실시간 운영체제의 효율적인 메모리 관리 설계 및 구현

박윤미[0], 이재규, 이철훈
충남대학교 컴퓨터공학과
(ympark[0], jklee)@pplab.ce.cnu.ac.kr, chlee@ce.cnu.ac.kr

Design and Implementation of Memory Management for preventing a memory leakage on Real-Time Operating System, iRTOS™

Yoon-Mi Park[0], Jae-Gyu Lee, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약

최근 임베디드 시스템 분야에서의 실시간 운영체제는 정보가전을 비롯한 임베디드 시스템등 적용범위가 점차 확대되는 추세이다. 실시간 운영체제는 다른 범용 운영체제와는 달리 시간 결정성을 보장하는 운영체제로서, 주로 자원(resource)이 한정된 시스템에 탑재되어야 하기 때문에 효율적인 자원관리가 필요하다. 시스템의 자원 중에서도 메모리는 실시간 운영체제의 실행에 있어서 꼭 필요한 자원이므로 이에 대한 효과적인 관리가 필수적이라 할 수 있다. 대부분 실시간 운영체제에서는 효율적인 메모리 관리를 위해서 동적 메모리 할당 방법을 채택하고 있다. 그러나 할당된 메모리를 해제하지 않고 종료되는 태스크로 인해 메모리 누수 문제가 발생 하였다. 본 논문에서는 동적 메모리 할당에서 메모리 누수를 최소화 할 수 있도록 개선한 메모리 관리 기법을 설계 및 구현 하였다.

1. 서론

임베디드 시스템은 우리 생활에서 쓰이는 각종 전자기기, 가전제품, 제어장치 등을 말한다. 이러한 장비의 특징은 단순히 전기, 전자회로로만 구성된 것이 아니라 마이크로프로세서가 내장되어 있는 시스템을 가리키며, 산업, 가전, 사무, 군사 등의 다양한 응용 분야를 가지고 있다. 최근 임베디드 시스템에서 네트워크나 멀티미디어가 자리잡으면서 임베디드 시스템의 기능이 복잡해졌고, 임베디드 시스템의 특정상 실시간이라는 요소를 만족해야 했으므로 실시간 운영체제가 임베디드 시스템에 도입되었다. 실시간 운영체제는 다른 범용 운영체제와는 달리 시간 결정성을 보장하는 운영체제로서, 주로 자원(resource)이 한정된 시스템에 탑재되어야 하기 때문에 자원 관리의 효율성이 요구 된다. 시스템의 자원 중에서도 메모리는 실시간 운영체제의 실행에 있어서 꼭 필요한 자원이므로, 이에 대한 효과적인 관리가 필수적이라 할 수 있다.

태스크에 메모리를 할당하는 방법은 정적 메모리 할당(Static Memory Allocation)과 실행 시간에 메모리를 할당하는 동적 메모리 할당(Dynamic Memory Allocation)이 있다. 대부분 실시간 운영체제에서는 효율적인 메모리 관리를 위해서 필요할 때 메모리를 할당하고, 그렇지 않을 경우에는 해제하는 동적 메모리 할당 방법을 채택하고 있다. 그러나 할당되 메모리를 해제하지 않고 종료되는 태스크로 인해 메모리 누수 문제가 발생 하였다. 이러한 현상이 반복 된다면 메모리 누수에 의한 메모리 공간 부족으로 애플리케이션이 동작하는데 필요한 메모리를 획득 할 수 없게 되어 시스템 성능을 저하 시킨다. 따라서 본 논문에서는 iRTOS™ 실시간 운영체제에서 메모리 누수를 최소화 할 수 있도록 개선한 메모리 관리 을 설계 및 구현 하였다.

본 논문은 2 장에서 관련 연구를, 3 장에서 메모리 관리 기법에 대한 설계 및 구현 내용을, 4 장에서는 테스트 환경 및 결과, 5 장에서는 결론 및 향후 연구과제를 기술한다.

2. 관련 연구

2.1 실시간 운영체제

우선순위 기반의 선점형 iRTOS™는 실시간 운영체제의 핵심이라고 할 수 있는 멀티태스킹(MultiTasking) 및 ITC 환경을 제공한다.

ITC 환경은 태스크와 태스크의 공동 작업(Cooperation)을 위한 동기화(Synchronization) 및 통신(Communication)을 위하여 세마포, 메시지 메일박스, 메시지 큐등을 지원하고 있다.

2.2 메모리 관리 체계

μRTOS 커널은 동적 메모리 관리를 위해 두 단계의 메모리 관리 기법을 제공한다. 하위 단계는 가변 크기의 메모리를 할당 또는 해제할 수 있는 힙 스토리지 매니저(Heap Storage Manager)이고 상위 단계는 고정 크기의 메모리를 할당, 해제할 수 있는 메모리 풀(Memory Pool)이다.

μRTOS™의 메모리 관리 체계는 그림 1 과 같다.

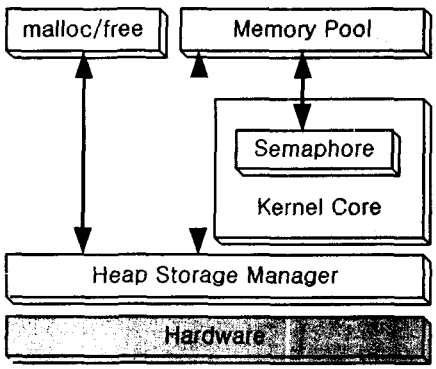

그림 1 메모리 관리 체계

동적 메모리 관리 체계의 힙 스토리지 매니저는 두 가지 문제들을 야기 할 수 있다. 첫째로, 힙(Heap)에서 메모리를 할당할 경우, 동적 메모리 할당 시간이 오래 걸리고 시간결정성을 저해하게 된다. 적당한 크기의 메모리 블록(Memory Block)을 찾기 위해서는 메모리 블록 프리 리스트(Free List)를 찾아야 하는데 이때 걸리는 시간은 시간 결정성을 심각하게 저해하기 때문이다. 둘째로, 힙(Heap)의 단편화(External Fragmentation)때문에 메모리 블록을 제대로 할당 할 수 없다. 이를 해결하기 위해서는 필요한 메모리 크기를 미리 추정하여 최대 크기의 고정된 메모리 풀(Memory Pool)로부터 메모리를 할당하는 것으로 해결 할 수 있다[3].

2.2.1 힙 스토리지 매니저(Heap Storage Manager)

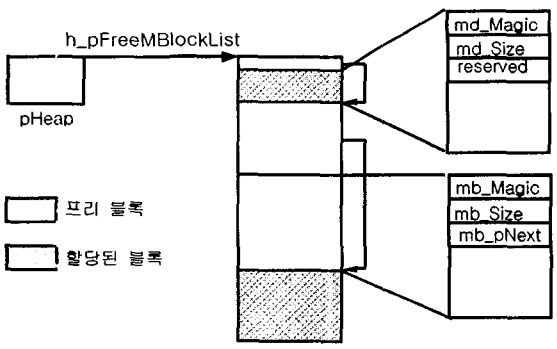

그림 2 힙 스토리지 매니저 구조

μRTOS™의 힙 스토리지 메지져 구조는 그림 2 과 같다. 힙 스토리지 매니저에서 할당되지 않은 프리 블록(Free Block)은 블록 앞에 12 바이트의 프리 블록을 관리하기 위한 구조체가 선언되어 있고, 이를 통해 프리 블록을 관리한다. 힙 스토리지 매니저에서 할당된 메모리는 사용자가 원하는 메모리 크기에 추가적으로 12 바이트의 구조체를 할당된 블록 앞에 선언하고, 이를 통해 할당된 블록을 관리한다.

[그림 3]은 힙 스토리지 매니저를 제어하는 API 함수들이다.

```
STATUS MK_CreateHeapManager(MK_HEAP *pHeap,
  char *pName, void *pAddr, UINT Length, UINT
  MinSize, BOOLEAN Options);
STATUS MK_DeleteHeapManager(MK_HEAP *pHeap);
```

그림 3 힙 스토리지 매니저 생성/삭제

MK_CreateHeapManager()는 힙 스토리지 매니저를 생성하는 함수이고, MK_DeleteHeapManager()는 힙 스토리지 매니저를 삭제하는 함수이다.

3 메모리 관리 체계 설계 및 구현

시스템이 부팅(Booting)되면 힙(Heap)영역은 초기화되고, 힙 스토리지 매니저에 의해 관리된다. 이 영역의 메모리를 할당하는 방법은 정적 메모리 할당과 동적 메모리 할당 두 가지가 있다. 특히 동적 메모리 할당은 실시간 운영체제의 설계에서 임베디드 시스템에 적용되는 특수한 상황 때문에 매우 중요하게 인식 된다.

μRTOS™에서 동작되는 애플리케이션내의 태스크들은 힙 스토리지 매니저에서 관리하고 있는 메모리 영역 내에서 정적 혹은 동적으로 메모리를 할당하고 태스크가 종료될 때 해제한다. 그러나 할당된 메모리를 해제하지 못하고 태스크가 종료되면 메모리 누수를 야기한다.

본 논문에서는 메모리 누수를 최소화 하기 위해서 태스크가 메모리를 할당 시 다른 태스크와 공유되는 공간과 그렇지 않은 공간으로 구분하여 힙 스토리지 매니저를 Global 과 Local 두 부분으로 나눈다.

```
STATUS
MK_CreateHeapManager(MK_HEAP *pHeap, CHAR
    *pName, VOID *pAddr, UINT Length,  UINT
      MinSize, BOOLEAN Options)

STATUS
MK_CreateLocalHeapManager(MK_HEAP *pHeap,
    MK_HEAP *pDstHeap, CHAR *pName,  UINT
    Length, UINT MinSize, BOOLEAN Options)
```

그림 4 Global / Local 힙 스토리지 매니저 생성

다른 태스크와 공유되는 메모리 영역을 할당할 때는

MK_CreateGlobalHeapManager()함수를 이용하고, 그렇지 않을 경우에는 MK_CreateLocalHeapManager()함수를 이용한다.

태스크가 메모리 영역을 해제하지 못하고 종료될 경우, 태스크가 종료될 때 호출되는 MK_DeleteTask()함수에서 Local 힙 스토리지 매니저로 할당 된 영역이 있다면 해당 힙 스토리지 매니저를 삭제 함으로써 메모리의 누수 문제를 최소화 할 수 있다.

```
STATUS
MK_DeleteTask(MK_TASK *pTask)
{
    .....
    MK_DeleteHeapManager(pTask->t_pLocalHeap);
    MK_FreeMemory(pTask->t_pLocalHeapAddr);
    pTask->t_pLocalHeapAddr = MK_NULL;
    pTask->t_pLocalHeap=MK_NULL;
.....
}
```

그림 5 Local 힙 스토리지 매니저 삭제

그림 5 에서 변수 t_pLocalHeap 은 태스크가 Local 힙 스토리지 매니저에 의해 생성된 힙 스토리지 매니저다.

4 테스트 환경 및 결과

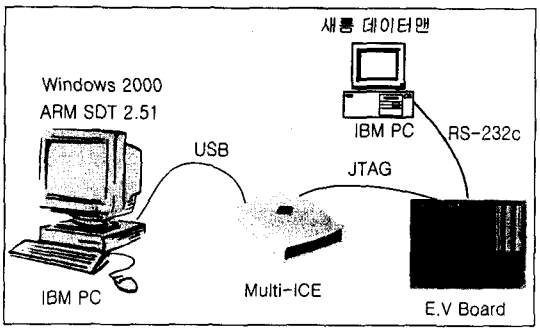

그림 6 테스트 환경

그림 6 에서 보는 바와 같이, 본 논문에서 기술하고 있는 실시간 운영체제는 iRTOS™이고, 컴파일러는 ARM SDT 2.51 을 사용하며, ARM 920T 를 기반으로 한 삼성 S3C2800™ 32-bit RISC MicroProcessor 에서 테스트 하였다.

특정 힙 스토리지 영역 내에서, Global 힙 스토리지 매니저를 생성하여 메모리를 할당하는 TASK2, Local 힙 스토리지 매니저를 생성하여 메모리를 할당하는 TASK1, TASK1 을 삭제하는 TASK3 으로 세개의 테스크를 수행시켰다. 그림 7 에서 보는 바와 같이, TASK1 이 TASK3 에게 삭제되었을 때 TASK1 에서 할당되었던 메모리를 TASK2 에서 할당 받을 수 있었음을 확인 할 수 있었다.

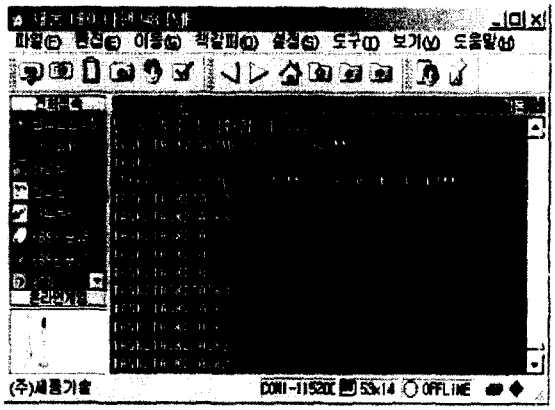

그림 7 테스트 결과

5. 결론 및 향후 연구 과제

특정 기능을 수행하는 임베디드 시스템에서는 적은 용량의 메모리를 가지고 있기 때문에 메모리 공간을 효율적으로 이용하는 것이 중요하다. 태스크를 위해 할당된 메모리를 해제하지 못하고 종료되기 때문에 메모리 누수의 문제가 발생할 수 있다. 그래서 본 논문에서는 Local 힙 스토리지 매니저를 이용하여 메모리 누수 문제를 최소화 할 수 있었다. 향후 연구 과제로는 메모리 누수를 방지하기 위한 가비지 컬렉터 모듈에 대해서 연구되어야 하겠다.

6. 참고문헌

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

[2] " User' s Guide for iRTOS kernel " , ㈜아이네스테크

[3] IEEE Std 1003.1b, *Portable Operating System*, 1993.

[4] David Stepner, Nagarajan Rajan, David Hui, " *Embedded Application Design Using a Real-Time OS*" , Design Automation Conference, 36th, pp.151-156, 1999

[5] 박희상, 안희중, 김용희, 이철훈, " *Design and Implementation of Memory Management Facility for Real-Time Operating System, iRTOS™*" , 한국정보과학회, Vol. 29, No. 1, pp.58-60, 2002.04.