객체지향 컴퓨터를 위한 확장-가중치 버디 시스템

김 관 중[†] · 김 병 기[††]

요 약

객체지향 컴퓨터의 동적 메모리 할당을 위해 가중치 버디 시스템을 확장한 확장-가중치 버디 시스템을 제안하였다. 가중치 버디 시스템이 블럭의 크기를 2^k와 $3*2^k$ 두가지로 제공하는데 비해 확장-가중치 버디 시스템은 블럭의 크기를 2^k, $3*2^k$, $5*2^k$, $7*2^k$등으로 제공한다. 이 확장은 각 블럭당 3비트의 추가 비용의 부담만으로 메모리 관리 장치를 실현할 수 있다.

그리고 본 기법과 가중치 버디 시스템을 비교 실험한 결과를 제시하였다. 확장-가중치 버디 시스템은 메모리 할당 요구 크기가 균일분포일 때 가중치 버디 시스템에 비해 내부단편화가 약 60% 정도 감소하였으며, 지수분포일 때 약 50% 정도 감소하였다. 외부단편화는 가중치 버디 시스템에 비해 본 논문에서 제안한 시스템이 크지만, 전체단편화는 지수분포시 확장-가중치 버디 시스템이 적으며 균일 분포시 두 시스템간에 큰 차가 없으므로 가중치 버디 시스템을 대체할 수 있다.

An Extended-Weighted Buddy System for an Object-Oriented Computer

Kwan-Joong Kim[†] · Byung-Gi Kim[††]

ABSTRACT

An extension of the weighted buddy system, called the extended-weighted buddy system, for dynamic memory allocation in an object-oriented computer is presented. The extended-weighted buddy system allows block sizes of 2^k, $3*2^k$, $5*2^k$, $7*2^k$, whereas the original weighted buddy system allowed block sizes of 2^k and $3*2^k$. This extension is achieved at only the cost of additional 3 bits per block for memory management unit.

Simulation results are presented which compare our method with the weighted buddy system. These results indicate that, for uniform request distributions, our system has less internal memory fragmentation than the weighted buddy system(approximately 60%). And, for exponential request distributions, it has less internal memory fragmentation than the weighted buddy method (approximately 50%). The external fragmentation is greater for this system than the weighted buddy system. But, our system has less total memory fragmentation for exponential request distributions, and two systems take a similar total memory fragmentation for uniform request distributions, so we can substitutes the extended-weighted buddy system for weighted buddy system.

† 정 회 원:한서대학교 전산학과
†† 정 회 원:숭실대학교 컴퓨터학부
논문접수:1997년 3월 11일, 심사완료:1997년 6월 5일

1. 서 론

객체지향 프로그래밍은 다형성(polymorphism), 캡슐화(encapsulation), 메시지 전송(message passing), 상속(inheritance), 동적 바인딩(dynamic binding)등의 특성으로 인해 새로운 프로그래밍 패러다임으로 자리잡고 있다. 객체지향 프로그래밍이 제공하는 잇점으로는 재사용성(reusability), 확장성(extensibility), 모듈성(modularity)등을 통한 소프트웨어의 질적 향상에 있다[1]. 그러나 객체지향 프로그램을 기존의 von Neumann 구조에서 실행시킬 때 객체의 병렬적 수행 특성과 순차처리에 적합한 하드웨어 사이의 차이로 인해 실행효율이 떨어진다[2, 3, 4]. 실행시간의 저하를 가져오는 오버헤드로는 메소드 열람(method lookup), 메모리 관리, 문맥 할당(context allocation)등이 있다.

특히 소프트웨어를 실행하기 위한 하드웨어 자원에 대한 실행속도를 측정하기 위한 요소로서 프로그래머의 작성시간, CPU의 처리시간, 메모리의 페이지 실패율(page fault rate), 가상 메모리 관리시간등 4가지 중에서 후자 3가지는 객체지향 언어를 구현하는데 전형적인 결점이 된다[5]. 이러한 관점하에서 객체의 특성에 맞는 메모리 관리 장치에 관한 연구[2, 6, 7]가 이루어져 왔으며, 시스템 객체 집합과 이를 처리할 명령을 제공하기 위한 객체지향 컴퓨터 구조의 연구도 진행되어 왔다. 대표적인 시스템으로는 Plessy S/250, Intel iAPX 432[8], IBM S/38[9], Caltech Object Machine(COM)[3]등이 있다.

객체지향 시스템에서 효율적인 메모리 관리를 위해 고려해야 할 특성은 객체의 크기와 그 수이다[2, 3, 7]. 객체지향 시스템의 경우 객체의 크기는 매우 작은 반면 그 수는 수천에서 수만개에 이르기 때문에 동적 메모리 할당을 위해서는 가능한 다양한 크기의 블럭을 필요로 한다. 하나의 객체를 하나의 세그먼트에 할당하는 것은 객체별로 액세스 권한 및 보호를 수행할 수 있으며, 다양한 세그먼트에 의해 메모리 이용률을 높일 수 있다. 2진 버디(binary buddy) 시스템은 주소결정 기능을 하드웨어로 간단하게 구성할 수 있고, 빠른 시간내에 할당과 회수를 수행할 수 있는 반면에 내부단편화가 크다는 문제점을 갖고 있다[6, 10]. 외부단편화는 크게 중요하지 않으나 내부단편화는 전체 메모리의 1/3 에서 1/4 정도를 낭비하게 되므로 내부단편화의 감소는 필수적이다[10].

이와 같은 관점하에서 본 논문에서는 객체지향 컴퓨터 구조의 연구를 위해 그 기초 이론을 제시할 목적으로 객체의 특성에 보다 적합하도록 다양한 크기의 블럭을 제공할 수 있는 가중치 버디 시스템[11]의 확장 방법을 제시하고 실험을 통해 그 성능을 분석하였다. 본 논문에서 제시한 확장-가중치 버디 시스템은 기존의 가중치 버디 시스템에 비해 메모리 할당 요구 크기가 균일분포일 때 약 60%, 지수분포일 때 약 50% 정도 내부단편화를 감소시킬 수 있다. 그리고 본 시스템은 [12]이 제안한 비트맵(bit-map) 기법에 의한 메모리 관리기 하드웨어 구현 방법을 도입하여 하드웨어 메모리 할당기를 설계하면 보다 향상된 메모리 관리 장치를 구현할 수 있다.

2장에서는 객체 특성에 적합한 메모리 관리 장치에 대한 관련 연구와 2진 버디 및 가중치 버디 시스템에 대해 고찰하고, 3장에서는 가중치 버디 시스템의 확장 방법에 대해 설명한다. 4장에서는 실험결과를 제시하고 성능을 분석하며, 5장에서는 결론을 맺고 향후 연구 방향에 대해 설명한다.

2. 배 경

2.1. 객체의 특성 및 메모리 관리 장치

객체지향 컴퓨터 구조의 기본 아이디어는 객체 개념에 있다. 이 객체에 대해서 주소 장치의 구성 방법, 액세스 제어 기법, 명령어 처리 기법 등의 문제를 다루어야 한다. 객체는 추상 자료형(abstract data type)의 소프트웨어 개념에 대한 구조적인 구성 대응체(architectural counterpart)로서 소형이면서 대량으로 존재하게 된다. 그리고 소량이라 하더라도 크기는 대형인 객체도 존재한다. 일반적으로 잘 알려진 객체지향 시스템의 객체 크기는 표 1[7]과 같다. 그리고 GNU Smalltalk의 상용 객체 크기 및 수는 표 2와 같다. 표에서 보는 바와 같이 객체의 평균 크기는 약 300바이트 정도가 되며, 그 수는 수천개에 이르고 있다.

이와 같은 특성의 객체에 대한 기존의 전통적인 페이징이나 세그먼테이션 기법을 사용한 가상메모리 공간 관리는 비효율적이다. 전통적인 페이징이나 세그먼테이션 방식은 고정된 폭의 페이지(또는 세그먼

〈표 1〉 객체의 평균 크기
〈Table 1〉 Mean object size

시스템	평균 객체 크기(bytes)
Berkeley Smalltalk	50
Hydra	326
iMAX	약 300

〈표 2〉 객체의 크기별 분포(GNU Smalltalk)
〈Table 2〉 Object size distribution(in GNU Smalltalk)

객체 크기	8	16	32	64	128	256	512	1024
객체 수	103	3410	2454	1503	123	1921	37	4

테이션)와 가지거리(offset)를 사용하기 때문에 다양한 크기의 대량 객체에 대한 주소지정을 위해 많은 비트를 할애하여야 한다. 즉, 대량의 객체를 수용하기 위해서는 세그먼트(또는 페이지) 필드가 커야 하고, 대형의 객체를 허용하기 위해서는 가지거리 필드가 커야 한다. [3]은 이와 같은 문제를 '작은 객체 문제(small object problem)'이라 하고, 이를 해결하기 위해 COM 프로젝트에서 부동소수점 주소(floating-point addressing) 방식을 제안하였다.

특히 객체는 독립적으로 생성될 수 있고 또한 소멸될 수 있기 때문에 메모리 할당 요구가 매우 빈번하게 발생하며, 작은 크기의 매우 많은 단편화 현상이 초래되고, 객체 자체가 메모리 관리의 단위가 되어야 하기 때문에 페이징 시스템은 적합한 기법이 못 된다[13]. 따라서 객체의 크기가 하나의 세그먼트가 될 수 있는 세그먼테이션 시스템이 객체의 액세스 제어에도 효율적이므로 세그먼테이션에 의한 메모리 관리 장치를 구현해야 한다. 대부분의 경우 액세스 제어에는 capability[14]에 의한 방법을 적용하고 있다. Intel iAPX 432는 객체의 주소지정을 위해 가상메모리를 가변 크기의 논리적 세그먼트로 구성하며 보다 속도 개선을 위해 주소 변환시 on-chip 캐시를 사용한다.

IBM S/38은 가상메모리를 64K 바이트 크기의 세그먼트로 분할하고, 이 세그먼트를 256개씩 통합하여 세그먼트 그룹으로 운영하고 있다. 이 세그먼트 그룹에 여러개의 객체를 할당한 후 시스템 포인터(가상주소)를 사용하여 각 객체를 액세스한다. 가상메모리가 매우 커서 동적 주소 변환시 탐색 속도가 떨어지므로 2단계 해싱 기법을 사용한다.

2.2. 2진 버디 시스템

버디 시스템[10]은 메모리를 동적으로 할당하기 위한 알고리즘으로서 자유 메모리 블럭을 필요한 크기만큼 분할하여 사용하고, 사용이 끝난 후 분할되었던 두 블럭은 필요에 따라 다시 결합하여 사용한다. 이때 분할된 두 개의 블럭을 버디라 부른다. 기본적인 규칙은 다음의 Hirschberg 일반화 공식으로 살펴 볼 수 있다.

$$L(i) = L(i-1) + L(i-n),\ n > 0$$

여기서, L(i)는 단계 i의 블럭 크기를 나타내고 n은 분리계수(splitting order)이다. 이 n값을 어떻게 결정하느냐에 따라 몇가지 알고리즘이 발생된다. 예를 들어, n=1이면 2진 버디 시스템이 된다. 즉, L(i)=L(i-1)+L(i-1)로써 블럭의 크기는

1, 2, 4, 8, 16, 31, 64, 128,...

의 순열이 된다. 그리고 n=2이면 피보나치 버디 시스템이 되므로 블럭의 크기는

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 79,...

의 순열이 된다.

이와 같은 특성 때문에 분리계수의 크기가 클수록 다루는 블럭의 크기는 작게 된다. 이 중에서 n=1인 2진 버디 시스템에 대해 고찰해 본다. 2진 버디 시스템은 메모리 할당 요구 크기가 2^k일 때, 사용가능 블럭이 없을 경우 2^{k+1} 크기의 사용가능 블럭을 찾아 이를 반으로 나누어 이 중의 한 버디를 사용한다. 2진 버디 시스템의 특성은 한 버디의 주소를 알고, 이 버디의 크기를 알고 있을 때 다른 한 버디의 주소를 쉽게 계산해 낼 수 있다는 것이다.

2진 버디 시스템에서 2^k 크기 버디 블럭의 주소는 2^k의 배수가 된다. 이를 2진수로 표현하면 오른쪽에

적어도 k개의 0을 갖는 것이 된다. 그러므로 크기가 32인 블럭의 주소 형태가 xx...x00000(x는 0이거나 1)일 때 이것을 반으로 나누는 경우 각 버디의 크기는 16이므로 주소는 xx...x00000과 xx...x10000이 된다. 예를 들어 한 버디 블럭의 크기가 16이고 시작주소가 2진수로 101110010110000이라면 다른 한 버디 블럭의 시작주소는 101110010100000이 된다. 따라서 버디의 주소는 다음과 같이 계산해 낸다[10].

$buddy_k(x) = x + 2^k$, x mod $2^{k+1} = 0$ 일 때,
$= x - 2^k$, x mod $2^{k+1} = 2^k$ 일 때.

2진 버디 시스템은 위의 식을 "exclusive OR" 명령만으로 수행할 수 있기 때문에 주소결정 기능을 하드웨어로 간단하게 구성할 수 있으며, 빠른 시간내에 할당과 회수를 수행할 수 있다. 반면에 버디 블럭의 크기를 2^k으로 제한하기 때문에 내부단편화가 크다는 결점이 있다. [10]에 의하면 외부단편화는 크게 중요하지 않으나 내부단편화는 전체 메모리의 1/3에서 1/4 정도를 낭비하게 된다고 하므로 내부단편화의 감소는 필수적이다.

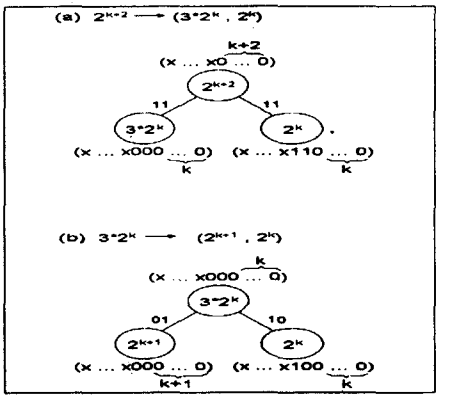
2.3. 가중치 버디 기법

[11]은 2진 버디 시스템에서 발생하는 많은 양의 내부단편화 현상을 감소시키기 위해 블럭의 크기를 2^k 외에 2^k에 가중치(weight)를 곱한 크기의 블럭을 제공하는 기법을 제안하였다. 이를 가중치 버디 시스템이라 하는데 블럭 분할식을

$B_s = c*2^k$, $0 \le k \le m$

이라 할 때, c값을 1과 3으로 제한한 것이다. 여기서 m은 전체 메모리 크기에 대한 지수값이다. c값이 1이라면 블럭의 크기는 2^k, $0 \le k \le m$이 되며, 3이면 블럭의 크기는 $3*2^k$, $0 \le k \le m-2$가 된다. 따라서 할당 가능한 블럭의 크기는 1, 2, 3, 4, 6, 8, 12,... 가 된다.

가중치 버디 시스템의 블럭 분할에 대한 개념은 그림 1과 같다. 그림 1(a)는 2^{k+2} 블럭이 $3*2^k$ 크기의 블럭과 2^k 크기의 블럭으로 분할되는 것이고, (b)는 $3*2^k$ 크기의 블럭이 2^{k+1} 크기의 블럭과 2^k 크기의 블럭으로 분할되는 것이다. 각 아크에 표시된 값은 분할되



(그림 1) 가중치 버디 시스템의 블럭 분할
(Fig. 1) Block splitting in the weighted buddy system

는 버디의 크기에 따라 주소를 계산하기 위한 TYPE 필드 값이다. 이 TYPE 필드는 각 블럭에 2 비트로 구성되는 데 코드 값은 다음과 같이 결정한다[11].

TYPE(p) = 11, 주소가 p인 블럭이 2^k 크기의
블럭으로부터 분할된 경우,
= 01, 주소가 p인 블럭이 $3*2^k$ 크기의
블럭으로부터 왼쪽으로 분할된 경우,
= 10, 주소가 p인 블럭이 $3*2^k$ 크기의
블럭으로부터 오른쪽으로 분할된 경우.

가중치 버디 시스템은 위의 TYPE 필드 정보를 이용하여 각 버디 블럭의 주소를 다음과 같이 계산한다[11].

$WB_k(x) = x + 3*2^k$, x mod $2^{k+2} = 0$이고
TYPE(x) = 11인 경우,
$= x - 3*2^k$, x mod $2^{k+2} = 3*2^k$이고
TYPE(x) = 11인 경우,
$= x + 2^{k+1}$, TYPE(x) = 01인 경우,
$= x - 2^{k+1}$, TYPE(x) = 10인 경우.

3. 가중치 버디 기법의 확장

2장에서 살펴본 바와 같이 객체지향 시스템에서 객체는 소형이면서 대량으로 존재하며, 소량이라 하더라도 대형인 객체도 존재한다. 이들 객체를 단위로

기억장치에 사상하기 위해서는 다양한 크기의 블럭이 생성될 수 있어야 하며, 하나의 객체가 하나의 세그먼트에 사상될 수 있어야 한다.

하나의 객체 및 객체의 일부가 독립적으로 생성되고 소멸되기 때문에 메모리 할당요구가 매우 빈번하게 발생되므로 단순하면서도 빠른 동적 메모리 할당 기법이 필요하다. 2진 버디 시스템은 하드웨어 구현이 간단하며, 실행 속도가 빠른 반면에 내부단편화가 크다는 단점이 있다. 내부단편화를 감소시키기 위해서는 생성되는 블럭의 크기를 되도록 작게 만들면 된다. [13]은 작은 객체를 다루기 위해서는 Hirschberg 일반화 공식에서 분리계수 n을 9 또는 10으로 결정하는 것이 바람직하다고 지적하고 있다.

3.1. 확장 방법

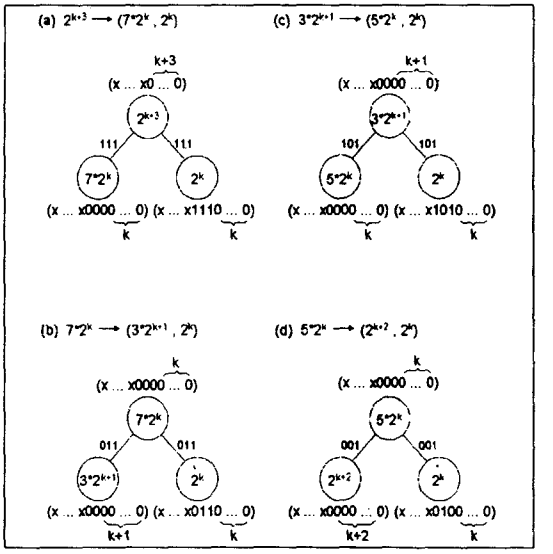
2.3절의 블럭 분할식에서 c의 값은 1과 3이다. 보다 다양한 크기의 블럭을 제공하기 위해서는 c의 값을 1과 3외의 다른 수를 적용하면 된다. 확장의 개념은 2진 버디를 기준으로 하였을 때 각 2^k와 2^{k+1} 크기의 블럭 사이에 새로운 c값을 곱한 크기의 블럭이 모두 존재해야 한다. 따라서 생성 가능한 블럭의 크기는 2^k, $c*2^{k-x}(0<x)$, 2^{k+1}의 순서로 성립되어야 한다. 그리고 블럭의 크기는 2의 승수 체계로 결정되므로 가중치 c값은 반드시 소수(prime)이어야 한다. 그러므로 확장을 위한 가중치 c의 결정조건은 다음과 같이 요약하게 된다.

(1) 3 이상의 소수,
(2) 각 2^k와 2^{k+1} 사이에 $c*2^{k-x}$가 모두 존재(단, $2 \le k$, $0 < x \le k$).

위의 두 조건에 의해 적용 가능한 가중치 c값은 {3, 5, 7}이 된다. k가 2인 경우 $3*2^{k-1}$은 6이며, $5*2^{k-2}$는 5이고, $7*2^{k-2}$는 7이므로 위의 두번째 조건을 만족한다. k가 3인 경우 $3*2^{k-1}$은 12이고, $5*2^{k-2}$는 10이며, $7*2^{k-2}$는 14이므로 역시 두 번째 조건을 만족한다. 즉 22과 23 사이에 확장한 크기의 블럭이 모두 존재한다. 따라서 사용 가능한 블럭의 크기는 2, 4, 5, 6, 7, 8, 10, 12,...이 된다. 반면에 첫 번째 조건을 만족하는 11의 경우를 보자. k가 2일 때 $11*2^{k-2}$는 11이 되므로 22과 23의 범위를 벗어나기 때문에 두 번째 조건을 만족할 수 없다. 그러므로 11이상의 소수는 가중치로 적용할 수 없다. 이와 같이 가중치 버디 시스템의 가중치 c값을 확장하였기에 이를 확장-가중치 버디 시스템이라 칭한다.

확장-가중치 버디 시스템의 블럭 분할 개념은 그림 2와 같다. 그림 2(a)는 2^{k+3} 크기의 블럭이 $7*2^k$ 크기의 버디와 2^k 크기의 버디로 분할되는 것이고, 그림 2(b)는 $7*2^k$크기 블럭이 $3*2^{k+1}$ 크기의 버디와 2^k 크기의 버디로 분할되는 것이다. 그림 2(c)는 $3*2^{k+1}$ 크기의 블럭이 $5*2^k$ 크기의 버디와 2^k 크기의 버디로 분할되는 것이고, 그림 2(d)는 $5*2^k$ 크기의 블럭이 2^{k+2} 크기 버디와 2^k 크기의 버디로 분할되는 것이다. 가중치 버디 시스템을 확장한 것이므로 모든 특성은 그대로 유지한다.

그림 2에서 각 아크에 표시된 값은 버디의 주소 계산을 위한 TYPE값으로써 가중치 c값을 2진수로 표현하여 사용하였다. 전체 메모리의 크기를 2^m, $m \ge 5$라 하였을 때, 발생 가능한 블럭의 크기 종류 수는 $4*m-9$가 된다. 가중치 버디 시스템은 $2*m$개가 되기 때문에 m이 충분히 크다면 가중치 버디 시스템보다 약 2배 정도 다양한 크기의 버디 블럭을 제공할 수 있다.



(그림 2) 확장-가중치 버디 기법의 블럭 분할
(Fig. 2) Block splitting in the extended-weighted buddy system

확장-가중치 버디 시스템은 할당 가능한 블럭의 크기를 가능한 객체의 크기와 같도록 하여 내부단편화를 감소시키기 위한 것이다. 그러나 분할되는 버디 블럭의 크기 종류수가 증가하면 할수록 실행 시간(분할 시간, 탐색 시간, 결합 시간등)은 증가한다. 확장-가중치 버디 시스템의 성능을 보다 향상시키기 위해 8바이트 미만의 버디 블럭이 생성되지 않도록 해야 한다. 확장-가중치 버디 시스템은 왼쪽 분할 버디 블럭의 크기와 오른쪽 분할 버디 블럭의 크기가 서로 다르다. 그리고 오른쪽 분할 버디 블럭의 크기가 왼쪽 분할 버디 블럭의 크기보다 항상 작다. 이 특성에 따라 다음의 조건을 추가 적용하면 8바이트 미만 블럭의 생성을 방지할 수 있다.

(1) 블럭의 최소 크기는 2^3,
(2) 외쪽 분할 블럭의 최소 크기는 2^5,
(3) 2^5이하의 블럭은 2진 버디 기법 적용.

첫 번째 조건은 블럭을 분할할 경우 블럭의 크기를 8바이트 까지만 생성되도록 하는 것이다. 두 번째 조건은 왼쪽 버디 블럭의 크기가 8바이트 보다 크더라도 이 블럭이 분할되면 오른쪽 버디 블럭의 크기가 8바이트 보다 작은 경우가 발생하므로 이를 제거하기 위한 것이다. 예를 들어 $5*2^3$ 블럭을 분할하면 왼쪽 버디 블럭은 2^5이고 오른쪽 버디 블럭은 2^3이다. 2^5 크기의 왼쪽 블럭을 다시 분할하면 왼쪽 버디 블럭은 $7*2^2$이고 오른쪽 버디 블럭은 2^2이다. 항상 2^5 크기 이하의 블럭을 분할했을 경우 8바이트 미만의 블럭이 생성되므로 왼쪽으로 분할되는 블럭의 크기를 32바이트로 제한시키면 오른쪽으로 분할되는 블럭의 최소 크기는 8이 된다.

세번째 조건은 왼쪽으로 분할된 32바이트 블럭을 더 작은 크기의 블럭으로 분할하기 위한 것이다. 32바이트 블럭을 확장-가중치 버디 기법에 따라 분할하면 두 번째 조건의 설명에서 본 바와 같이 8바이트 미만의 블럭이 생성된다. 따라서 32바이트 블럭을 더 분할하기 위해서는 2진 버디 기법을 적용하면 된다.

3.2. 주소 표현 및 계산

그림 2와 같이 분할되는 확장-가중치 버디 시스템의 각 블럭 주소는 기존의 버디 시스템이나 가중치 버디 시스템과 마찬가지로 2진수로 표현했을 때, 오른쪽 최하위 비트로부터 $k+x(0 \le x)$개의 0을 갖는다 ($k+x$는 블럭 크기 지수값). 그림 2의 블럭 분할에 대응되는 주소 표현에서 특히 주목할 만한 점은 Ⅲ장 1절에서 언급한 것처럼 주소 계산을 위한 TYPE값을 가중치 c와 일치하도록 했는데, 이것은 오른쪽 버디의 주소에서 $k+x$개의 0 다음에 있는 3개의 비트값과도 일치한다. 버디 블럭의 실제 주소 표현 속에 TYPE값이 사상되어 있는 것이므로 오른쪽 버디의 짝인 왼쪽 버디의 주소 계산은 보다 간단하게 구현할 수 있다.

이제 버디의 주소를 계산하는 방법을 보자. 어느 한 블럭의 크기와 주소가 주어졌을 경우, 이 블럭의 버디 주소 계산식은 Ⅱ장 3절에서 기술한 식을 본 확장-가중치 버디 시스템의 주소 표현 특성에 적용하면 된다. 즉, 다음의 식을 사용하여 버디의 주소를 계산한다.

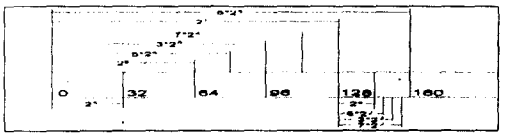
$$
\begin{aligned}
EWBk(x) &= x+7*2^k,\ x \bmod 2^{k+3}=0\ \&\ TYPE(x)=111,\\
&= x-7*2^k,\ x \bmod 2^{k+3}=7*2^k\ \&\ TYPE(x)=111,\\
&= x+3*2^k,\ x \bmod 2^{k+2}=0\ \&\ TYPE(x)=011,\\
&= x-3*2^k,\ x \bmod 2^{k+2}=3*2^k\ \&\ TYPE(x)=011,\\
&= x+5*2^k,\ x \bmod 2^{k+3}=0\ \&\ TYPE(x)=101,\\
&= x-5*2^k,\ x \bmod 2^{k+3}=5*2^k\ \&\ TYPE(x)=101,\\
&= x+2^{k+2},\ TYPE(x)=001,\\
&= x-2^{k+2},\ TYPE(x)=010.
\end{aligned}
$$

위의 주소 계산식은 블럭을 분할하는 방식에 따라 결정되는 버디의 주소(버디의 첫 단어 주소) 형식에 다음과 같은 특성이 포함되어 있기 때문에 성립된다.

(1) 2^k 크기 버디 블럭의 주소는 2^k의 배수,
(2) $7*2^k$ 크기 버디 블럭의 주소는 2^{k+3}의 배수,
(3) $3*2^k$ 크기 버디 블럭의 주소는 2^{k+2}의 배수,
(4) $5*2^k$ 크기 버디 블럭의 주소는 2^{k+3}의 배수.

이와 같은 특성은 다음 그림 3으로 설명할 수 있다. 2^m 크기의 블럭을 분할하는 경우 첫 분할의 왼쪽 버디 블럭의 주소는 0이다. 이 왼쪽 버디 블럭을 분할하는 경우에도 왼쪽 버디 블럭의 주소는 부모 블럭의 주소와 동일한 0이다. 따라서 블럭의 주소가 0인 왼

쪽 버디의 짝인 오른쪽 버디의 주소는 0에 왼쪽 버디의 크기를 더한 값이 된다. 반대로 오른쪽 버디의 짝인 왼쪽 버디의 주소는 오른쪽 버디의 주소값에서 왼쪽 버디의 크기를 빼면 된다.



(그림 3) 확장-가중치 버디 기법의 주소 특성
(Fig. 3) Address feature in the extended-weighted buddy system

그리고 그림 3에서 보는 바와 같이 가중치를 곱한 크기의 모든 블록은 2^k 크기 블럭내에 존재하며, 이들의 시작 주소는 2^k의 배수가 된다. 따라서 이 주소값을 2^k로 나누어 그 나머지가 0이면 앞서 설명한 거처럼 왼쪽 버디의 크기 값을 더하면 되고, 반대로 나머지가 $c*2^{k-x}$, $x>0$가 된다면 이를 빼면 된다.

4. 실험 및 결과

4.1. 실험 배경

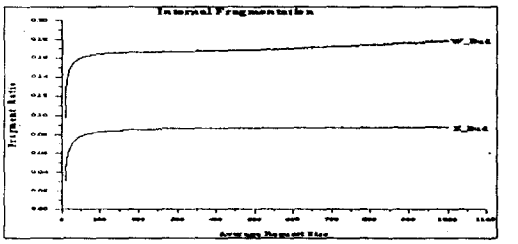
실험은 본 논문에서 제시한 확장-가중치 버디 시스템과 가중치 버디 시스템을 중심으로 수행한다. [11]에서 사용한 실험 알고리즘도 Knuth[9]가 제시한 것이므로 타당한 실험 결과를 보이기 위해 본 논문에서도 Knuth의 알고리즘을 기초로 수행한다. 실험을 위한 전체 메모리의 크기는 2^{20}(1 Mbytes)로 하였다.

객체 생성시 메모리 할당 요구가 발생하면 메모리를 분할하여 할당해야 하므로 블럭 할당 요구 크기는 난수를 사용한다. 생성되는 난수의 크기는 최소 크기를 8로 하고, 최대 크기는 2500으로 제한하였다. 난수의 생성 방법은 [11]과 마찬가지로 균일분포와 지수분포 두 가지로 한다. 객체의 메모리 점유시간 - 즉, 생존기간 역시 난수를 사용한다. 이 난수는 객체 크기를 생성하는 난수와는 별도로 구성하며 범위는 1 부터 100000까지 제한한다. 확장-가중치 버디 기법은 생성되는 블럭의 크기 종류가 많기 때문에 메모리 할당 오버플로우 발생 간격이 크므로 10만 단위까지 설정한 것이다.

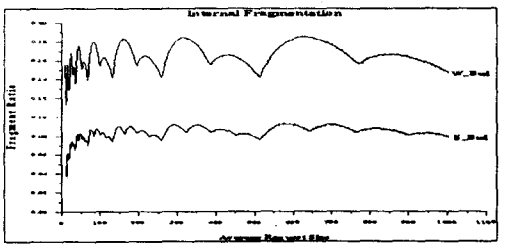
메모리 할당 오버플로우 발생시 내부단편화와 외부단편화를 산출하여 전체 메모리에 대한 손실비를 계산한다. 할당된 객체의 크기는 지수분포와 균일분포로 생성되기 때문에 오버플로우가 발생한 시점까지 할당된 객체의 평균을 계산하여 이를 평균 요구 크기라 하고, 전체 메모리에 대한 손실비 그래프의 X 축 값으로 사용한다. 1 회의 오버플로우만으로는 타당한 결과를 얻을 수 없으므로 1000회를 수행한 후 평균치를 계산하여 그래프로 표현한다. 실행시간 비용도 단편화 산출과 마찬가지로 메모리 할당 오버플로우 발생 시점에서 그동안 일어난 블럭의 분할횟수, 사용가능 리스트의 탐색횟수를 산출한다. 물론 이 값은 매 객체를 할당할 때 발생한 것이므로 오버플로우 발생 시점까지의 평균치이다.

4.2. 결 과

메모리 이용률은 내부단편화와 외부단편화에 의한 손실량에 따라 결정된다. 내부단편화는 가능한 요구 크기에 맞는 블럭을 제공해 줄수록 감소하는 반면, 외부단편화는 할당 불가능한 작은 블럭의 갯수가 많을 수록 증가하게 된다. 그림 4는 할당 요구 크기가



(그림 4) 지수분포시 내부 단편화
(Fig. 4) Internal fragmentation for an exponential request distribution.



(그림 5) 균일분포시 내부 단편화
(Fig. 5) Internal fragmentation for an uniform request distribution.

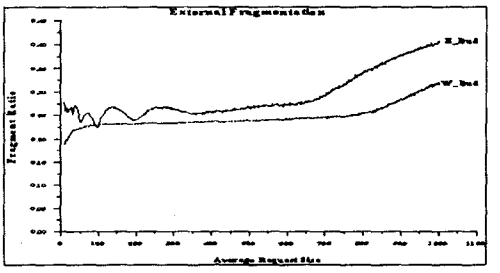
지수분포일 때 전체 메모리에 대한 내부단편화의 비율이다. 가중치 버디 시스템에 비해 확장-가중치 버디 시스템이 약 50% 정도 내부단편화를 감소시킬 수 있음을 볼 수 있다.

그림 5는 할당 요구 크기가 균일분포일 때 전체 메모리에 대한 내부단편화의 비율로서 지수분포와는 다르게 각 각의 블럭 분할 방식을 반영하여 산과 계곡이 반복되어 나타나고 있다. 가중치 버디 시스템보다 분할되는 블럭의 수가 많을 수록 산과 계곡의 차가 줄어 들며 그 수는 늘어 나고 있다. 가중치 버디 시스템에 비해 확장-가중치 버디 시스템이 약 60% 정도 내부단편화를 감소시킬 수 있음을 볼 수 있다.

그림 6과 7은 할당 요구 크기가 지수분포일 경우와 균일분포일 경우의 외부단편화 비율이다. 지수분포의 경우는 가중치 버디 시스템과 확장-가중치 버디 시스템이 큰 차이를 보이지 않는 것에 비해 균일분포의 경우는 확장-가중치 버디 시스템에서 많은 외부단편화 현상을 보이고 있다. 이 같은 현상은 지수분포시 비교적 같은 크기의 객체가 할당되기 때문이고, 균일 분포시에는 연속된 이웃 블럭에 할당되지 못하기 때문이다. 사용 가능 블럭 관리시 링크리스트를 주소 크기에 따라 정렬시키는 알고리즘을 구현하면 외부단편화를 더욱 감소시킬 수 있다.

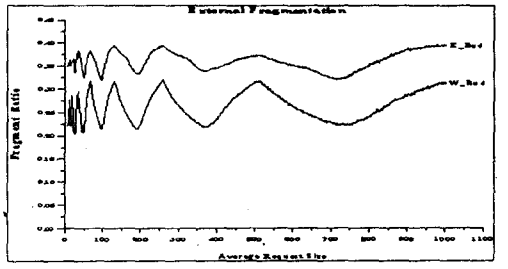
다음의 그림 8과 9는 전체단편화의 그래프로서 내부단편화와 외부단편화의 단순한 합으로 표현할 수 없기 때문에 [10]과 [11]의 방법과 마찬가지로 {(외부단편화+내부단편화)-내부단편화*외부 단편화} 식을 적용하여 구성하였다. 지수분포시 전체단편화는 가중치 버디 시스템보다 확장-가중치 버디 시스템이 우수하다. 이것은 내부단편화의 감소에 비해 외부단편화 현상이 그리 크지 않았기 때문이다. 균일분포의 경우 두 시스템간에 큰 차이는 나지 않는다. 결론적으로 내부단편화는 약 50~60% 감소하였고, 전체단편화 측면에서도 가중치 버디 시스템에 비해 큰 차를 보이지 않으므로 객체지향 컴퓨터를 위한 동적 메모리 관리 장치 구현에 활용될 수 있다.

그림 10과 11은 각 각 지수분포시 평균 분할횟수와 균일분포시 평균 분할횟수를 나타낸 것이다. 확장-가



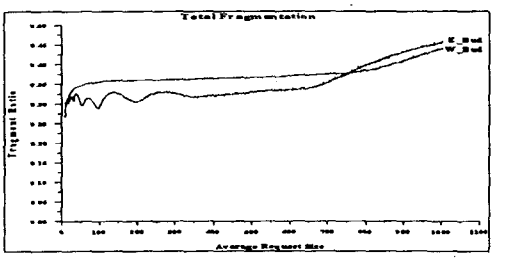
(그림 6) 지수분포시 외부 단편화

(Fig. 6) External fragmentation for an exponential request distribution.



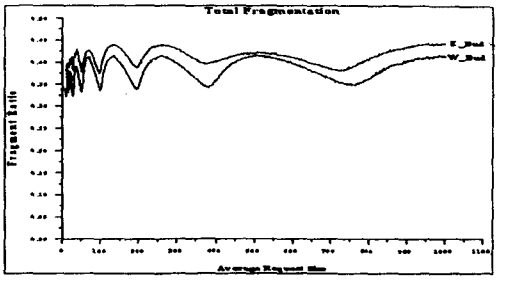
(그림 7) 균일분포시 외부 단편화

(Fig. 7) External fragmentation for an uniform request distribution.



(그림 8) 지수분포시 전체단편화

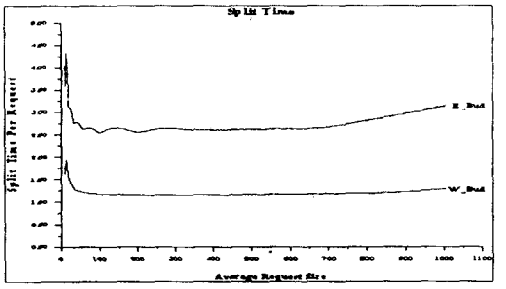
(Fig. 8) Total fragmentation for an exponential request distribution.



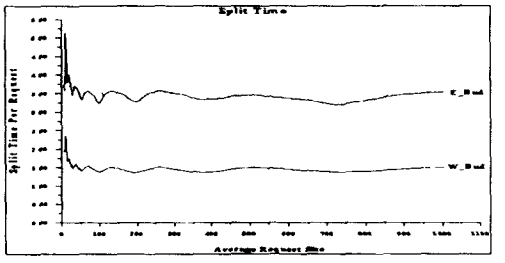
(그림 9) 균일분포시 전체단편화

(Fig. 9) Total fragmentation for an uniform request distribution.

중치 버디 시스템이 가중치 버디 시스템보다 생성하는 블럭의 크기 종류수가 약 두 배 정도 다양하므로 블럭 분할횟수도 약 두 배 정도 많이 발생함을 볼 수 있다.

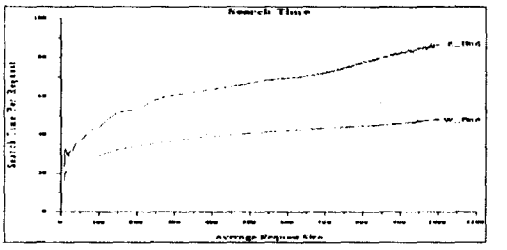


(그림 10) 지수분포시 평균 분할횟수
(Fig. 10) Average number of splitting for an exponential request distribution.

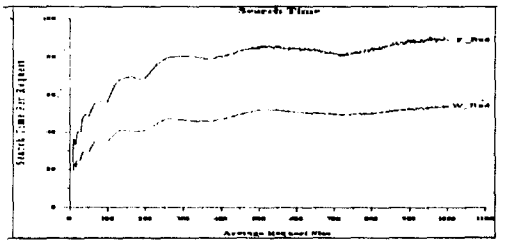


(그림 11) 균일분포시 평균 분할횟수
(Fig. 11) Average number of splitting for an uniform request distribution.

그림 12와 13은 각 각 지수분포시 사용가능 블럭 평균 탐색횟수와 균일분포시 평균 탐색횟수를 나타낸 것이다. 블럭 분할횟수의 경우와 마찬가지로 두 배 정도 많이 탐색함을 볼 수 있다. 가중치 버디 시스템은 지수분포시 약 1.25회, 균일분포시 약 1.5회 분할을 수행하며 확장-가중치 버디 시스템은 지수분포시 약 2.5회, 균일분포시 약 3.5회 수행한다. 이것은 할당 요구의 평균 크기당 평균 분할횟수를 의미한다. 즉, 할당 요구의 평균 크기가 100바이트라 할 경우 이 100바이트 크기의 블럭을 생성하기 위해 분할하는 평균횟수인 것이다. 메모리 할당을 시작한 초기에는 필요한 크기의 블럭을 생성하기 위해 분할을 반복해야 하기 때문에 분할횟수가 크다. 그러나 안정된 상태가 되면 만족한 크기의 블럭을 생성하기 위한 분할횟수와 사용이 끝난 버디 블럭의 재결합횟수는 같게 되므로 재결합횟수는 분할횟수를 참고로 하면 된다.



(그림 12) 지수분포시 탐색 횟수
(Fig. 12) Average number of searches for an exponential request distribution.



(그림 13) 균일분포시 탐색 횟수
(Fig. 13) Average number of searches for an uniform request distribution.

이들 알고리즘의 실행시간은 분할과 재결합횟수에 비례하게 되는 데 확장-가중치 버디 시스템이 가중치 버디 시스템에 비해 약 2.5배 증가한다. 이것은 할당 요구와 폐기 메카니즘의 진행 상태에 따라 증감하는 것이며, 전체적인 컴퓨팅 시간의 아주적은 부분임을 고려한다면 동적 메모리 할당에 확장-가중치 버디 시스템을 활용할 수 있다.

5. 결 론

보다 다양한 크기의 블럭을 제공하여 내부단편화를 감소시키고자 가중치 버디 시스템을 확장하였다. 본 확장-가중치 버디 시스템은 객체의 크기가 평균적으로 작으며 그 수는 수천개 이상인 객체지향 시스템에서 동적 메모리 할당 방법으로 대체하여 사용할 수 있다. 확장-가중치 버디 시스템은 2진 버디 시스템에

비해 약 3.5배, 가중치 버디 시스템에 비해 약 2배 정도의 다양한 크기의 블럭을 제공한다. 전체 메모리의 크기가 2m일 경우 가중치 버디 시스템은 2*m개의 블럭이 생성되며, 확장-가중치 버디 시스템은 4*m-9개의 블럭이 생성된다. 따라서 내부단편화는 메모리 요구 크기가 균일분포일 때 가중치 버디 시스템보다 약 60% 정도 감소하였으며, 지수분포일 때 약 50% 정도 감소하였다.

주소계산을 위한 TYPE 필드에 의한 블럭당 오버헤드는 가중치 버디 시스템보다 1비트가 더 많은 3비트가 소요된다. 외부단편화의 증가는 2^k 블럭내의 오른쪽에 생성된 작은 블럭이 흩어져 있고, 버디 시스템의 특성상 재결합할 수 있는 블럭은 오로지 각 버디 쌍이기 때문이다. 그러나 객체지향 시스템에서 객체의 생존기간이 매우 짧기 때문에 외부단편은 바로 재결합하여 할당할 수 있으며, 만일 비슷한 크기의 객체가 계속 할당될 수 있음을 예측할 수 있다면 외부단편화는 상당히 감소시킬 수 있다.

확장-가중치 버디 시스템의 모체는 가중치 버디 시스템과 마찬가지로 2진 버디 시스템이므로 생성된 블럭의 주소 형태가 항상 2k 크기의 블럭내에서 구성되기 때문에 [12]에서 제안한 하드웨어 지원 비트-벡터 할당 기법과 접목시키면 내부단편화를 더욱 감소시킬 수 있으며, 실행 시간이 증가하는 오버헤드는 제거해 낼 수 있다. 앞으로 더욱 연구할 일은 [12]의 기법과 접목시키는 일이며, 외부단편화를 보다 감소시키기 위해서 산재해 있는 할당 블럭을 한쪽으로 모으는 방법과, 버디 쌍 중에서 어느 한쪽만이 할당되어 있는 경우 나머지 한쪽에 채워 넣는 방법의 연구이다.

참 고 문 헌

[1] Meyer, B., 'Object-Oriented Software Construction', Prentice-Hall, 1988.

[2] Cunha, A.R., Ribeiro, C.N., and Marques, J.A., "The Architecture of a Memory Management Unit for Object-Oriented Systems," ACM SIGARC Computer Architecture News, Vol.19, No.4, pp.109-116, June 1991.

[3] Dally, W. J. and Kajiya, J. T., "An Object Oriented Architecture," The 12th Int'l Symp. on Computer Architecture Conference Proceedings, pp.154-161, 1985.

[4] Lewis, D. M., Galloway, D. R., Francis R. J., and Thomson, B. W., "Swamp:A Fast Processor for Smalltalk-80," Proc. of OOPSLA'86, pp. 131-139, 1986.

[5] Moon, D. A., "Object-Oriented Programming with Flavors," Proc. of OOPSLA'86, pp.1-8, 1986.

[6] Hyatt, C., "A High-Performance Object-Oriented Memory," ACM SIGARCH Computer Architecture News, Vol.21, No.4, pp.11-19, Sep. 1993.

[7] Kaiser, J., "MUTABOR, A Coprocessor Supporting Memory Management in an Object-Oriented Architecture," IEEE MICRO, pp.30-46, 1988.

[8] Myers, G. j., 'Advances in Computer Architecture', 2nd Edition, John Willy & Sons, New York, 1982.

[9] Houdek, M. E., Soltis, F. G., and Hoffman, R. L., "IBM S/38 Support for Capability-Based Addressing," The 8th Int'l Symp. on Computer Architecture Conference Proc., pp.341-344, 1981.

[10] Knuth, D. E., 'The Art of Computer Programming, Volume 1:Fundamental Algorithm', 2nd Edition, Addison-Wesley, pp.435-455, 1973.

[11] Shen, K. K. and Peterson, J. L., "A Weighted Buddy Method for Dynamic Storage Allocation," CACM, Vol.17, No.10, pp. 558-562, Oct. 1974.

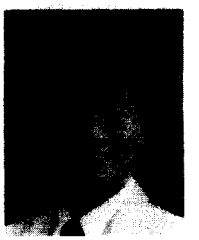
[12] Chang, J. M., Gehringer, E. F., "A High-Performance Memory Allocator for Object-Oriented Systems," IEEE Transactions on Computers, 45, No.3, pp.357-366. March 1996.

[13] Decouchant, D., "A Distributed Object Manager for the Smalltalk-80 System," Object-Oriented Concepts, Databases, and Applications, Chap. 19, Edited by Won Kim and Frederick H. Lochovsky, Addison Wesley, pp.487-520, 1989.

[14] Fabry, R.S., "Capability-Based Addressing," CACM, Vol.17, No.7, July 1974.

김 관 중

1983년 숭실대학교 전자계산학과 졸업(학사)
1988년 숭실대학교 대학원 전자계산학과 졸업(석사)
1993년 숭실대학교 대학원 전자계산학과 박사과정 수료
1997~현재 한서대학교 전산학과 전임강사
관심분야: 컴퓨터구조, 병렬처리 등



김 병 기

1977년 서울대학교 전자공학과 졸업(학사)
1979년 한국과학기술원 전산학과 졸업(이학석사)
1997년 한국과학기술원 전산학과 졸업(공학박사)
1979년~1982년 경북대학교 전자공학과 전임강사
1982년~현재 숭실대학교 정보과학대학 컴퓨터학부 교수
관심분야: 컴퓨터구조, 컴퓨터 통신, 병렬처리