# 실시간 운영체제에서 최대 힙 알고리즘을 이용한 메모리 할당 기법 설계 및 구현*

이정원°, 최인범, 김용희, 이철훈
충남대학교 컴퓨터공학과
{jwlee°, ibchoi, yonghee, chlee}@ce.cnu.ac.kr

# The Design and Implementation of Memory Allocation using Max Heap Algorithm on Real-time Operating System

Jung-Won Lee°, In-Bum Choi, Yong-Hee Kim, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

**요 약**

실시간 운영체제는 멀티태스킹 및 ITC(Inter Task Communication)를 제공한다는 면에서는 범용 운영체제와 비슷하나, 시간 결정성을 보장해야 한다는 면에서는 일반 운영체제와 다르다. 실시간 시스템에서는 메모리를 할당하는데 있어서 시간 제약을 어기지 않아야 하기 때문에 동적 메모리 할당은 효율적으로 구성되여야 한다. 본 논문에서는 실시간 운영체제 ARTOS™에서 메모리 할당에 소요되는 시간을 향상시키기 위해 최대 힙 알고리즘을 적용한 메모리 할당 기법을 설계 및 구현하였다.

## 1. 서론

실시간 운영체제는 임베디드 시스템에 사용되는 대표적인 운영체제이다. 임베디드 시스템은 자원이 제한된 환경에서 미리 정해진 특정 기능을 수행하도록 설계된 시스템을 의미한다. 실시간 운영체제는 시간 결정성과 작은 크기의 실행이미지를 특징으로 하며, 시간 결정성을 제공할 수 있도록 내부적으로 특화된 자료구조 및 수행루틴을 포함하게 된다[1][2][3].

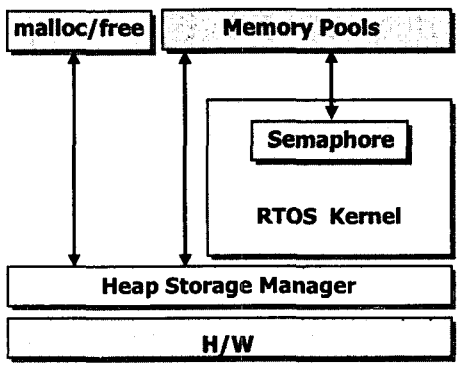
본 논문에서는 단방향 연결리스트로 프리 메모리를 관리하는 실시간 운영체제인 ARTOS™에서 메모리 할당에 소요되는 시간을 줄일 수 있도록 최대 힙 알고리즘을 적용한 메모리 할당 기법을 제안하고자 한다.

본 논문의 구성은 2 장에서 관련연구로 실시간 운영체제의 메모리 관리 기법을 소개하고, 3 장에서 실시간 운영체제에서 최대 힙 알고리즘을 적용한 동적 메모리 할당 기법 설계 및 구현을, 4 장에서 테스트 환경을 보이고 결과를 분석한다. 마지막으로, 5 장에서는 결론 및 향후 연구 계획에 대해 기술한다.

## 2. 관련 연구

### 2.1 실시간 운영체제의 메모리 관리

ARTOS™에서는 2 단계의 메모리 관리기법을 제공한다. 하위 단계는 가변 크기의 메모리를 할당, 해제할 수 있는 힙 스토리지 매니져(Heap Storage Manager)이고 상위 단계는 고정 크기의 메모리를 할당, 해제할 수 있는 메모리 풀(Memory Pool)이다.




[그림 1] Storage Allocation 관리 체계

[그림 1]은 메모리 관리의 구성을 보여준다. 하드웨어 상의 메모리는 힙 스토리지 매니져에 의해 관리되며, 가변 크기의 메모리를 할당, 해제한다. 메모리 풀은 이미 생성된 힙

* 본 논문은 국방과학연구소의 실시간 운영체제 인터페이스용 미들웨어 연구과제의 수행결과임.

스토리지 매니저에서 시스템 초기화 시에 미리 할당 받아 여러개의 고정 크기의 버퍼로 관리하며, 고정 크기의 버퍼를 할당, 해제한다.

### 2.2 힙 스토리지 매니저에서 메모리 할당 및 해제

힙 영역의 동적 메모리 할당은 MK_GetMemory()함수를 사용하여 퍼스트 핏(First-fit) 알고리즘에 따라 프리 메모리 리스트로부터 검색하여 요구하는 크기보다 큰 첫번째 메모리 블록을 얻어 오게 된다. 만약, 할당 가능한 메모리가 존재하지 않으면 메모리가 생길 때까지 태스크는 대기하게 된다.

메모리 해제는 MK_FreeMemory() 함수로 프리 메모리 리스트에서 인접한 프리 블록을 확인하고 존재 할 경우 병합하여 프리 메모리 리스트에 추가한다.

### 2.3 메모리 풀에서 메모리 할당 및 해제

메모리 풀은 힙에서 할당 받은 메모리를 주어진 고정 크기의 버퍼 단위로 나누며, 버퍼의 크기에 따라 다양한 메모리 풀을 생성할 수 있다.

메모리 풀은 MK_GetBuf() 함수로 프리 리스트의 맨 앞의 버퍼를 할당한다. 할당 가능한 버퍼가 없으면, 태스크는 대기하게 된다.

메모리 버퍼는 MK_FreeBuf() 함수로 버퍼가 속한 프리 리스트에 연결하고 해제된다.

## 3. 최대 힙의 설계 및 구현

### 3.1 우선 순위 큐 - 최대 힙

자료구조 힙은 우선순위 큐를 구현하는 데 자주 사용된다. 우선순위 큐는 우선순위가 가장 높거나 낮은 원소를 먼저 삭제하고 임의의 원소를 우선순위 큐에 언제든지 삽입할 수 있는 구조이다. 만일 가장 높은 우선순위를 가진 원소를 삭제하는 경우는 우선순위 큐인 최대 힙을 사용한다. 최대 힙은 완전 이진 트리로서 배열을 사용하여 구현한다.

[표 1] 우선순위 큐의 표현방법과 수행시간

| 표현방법 | 삽 입 | 삭 제 |
|---|---|---|
| 순서 없는 배열 | $\Theta(1)$ | $\Theta(n)$ |
| 순서 없는 연결 리스트 | $\Theta(1)$ | $\Theta(n)$ |
| 정렬된 배열 | $O(n)$ | $\Theta(1)$ |
| 정렬된 연결 리스트 | $O(n)$ | $\Theta(1)$ |
| 최대 힙 | $O(\log_2 n)$ | $O(\log_2 n)$ |

[표 1]은 최대 힙을 포함한 여러 표현방법에 대한 삽입과 삭제 시간을 보여주고 있다. 최대 힙으로 우선순위 큐를 표현할 경우 삽입과 삭제 시간이 모두 $O(\log_2 n)$ 이 되기 때문에 최대 힙이 효율적이다[4]

### 3.2 최대 힙의 구현

기존에는 메모리 할당에 단방향 리스트를 사용하여 원하는 프리 메모리 크기보다 큰 첫번째 프리 메모리를 탐색하는 구조였다(First-fit). 본 연구에서는 프리 메모리 리스트를 가지고 최대 힙을 구성하여, 루트 노드는 남은 프리 메모리 블록의 가장 큰 메모리 블록이 된다. 루트에서 삭제(배열의 첫번째 원소)로 메모리를 할당함으로써 검색시간이 단축된다. 이때 힙을 재구성한다.

이러한 힙을 구현하기 위하여 메모리를 관리하는 힙 스토리지 관리자에 최대 힙의 배열, 현재 우선순위 큐에 들어 있는 프리 블록의 수, 그리고 삽입/삭제를 위한 함수를 추가하게 된다.

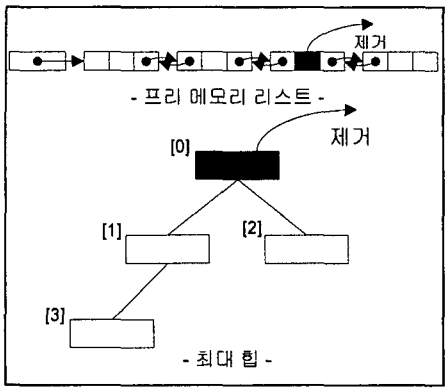
```
#define MK_FREE_SIZE          200

#define SWAPND    (x, y, t)   ((t)=(x), (x)=(y), (y)=(t))
#define HEAP_FULL (n) (n == MK_FREE_SIZE - 1)
#define HEAP_EMPTY (n) (!n)

typedef struct mk_heap_struct {
          ...

  /* 추가된 최대 힙(배열)에 대한 포인터 필드*/
  struct mk_memory_block_struct*
    h_pMaxHeap[MK_FREE_SIZE];
  /* 프리 메모리 블록의 수 */
  int h_FreeMBlockCnt
          ...

} MK_HEAP;

/* 추가된 함수들 */
void MK_InsertMaxHeap(MK_HEAP* pHeap, MK_MBLOCK*
pTemp);
void MK_DeleteMaxHeap(MK_HEAP* pHeap, MK_MBLOCK*
pTemp);

  <생략>
```

[그림 2] 최대 힙 제어 함수 헤더

[그림 2]에서 MK_InsertMaxHeap() 함수는 메모리 해제시에 최대 힙의 리프 노드에 새로운 프리 메모리를 삽입하는 함수로 결합될 프리 메모리가 있으면 해당 메모리를 최대 힙에서 삭제한 후 결합하여 리프 노드로 삽입하고 최대 힙을 재구성하는 함수이다.

MK_DeleteMaxHeap() 함수는 메모리를 할당하고 남은 메모리가 없을 경우 루트 노드를 삭제하고 재구성하는 함수이다. 남는 메모리가 있을 경우, 루트 노드에 남은 메모리 블록을 삽입하고 힙을 재구성하는 함수이다.

### 3.2.1 메모리 할당과 힙의 재구성

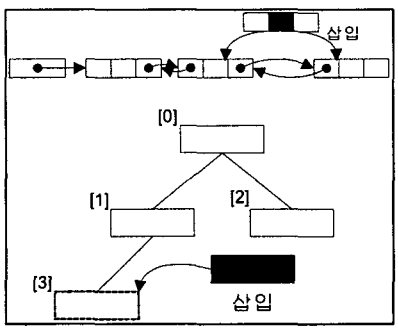



[그림 3] 최대 사이즈 메모리 할당

- 루트 노드의 메모리를 할당하고 남은 메모리가 발생하면, 즉, 요청한 메모리 사이즈보다 루트 노드의 프리 메모리가 클 경우 남은 메모리를 새로운 프리 블록으로 만들어서 최대 힙의 루트에 삽입하고 재구성을 한다.

- 할당할 메모리의 크기보다 루트 노드의 프리 메모리가 작거나 같을 경우 루트 노드의 프리 메모리는 전부 할당되고 리프 노드를를 루트 노드에 삽입하고 최대 힙을 재구성한다.

### 3.2.2 메모리 해제와 힙의 재구성



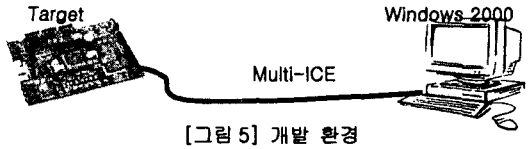

[그림 4] 프리 메모리 삽입 후 재구성

- 메모리를 해제할 경우는 생성된 프리 메모리 블록의 다른 프리 메모리 블록들이 인접한지를 판단한다. 인접할 경우에는 결합하려는 블록의 노드를 삭제하여 힙을 재구성하고 결합한 노드를 리프 노드에 삽입한다.

- 메모리 해제시 결합될 다른 프리 메모리가 없을 경우 최대 힙의 리프 노드의 마지막에 삽입한다.

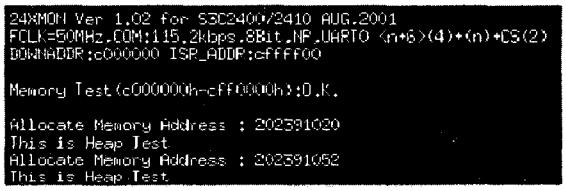
## 4. 테스트 환경 및 결과

본 논문에서 기술하고 있는 메모리할당 기법은 ARM 920T 를 기반으로 한 삼성 S3C2400™ 32-bit RISC Micro Processor 에 *i*RTOS™를 탑재하여 테스트 하였다. 컴파일러는 ARM Developer Suite 1.2 을 사용하였다.




[그림 5] 개발 환경

[표 1]에서 보인 것처럼 기존의 단일 연결 리스트에 비하여 최대 힙을 사용하여 메모리를 할당하면 수행시간이 감소한다. 또한, 프리 메모리 리스트의 메모리 블록들이 증가할수록 메모리 할당 시간은 더욱 감소하게 된다. 메모리 할당을 테스트한 결과는 다음과 같다.




[그림 6] 테스트 결과

## 5. 결론 및 향후 연구 과제

본 논문은 메모리 할당의 소요시간을 단축하기 위하여 최대 힙을 사용한 메모리 할당기법을 설계 및 구현하였다. 최대 힙을 적용한 메모리 할당 기법으로 메모리 할당 시간은 향상되었지만, 해제 후 서로 인접한 메모리 블록들을 결합하기 위한 연산이 많아지는 문제가 발생한다.

향후 연구 과제로는 해제 후 결합 연산을 감소시킬 수 있는 효율적인 알고리즘에 대한 연구가 필요하다.

## 6. 참고문헌

[1] http://www.inestech.com

[2] David Stepner 외 2 명, " *Embedded Application Design Using a Real-Time OS*" , IEEE, 1999

[3] C.M.Krishna, Kang G.Shin, " *Real-Time Systems*" , The McGraw-Hill Companies, Inc.1997

[4] Ellis Horowitz 외 2 명, " *Fundamentals of Data Structures in C*" , W. H. Freeman and Company