라, 실시간 임베디드 시스템에서도 demand paging의 필요성이 커지고 있다. 단, 기존의 요구 페이징 기법을 플래시 메모리 기반의 실시간 임베디드 시스템에 적용하기 위해서는 다음과 같은 문제를 고려해야 한다. 첫째, 기존의 demand paging 기법은 대부분 디스크 기반의 저장장치를 가정하기 때문에, 플래시 메모리의 특성을 고려한 demand paging 기법이 필요하다. 둘째, demand paging 시스템에서는 page fault에 의한 시간 지연이 발생하기 때문에, 실시간 시스템에서 요구하는 엄격한 시간 제약을 보장하지 않을 가능성이 있다. 더 큰 문제는 page fault의 발생 빈도 및 발생 양상을 예측하기 어렵기 때문에, demand paging 환경에서 수행되는 태스크의 최악 수행 성능을 예측하기 어렵다는 점이다. 이러한 특성 때문에 demand paging 기법은 따라서 demand paging 기법을 실시간 시스템에 적용하기 위해서는 먼저 demand paging 환경에서의 태스크 최악 실행 성능을 파악할 필요가 있다.

본 논문에서는 NAND 플래시 메모리 기반의 임베디드 실시간 시스템에서 demand paging 비용을 고려한 태스크 최악 응답 시간[2] 분석 기법을 제안한다. 제안하는 기법에서는 각 태스크에 대한 demand paging 비용을 계산하고, 이를 전통적인 WCRT 분석 기법과 결합하는 방법을 사용한다.

Y. Lee[8] 등은 플래시 메모리 기반의 실시간

[1] 프로그램 실행 이전에 전체 프로그램 이미지를 RAM에 복사하는 방식

[2] Worst Case Response Time : WCRT

시스템에서 demand paging 비용을 고려한 태스크 최악 응답 시간을 제안하였다. [8]에서는 각 태스크 인스턴스에 대한 최악 demand paging 비용을 계산하기 위해, 태스크의 각 실행 경로에 대한 최악 실행 시간과 demand paging 비용을 결합하는 방법을 사용하였다. [1]에서 제안한 기법은 각 태스크 인스턴스에 대해 최적의 demand paging 비용을 얻을 수 있다는 장점이 있는 반면, demand paging 비용을 계산하기 위해 각 실행 경로에 대한 WCET 값을 모두 알아야 한다는 단점이 있다. 특히 최근 WCET 분석 관련 연구[9][10][11]에서는 분석의 효율을 위해 모든 실행 경로에 대한 WCET을 예측하는 방식은 채택하지 않고 있다. 따라서 본 논문에서는 각 태스크 내의 페이지 정보만을 사용하여 demand paging 비용을 분석하는 데 초점을 맞춘다. 이는 WCET 분석과 demand paging 비용 분석 과정을 독립적으로 수행함으로써, 분석 기법을 단순화하고 분석의 복잡도를 낮추는데 목적이 있다.

시뮬레이션을 통해 본 논문에서 제안한 분석 기법의 정확도를 검증한 결과 페이지 적재 양상을 고려하지 않는 naïve 분석 기법에 비해 더 정확한 최악 응답 시간을 얻을 수 있음을 보였다.

논문의 구성은 다음과 같다. 2장에서는 WCRT 분석 기법에 대한 기존 연구를 살펴본다. 3장에서는 논문에서 제안하는 demand paging 환경에서의 태스크 최악 응답 시간 분석 기법을 제시한다. 4장에서는 가상의 태스크 집합을 사용한 시뮬레이션 결과를 보인다. 마지막으로, 5장에서는 결론 및 향후 과제를 제시한다.

2. 관련 연구

실시간 시스템을 구성할 때는 스케줄링 가능성 분석(schedulability analysis)을 통해 시스템 내의 태스크 집합이 데드라인(deadline)과 같은 실시간 성능 요구조건을 만족시키는지를 판단할 수 있어야 한다. 본 논문에서 제안하는 기법은 최악 응답 시간 (WCRT) 분석 기법[1][2][3][4][5][6]을 바탕으로 한다.

최악 응답 시간 분석 기법에서는 각 태스크에 대해 최악 응답 시간을 계산하고, 각 태스크의 최악 응답 시간이 해당 태스크의 데드라인보다 작거나 같을 경우 해당 시스템은 스케줄 가능하다고 한다. 태스크 τ_i의 최악 응답 시간 R_i는 τ_i의 최악 실행 시간(WCET)과 상위 우선순위 태스크에 의한 시간 지연의 합으로 나타낸다(식 1).

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \qquad (1)$$

식 1 에서 $hp(i)$는 τ_i보다 높은 우선순위에 해당하는 태스크 집합을 의미한다. 이후 최악 응답 시간 분석 기법은 다양한 추가 비용을 고려하도록 확장되었다.

C. Lee[6] 등은 태스크가 선점될 때 캐시에 의해 발생하는 선점 비용(preemption cost)를 고려한 최악 응답 시간을 제시하였다. 식 2 는 [10]에서 제안한 태스크 최악 응답 시간을 나타내며, 이때 $PC_i(R_i)$은 τ_i에 의해 발생한 총 선점 비용을 나타낸다.

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + PC_i(R_i) \qquad (2)$$

본 논문에서는 WCRT 분석 기법을 바탕으로 demand paging 환경에서의 태스크 최악 응답 시간을 제안한다. demand paging 환경에서는 각 태스크가 page fault 를 처리하는데 걸리는 시간에 따라 태스크 최악 응답 시간이 결정된다. 따라서 각 태스크에 대한 demand paging 비용을 계산하고, 이를 WCRT 분석 이론과 결합하는 방법이 필요하다.

3. demand paging 환경에서의 태스크 최악 응답 시간 분석

본 논문에서는 demand paging 환경에서의 태스크 최악 응답 시간을 얻기 위해 각 태스크 별 demand paging 비용을 계산하고, 이를 전통적인 WCRT 분석 기법과 결합하는 방법을 사용한다. 따라서 demand paging 비용 분석의 정확도에 따라 분석 기법의 정확도가 결정된다.

[8]에서는 demand paging 비용 분석의 정확도를 높이기 위해 각 태스크의 실행 경로에 포함된 페이지 집합과 WCET 정보를 바탕으로 각 태스크 인스턴스에 대한 최대 demand paging 비용을 계산하였다. 그러나 본 논문에서는 각 태스크의 실행 경로에 포함된 페이지 집합 정보만을 사용하여 최대 demand paging 비용을 계산한다. 이는 WCET 분석과 demand paging 비용 분석을 독립적으로 수행함으로써, 분석 시 계산 복잡도를 줄이면서도 demand paging 비용이 최악임을 보장하기 위한 것이다.

각 태스크의 demand paging 비용을 예측하기 위한 가장 간단한 방법은 각 태스크 인스턴스에서 적재 가능한 최대 페이지 개수와 최악 demand paging 비용을 서로 곱하는 것이다. 이 경우, demand paging 비용을 고려한 태스크 최악 실행 시간은 식 3과 같이 나타낼 수 있다.

$$C_i' = C_i + \pi \times max\,(P_i) \qquad (3)$$

식 3에서 π 는 최악 page fault 처리 시간을, $max\,(P_i)$는 τ_i의 각 태스크 인스턴스에서 적재 가능한 최대 페이지 개수를 나타낸다. 따라서 태스크 최악 응답 시간은 식 1에서 C_i 를 식 3에 정의된 C_i' 로 대치함으로써 얻을 수 있다. 본 논문에서는 이러한 방법을 naïve 분석 기법으로 정의한다. 그러나 naïve 분석 기법은 demand paging 기법의 특성을 전혀 고려하지 않기 때문에, demand paging 비용을

실제보다 지나치게 높게 예측할 수 있다. 따라서 demand paging 비용을 정확하게 예측하기 위해서는 다음과 같은 사항을 고려해야 한다.

첫째, demand paging 비용은 태스크의 실행 경로에 따라 달라질 수 있기 때문에 태스크의 각 실행 경로에 대한 demand paging 비용을 고려해야 한다. 둘째, demand paging 비용은 매 태스크 인스턴스마다 달라질 수 있다. 실제로 demand paging 환경에서는 이전 태스크 인스턴스에서 적재한 페이지가 이후 태스크 인스턴스 demand paging 비용에 따라 달라질 수 있다. 따라서 정확한 demand paging 비용을 얻기 위해서는 이러한 고려사항을 반영한 분석 기법이 필요하다. 본 논문에서는 이를 반영하기 위해 2가지 분석 단계를 도입한다. 1) demand paging 비용 분석 단계에서는 각 태스크 실행 경로에 대한 demand paging 비용을 계산한다. 2) 페이지 재사용 양상 분석 단계에서는 이전 태스크 인스턴스에 의한 페이지 재사용 양상을 분석한다.

3.1 가정

demand paging 시스템에서는 메모리 크기, page fault 처리 시간, 페이지 교체 정책 등 태스크의 수행에 영향을 주는 많은 변수가 존재한다. 실제 실시간 시스템에서는 이러한 변수를 모두 고려한 분석 기법이 필요하지만, 본 논문에서는 page fault 발생 양상이 태스크 최악 응답 시간에 어떻게 영향을 미치는 지에만 초점을 맞춘다. 따라서 대상 demand paging 환경에 대해 다음과 같은 가정을 한다. 첫째, 모든 페이지에 대해 최악 page fault 처리 시간은 항상 같다. 둘째, 대상 시스템에는 충분한 양의 메모리가 존재하며, 페이지 교체는 가정하지 않는다. 셋째, 서로 다른 태스크 간에 중복해서 적재하는 페이지는 존재하지 않는다.

3.2 demand paging 비용 분석

demand paging 비용 분석 단계의 목적은 태스크의 각 실행 경로에 대해 demand paging 비용을 얻기 위한 것이다. demand paging 비용은 태스크의 실행 경로마다 달라질 수 있기 때문에, 본 논문에서는 분석의 정확도를 높이기 위해 제어 흐름(control-flow) 기반 방법을 사용한다. 분석 과정은 다음과 같다. 먼저 각 태스크에 대해 제어 흐름 그래프를 생성하고, 그래프 내의 기본 블록(basic block)과 해당 기본 블록을 포함하는 메모리 페이지 간에 연관 관계를 생성한다. (그림 1)은 예제 제어 흐름 그래프를 사용한 각 기본 블록과 메모리 페이지 간의 관계를 보여준다. 예제 제어 흐름 그래프는 4개의 실행 경로를 포함하며, 각 기본 블록은 6개의 메모리 페이지에 사상되어 있다.

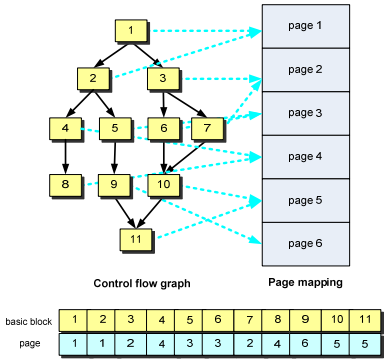

그림 1. 제어 흐름 그래프 상에서 basic block과 페이지 대응

본 논문에서는 분석을 위해 다음과 같은 수식을 정의한다. $C_{i,j}$, $P_{i,j}$ 는 각각 태스크 τ_i 의 j번째 실행 경로에 대한 최악 실행 시간(WCET)과 페이지 집합 나타내며, $n(P_{i,j})$ 는 $P_{i,j}$ 에 속한 페이지의 개수를 의미한다. 그리고 π 는 최악 page fault 처리 시간을 나타낸다. 따라서 τ_i 의 각 실행 경로에 대한 demand paging 비용은 식 4와 같이 나타낼 수 있다.

$$demand\ paging\ cost_{i,j} = \pi \times n(P_{i,j}) \quad (4)$$

3.3 페이지 재사용 양상 분석

demand paging 환경에서는 특정 태스크 인스턴스의 demand paging 비용이 이전에 수행된 태스크 인스턴스가 적재한 페이지 때문에 줄어들 가능성이 있다. 페이지 재사용 양상 분석의 목적은 태스크 인스턴스간에 재사용되는 페이지를 고려하여 각 태스크 인스턴스에 대한 최악 실행 시간을 구하는 데 있다.

페이지 재사용 양상 분석 기법은 그래프 이론(graph-theoretical)을 바탕으로 하며, 분석 단계는 다음과 같다.

첫째, demand paging 비용 분석 단계로부터 얻어진 제어 흐름 그래프를 사용하여 그래프를 생성한다. 그래프에서 각 노드는 실행 경로에 대한 고유번호를 의미하며, 각 열은 인스턴스의 순서를 뜻한다. 그리고 첫 번째 노드 'S'는 그래프의 논리적인 시작을 의미하고, 간선의 가중치는 특정 인스턴스에서 특정 실행 경로에 대한 최악 demand paging 비용을 의미한다.

둘째, 첫 번째 단계로부터 생성된 그래프를 사용하여 페이지 재사용 양상을 분석하고, DPC(Demand Paging Cost) 테이블을 생성한다. DPC 테이블 내의 원소 $\theta_{i,j}$는 각 태스크 인스턴스에 대한 최악 demand paging 비용을 나타낸다. DPC 테이블은 최종적으로 태스크 최악 응답 시간을 계산하는데 사용된다. 그림 2는 DPC 테이블의 구성을 보인다.

Tasks	Demand Paging Cost Table			
τ_1	$\theta_{1,1}$	$\theta_{1,2}$	...	$\theta_{1,k}$
τ_2	$\theta_{2,1}$	$\theta_{2,2}$	...	$\theta_{2,k}$
...	...	...	...	...
τ_n	$\theta_{n,1}$	$\theta_{n,2}$	...	$\theta_{n,k}$

그림 2. DPC 테이블

예를 들어, 페이지 재사용 양상을 고려한 태스크 인스턴스 $\tau_{i,0}$, $\tau_{i,1}$, $\tau_{i,2}$의 최악 demand paging 비용은 각각 $\theta_{i,1}$, $\theta_{i,2}$, $\theta_{i,3}$ 이다. 그림 3은 $\theta_{i,j}$ 를 계산하는 과정을 보여준다.

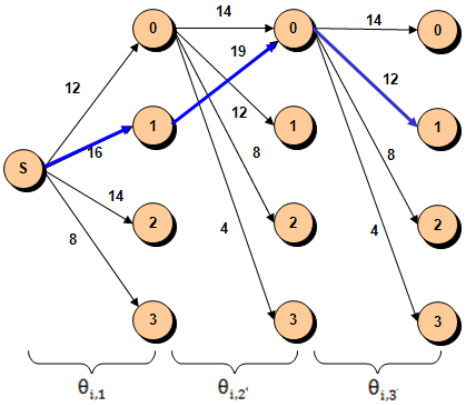

그림 3. 페이지 재사용 양상 분석 과정

그림 3에서 첫 번째 태스크 인스턴스는 1번 실행 경로를 수행할 때 demand paging 비용이 최대가 되며, 이때의 $\theta_{i,1}$ 값은 16이다. 다음으로 두 번째 태스크 인스턴스는 첫 번째 태스크 인스턴스의 실행 경로에 해당하는 1번 실행경로로부터 최악 demand paging 비용을 갖는 실행 경로를 취한다. 이 경우 두 번째 태스크 인스턴스는 실행 경로 0일 때 최대 demand paging 비용을 가지며, 이때의 $\theta_{i,2}$ 값은 19이다.

3.4 태스크 최악 응답 시간 계산

demand paging 비용 분석 단계 및 페이지 재사용 양상 분석 단계를 통해 각 태스크 인스턴스에 대한 최악 실행 시간을 얻을 수 있다. demand paging 비용을 고려한 태스크 최악 응답 시간은 식 5와 같이 나타낸다.

$$R_i^{k+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil C_j + DPC_i(R_i^k) \qquad (5)$$

식 5에서 태스크 최악 응답 시간은 $R_i^{k+1} = R_i^k$ 를 만족할 때까지 재귀적으로 계산된다. $DPC_i(R_i^k)$는 τ_i에 대한 demand paging 비용을 나타내며, 각 태스크의 요청 비율(request rate)에 따라 달라진다. $DPC_i(R_i^k)$의 정의는 식 6과 같이 나타낸다.

$$DPC_i(R_i) = \sum_{j=1}^{i} \sum_{l=1}^{g_j} \theta_{j,l} \qquad (6)$$

식 6에서 $\theta_{j,l}$ 는 demand paging 비용을 고려한 태스크 인스턴스 $\tau_{j,l}$ 의 최악 실행 시간을 나타내며, 페이지 재사용 양상 분석 단계로부터 얻어진다. g_j 는 R_i^k단계로부터 얻어진 τ_j의 최대 요청 비율을 나타내며, 식 7을 만족한다.

$$g_j \le \left\lceil \frac{R_i^k}{T_j} \right\rceil \qquad (7)$$

$\tau_{j,l}$에 해당하는 실행 경로의 WCET를 $wcet_{j,l}$라고 하자. 즉, $\theta_{j,l}$는 $\tau_{j,l}$에 해당하는 demand paging 비용이다. 이 경우 $wcet_{j,l}$는 항상 C_j보다 작거나 같다. 또한 $\theta_{j,l}$는 매 태스크 인스턴스에 대한 최악 demand paging 비용에 해당하므로, $DPC_i(R_i^k)$ 또한 최악임을 보장할 수 있다. 따라서 식 5로부터 얻어진 태스크 최악 응답 시간은 최악의 경우에도 안전함을 보장할 수 있다.

4. 실 험

본 논문에서는 제안한 태스크 최악 응답 시간 분석 기법의 정확성을 검증하기 위해, 가상의 태스크 집합에 대해 태스크 수행을 시뮬레이션 하였다. 표 1은 실험에 사용된 가상의 태스크 집합과 매개변수 정보를 나타낸다.

표 1. 태스크 집합 및 매개변수

태스크 집합	T_i	실행 경로 수	$C_{i,j}$	$P_{i,j}$
1	5	1	1	{45}
	15	1	2	{46}
	60	5	5	{1, 2, 3}
				{4, 5, 6}
				{7, 8}
				{9, 10, 11, 12, 13}
				{14, 15, 16, 17}
	180	1	60	{47}
2	5	1	1	{45}
	15	1	2	{46}
	60	5	5	{1, 2, 3}
				{3, 4, 5}
				{1, 7, 8}
				{4, 5, 7, 13, 14}
				{14, 15, 16, 17}
	180	1	60	{47}
3	5	1	1	{45}
	15	1	2	{46}
	60	5	5	{1, 2, 3}
				{2, 3, 4}

				{1, 3, 7, 8}
				{7, 8, 13, 14}
				{1, 2, 15, 16, 17}
	180	1	60	{47}

실험에 사용한 태스크 집합은 3개이며, 각 태스크 집합은 각각 4개의 태스크로 구성되어 있다. 표 1에서 T_i 및 $C_{i,j}$ 의 단위는 모두 ms(*millisecond*)이고, π 는 2이며, 각 태스크의 데드라인은 주기와 동일하다고 가정한다.

논문에서 제안한 분석 기법은 페이지 적재 양상을 고려하기 때문에, 페이지 적재 양상에 따른 태스크 최악 응답 시간 분석 결과의 정확성을 검증하기 위해 본 실험에서는 태스크 3의 매개변수 중 페이지 적재 양상만을 다양하게 변화시켰다. 태스크 3의 경우, 태스크 집합 1에서는 각 실행경로마다 중복해서 적재하는 페이지가 없기 때문에 가장 단순한 페이지 적재 양상을 가지며, 태스크 집합 3에서는 여러 실행 경로에 공통으로 포함된 페이지가 많기 때문에 가장 복잡한 페이지 적재 양상을 가진다.

페이지 적재 양상에 따른 분석 결과의 정확성을 검증하기 위해, naïve 분석 기법과 제안한 기법에 따른 태스크 최악 응답 시간을 비교하였다. 표 2는 표 1에서 정의한 가상 태스크 집합을 사용한 시뮬레이션 결과를 보여준다.

표 2. 태스크 최악 응답 시간 분석 결과

태스크 집합	R_i (naïve 분석 기법)	R_i (제안한 기법)
1	180 초과	159
2	180 초과	157
3	180 초과	157

먼저 naïve 분석 기법은 모든 태스크 집합에 대해 태스크 최악 응답 시간이 데드라인인 180ms를 초과한 것으로 나타난 반면 제안한 기법에서는 모두 데드라인보다 작은 것으로 나타났다. 이는 페이지 적재 양상을 고려하지 않을 경우 지나치게 높은 태스크 응답 시간을 얻을 가능성이 있음을 보여준다. 특히 상위 우선순위를 갖는 태스크에 의한 응답 시간 지연이 발생할 수 있다. 태스크 집합 1에서 태스크 3의 각 실행 경로는 모두 다른 페이지 집합을 적재함에도 불구하고 naïve 분석 기법과 제안한 기법간에 최악 응답 시간 차이가 발생했다. 높은 우선순위를 갖는 태스크일수록 요청 비율이 높아지기 때문에, 페이지 적재를 고려하지 않는 경우 상위 우선순위 태스크에 의한 demand paging 비용이 최악 응답 시간을 지나치게 높일 수 있다. 예를 들면, 태스크 1의 요청 비율은 최대 36($180/5$)이므로, naïve 분석 기법에서는 태스크 1의 demand paging 비용을 $2 \times 36 = 72$인 반면, 제안한 기법에서는 2가 된다.

또한 태스크 집합 1보다 태스크 집합 3에서 최악 응답 시간이 더 작은 것으로 나타났는데, 이는 태스크 집합 3의 페이지 적재 양상이 태스크 집합 1에 비해 더 복잡하기 때문이다. 실험 결과는 논문에서 제안한 분석 기법이 페이지 적재 양상을 고려하지 않는 naïve 분석 기법에 비해 더 정확한 분석 결과를 나타냄을 보여준다.

5. 결론 및 향후 과제

본 논문에서는 플래시 메모리를 사용하는 demand paging 환경에서의 태스크 최악 응답 시간 분석 기법을 제안하였다. 제안한 기법은 각 태스크의 실행 경로와 이전 태스크 인스턴스에 의한 페이지 적재 양상에 따라 demand paging 비용이 달라진다는 점을 반영한다. 분석 단계는 크게 demand paging 비용 분석 단계와 페이지 재사용 양상 분석 단계로 나누어진다.

본 논문에서 제안한 방법은 다음과 같은 개선점이 있다. 첫째, 반복문과 같은 복잡한 프로그램 구조, 페이지 교체 정책(page replacement policy)이나 공유 코드(shared code)등을 고려함으로써 실제 실시간 시스템에서 사용 가능하도록 분석 기법을 확장할 필요가 있다. 둘째, 분석 결과를 바탕으로 예측 가능한 demand paging 시스템을 설계할 수 있다. 예를 들면, 태스크 수행 이전에 일련의 페이지를 적재하는 pre-paging 등의 방법을 사용함으로써 태스크 수행 중에 page fault가 일어나지 않도록 할 수 있다.

참 고 문 헌

[1] K. Tindell.: Adding Time-Offsets to Schedulability Analysis. Technical Report YCS 221, Dept. of Computer Science, University of York, England (1994)

[2] Palencia, J. C. and Gonzlez Harbour, M.: Schedulability Analysis for Tasks with Static and Dynamic Offsets. In Proceedings of the IEEE Real-Time Systems Symposium (1998)

[3] N.Audsley et al.: Applying new scheduling theory to static priority preemptive scheduling. Software Engineering Journal. (1993) 284-292

[4] J. V. Busquets-Mataix, J. J. Serrano-Martin, R. Ors, P. Gil, and A. Wellings.: Adding Instruction Cache Effect to Schedulability Analysis of Preemptive Real-Time Systems. Proceedings of the 2nd Real-Time Technology and Applications Symposium (1996)

[5] C. Lee, J. Hahn, Y. Seo, S. Min, R. Ha, S. Hong, C. Park, M. Lee, and C. Kim.: Analysis of Cache-related Preemption Delay in Fixed-Priority Preemptive Scheduling. IEEE Transactions on Software Engineering (1996) 264-274

[6] K. Tindell, A. Burns, and A. Wellings.: An Extendible Approach for Analysing Fixed-Priority Hard Real-Time Tasks. Journal of Real-Time Systems, 6(2). (1994) 133-151

[7] Scott F. Kaplan, Lyle A. McGeoch, Megan F. Cole.: Adaptive caching for demand prepaging. Proceedings of the 3rd international symposium on Memory management. Berlin, Germany (2002)
[8] Young-Ho Lee, Sung-Soo Lim.: Worst Case Response Time Analysis for demand paging on Flash Memory. Proceedings of the 2nd International Workshop on Software Support for Portable Storage. (2006)
[9] A. Ermedahl. A Modular Tool Architecture for Worst-Case Execution Time Analysis. PhD thesis, Uppsala University, June 2003.
[10] Iain Bate , Ralf Reutemann, Worst-Case Execution Time Analysis for Dynamic Branch Predictors, Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04), p.215-222, June 30-July 02, 2004
[11] A. Ermedahl, J. Gustafsson, and B. Lisper. Experiences from industrial WCET analysis case studies. In Reinhard Wilhelm, editor, Proc. 5th International Workshop on Worst-Case Execution Time Analysis, (WCET'2005), pages 19--22, Palma de Mallorca, July 2005.