# 실시간 운영체제의 메모리 관리에서 메모리 풀의 메모리 누수 방지 기법 설계 및 구현*

조문행[0], 정명조, 유용선, 이달한[†], 이철훈
충남대학교 컴퓨터공학과, [†]삼성탈레스㈜ R&D 팀
{mhcho[0], mjjung, ysryu}@ce.cnu.ac.kr, [†]dh308.lee@samsung.com

# The Design and Implementation for Preventing A memory leakage of Memory Pool on Memory Management of Real-Time Operating Systems

Moon-Haeng Cho[0], Myoung-Jo Jung, Yong-Sun Ryu, Dal-Han Lee[†] and Cheol-Hoon Lee
[0]Dept. of Computer Engineering, Chungnam National University
[†]Dept. of R&D Team, SAMSUNG THALES CO.,LTD

요 약

실시간 운영체제가 탑재되는 임베디드 시스템의 공간제약 특성상 한정된 자원을 가질 수 밖에 없기 때문에 자원의 효율적인 사용 및 관리가 필수적이다. 특히, CPU 와 함께 한정된 크기의 메모리는 운영체제의 중요한 자원으로, 효율적인 메모리 관리를 통해 시스템의 성능을 향상 시킬 수 있다. 본 논문에서는 실시간 운영체제의 동적 메모리 관리기법 중 메모리 풀에서의 메모리 누수 방지 기법을 설계 및 구현하였다.

## 1. 서 론

최근 유비쿼터스 컴퓨팅과 차세대 컴퓨터에 대한 관심으로 임베디드 시스템 분야가 급성장하고 있다. 이에 따라, 임베디드 시스템에 탑재되는 실시간 운영체제(Real-Time Operating System)에 대한 많은 연구들도 진행되고 있다.

실시간 운영체제는 시간 결정성을 보장해야 하는 특성을 지니며, 데드라인(deadline)내에 수행한 결과만 신뢰하는 경성 실시간 운영체제(Hard Real-Time Opera-ting System)와 주어진 데드라인 이후의 수행 결과도 어느 정도 신뢰하는 연성 실시간 운영체제(Soft Real-Time Operating System)로 나뉘어진다.

임베디드 시스템에 탑재되는 실시간 운영체제는 한정된 자원을 효율적으로 관리할 수 있는 기법이 필요하다. 특히, 한정된 자원인 메모리는 CPU 와 함께 운영체제의 성능에 큰 영향을 미치는 자원으로, 보다 효율적인 관리기법을 통해 시스템 성능을 향상시킬 수 있다.

본 논문의 구성은 2 장에서는 관련연구로서 실시간 운영체제인 /RTOS™에서 사용하고 있는 메모리 관리 기법과 힙 스토리지에서의 메모리 누수 방지기법에 대해 소개하고, 3 장에서는 실시간 운영체제인 /RTOS™의 동적 메모리 관리 기법 중 메모리 풀의 메모리를 보다 효율적으로 관리하고 메모리 누수 현상을 줄일 수 있도록 설계 및 구현한 메모리 관리기법을 제시하며, 4 장에서는 테스트 환경 및 결과를 기술한다. 마지막으로, 5 장에서는 결론 및 향후 연구과제에 대해서 기술한다. [1][2].

## 2. 관련 연구

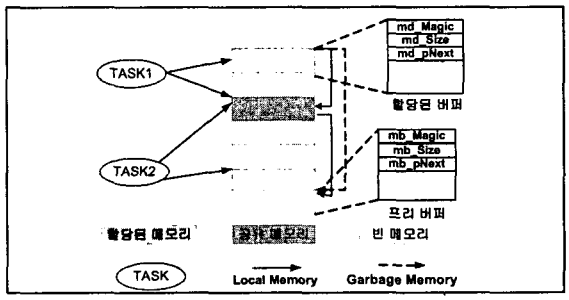
태스크에 메모리를 할당하는 방법은 전역변수로 선언하는 정적메모리 할당(Static Memory Allocation)과 실행 시간에 메모리를 할당하는 동적메모리 할당(Dynamic Memory Alloca-tion)이 있다.

임베디드 환경에서 실시간 운영체제의 실행 이미지는 롬(ROM)에 다운로드 하므로 가능한 작아야 한다. 그러나 전역 변수를 사용하여 정적으로 메모리를 할당하게 되면 실행이미지가 증가하게 되므로 실행 시간에 동적으로 메모리를 할당 받아 사용하는 것이 보다 효율적이다.

/RTOS™에서는 동적 메모리 관리를 위해 가변 크기 메모리를 할당하는 힙 스토리지 매니저(Heap Storage Manag-er)와 고정 크기의 메모리를 할당하는 메모리 풀(Memory Pools)을 사용한다. 또한, 힙 스토리지 에서는 누수된 메모리를 관리하기 위한 기법이 존재한다. [3][4].

### 2.1 힙 스토리지 매니저와 누수 메모리 관리 기법

임의의 메모리 크기를 동적으로 할당하고 시스템 메모리의 힙(Heap)영역을 관리하기 위하여 힙 스토리지 매니저를 사용한다.




[그림 1] 할당된 힙 영역

* 본 논문은 산업자원부의 지역전략산업 석,박사 연구인력 양성사업의 지원으로 수행된 것임

[그림 1]의 공유 메모리는 힙 영역에서 [그림 2]의 MK_GetMemory()를 통해 메모리를 할당하는데, 퍼스트 핏(First-Fit) 알고리즘에 따라 프리 블록 리스트로부터 적당한 메모리를 선택하여 할당하고, 공유 메모리가 아닌 메모리는 MK_GetLocalMem()를 통해 할당한다. 만약, 가용한 메모리가 존재하지 않으면, 메모리를 요청한 태스크는 가용한 메모리가 생길 때까지 대기(Pending)한다. 메모리 해제는 MK_FreeMemory()에 의해 이루어진다.

한편, 태스크가 종료(Termination)될 때 태스크가 직접 자신이 사용했던 메모리 영역을 스스로 제거할 수 있는 것이 아니고, 다른 태스크에 의해서 제거되어져야 하는데 그 제거 시점이 명확하지 못할 뿐만 아니라, 제거되지 않은 채 메모리에 남아 있는 경우가 발생하였다. 이런 메모리 누수 현상을 방지하기 위하여, 공유메모리가 아닌 메모리를 MK_G-etLocalMem()를 통해 할당하고, [그림 1]의 GarbageMemory 리스트를 생성하여 MK_GarbageMemFree()를 통해 해제한다.

```
struct  mk_task_struct *t_pMemHead;
        // Task에 할당된 메모리를 가리키는 포인터
        //TCB 구조체의 멤버
UINT  *pLocalMemHead;
    // 해제시킬 메모리를 가리키는 포인터
MK_GetMemory(UINT *pLocalMemHead);
        // 공유메모리를 할당해 주는 함수
MK_GetLocalMem(UINT *pLocalMemHead);
        // 공유메모리가 아닌 메모리를 할당해 주는 함수
MK_GarbageMemFree();
        // 가용한 메모리로 만들어주는 함수
```

[그림 2] 메모리 해제를 위한 포인터와 변수

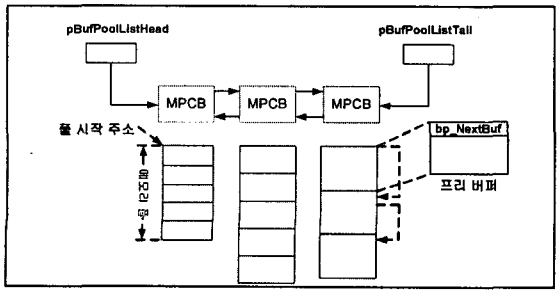
### 2.2 메모리 풀(Memory Pool)

힙 스토리지 매니저를 통한 임의의 메모리 크기를 할당함으로써 메모리 단편화(External Fragmentation)가 생길 수 있으며, 가용 메모리 검색 시간으로 인해 시간 결정성을 저해한다. 이를 해결하기 위해 고정 크기의 버퍼를 할당하는 메모리 풀을 사용한다.

메모리 풀은 힙 스토리지 매니저에서 스케줄링을 시작하기 전에 메모리 공간을 계산한다.

{ 생성할 버퍼 개수(count) × ( 버퍼 크기 + 다음 버퍼의 포인터(4Byte)크기 ) }

메모리 풀은 힙에서 미리 할당받은 시작주소(pAddr)와 버퍼의 개수(Count), 버퍼의 크기(Size)를 인자로 하여 생성한다. 메모리 풀의 구성은 할당받은 메모리를 주어진 고정 크기의 버퍼단위로 나누며, 각 버퍼는 세마포의 자원이 되고, 버퍼의 개수는 세마포의 카운트 값이 된다.




[그림 3] 메모리 풀

한편, 메모리 할당은 메모리 풀에서 MK_GetBuf()에 의해 이루어지며, 세마포 메커니즘에 따라 Count 값을 하나 감소하고 Count 가 0 이하가 되면 메모리를 요청한 태스크는 대기(Pending)한다. 프리 버퍼는 메모리 풀에 리스트로 연결되어 있어 리스트의 맨 앞에서 버퍼를 할당한다(First-Fit). 메모리의 해제는 MK_FreeBuf()에 의해 해제되고, 세마포 Count 를 하나 증가시킨다. 반납된 버퍼의 앞 4Byte 를 이용하여 버퍼가 속한 메모리 풀을 찾아, 메모리 풀의 프리 리스트에 연결하고 해제한다[4][5].

## 3. 메모리 풀의 메모리 누수 방지 기법 설계 및 구현

### 3.1 메모리 풀의 메모리 누수 현상

시스템 생성시 또는 특정 태스크의 필요에 의해 만들어진 메모리 풀은 메모리 풀의 해제 시점이 명확하지 않을 뿐만 아니라, 다른 태스크에 의해서 해제되어져야 하는데 그 시점 또한 명확하지 못하여 해제되지 않은 채 메모리에 남아 있는 경우가 발생하였다. 또한, 버퍼를 생성할 때 필요이상으로 생성된 버퍼, 즉 태스크가 사용하지 않는 버퍼는 메모리의 낭비가 된다. 이런, 해제되지 않은 채 방치된 버퍼와 사용하지 않는 버퍼로 인한 메모리 낭비는 한정된 메모리를 갖는 임베디드 시스템의 성능을 떨어뜨리고, 태스크 생성과 실행에 있어서의 제한을 가져올 뿐만 아니라 시스템 장애에 이르게 한다.

### 3.2. 메모리 풀의 메모리 누수 방지 기법의 설계 및 구현

본 논문에서는 위와 같은 문제를 해결하기 위해서 메모리 풀 생성시 각 버퍼에 1:1 대응하는 할당 버퍼 비트맵과 접근 개수 비트맵을 생성한다. 생성된 버퍼를 할당 할 때, 할당 버퍼 비트맵의 대응되는 비트(bit)를 1 로 set 하고, 접근 개수 비트맵은 다른 태스크가 그 버퍼에 접근할 때 즉, 버퍼의 acc_count 필드가 2 이상이 되면 1 로 set 한다. 접근 개수 비트맵이 set 된 버퍼는 공유 버퍼가 된다. 그리고 버퍼가 해제될 때, 할당 버퍼 비트맵을 0 으로 set(clear)하고, 접근 개수 비트맵은 버퍼의 acc_count 가 0 이 되면 0 으로 set 한다. 하나의 메모리 풀 전체에 대한 해제는 할당 버퍼 비트맵과 접근 버퍼 비트맵이 0 으로 set 된 상태에 있고, 특정 태스크가 메모리가 부족하여 대기(Pending)할 때 메모리 해제가 이루어진다.

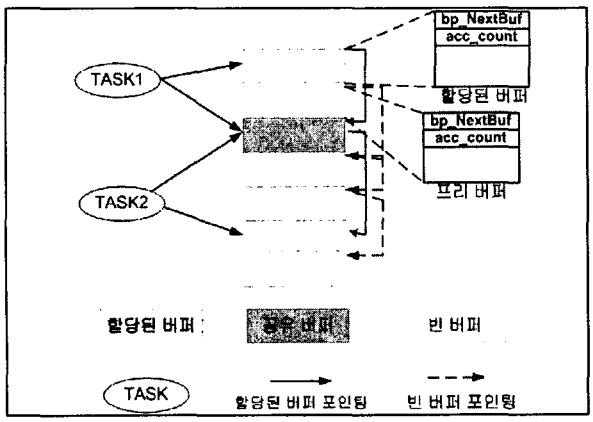
또한, 불필요하게 많이 생성되어 사용되지 않는 버퍼가 있을 경우, 메모리 풀 중간에 사용되는 버퍼의 내용을 앞쪽 빈 버퍼로 복사한 후, 사용되지 않는 공간을 합병하여 해제한다. [그림 4]는 메모리 풀을 관리하기 위한 구조체와 접근 개수 비트맵을 1 로 set 하기 위한 변수 그리고 메모리 풀을 해제하기 위한 함수들이다.

```
typedef struct mk_buffer_pool_struct {
    . . .
    ULONG  al_buf_bitmap;    // 할당 버퍼 비트맵
    ULONG  al_buf_bitmap;    // 접근 개수 비트맵
} MK_POOL;
    // 버퍼를 관리하는 구조체에 멤버 추가

UINT acc_count;   // 버퍼에 대한 접근 개수
```

```
MK_BufCopy();
        // 버퍼의 내용을 앞쪽 빈 버퍼에 복사하는 함수
MK_BufMerge();
        // 사용하지 않은 버퍼를 합병하는 함수
MK_GarbageBufMemFree();
        // 가용한 메모리로 만들어주는 함수
```

[그림 4] 메모리 풀 관리 구조체와 함수




[그림 5] 할당된 메모리 구조 예

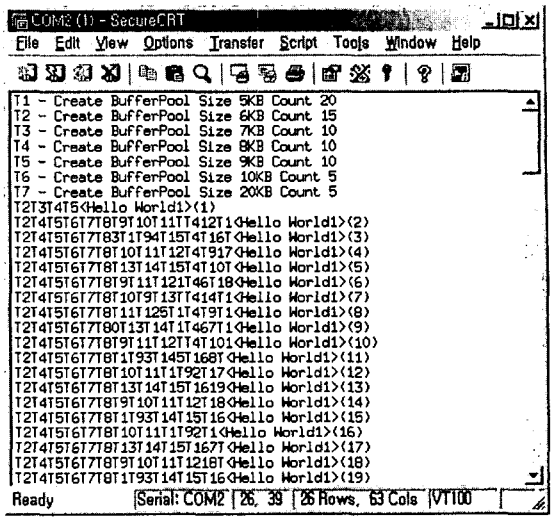
[그림 5] 에서와 같이 Task1, Task2 가 메모리 풀에서 버퍼를 할당 받아 사용하며, 태스크 사이에 공유 버퍼가 있다. Task1 이 MK_FreeBuf()로 버퍼를 해제하고 Task2 는 계속 사용하고 있다면 Task2 의 공유 버퍼가 아닌 버퍼는 메모리 풀의 첫 빈 버퍼에 [그림 4]의 MK_BufCopy()에 의해 복사하고, Task2 의 버퍼에 대한 포인터, 버퍼 풀 관리자의 할당 버퍼 비트맵과 접근 개수 비트맵을 재 조정한 후, 할당 버퍼 비트맵과 접근 개수 비트맵이 모두 clear 된 곳을 확인한다. 그리고 나서 연속된 빈 버퍼는 MK_BufMerge()를 통해 합병하여 MK_GarbageBufMemFree()를 통해 해제한다. 단, 공유버퍼는 상위의 빈 버퍼로 복사하지 않고 acc_count 가 0 이 되면 해제한다.

태스크 생성 또는 수행 시 가용한 메모리가 없어 태스크가 생성되지 못하거나 대기(Pending)할 때, MK_GarbageBuf-MemFree()함수에 메모리 풀 관리자구조체(MK_POOL)를 인자로 하여 메모리를 해제시켜줌으로써 가용한 메모리를 확보한다.

## 4. 테스트 환경 및 결과

본 논문에서 구현한 실시간 운영체제의 메모리 풀의 메모리누수 방지 기법은 삼성에서 제작한 ARM920T 기반의 S3C28-00 Evaluation Board 에 *i*RTOS™를 탑재하여 구현하였으며, 컴파일러는 Metrowwerks 사의 Code Warrior 를 사용하였다.

[그림 6]에서 100 개의 태스크를 생성하고 6 가지 크기의 버퍼 풀을 생성한 결과를 보여주고 있으며, 메모리 부족으로 인한 시스템 성능 저하와 시스템 장애가 일어나지 않았다.




[그림 6] 테스트 결과

## 5. 결론 및 향후 연구 과제

본 논문에서는 실시간 운영체제의 메모리 풀에서의 메모리 관리에 있어서 태스크가 종료된 이후, 메모리가 누수되는 현상을 방지할 수 있는 기법과 보다 효율적으로 메모리를 사용할 수 있는 방법에 대해 설계하고 구현한 내용을 기술하였다.

향후 연구과제로는 실시간 운영체제에서 사용하는 메모리를 보다 효율적이고 시간결정성을 보장할 수 있도록, 전체 메모리를 모니터링하여 할당 및 해제하는 관리기법을 연구할 것이다.

## 참고문헌

[1] Embedded System & RTOS, http://www.inestech.com.

[2] Jean J. Labrosse, "uC/OS The Real-Time Kernel, R&D Publications", 1995.

[3] Brett Hammond, "*Software Quality Vs. Dynamic Memory Allocation*", Embedded System Programming, June 1995

[4] David Lafreniere, "An Efficient Dynamic Storage Allocator", Embedded Systems Programming, Sept. 1998.

[5] 안희중, 박윤미, 성영락, 이철훈, "*Implementation of Memory Management for Real-Time Operating System, iRTOS™*", 한국정보처리학회, 제 10권 제 1호, p.483~486, 2003.05.

[6] IEEE Std 1003.1b, Portable Operating System, 1993