2002년 한국정보처리학회 춘계학술발표논문집 제9권 제1호

BST 기반 보완된 절반-적합 메모리 할당 방법

류제영, 추현승, 윤희용
성균관대학교 전기전자 및 컴퓨터공학부
E-mail : mazelan95@ece.skku.ac.kr

Modified Half-Fit Memory Allocation Scheme Based on BST

Je Young Ryu, Hyunseung Choo, Hee Yong Youn
School of Electrical and Computer Engineering, Sungkyunkwan University

요 약

동적 메모리 관리는 컴퓨터 시스템의 중요하고 본질적인 동작이다. 메모리를 얼마나 효율적으로 이용 하느냐에 따라 시스템의 성능이 달라진다. 따라서 본 논문에서는 실시간 시스템을 위해 보다 효율적으로 메모리를 사용하는 동적 메모리 할당 알고리즘, BHF(Binary-search-tree-Half-Fit)를 제안한다. 제안된 알고리즘은 메모리 요청을 위해 2 의 거듭제곱의 프리 블럭 리스트를 이진 탐색 트리로 사용한다. 제안된 알고리즘의 효율성을 나타내기 위하여 절반-적합 알고리즘과 이진 버디 시스템과 비교, 분석하였다.

1. 서론

메모리는 많은 실시간 시스템에서 전체 시스템 가격을 결정하는 중요한 H/W 자원 중에 하나이다. 메모리가 충분치 않은 시스템에서 작업을 하는 것이 어떠한지 알고 있으리라 생각한다. 때때로 실행이 늦어지고 메모리 오류는 더 자주 발생하며 이따금 다른 애플리케이션을 닫거나 중지하기 전에는 애플리케이션 또는 파일을 새로 실행할 수 없다. 충분한 메모리가 있는 시스템에서는 한 개의 문서에서 작업을 하면서 다른 문서를 인쇄하는 것과 같은 복수 작업을 동시에 할 수 있으며 여러 개의 애플리케이션을 동시에 열어 놓을 수도 있다. 이러한 메모리 자원을 효과적으로 사용할 수 있는 기술이 있다면 동일한 H/W 자원에서 시스템 성능을 개선할 수 있는 좋은 방법이 될 것이다. 또한, 메모리를 동적으로 사용하기 위해서는 사용자가 반환한 메모리 영역을 미래의 할당 요구에 이용되어야 하고, 사용되지 않는 메모리 조각(Fragmentation)이 최소한으로 유지되어야 한다. 이는 시스템이 실행되면서 사용되고 남은 메모리 공간이 점차 조각화(Fragmentation)되어 충분한 공간의 메모리가 있음에도 더 이상의 메모리를 할당 할 수 없는 상황이 발생할 수 있다는 것이다.

본 논문에서 제안하는 동적 메모리 할당 알고리즘은 2 의 거듭제곱 크기별로 프리 블럭 리스트를 관리하는 이진 탐색 트리의 구조로 되어있다.

본 논문의 구성은 다음과 같다. 2 절에서는 기존에 연구되었던 동적 메모리 할당 알고리즘을 분석하고, 3 절에서는 조각화를 최소화하며 메모리 효율성을 높이는 동적 메모리 할당 알고리즘을 제안한다. 4 절에서는 제안된 알고리즘과 기존의 알고리즘을 비교, 분석하고 5 절에서는 결론과 향후 연구 계획에 대해서 논의한다.

2. 연구 배경

본 절에서는 동적 메모리 할당 알고리즘 결정시 고려되어야 하는 점들에 대해서 생각해 보고, 기존의 동적 메모리 할당 알고리즘에 대하여 분석하여 문제점을 검토해 보고자 한다.

2.1 동적 메모리 할당 알고리즘의 고려사항

동적 메모리 할당은 요구되어질, 그 크기를 미리 알 수 없는 메모리 요청에 효과적으로 대처하기 위하여 메모리를 할당해 주고 관리하는 것을 말한다. 이러한 메모리를 할당하고 남은 부분을 다시 어떻게 관리하는지, 그리고 사용 후 반환(deallocation)되는 블럭들을 어떻게 관리할 것인지 고려해야 한다. 사용 후 반환되는 프리 블럭들은 나중에 메모리 요구시 다시 할

당해 주기 위해서 인접한 블럭들과 서로 합병을 해 주어야 한다. 이러한 합병 전략에는 두 가지가 있는데, 즉시 합병(immediate coalescing)과 지연 합병(deferred coalescing)이 그것이다. 즉시 합병은 반환되는 메모리 프리 블럭이 있을 때 마다 인접한 블럭이 있는지 체크하여 즉시 합병하는 방법이고, 지연 합병은 반환된 메모리 블럭을 즉시 합병하는 것이 아니고, 요구된 메모리에 대해 할당해 줄 메모리가 부족할 때 인접한 블럭들과 합병하여 메모리 요구에 대처하는 방법이다.

요구되어진 크기의 프리 메모리 블럭을 할당해 주면, 남는 부분이 생기게 되는데 이것이 메모리 조각(fragmentation)이다. 동적 메모리 할당이나 특히 메모리 크기가 부족한 이동 컴퓨팅에서는 이 메모리 조각화를 최소화 시켜 메모리 사용의 극대화를 추구하는 것이 중요하다.

2.2 기존의 동적 메모리 할당 알고리즘

메모리 할당 방법으로 크게 두 부류로 나눌 수 있는데, 정적 할당(static allocation) 방법과 동적 할당(dynamic allocation) 방법이 그것이다. 고정 할당(static allocation)은 시스템 초기에 미리 메모리의 크기를 정해 놓고, 메모리 요청시 그 고정된 크기의 메모리를 할당해 주는 방법이다. 메모리가 초기화시에 미리 정해져 있기 때문에 최악의 경우 실행 시간을 미리 예측할 수 있으며 메모리 관리가 간단하다는 잇점이 있지만 메모리 조각화에 의한 불필요한 메모리 낭비가 불가피하다. 그에 대한 해결책으로 동적 할당(dynamic allocation) 방법, 즉 사전에 그 크기를 정해 놓지 않고 임의의 크기에 대한 요청이 있을 때 알맞은 크기의 프리 메모리를 찾아 할당해 주는 방법이 고려되고 있다. 이 방법은 메모리 관리에 대해 정적 방법에 비해 오버헤드가 있지만 효율적인 메모리 사용과 조각화를 최소화 할 수 있는 장점이 있다.

기존의 대부분의 동적 메모리 할당 알고리즘은 프리 리스트 구조를 가지고 있다. 하나의 리스트를 가지고 모든 프리 블럭들을 관리하는 단일 프리 리스트(single free list)와, 여러개의 프리 리스트를 사용하는 다중 프리 리스트(multiple free list)구조가 있다.

버디 시스템(buddy system)은 메모리 블럭의 분할과 합병 작업을 제한적이지만 효과적으로 지원하는 다중 프리 리스트의 병형된 알고리즘이다. 이 알고리즘에서는 하나의 메모리를 버디(buddies)라 부르며, 각 버디의 크기는 2의 거듭제곱이며, 이 버디는 다시 두개의 같은 크기의 버디로 나누어 관리되며, 양분되면 이들은 서로의 버디가 된다.

절반-적합(Half-Fit)은 실시간 시스템 환경에서 사용할 목적으로 개발된 시간 복잡도가 O(1)으로 한정(bounded)된 다중 프리 리스트를 이용하는 동적 메모리 할당 알고리즘이다. 크기 범위 체계로는 2의 거듭제곱 크기들을 사용하여 블럭의 크기 s 가 $2^i \le s < 2^{i+1}$ 인 프리 블럭들은 i 를 색인으로 사용하는 하나의 프리 리스트에서 관리된다. 따라서 $[2^{i-1}+1, 2^i]$ 크기 영역 내의 메모리 요청이 있을 경우 프리 리스트 i 에서 프리 블럭이 할당되고 반환된다. 만약 프리 리스트 i 가 비어 있다면, i 보다 크면서 비어 있지 않은 가장 작은 프리 리스트에서 하나의 블럭을 빼낸다. 이 블럭을 분할하여 요청된 크기만큼의 메모리를 할당하고, 나머지는 해당 크기의 프리 리스트에 다시 삽입한다. 반환된 블럭은 즉시 합병 방식에 의해 합병되며, 합병된 프리 블럭은 기존의 프리 리스트에서 삭제되고 해당 크기의 프리 리스트로 삽입된다.[1][2]

Segregated fit 은 프리 리스트의 배열에 각각 같은 크기의 프리 블럭들이 다중 연결 리스트에 포함되어 있다. 크기 n 인 하나의 블록을 할당하기 위해서 n 보다 큰 크기의 블록을 위해 알맞은 프리 리스트를 찾아서 할당해 주고, 그렇지 못한 경우에는 알맞은 크기의 프리 블록을 찾을 때 까지 다음 리스트를 찾게된다. 이 정책은 할당하고 남은 프리 블럭이 인접한 블록과 합병하거나 분리되지 않는 특징이 있다. Segregated fit 의 주요 이점은 순차 적합(sequential fit)보다 할당과 프리시키는 메모리 블럭의 효과적인 실행에 있지만, 메모리의 사용이 비효율적인 조각화가 발생하는 문제가 있다.

3. Binary-tree-Half-Fit(BHF) 알고리즘

본 절에서는 기존의 동적 메모리 할당 알고리즘에서 지적된 비효율적인 메모리 사용에 보다 효과적인 메모리 사용을 위해 본 논문에서 제안하는 이진 탐색 트리 절반-적합(Binary-tree-Half-Fit) 알고리즘에 대해서 알아본다.

3.1 Binary-search-tree-Half-Fit(BHF) 알고리즘

절반-적합(Half-Fit) 정책은 크기 범위 체계로 2 의 거듭제곱 크기들을 사용하고, 블록의 크기 s 가 $2^i \le s < 2^{i+1}$ 인 프리 블럭들을 i 를 색인으로 사용하여 $[2^{i-1}+1, 2^i]$ 크기의 메모리 요청이 있을 때 프리 리스트 i 에서 프리 블록을 할당하고 반환한다. 따라서 요청하는 메모리 크기보다 2 배 더 큰 리스트에서 찾기 때문에 시간 복잡도가 O(1)의 빠른 메모리 할당이 가능하다. 하지만, 충분히 같은 크기의 메모리 할당이 가능함에도 불구하고 더 큰 메모리의 할당은 메모리 조각(Fragmentation)이 많이 발생하므로 비효율적이다. 이러한 메모리의 낭비를 최소화하면서 시스템 성능을 향상시키기 위해 제안한 Binary-tree-Half-Fit(BHF) 알고리즘은 블록의 크기를 2 의 거듭제곱으로 사용하고, $[2^i, 2^{i+1}-1]$ 크기의 메모리 요청에 대해서 $2^i \le s < 2^{i+1}$ 인 프리 블록을 i 를 색인으로 찾아서 할당해 준다. 또한 연결 리스트로 연결된 프리 블럭들을 찾는데 대한 오버헤드를 줄이기 위해 각 프리 블록들을 이진 탐색 트리로 구성하였다.

이진 트리는 모든 노드의 차수(degree)가 2 를 넘지 않는 특수한 트리(Tree)로 하나의 노드는 두개의 자식 노드(child node)를 가질 수 있다. 각각을 왼쪽 자식 노

드(left child)와 오른쪽 자식 노드(right child)라 부른다. 일반 트리에서는 자식 노드들의 순서를 구별하지 않지만 이진 트리에서는 자식 노드의 순서를 구별한다. 노드(node)는 정보가 저장되는 기억장소를 말하고, 가지(edge)는 노드와 노드를 연결하는 포인터이다. 이진 트리는 루트(root) 노드를 포함하여 각 노드는 최대로 두 개의 자식 노드를 가지는 추상적 자료 구조를 말하며, 자식 노드에 연결된 위치에 따라 노드의 순서를 갖게 된다.

그림 1 은 BHF 의 구조를 설명하고 있다. 그림 1 에서 보는 바와 같이, 2 의 거듭 제곱 크기별로 이진 탐색 트리로서 관리되고, 각 노드는 해당 크기로 다양한 크기의 블록들이 포함되어 있고, 각 노드의 크기의 단위는 워드(word)로 구성되어 있다. 그러므로, 인덱스가 1 인 루트에 연결된 트리의 노드들은 메모리 크기가 $[2^1, 2^2)$ 범위의 2 와 3 워드, 인덱스 2 를 갖는 트리의 노드는 $[2^2, 2^3)$ 범위의 4, 5, 6, 7 워드 크기의 프리 노드가 연결되어 있다. 이 트리의 루트 노드는 포인터들의 배열로서 포함된다.

BHF 는 실시간 특성을 위해 반환되는 메모리 발생 시 서로 인접한 프리 노드들끼리 즉시 합병하는 방법이 수행된다. 따라서 반환된 노드와 인접한 프리 노드가 있는지 조사를 하고, 인접한 프리 노드가 있다면 서로 합병하여 해당되는 프리 리스트에 삽입이 되고, 만약, 존재하지 않는 다면 반환된 프리 노드 삽입 전략에 따라 해당되는 프리 리스트에 삽입이 될 것이다.

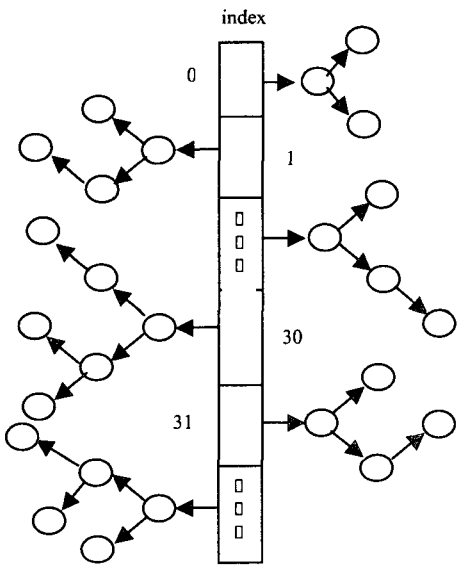

그림 1. BHF 알고리즘의 구성

3.2 AVL-tree-Half-Fit(AHF) 알고리즘

리눅스에서는 대량의 데이터등으로 인해 노드의 수가 커지면 노드 탐색기법으로 AVL 트리를 사용한다. AVL 트리는 그림 2 에서 보는 것과 같이 각 노드를 중심으로 양 서브트리의 깊이가 2 이상 나지 않는 범위 내에서 균형을 유지하는 트리이다. 이 방법을 제안한 사람은 Adelson-Velskii 와 Landis 로서 이 트리를 우리는 이들 이름의 첫 문자를 따서 AVL 트리 또는 균형트리(Balanced tree)라고 한다. 트리 T 가 왼쪽과 오른쪽 서브트리인 TL 과 TR 을 가진 이진 트리라고 할 때 아래 조건을 만족하면 T 는 높이 균형을 이루며 그 역도 성립한다.

(1) TL 과 TR 이 높이 균형을 이루며
(2) $|hL - hR| \leq 1$ (hL 과 hR 은 각각 TL 과 TR 의 높이)

이진 트리에서 어떤 노드, T 의 균형 인수 BF(T)는 hL − hR 로 정의하고, AVL 트리의 어떠한 노드 T 에 대해서도 BF(T)는 −1, 0, 1 이 된다.

AVL 트리의 장점은 완전 균형 트리의 단점을 보완하기 위하여 완화된 균형화 방법이며 정의가 단순하고, 균형을 위한 연산이 용이하며, 탐색시간은 완전 균형 트리와 동일한 평균 경로길이, 즉 연산시간이 최악의 경우 O ($\log_2 n$)이다. 따라서 이진 탐색 트리의 최악의 경우 모든 노드가 한쪽으로 몰리는 경우를 배제할 수 있다. 트리의 깊이가 2 이상의 불균형 트리가 구성되었을 때 재균형을 하여 이진 탐색 트리보다 빠르게 메모리 할당이 가능한 대안책이다.

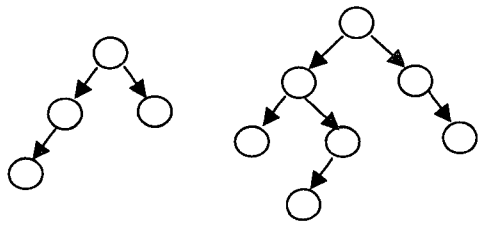

그림 2. AVL 트리

4. 성능 평가

본 논문에서 제안한 BHF 알고리즘과 절반-적합, 이진 버디 시스템의 메모리 사용 효율성을 비교하기 위하여 몇 가지 논문의 결과를 참고 하였다.

먼저, 절반-적합 알고리즘과 이진 버디 시스템에서의 메모리 할당에 소요되는 최장 실행 시간, 메모리 할당 실패율과 메모리 조각율은 [2]의 실험 결과에서 알 수 있다. 점차 요청되는 메모리의 크기가 커질수록 메모리 할당에 소요되는 최장 실행 시간(WCET)이 이진 버디 시스템에서는 증가함을 알 수 있지만, 절반-적합은 거의 동일한 시간을 나타내고 있다. 그러나 메모리 할당 실패율에서는 거의 동일한 형태의 그래프를 유지 하지만 이진 버디 시스템이 보다 크게 나타남을 알 수 있다. 또한, 그림 3 과 그림 4 에서 알 수 있듯이 내부(internal fragmentation) 및 외부 조각화(external fragmentation)를 합친 총 조각율을 비교할 때 작은 크기의 메모리 요청에 대해서는 근소하게 이진

버디 시스템이 우의를 보이지만 점차 큰 크기의 메모리 요청에 대해서는 절반-적합 알고리즘이 보다 작은 조각화를 보인다. 하지만, 두 알고리즘 모두 총 조각율이 1.3 이상의 높은 조각화가 발생하고 있음을 나타내고 있다.[2]

[4]에서는 이진 트리구조로 구성된 메모리 할당 알고리즘과 연결 리스트로 구성된 메모리 할당 알고리즘을 비교하였다. 실험 결과에서 메모리 할당을 위해 탐색하게 되는 평균 프리 노드의 수가 이진 트리로 구성된 것 보다 연결 리스트로 구성된 알고리즘이 수십배 이상을 탐색하는 것을 알 수 있다. 또한, 메모리 할당에 필요한 노드의 수도 수배에서 수십배 이상이 필요하다는 것을 알 수가 있었다.

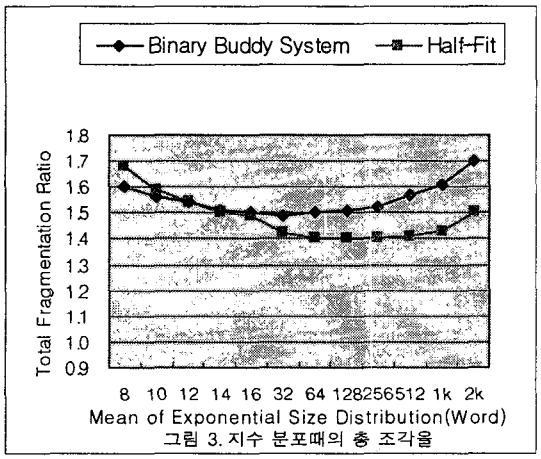

그림 3. 지수 분포때의 총 조각율

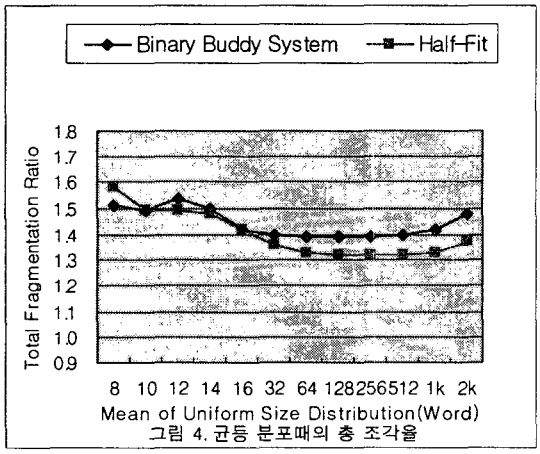

그림 4. 균등 분포때의 총 조각율

따라서, 본 논문에서 제안한 BHF 알고리즘은 요청되는 메모리 크기에 상관없이 메모리 할당 및 반환에 소요되는 최장 실행 시간(WCET)이 거의 1~2 마이크로 초로 동일하게 제공할 수 있으며[2], 현재 종합적으로 진행중인 시뮬레이션의 초기 결과치에 기반하여, 요청되는 메모리 크기와 비슷한 크기의 메모리 할당이 가능하므로 메모리 조각율이 최소한 1.3 보다 작은 값을 얻을 수 있다.

한 가지 고려할 사항은 이진 탐색 트리로 구성되어 있기 때문에 메모리 할당이 실행됨에 따라서 최악의 경우에 트리가 한쪽으로 치우치게 될 수도 있다. 그렇게 되면, 프리 노드를 찾아서 할당하는데 걸리는 시간이 연결 리스트에서와 같이 O(n)의 시간 복잡도가 될 수도 있다. 그에 대한 해결책으로 이진 탐색 트리를 균형 탐색 트리인 AVL 트리로 구성한다면, 트리 구조의 재균형으로 인한 오버헤드가 추가되긴 하지만, 항상 최적의 트리 균형을 이루고 있기 때문에 항상 O(log n)의 프리 노드 할당이 가능하다.

5. 결론 및 향후 계획

본 논문에서는 보다 메모리를 효율적으로 사용하기 위해서 BHF 알고리즘을 제안하였다. 이는 동적 메모리 할당 알고리즘으로서 half-fit 알고리즘의 조각화로 인한 메모리 낭비를 최소화하기 위해 half-fit 알고리즘을 이진 탐색 트리 구조로 구성하였다. Half-fit 알고리즘의 시간 복잡도가 O(1)일지라도 메모리 할당이 실행됨에 따라 프리 노드의 탐색 시간이 증가하게 된다. 또한, 위의 그래프에서 알 수 있듯이 half-fit 알고리즘은 이진 버디 시스템보다는 좋은 조각화를 보이지만 그 자체도 130% 이상의 높은 조각화를 나타내고 있다. 따라서, 제안된 알고리즘은 빠른 메모리 할당을 제공하고, 조각화를 최소화하여 보다 효율적인 메모리 사용이 가능하다.

향후 본 논문에서 제안한 알고리즘의 메모리 효율성의 우수함을 증명하기 위해서 실험 및 구현을 계속 연구할 계획이다.

참고문헌

[1] Takeshi Ogasawara, "An Algorithm with Constant Execution Time for Dynamic Storage Allocation", Proc. of 2nd Intn. Workshop on Real-Time Computing Systems and Applications, pp.21-25, 1995.

[2] 정성무, 유해영, 심재홍, 김하진, 최경희, 정기현, "예측 가능한 실행 시간을 가진 동적 메모리 할당 알고리즘" , 한국정보처리학회논문지, 제 7 권 제 7 호, pp. 2204-2218, Jul. 2000.

[3] 심재홍, 정석용, 강봉직, 최경희, 정기현, "Real-Time Linux 시스템을 위한 재구성 가능한 메모리 할당 모델" , 한국정보처리학회논문지 제 8-A 권 제 3 호, Sep. 2001.

[4] Mehran Rezaei and Krishna M. Kavi, "A New Implementation Technique for Memory Management", Southeastcon 2000, Proc. of the IEEE, 2000, pp. 332-339

[5] Iyengar, A. "Parallel dynamic storage allocation algorithms", Parallel and Distributed Processing, 1993. Proceedings of the Fifth IEEE Symposium on , 1993 Page(s): 82 -91

[6] Iyengar, A. "Scalability of dynamic storage allocation algorithms", Frontiers of Massively Parallel Computing, 1996. Proceedings Frontiers '96., Sixth Symposium on the , 1996. Page(s): 223 -232