韓國 컴퓨터情報學會 論文誌
第11卷 第6號, 2006. 12

2006-11-6-1-13

플래시 메모리를 사용하는 demand paging 환경에서의 태스크 최악 응답 시간 분석

이 영 호*, 임 성 수**

Worst Case Response Time Analysis for Demand Paging on Flash Memory

Young-Ho Lee *, Sung-Soo Lim **

요 약

최근 NAND 플래시 메모리를 데이터뿐만 아니라 프로그램 코드를 저장하기 위한 목적으로 사용하는 실시간 시스템이 증가하고 있다. 그러나 데이터의 순차 접근만을 허용하는 NAND 플래시의 물리적인 특성 때문에, NAND 플래시 메모리 기반의 시스템에서는 일반적으로 shadowing 기법을 통해 프로그램을 수행한다. 그러나 shadowing 기법은 시스템의 부팅 시간을 증가시키고 불필요한 DRAM 영역을 차지한다는 단점이 있다. 이에 대한 대안 중 하나는 demand paging 기법을 활용하는 것이다. 그러나 demand paging 환경에서는 프로그램 실행 도중 임의로 발생하는 page fault 때문에 프로그램의 최악 응답 시간을 예측하기 어렵다. 본 논문에서는 demand paging 환경에서의 태스크 최악 응답 시간 분석 기법을 제안한다. 분석 기법은 분석의 정확도와 시간 복잡도에 따라 DP-Pessimistic, DP-Accurate으로 나뉜다. 또한 시뮬레이션을 통해 DP-Pessimistic과 DP-Accurate 분석 기법의 정확도를 비교한다.

Abstract

Flash memory has been increasingly used in handheld devices not only for data storage, but also for code storage. Because NAND flash memory only provides sequential access feature, a traditionally accepted solution to execute the program from NAND flash memory is shadowing. But, shadowing has significant drawbacks: increasing a booting time of the system and consuming severe DRAM space. Demand paging has obtained significant attention for program execution from NAND flash memory. But, one of the issues is that there has been no effort to bound demand paging cost in flash memory and to analyze the worst case performance of demand paging. For the worst case timing analysis of programs running from NAND flash memory, the worst case demand paging costs should be estimated. In this paper, we propose two different WCRT analysis methods considering demand paging costs, DP-Pessimistic and DP-Accurate, depending on the accuracy and the complexity of analysis. Also, we

• 제1저자 : 이영호, 교신저자 : 임성수
• 접수일 : 2006.11.24, 심사일 : 2006.12.06, 심사완료일 : 2006. 12.25
* 국민대학교 전산과학과 석사과정, ** 국민대학교 전산과학과 교수
※ 본 연구는 국민대학교 교내연구비의 지원으로 수행되었습니다.

compare the accuracy between DP-Pessimistic and DP-Accurate by using the simulation.

▸ Keyword : 최악 응답 시간 분석(Worst Case Response Time Analysis), WCRT 분석(WCRT analysis), 요구 페이징(Demand Paging), 플래시 메모리(Flash Memory)

Ⅰ. 서론

최근 플래시 메모리를 저장장치로 사용하는 임베디드 시스템이 증가하고 있다. 플래시 메모리는 NOR 형과 NAND 형으로 구분된다. 일반적으로 NOR 플래시 메모리는 주로 프로그램 코드를 저장하기 위한 목적으로 사용되는 반면, NAND 플래시 메모리는 멀티미디어 데이터와 같이 고용량 데이터를 저장하기 위한 목적으로 사용된다. 이러한 차이는 NOR 플래시 메모리와 NAND 플래시 메모리의 물리적인 특성 때문이다. NOR 플래시 메모리에서는 임의 접근(random access)이 가능하기 때문에, 프로그램을 NOR 플래시 메모리로부터 직접 실행하는 것이 가능하다(eXecute-In-Place : XIP). 반면, NAND 플래시 메모리에서는 순차 접근(sequential access)만 허용하기 때문에, 프로그램을 NAND 플래시 메모리로부터 직접 실행할 수 없으며, 실행 이전에 프로그램을 DRAM으로 먼저 복사해야 한다.

최근에는 NAND 플래시 메모리의 가격이 낮아지고 용량이 커짐에 따라 NAND 플래시 메모리의 비율이 높아지고 있다. 이러한 추세는 고용량 데이터를 기반으로 하는 멀티미디어 프로그램이 증가함에 따라 강화되고 있다. 따라서 NOR 플래시 메모리보다는 NAND 플래시 메모리와 DRAM의 조합으로 시스템을 구성하는 것이 가격 측면에서 더 유리하게 되었으며, 최근에는 NAND 플래시 메모리와 DRAM의 조합으로 시스템을 구성하는 사례가 증가하고 있다.

NAND 플래시 메모리에서 프로그램을 실행하는 가장 일반적인 방법은 shadowing이다. shadowing이란 프로그램이 수행되기 이전에 프로그램의 전체 이미지를 NAND 플래시 메모리로부터 DRAM에 복사하는 것을 의미한다. shadowing은 휴대폰과 같이 실시간 성능을 요구하는 임베디드 시스템에서 널리 사용되어 왔다. 그러나 shadowing은 실행되지 않는 코드 페이지도 DRAM에 복사하기 때문에 DRAM 영역을 지나치게 낭비하고, 태스크 실행 전에 전체 이미지를 DRAM에 적재해야 하기 때문에 부팅 시간을 높일 수 있다는 단점이 있다. 따라서 휴대폰과 같은 무선 통신 장치에서는 코드 저장장치로 NOR 플래시 메모리를 주로 사용해왔다.

그러나 무선 통신 장치에서도 고용량 데이터를 요구하는 멀티미디어 관련 어플리케이션이 주가 됨에 따라 고용량 데이터를 저장하기에 적합한 NAND 플래시 메모리를 사용하는 것은 필수 요구사항이 되었다. 뿐만 아니라, 하나의 시스템에서 코드 저장장치로 NOR 플래시 메모리를 사용하고 데이터 저장장치로 NAND 플래시 메모리를 사용하는 것은 불필요하게 시스템의 단가를 높이는 원인이 된다. 따라서 NAND 플래시 메모리를 코드와 데이터에 대한 저장장치로써 효과적으로 사용하는 것이 중요한 이슈가 되었다. 이러한 점에서 볼 때, shadowing은 실시간 임베디드 시스템에 적합한 기법이라고 할 수 없다.

NAND 플래시 메모리에서 프로그램을 실행하기 위한 방법 중 하나로 demand paging 기법이 관심을 얻고 있다. shadowing과는 달리 demand paging은 실제로 실행될 페이지만 NAND 플래시 메모리로부터 DRAM으로 복사하기 때문에, shadowing에 비해 불필요한 DRAM 영역을 낭비하지 않으며 부팅 시간을 단축시킨다는 장점이 있다. 그러나 demand paging 기법에는 몇 가지 치명적인 문제점이 존재한다. 첫 번째는 page fault의 발생 양상을 예측할 수 없기 때문에 시스템의 예측성을 크게 떨어뜨릴 수 있다는 점이다. 이는 각 태스크가 실시간 성능 요구사항을 만족시키지 못하는 원인이 될 수 있다. 두 번째는 플래시 메모리를 기반으로 하는 demand paging 시스템의 최악 성능을 측정하기 위한 기존의 연구가 거의 없다는 점이다.

따라서 본 논문에서는 플래시 메모리를 기반으로 하는 demand paging 환경에서의 태스크 최악 응답 시간 분석 기법을 제안한다. 제안한 방법은 최악 응답 시간(worst case response time: WCRT) 분석 기법을 바탕으로 한다. 본 논문에서는 demand paging 비용을 계산하고 이를 기존의 WCRT 분석 이론과 결합하는 방법을 사용한다. 제안한 방법은 분석 기법의 정확도와 시간 복잡도에 따라 DP-Pessimistic과 DP-Accurate으로 나뉜다. DP-Pessimistic은 demand paging 비용만을 고려한 분석 방법이며, DP-Accurate은 이전 태스크 인스턴스에 의해 재사용되는 페이지까지 고려한 분석 기법이다.

시뮬레이션을 통해 본 논문에서 제안한 두 가지 분석 기법의 정확도를 비교해 본 결과 DP-Accurate이 DP-Pessimistic에 비해 최대 57% 정확함을 보였다.

본 논문의 구성은 다음과 같다. 2장에서는 최악 응답 시간 분석 이론에 대한 관련 연구를 소개하고, 3장에서는 본 논문에서 제안한 demand paging 환경에서 태스크 최악 응답 시간 분석 기법에 대해 설명한다. 4장에서는 시뮬레이션을 통해 본 논문에서 제안한 2가지 기법의 정확도를 비교한다. 마지막으로 5장에서 결론을 맺는다.

II. 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. 스케줄링 가능성 분석 기법에는 이용률 기반(utilization-based) 분석 기법[1][2]과 최악 응답 시간 (worst case response time : WCRT) 분석 기법[3][4][5]이 있다.

Liu와 Layland [1]는 고정 우선순위를 가지는 주기 태스크를 위한 스케줄링 가능성 테스트(schedulability test)를 제안했다. [1]에서는 태스크 집합이 스케줄 가능하기 위한 CPU 이용률의 상한선을 제시했는데, n이 무한히 클 때 U가 최대 약 69.32보다 작거나 같다면 해당 태스크 집합은 비율 단조(rate monotonic) 우선순위 할당[2] 시 스케줄 가능하다고 알려져 있다 (식 1).

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \le n(2^{\frac{1}{n}} - 1) \quad \text{..... (식 1)}$$

단, 위의 방법은 스케줄링 가능하기 위한 충분조건은 만족하지만 필요조건은 만족하지 못한다는 단점이 있다. 이후 Lehoczky[2] 등은 CPU 이용률 분석 기법을 사용한 스케줄링 가능성 테스트에 대한 필요충분조건을 제시했다.

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 시간과 상위 우선순위 태스크에 의한 시간 지연의 합으로 나타내는데, 일반적으로 (식 2)와 같이 나타낸다.

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \quad \text{..... (식 2)}$$

(식 2)에서 $hp(i)$는 τ_i보다 높은 우선순위를 갖는 태스크의 집합을 뜻한다. 최악 응답 시간은 반복적으로 계산되는데, 초기 조건은 $R_i^0 = C_i$이고, $R_i^{n+1} = R_i^n$일 때까지 계속된다. 이후 최악 응답 시간 분석 기법은 다양한 추가 비용을 고려하도록 확장 되었는데, release jitter[6][7][8], preemption cost[9][10], arbitrary deadlines[11] 등을 고려한 최악 응답 시간 모델이 여기에 해당한다. N. Audsley [8] 등은 release jitter를 고려한 경우의 태스크 최악 응답 시간을 제안했는데, 이는 (식 3)과 같이 나타낸다.

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n + J_j}{T_j} \right\rceil C_j \quad \text{..... (식 3)}$$

(식 3)에서 J_i는 τ_i의 release jitter를 나타내며, 태스크 최악 응답 시간은 (식 2)와 마찬가지로 $R_i^{n+1} = R_i^n$일 때까지 반복적으로 계산된다. 이때, R_i는 τ_i가 release될 때부터 종료될 때까지 걸린 시간이므로, τ_i의 최악 응답 시간은 $R_i + J_i$로 나타낸다.

C. Lee[10] 등은 태스크가 선점될 때 캐시에 의해 발생하는 선점 비용(preemption cost)를 고려한 최악 응답 시간을 제시하였다. (식 4)는 [10]에서 제안한 태스크 최악 응답 시간을 나타내며, 이때 $PC_i(R_i)$은 τ_i에 의해 발생한 총 선점 비용을 나타낸다.

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + PC_i(R_i) \quad \text{..... (식 4)}$$

본 논문에서는 WCRT 분석 기법을 바탕으로 demand paging 환경에서의 태스크 최악 응답 시간을 제안한다. dem

2) 주기가 짧은 태스크에 더 높은 우선순위를 할당하는 방식

and paging 환경에서는 각 태스크가 page fault를 처리하는데 걸리는 시간에 따라 태스크 최악 응답 시간이 결정된다. 따라서 각 태스크에 대한 demand paging 비용을 계산하고, 이를 WCRT 분석 이론과 결합하는 방법이 필요하다.

III. demand paging 환경에서의 태스크 최악 응답 시간 분석

본 장에서는 논문에서 제안하는 demand paging 환경에서의 태스크 최악 응답 시간 분석 기법에 대해 설명한다. 제안한 방법은 분석의 정확도와 시간 복잡도에 따라 DP-Pessimistic, DP-Accurate으로 나뉜다.

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 최악 응답 시간 모델이 필요하지만, 본 논문에서는 page fault 발생 양상이 태스크의 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 같다고 가정한다. 디스크에서는 지연 시간(latency time) 때문에 특정 페이지를 읽는데 걸리는 시간이 일정하지 않다[12]. 반면 플래시 메모리의 경우 디스크와 같은 latency time이 존재하지 않기 때문에, 임의의 페이지를 읽는데 걸리는 시간은 위치에 상관없이 거의 같다. 따라서 최악 page fault 처리 시간은 항상 같다고 가정할 수 있다.

둘째, 대상 시스템 내에는 충분한 양의 메모리가 존재한다고 가정한다. 따라서 페이지 교체는 발생하지 않는다. 만약 시스템이 제한된 양의 메모리를 가진다고 할 경우 단순히 태스크의 페이지 적재 양상만을 통해서는 태스크 최악 응답 시간을 얻을 수 없다. 왜냐하면 특정 시점에 메모리가 부족한 경우, 이미 적재된 페이지도 이후 인스턴스에서 다시 적재될 수 있기 때문이다. 반면 메모리가 충분하다고 가정할 경우, 한번 적재된 페이지는 이후 인스턴스에서 적재되지 않기 때문에 페이지 적재 양상에 따른 영향을 최소화할 수 있다.

셋째, 서로 다른 태스크 간에 겹치는 페이지는 존재하지 않는다고 가정한다. 예를 들면 서로 다른 태스크 간에 공유 라이브러리를 사용하는 경우는 발생하지 않는다. 만약 서로 다른 태스크 간에 겹치는 페이지가 존재한다고 가정할 경우, 특정 태스크에서 적재한 페이지가 다른 태스크에 영향을 줄 가능성이 있으며 이는 페이지 적재 양상을 복잡하게 만들 수 있기 때문이다.

3.2 고려사항

demand paging 비용을 고려한 태스크 최악 응답 시간을 얻기 위해서는 demand paging 시스템의 특성을 반영한 분석 기법이 필요하다. 기존의 최악 응답 시간 이론과 demand paging 시스템에서의 최악 응답 시간 이론 사이에는 다음과 같은 차이점이 존재한다.

첫째, demand paging 비용은 태스크의 특정 실행 경로에 종속적이다. 기존의 최악 응답 시간 분석 이론에서는 추가적인 비용(release jitter, preemption cost 등)을 처리하는데 걸리는 시간은 실행 시간과 독립적이라고 가정한다. 따라서 추가적인 비용을 처리하는데 걸리는 시간은 $C_i+\alpha_i$ (단, α_i는 추가 비용을 처리하는데 걸리는 시간)로 나타낼 수 있다. 반면 demand paging 시스템에서 page fault의 발생 빈도는 특정 실행 경로에 종속적이기 때문에 demand paging 비용은 태스크의 실행 경로에 따라 달라질 수 있다. (표 1)은 태스크의 실행 경로와 demand paging 비용 간의 관계를 나타낸다.

표 1. 실행 경로와 demand paging 비용 간의 관계
Table 1. Demand paging costs for execution paths

실행 경로	C_i	α_i	$C_i+\alpha_i$
1	10	9	19
2	15	3	18

(표 1)에서 태스크 τ_i는 두 개의 실행 경로를 포함하며, 각 실행 경로의 최악 실행 시간은 각각 10, 15이고 demand paging 비용은 각각 9, 3이라고 하자. 만약 태스크 τ_i의 최악 실행 시간이 demand paging 비용과 독립적이라고 가정할 경우, demand paging 비용을 고려한 τ_i의 최악 실행 시간은 2번 실행 경로의 최악 실행 시간 15와 1번 실행 경로의 최대 demand paging 비용 9의 합인 24가 된다. 그러나 demand paging 비용은 특정 실행 경로에 종속적이기 때문에 demand paging 비용을 고려한 τ_i의 실제 최악 실행 시간은 19가 된다. 만약 최악 실행 시간에 해당하는 실행 경로와 최대 demand paging 비용에 해당하는 실행 경로가 일치하지 않는 경우 demand paging 비용을

고려한 의 최악 실행 시간은 실제 가능한 시간보다 더 높은 것으로 나타날 수 있다. 따라서 demand paging 비용을 고려한 태스크의 최악 응답 시간을 정확하게 얻기 위해서는 각 실행 경로에 대한 demand paging 비용과 최악 실행 시간을 고려해야 한다.

둘째, 이전 태스크 인스턴스에 의한 페이지 적재를 반영해야 한다. 기존의 최악 응답 시간 이론에서는 각 태스크 인스턴스가 서로 독립적이라고 가정한다. 다시 말하면, 각 태스크 인스턴스는 다른 태스크 인스턴스의 최악 실행 시간에 영향을 미치지 않는다. 따라서 모든 태스크 인스턴스의 최악 실행 시간은 항상 같다. 그러나 이러한 가정은 이전 태스크 인스턴스에 의해 이미 적재된 페이지를 이후 태스크 인스턴스가 재사용할 가능성을 고려하지 않는다는 단점이 있다. 실제로 demand paging 시스템에서는 이전 태스크 인스턴스에서 적재한 페이지가 이후 태스크 인스턴스의 demand paging 비용에 영향을 줄 수 있다. (그림 1)은 demand paging 환경에서 태스크 인스턴스 간의 페이지 재사용 양상을 보여준다.

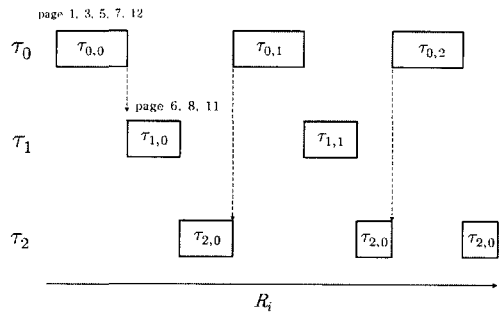

그림 1. 태스크 인스턴스 간의 페이지 재사용 양상
Fig. 1. Page reuses of task instances

(그림 1)에서 τ_0의 최악 실행 시간 C_0은 100이고, 이때의 실행 경로는 1, 3, 5, 7, 12번 페이지를 적재한다고 가정한다. 만약 $\tau_{0,1}$과 $\tau_{0,2}$가 동일한 실행 경로를 수행한다고 가정할 경우, $\tau_{0,1}$을 실행하는 시점에는 추가적인 page fault가 발생하지 않는다. 왜냐하면 $\tau_{0,0}$의 실행 시점에 1, 3, 5, 7, 12번 페이지가 이미 적재되었기 때문이다. 그 결과 $\tau_{0,1}$, $\tau_{0,2}$에서의 Demand paging 비용은 0이 되며, $\tau_{0,1}$의 최악 실행 시간은 $100+5\pi$가 아니라 100이 된다. 위의 예는 태스크 인스턴스 간의 페이지 재사용 양상을 고려하지 않을 경우, 지나치게 높은 최악 응답시간을 얻게 될 가능성이 있음을 보여준다.

3.3 demand paging 환경에서의 WCRT 분석 과정

demand paging 환경에서의 태스크 최악 응답 시간을 정확하게 얻기 위해서는 앞서 서술한 고려사항을 반영해야 한다. 본 논문에서는 이를 반영하기 위해 다음과 같은 2가지 분석 단계를 도입한다.

첫째, demand paging 비용 분석 단계에서는 태스크의 각 실행 경로에 대해 demand paging 비용을 계산한다. 이를 위해 본 논문에서는 제어 흐름 그래프(control flow graph)를 사용한다. 분석 결과는 각 태스크의 실행 경로 별 최악 실행 시간과 해당 실행 경로에 속하는 페이지의 집합이다.

둘째, 페이지 재사용 양상 분석 단계에서는 이전 태스크 인스턴스에 의해 재사용되는 페이지를 고려하여 각 태스크 인스턴스에 대한 DPRC 값을 구한다. 여기서 DPRC란 demand paging 비용과 페이지 재사용 양상이 반영된 태스크 인스턴스의 최악 실행 시간을 뜻한다. 이를 위해 본 논문에서는 그래프 기반(graph-based)의 방법을 사용한다. 페이지 재사용 양상 분석 결과는 DPRC 테이블이며, DPRC 테이블에는 각 태스크 인스턴스에 대한 DPRC 값이 포함된다.

본 논문에서 제안하는 태스크 최악 응답 시간 기법은 분석의 정확도와 시간 복잡도에 따라 DP-Pessimistic과 DP-Accurate으로 나뉜다. (그림 2)는 본 논문에서 제안한 demand paging 환경에서의 WCRT 전체 분석 과정을 보여

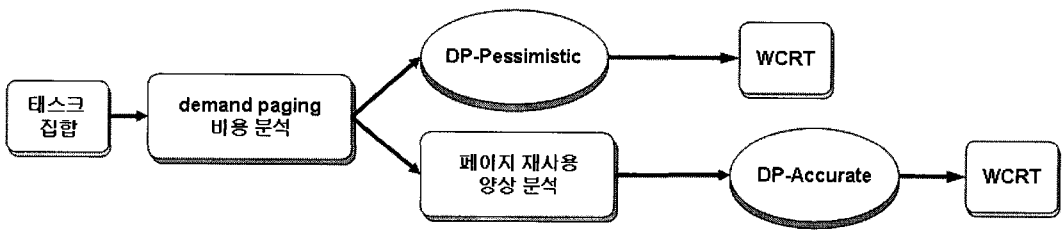

그림 2. demand paging 환경에서의 WCRT 전체 분석 과정
Fig. 2. WCRT analysis for demand paging overview

준다. DP-Pessimistic에서는 demand paging 비용 분석만을 수행하여 최악 응답 시간을 계산하는 반면, DP-Accurate에서는 demand paging 비용 분석과 페이지 재사용 양상 분석을 수행하여 최악 응답 시간을 계산한다.

3.4 demand paging 비용 분석

본 논문에서는 각 실행 경로에 대한 demand paging 비용을 분석하기 위해 제어 흐름 그래프(control flow graph)를 사용한다. 분석 단계는 다음과 같다. 먼저 각 태스크에 대한 제어 흐름 그래프를 생성한다. 제어 흐름 그래프에서 각 태스크는 basic block과 각 basic block간의 흐름으로 나타낼 수 있다. 다음으로 각 basic block을 해당 basic block이 포함되는 페이지와 대응 시킨다. 그 결과, 각 태스크에 대한 실행 경로와 각 실행 경로에 포함되는 페이지 정보를 얻을 수 있다. (그림 3)은 예제 태스크에 대해 각 basic block을 페이지와 대응한 결과를 보여준다.

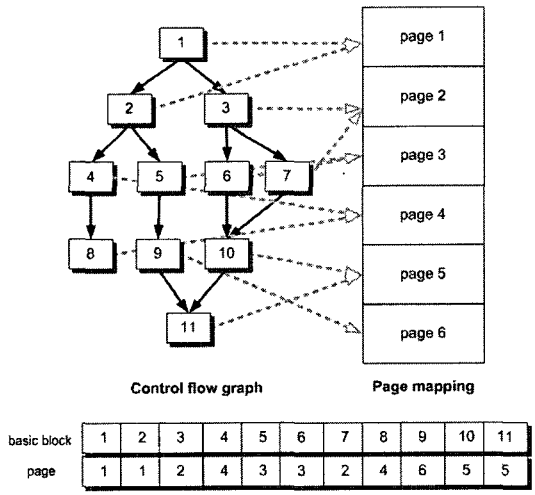

그림 3. 제어 흐름 그래프 상에서 basic block과 페이지 대응
Fig. 3. Basic block and page mapping for a control flow graph

(그림 3)에서 제어 흐름 그래프는 4개의 실행 경로를 가지고 있으며, 각 basic block은 6개의 페이지에 대응되어 있음을 알 수 있다. 본 논문에서는 분석을 위해 다음과 같은 수식을 정의한다.

- $C_{i,j}$: 태스크 τ_i의 j번째 실행 경로에 대한 최악 실행 시간(WCET)
- $P_{i,j}$: 태스크 τ_i의 j번째 실행 경로에 속하는 페이지의 집합
- $n(P_{i,j})$: $P_{i,j}$에 속하는 페이지의 개수
- π : 최악 page fault 처리 시간

예를 들어, (그림 3)에서 태스크 τ_i는 위에서 정의한 수식을 사용하여 다음과 같이 나타낼 수 있다.

$$P_{i,0} = \{1, 4\},\ n(P_{i,0}) = 2$$

$$P_{i,1} = \{1, 3, 5, 6\},\ n(P_{i,1}) = 4$$

$$P_{i,2} = \{1, 2, 3, 5\},\ n(P_{i,2}) = 4$$

$$P_{i,3} = \{1, 2, 5\},\ n(P_{i,3}) = 3$$

태스크 τ_i의 각 실행 경로에 대한 demand paging 비용은 다음과 같다.

$$demand\ paging\ \cos t_{i,j} = \pi \times n(P_{i,j}) \quad \cdots\cdots\cdots\cdots \text{(식 5)}$$

따라서 demand paging 비용을 고려한 태스크 τ_i의 최악 실행 시간은 다음과 같이 계산할 수 있다.

$$C''_i = \max \begin{Bmatrix} C_{i,0} + demand\ paging\ cost_{i,0} \\ C_{i,1} + demand\ paging\ cost_{i,1} \\ \cdots \\ C_{i,n-1} + demand\ paging\ cost_{i,n-1} \end{Bmatrix} \quad \cdots\cdots\cdots \text{(식 6)}$$

예를 들어, (그림 3)에서 π를 2로, $C_{i,j}$를 각각 3, 6, 7, 5라고 가정할 경우 demand paging 비용을 고려한 태스크 τ_i의 최악 실행 시간은 다음과 같이 계산할 수 있다.

$$demand\ paging\ cost_{i,0} = 2 \times 2 = 4$$

$$demand\ paging\ cost_{i,1} = 2 \times 4 = 8$$

$$demand\ paging\ cost_{i,2} = 2 \times 4 = 8$$

$$demand\ paging\ cost_{i,3} = 2 \times 3 = 6$$

$$C''_i = \max \begin{Bmatrix} C_{0,0} + demand\ paging\ cost_{0,0} \\ C_{0,1} + demand\ paging\ cost_{0,1} \\ C_{0,2} + demand\ paging\ cost_{0,2} \\ C_{0,3} + demand\ paging\ cost_{0,3} \end{Bmatrix} = \max \begin{Bmatrix} 3+4 \\ 6+8 \\ 7+8 \\ 5+6 \end{Bmatrix}$$

$$= 15$$

그러나 본 논문에서 사용한 demand paging 비용 분석

방법은 반복문과 같이 복잡한 구조의 프로그램은 고려하지 않는다는 단점이 있다. 예를 들어 반복문을 고려할 경우, 특정 실행 경로뿐만 아니라 다른 실행 경로에 대한 demand paging 비용도 반영해야 한다. 예를 들어, (그림 3)에서 basic block 10에서 basic block 2로의 분기 흐름이 존재한다고 가정할 경우, basic block 1-3-6-10-2-4-8 과 같은 실행 흐름이 존재할 수 있으며, 그 결과 페이지 적재 양상이 매우 복잡해지게 된다. 따라서 이를 고려할 수 있도록 demand paging 비용 분석 과정을 확장할 필요가 있다.

3.5 페이지 재사용 양상 분석

demand paging 환경에서는 특정 태스크 인스턴스의 demand paging 비용이 이전에 수행된 태스크 인스턴스가 적재한 페이지 때문에 줄어들 가능성이 있다. 따라서 페이지 재사용 양상 분석 단계의 목적은 이러한 태스크 간에 재사용되는 페이지를 고려하여 각 태스크 인스턴스에 대한 최악 실행 시간을 구하는 데 있다. 이 단계에서는 그래프 기반(graph-based)의 방법을 사용한다. 분석 단계는 다음과 같다.

첫째, demand paging 비용 분석 단계로부터 얻어진 제어 흐름 그래프를 사용하여 그래프를 생성한다. 그래프에서 각 노드는 실행 경로의 인덱스를 의미하며, 각 열은 인스턴스의 순서를 뜻한다. 그리고 첫 번째 노드 S는 그래프의 논리적인 시작을 의미하며, 간선의 가중치는 특정 인스턴스에서 특정 실행 경로에 대한 최악 실행 시간을 의미한다. 여기서 최악 실행 시간은 demand paging 비용과 페이지 재사용 양상을 모두 고려한 값이다.

둘째, 첫 번째 단계로부터 얻어진 그래프를 사용하여 페이지 재사용 양상을 분석한다. 분석 결과 DPRC 테이블을 얻게 되는데, DPRC란 demand paging 비용과 페이지 재사용 양상을 고려한 특정 태스크 인스턴스의 최악 실행 시간을 뜻한다. 따라서 DPRC 테이블을 통해 각 태스크 인스턴스에 대한 DPRC 값을 얻을 수 있다.

(그림 4)는 페이지 재사용 양상 분석 과정을 개략적으로 보여준다.

본 논문에서는 페이지 재사용 양상 분석을 위해 다음과 같은 수식을 정의한다.

- $DPRC_i(j)$ (Demand Paging Related Cost) : demand paging 비용과 페이지 재사용 양상을 고려한 태스크 인스턴스 τ_{ij}의 최악 실행 시간
- $PLOAD_i(j)$ (Previously LOADed pages) : 태스크 인스턴스 τ_{ij} 이전에 적재된 모든 페이지의 집합. 예를 들어 $PLOAD_i(2)$는 태스크 인스턴스 τ_{i0} 부터 τ_{i2}에 의해 적재된 모든 페이지를 의미한다.

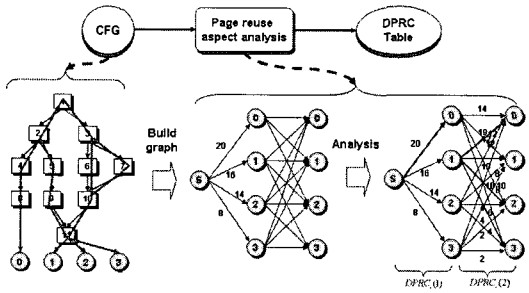

그림 4. 페이지 재사용 양상 분석 과정
Fig. 4. The page reuses aspect analysis overview

예를 들어, (그림 3)에서 demand paging 비용과 페이지 재사용 양상을 고려한 태스크 인스턴스 τ_{i0}, τ_{i1}, τ_{i2}의 최악 실행 시간은 각각 $DPRC_i(1)$, $DPRC_i(2)$, $DPRC_i(3)$이다.

각 태스크 인스턴스에 대한 DPRC 값은 이전 태스크 인스턴스에 의한 페이지 적재 양상을 고려하여 얻을 수 있다. 그러나 $DPRC_i(j)$값이 이미 결정되었더라도 최악 실행 시간을 가지는 $DPRC_i(j+1)$을 얻기 위해서는 이전 태스크 인스턴스에 의한 모든 페이지 적재 가능성을 고려해야 한다. (그림 5)는 각 태스크 인스턴스에 대해 DPRC 값을 계산하는 과정을 보여준다.

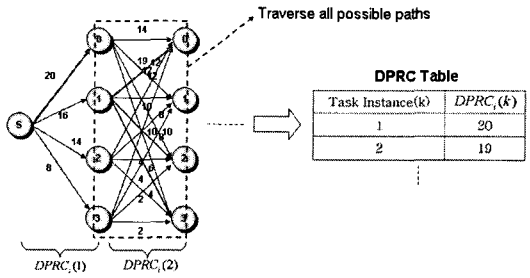

그림 5. DPRC 값의 계산
Fig. 5. Calculation of DPRC values

(그림 5)에서 첫 번째 태스크 인스턴스는 0번 실행 경로를 수행할 때 최대가 되며, 이때의 DPRC 값은 20이다. 그러나 두 번째 태스크 인스턴스의 DPRC 값을 구하기 위해서는 첫 번째 인스턴스의 최악 실행 경로인 0번이 아니라

모든 실행 경로에 대해 재계산을 수행해야 한다. 왜냐하면 이전 인스턴스의 최악 실행 시간이 해당 인스턴스의 DPRC 값보다 작더라도 이후 인스턴스에서 많은 페이지를 적재하는 경우 결과적으로 더 높은 최악 응답 시간을 얻을 수도 있기 때문이다. 따라서 $DPRC_i(2)$를 구하기 위해서는 실행 경로 0이 아니라 모든 실행 경로에 대해 최악 실행 시간을 재계산하고, 이 중 가장 큰 값을 취해야 한다. 이 경우 $DPRC_i(2)$는 19가 된다. 이를 반복하면 모든 태스크 인스턴스에 대한 DPRC 값을 얻을 수 있다. 이를 수식으로 나타내면 다음과 같다.

먼저 $DPRC_i(1)$는 다음과 같이 나타낸다.

$$DPRC_i(1) = max_{0 \le x < p}(C_{i,x} + \pi \times n(P_{i,x})) \quad \cdots\cdots \text{(식 7)}$$

(식 7)에서 p는 실행 경로의 수를 나타낸다. 즉 첫 번째 인스턴스의 DPRC 값은 모든 실행 경로 중 $C_{i,x} + \pi \times n(P_{i,x})$ 값이 최대일 때를 나타낸다. $DPRC_i(1)$을 바탕으로 $DPRC_i(j)$는 다음과 같이 계산할 수 있다.

$$DPRC_i(j) = max_{0 \le x < p}(C_{i,x} + \pi \times n(P_{i,x} - PL_l)) \quad \text{(식 8)}$$
$$PL_l \in PLOAD_i(j-1)$$

(식 8)에서 PL_l($0 \le l < n(PLOAD_i(j-1))$)은 $PLOAD_i(j-1)$에 포함된 페이지 집합을 나타낸다. (식 8)은 이전 태스크 인스턴스에 의해 발생 가능한 모든 경우를 고려할 때, 태스크 인스턴스 $\tau_{i,j}$의 최악 실행 시간을 나타낸다.

3.6 demand paging 환경에서의 태스크 최악 응답 시간

본 논문에서 제안한 WCRT 분석 방법은 분석의 정확도와 시간 복잡도에 따라 DP-Pessimistic, DP-Accurate으로 나뉜다.

DP-Pessimistic 에서 태스크 최악 응답 시간은 demand paging 비용만을 고려하는데, 다음과 같이 나타낼 수 있다.

$$R_i = C_i'' + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j'' \quad \cdots\cdots \text{(식 9)}$$

(식 9)에서 C_i''는 demand paging 비용을 고려한 태스크 τ_i의 최악 실행 시간이며, demand paging 비용 분석 단계로부터 얻어진다.

반면 DP-Accurate에서 태스크 τ_i의 최악 응답시간은 다음과 같이 나타낸다.

$$R_i = DPRC_i(1) + \sum_{j \in hp(i)} \sum_{k=1}^{\left\lceil \frac{R_i}{T_j} \right\rceil} DPRC_j(k) \quad \cdots\cdots \text{(식 10)}$$

DP-Accurate은 태스크 인스턴스간의 페이지 재사용 양상을 고려하기 때문에, 매 태스크 인스턴스마다 최악 실행 시간이 달라질 수 있다. (식 10)에서 각 DPRC 값은 페이지 재사용 양상 분석 단계의 결과인 DPRC 테이블로부터 얻을 수 있다. 만약 $DPRC_i(1)$을 C_i''로, $DPRC_j(k)$를 C_j''로 바꿀 경우, DP-Accurate에서의 WCRT 식(식 10)은 DP-Pessimistic에서의 WCRT 식(식 9)과 동일하게 된다.

IV. 실험 결과

DP-Pessimistic 및 DP-Accurate의 정확성을 비교하기 위해, 본 논문에서는 가상의 태스크 집합에 대해 태스크 수행을 시뮬레이션 하였다. 이를 위해, 본 논문에서는 windows 환경에서 동작하는 시뮬레이터를 직접 제작하였다. (그림 6)은 실험에 사용한 시뮬레이터를 보인다. 이 시뮬레이터는 임의의 태스크 집합을 입력으로 받아 Shadowing, DP-Pessimistic 및 DP-Accurate 을 바탕으로 한 최악 응답 시간을 계산한다.

DP-Pessimistic 및 DP-Accurate 의 정확성을 비교하기 위해서는 입력 태스크 집합을 올바르게 정의하는 것이 중요하다. demand paging 환경에서 태스크의 최악 응답 시간에 영향을 미치는 매개변수에는 최악 page fault 처리 시간, 태스크의 실행 경로 별 최악 실행 시간, 페이지 적재 양상 등이 있는데, 이 중 DP-Pessimistic과 DP-Accurate의 차이를 직접적으로 반영하는 것은 페이지 적재 양상이다. 왜냐하면 DP-Pessimistic과는 달리 DP-Accurate에서는 페이지 적재 양상을 고려하기 때문이다. 따라서 본 실험에서는 태스크의 매개변수 중 페이지 적재 양상만을 다양하게 변화시킴으로써 DP-Pessimistic 과 DP-Accurate의 정확성을 비교한다.

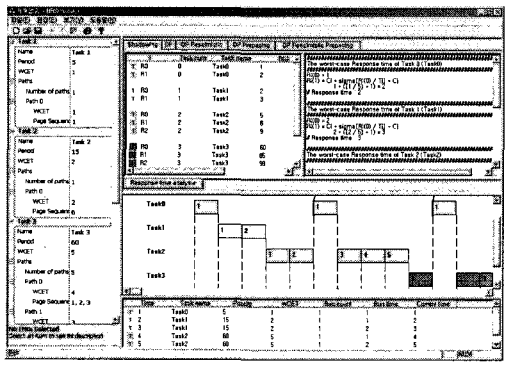

그림 6. 최악 응답 시간 분석기
Fig.6 WCRT analysis analyzer

DP-Pessimistic과 DP- Accurate의 차이가 가장 큰 경우는 특정 인스턴스의 DPRC값이 다른 인스턴스에 비해 매우 높을 때이다. 예를 들면 각 실행 경로의 최악 실행 시간은 비슷하지만 특정 실행 경로의 페이지 적재 빈도가 매우 높은 태스크가 이에 해당한다. 다시 말하면, 매 인스턴스마다 DPRC 값의 감소 폭이 커질수록 DP-Pessimistic과 DP-Exact 모델간의 차이가 더 뚜렷하게 나타난다. 반대로 DP-Pessimistic과 DP-Accurate의 차이가 가장 작은 경우는 각 인스턴스간의 DPRC값의 차이가 거의 없는 경우이다. 이를 바탕으로 본 실험에서 정의한 태스크 집합과 매개변수 정보가 (표 2)에 나타나 있다.

표 2. 태스크 집합 매개변수
Table 2. Task set parameters

Task set	T_i	# of paths	$C_{i,j}$	$P_{i,j}$
1	5	1	1	{45}
	15	1	2	{46}
	60	5	4	{1, 2}
			3	{8, 9}
			5	{15, 16}
			4	{24, 25, 26, 27, 28, 29}
			5	{33, 34, 38, 39}
	240	1	60	{47}
2	5	1	1	{45}
	15	1	2	{46}
	60	5	4	{1, 2}
			3	{8, 9}
			5	{15, 16}
			4	{24, 25, 26, 27, 28, 29, 30, 31}
			5	{38, 39}
	240	1	60	{47}
3	5	1	1	{45}
	15	1	2	{46}
	60	5	4	{1, 2}
			3	{8, 9}
			5	{15, 16}
			4	{24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34}
			5	{38, 39}
	240	1	60	{47}
4	5	1	1	{45}
	15	1	2	{46}
	60	5	4	{1, 2, 3}
			3	{8, 9, 10}
			5	{15, 16}
			4	{24, 25, 26, 27}
			5	{38, 39, 40}
	240	1	60	{47}

(표 3)은 각 태스크 집합에 대한 WCRT 분석 결과를 보여준다. (표 3)에서 실행 시간의 단위는 ms이다. 각 태스크 집합에 대해 DP-Pessimistic, DP-Accurate 그리고 shadowing 환경에서의 WCRT 예측 값이 나타나 있으며, DP-Pessimistic에 대한 DP-Accurate의 상대적인 정확도가 비율로 표시되어 있다. 먼저 태스크 집합 4의 경우 DP-Accurate과 DP-Pessimistic에 의한 최악 응답 시간이 비슷한 것으로 나타났는데 이는 태스크 집합 4에 속하는 각 태스크 간의 demand paging 비용이 비슷하기 때문이다. 반면 태스크 집합 2와 3의 경우 DP-Accurate와 DP-Pessimistic 간의 차이가 매우 큰 것으로 나타났는데, 이는 해당 태스크 집합이 다른 실행 경로에 비해 demand paging cost가 매우 큰 실행 경로를 포함하기 때문이다. 위의 실험 결과는 DP-Pessimistic이 DP-Accurate에 비해 지나치게 높은 최악 응답 시간을 도출할 수 있음을 보여준다. 그 결과 DP-Pessimistic 분석 기법은 태스크 집합이 스케줄 가능한지를 올바르게 결정하지 못할 가능성이 있다. 예를 들면, DP-Accurate 에서는 모든 태스크 집합이 데드라인을 만족시키는 것으로 나타난 반면, DP-Pessimistic에서는 태스크 집합 1, 2, 3이 데드라인을 만족시키지 못하는 것으로 나타났다.

표 3. 최악 응답 시간 분석 결과
Table. 3 WCRT analysis results

Task set	DP-Accurate	DP-Pessimistic	DP-Accurate / DP-Pessimistic	shadowing	deadline
1	235	280	83.9%	105	240
2	235	338	69.5%		
3	240	418	57.4%		
4	233	235	99.1%		

V. 결론 및 향후 과제

본 논문에서는 플래시 메모리를 사용하는 demand paging 환경에서의 태스크 최악 응답 시간 분석 기법을 제안하였다. 제안한 모델은 2가지 고려사항을 바탕으로 한다. 첫째는 태스크의 각 실행 경로에 따라 demand paging 비용이 달라진다는 점이고, 둘째는 같은 태스크 인스턴스라도 페이지 적재 양상에 따라 demand paging 비용이 달라질 수 있다는 것이다. 제안한 방법은 분석 기법의 정확도와 시간 복잡도에 따라 DP-Pessimistic과 DP-Accurate으로 나뉘며, 이 중 DP-Accurate은 페이지 적재 양상까지 고려한 분석 기법이다.

실험 결과 DP-Accurate이 DP-Pessimistic에 비해 더 정확한 결과를 나타냄을 보였다. 특히 이전 태스크 인스턴스에 의해 재사용되는 페이지의 비율이 높을수록 DP-Accurate이 DP-Pessimistic에 비해 최대 57% 정도 정확한 것으로 나타났다.

본 논문에서 제안한 방법은 다음과 같은 개선점이 있다. 첫째, 반복문과 같은 복잡한 구조를 가진 프로그램에 대한 demand paging 비용을 정확하게 계산할 수 있도록demand paging 비용 분석을 확장하는 것이다. 둘째, 실제 demand paging 환경을 반영할 수 있도록 DP-Accurate을 확장하는 것이다. 예를 들면, 페이지 교체 정책이나 공유 라이브러리 등을 고려할 경우, 실제 실시간 임베디드 시스템에서 적용 가능한 태스크 최악 응답 시간 모델을 얻을 수 있다. 셋째, 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.

참고문헌

[1] C.L. Liu, and J.W. Layland, "Scheduling Algorithms Multiprogramming in a Hard Real-Time Environment". Journal of the ACM, 20 (1), pages 46-61, 1973.

[2] J. Lehoczky, L. Sha, and Y. Ding. "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior". Proceedings of the 10th Real-Time Systems Symposium, pages 166-171, 1989.

[3] M. Joseph and P. K. Pandya, "Finding response times in a real-time system," The Computer Journal (British Computer Society), vol. 29, pages 390-395, Oct. 1986.

[4] A. Burns, "Preemptive priority based scheduling: An appropriate engineering approach," in Advances in Real-time Systems, pp. 225-248, Prentice-Hall International, Inc., 1994.

[5] P. K. Harter, "Response times in level-structured systems," ACM Transactions on Computer Systems, vol. 5, pp. 232-248, Aug. 1987.

[6] K. Tindell, "Adding Time-Offsets to Schedulability Analysis", Technical Report YCS 221, Dept. of Computer Science, University of York, England, January 1994.

[7] Palencia, J. C. and Gonz?lez Harbour, M. 1998. "Schedulability Analysis for Tasks with Static and Dynamic Offsets". In Proceedings of the IEEE Real-Time Systems Symposium, December, 1998.

[8] N.Audsley et al., "Applying new scheduling theory to static priority preemptive scheduling", Software Engineering Journal, pp. 284-292, September 1993.

[9] 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, June 1996.

[10] 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, pages 264-274, 1996.

[11] 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):133-151, March 1994.

[12] Scott F. Kaplan , Lyle A. McGeoch , Megan F. Cole.: Adaptive caching for demand prepaging, Proceedings of the 3rd international symposium on Memory management, June 20-21, 2002, Berlin, Germany

저 자 소 개

이 영 호

2005년 2월 : 국민대학교 컴퓨터학부 학사

2005년 ~ 현재 : 국민대학교 전산과학과 석사과정

관심분야 : 실시간 시스템, 운영체제, 컴파일러, 컴퓨터 음악

임 성 수

2004년3월~현재: 국민대학교 컴퓨터학부 조교수

2002년2월: 서울대학교 전기컴퓨터공학박사

2000년~2001년: 미국일리노이주립대 방문연구원

1995년2월: 서울대학교 컴퓨터공학 석사

1993년2월: 서울대학교 컴퓨터공학 학사

관심분야: 실시간임베디드시스템, 컴퓨터구조, 운영체제