실시간 운영체제에서 메모리 누수 방지 기법 설계 및 구현*

조문행[0], 양회권, 이철훈
충남대학교 컴퓨터공학과,
{mhcho[0], hkyang, chlee}@ce.cnu.ac.kr

The Design and Implementation for Preventing A memory leakage on Real-Time Operating Systems

Moon-Haeng Cho[0], Hui-Kwan Yang, and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam Nationai University

요 약

임베디드 시스템에 사용되는 실시간 운영체제는 한정된 자원의 효율적인 관리가 필수적이다. 특히, 메모리는 운영체제의 실행에 있어서 꼭 필요한 자원으로써, 메모리 관리 기법은 시스템의 성능에 영향을 미칠 수 있기 때문에 실시간 운영체제뿐만 아니라 범용 운영체제에서도 매우 중요하게 다루어지고 있다. 본 논문은 실시간 운영체제에서 발생할 수 있는 메모리 누수 문제를 최소화하기 위한 기법을 설계 및 구현하였다.

1. 서 론

최근 유비쿼터스 컴퓨팅이 대두되면서 임베디드 시스템 및 실시간 운영체제(Real-Time Operating System)에 대한 많은 연구들이 진행되고 있다.

실시간 운영체제는 시간결정성이 보장되는 운영체제로 크게 경성 실시간 운영체제(Hard Real-Time Operating System)와 연성 실시간 운영체제(Soft Real-Time Operating System)로 구분된다. 실시간 운영체제는 주로 시스템의 자원이 제한된 임베디드 시스템에 탑재되기 때문에 효율적인 자원관리가 필요하다. 특히, 메모리는 시스템에서 가장 빈번하게 사용되는 자원으로써 효과적인 메모리 관리 기법을 통하여 시스템의 성능을 향상시킬 수 있다[1][2].

본 논문에서는 실시간 운영체제인 /RTOS™ 기반에서 보다 효율적으로 메모리를 관리하고 메모리 누수 현상을 최소화할 수 있도록 설계 및 구현한 메모리 관리기법을 기술한다[1].

본 논문의 구성은 2 장에서는 관련연구로서 실시간 운영체제인 /RTOS™ 에서 사용하고 있는 메모리 관리 기법에 대해 소개하고, 3 장에서는 본 논문에서 설계 및 구현한 메모리 누수 현상을 효과적으로 제어할 수 있는 기법을, 4 장에서는 테스트 환경 및 결과를 기술한다. 마지막으로, 5 장에서는 결론 및 향후 연구과제에 대해서 기술한다.

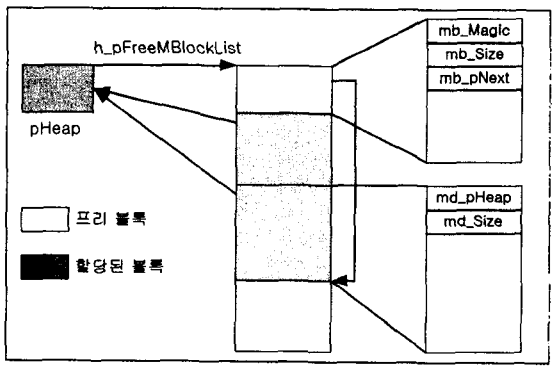
2. 관련 연구

/RTOS™ 에서는 힙 스토리지 매니저(Heap Storage Manager)와 메모리 풀(Memory Pools)을 통해 가변 크기 메모리와 고정 크기 메모리를 할당 및 해제할 수 있는 동적 메모리 관리를 제공한다[3][4].

2.1 힙 스토리지 매니저

시스템 메모리의 힙(Heap)영역을 관리하기 위해 힙 스토리지 매니저를 사용한다.

힙 스토리지 매니저의 구조는 [그림 1]과 같다.



[그림 1] 힙 스토리지 매니저

* 본 논문은 산업자원부 중기거점과제 연구비 지원에 의한 것임

힙 영역에서 동적 메모리의 할당은 MK_GetMemory()를 통해 이루어진다. 먼저, 퍼스트 핏(First-Fit) 알고리즘에 따라 프리 블록 리스트로부터 적당한 메모리를 선택하여 Heap 을 가리키는 포인터(md_pHeap)와 할당된 메모리 크기(md_size)로 구성된 struct mk_heap_dummy 구조체인 헤더를 포함하여 할당한다. 만약, 할당 가능한 메모리가 존재하지 않으면 메모리를 요청한 태스크는 적당한 메모리가 생길 때까지 Pending 된다. 메모리 해제는 MK_FreeMemory()에 의해 이루어진다. 먼저, 프리 블록 리스트에서 해제할 메모리 블록에 인접한 다른 프리 블록이 있는지 찾아보고 존재하면 병합(Merge)하고, 없으면 프리 블록 리스트에 추가한다. 프리 블록에 대해서는 메모리에 대한 식별자(mb_Magic)와 프리 블록의 사이즈(mb_Size), 다음 프리 블록을 가리키는 포인터(mb_pNext)로 구성된 struct_memory_block_struct 가 포함된다[2][4].

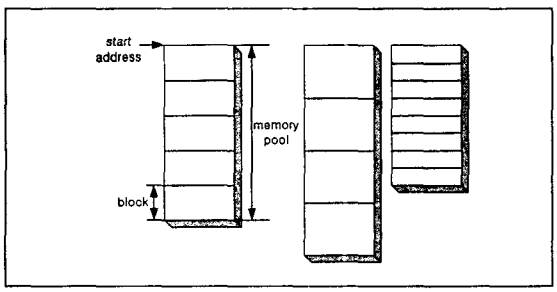
2.2 메모리 풀(Memory Pool)

힙 스토리지 매니저는 메모리 할당을 위해 필요한 가용 메모리 검색 시간에 있어서의 시간 결정성과 메모리 단편화(External Fragmentation)문제가 있기 때문에 메모리 풀을 사용한다.

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

{ 생성할 버퍼의 개수(Count) × (버퍼의 크기(Size) + 다음 버퍼의 포인터(4Byte)) }

[그림 2]에서 힙에서 할당 받은 메모리의 시작주소(pAddr)와 버퍼의 개수(Count), 버퍼의 크기(Size)를 인자로 하여 생성된 메모리 풀은 할당 받은 메모리를 주어진 고정 크기의 버퍼 단위로 나뉘게 되며, 버퍼의 크기에 따라 여러 크기의 메모리 풀을 생성할 수 있다. 각 버퍼는 세마포의 자원이 되고, 개수는 세마포의 카운트 값이 된다.



[그림 2] 메모리 풀

메모리 풀에서 MK_GetBuf()에 할당 받을 버퍼의 메모리 풀(pPool), 할당 받은 버퍼의 주소(pAddr), 시간틱(Ticks)을 인자로 하여 메모리를 할당 한다. 메모리의 할당은 세마포 메커니즘에 따라 버퍼를 할당하면 Count 를 하나 감소하고, Count 가 0 이하가 되면 메모리를 요청한 태스크는 PendingList 에 삽입되고 스케줄링 된다. 또한, 할당된 버퍼는 4Byte 의 자료구조로 자신이 속한 메모리 풀을 가리키며, 프리 버퍼는 메모리 풀에 리스트로 연결되어 있어 리스트의 맨 앞에서 버퍼를 할당한다(First-Fit). 메모리의 해제는 MK_Fre-eBuf()에 해제할 메모리 주소를 매개변수로 하여 이루어진다. 버퍼의 할당과 마찬가지로 세마포 메커니즘에 따라 버퍼를 해제하면 Count 를 하나 증가시키고 반납된 버퍼의 앞 4Byte 를 이용하여 버퍼가 속한 메모리 풀을 찾아, 메모리 풀의 프리 리스트에 연결하고 해제한다[2][4].

3. 메모리 누수 방지 기법 설계 및 구현

3.1 메모리 누수 현상

태스크가 실행 중에 할당 받는 메모리는 다른 태스크 또는 ISR 이 공유하는 것과 그렇지 않은 것이 있다. 할당된 메모리 중 공유하지 않는 메모리는 더 이상 사용되지 않으면 반드시 해제되어야 한다. 그렇지 않고 방치된 메모리는 시스템의 성능을 떨어뜨리거나 더 큰 문제를 야기시킬 수 있다. 예를 들어, Task A 가 수행 중에 할당 받은 메모리를 해제하지 않고 종료되면, Task A 가 사용하던 메모리는 더 이상 사용할 수 없게 된다. 이렇게 사용되지 않는 메모리는 더 이상 사용할 수 없게 되어 메모리 누수로 이어진다. 메모리 누수현상이 누적되면 시스템의 성능은 점차 떨어지고 급기야 시스템 장애에 이르게 된다.

3.2. 메모리 누수 방지 기법의 설계 및 구현

본 논문에서는 위와 같은 문제를 해결하기 위해서 태스크의 메모리 중 태스크들 사이에서 공유하지 않는 메모리를 로컬 메모리로 하여 로컬 메모리 리스트로 관리한다. 공유메모리는 함부로 해제할 수 없기 때문에 로컬 메모리 리스트에 추가하지 않는다.

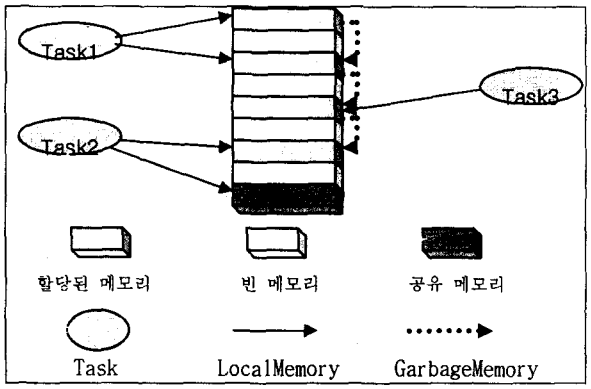
태스크의 로컬 메모리는 MK_GetLocalMem()로 할당하며, 나머지는 MK_GetMemory()로 할당한다. 해제 시킬 메모리들의 리스트를 가리키는 포인터인 pLocalMemHead 를 전역변수로 두고, 할당된 메모리를 가리키는 포인터를 Task Control Block 에 추가하고, 할당된 블록을 나타내기 위한 struct mk_heap_dummy_struct 에 md_pNext 를 추가한다. 이렇게, MK_GetLocalMem()로 할당된 메모리는 pLocalMemHead 로 리스트를 유지하여 이를 통해 메모리 누수를 방지 한다. [그림 3], [그림 4]는 필요한 변수와 함수에 대한 선언이다.

```
typedef struct mk_task_struct {
   . . .
  struct  mk_task_struct *t_pMemHead;
            // Task에 할당된 메모리를 가리키는 포인터
   . . .
} MK_TASK;
```

[그림 3] Task Control Block

```
UINT  *pLocalMemHead;
    // 해제시킬 메모리를 가리키는 포인터
MK_GetLocalMem(UINT *pLocalMemHead);
      // 메모리를 할당해 주는 함수
MK_GarbageMemFree();
     // 가용한 메모리로 만들어주는 함수
```

[그림 4] 메모리 해제를 위한 포인터와 함수



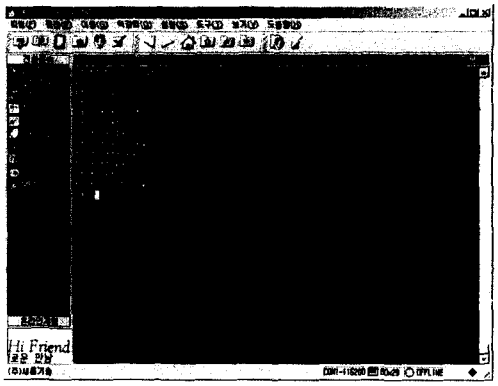
[그림 5] 할당된 메모리 구조 예

[그림 5] 에서와 같이 Task1, Task2 및 Task3 를 위하여 할당된 메모리가 존재하며, 메모리에는 Task2 와 Task3 의 공유메모리가 존재한다. Task1, Task3 순으로 종료되었을 때, Task1 에서 사용했던 LocalMemory 블록 1 의 md_pNext 는 블록 3 을 가리키고 있고 블록 3 의 md_pNext 는 NULL 이므로 pLocalMemHead 가 pMemHea-d 를 가리키게 하여 해제할 메모리 블록 리스트를 생성한다. 이와 같이, Task3 의 LocalMemory 의 pMemHead 를 이전 태스크의 md_pNext 가 NULL 인 곳을 가리키도록 함으로써 해제할 메모리 블록 리스트를 생성한다.

태스크 수행 또는 생성시 메모리 부족으로 인해 Pen-ding 되는데, 그전에 MK_GarbageMemFree()에 해제할 메모리 블록 리스트의 헤더인 pLocalMemHead 를 인자로 넘겨주어 메모리를 해제시켜줌으로써 가용한 메모리를 확보할 수 있다.

4. 테스트 환경 및 결과

본 논문에서 구현한 실시간 운영체제에서의 메모리 누수 방지 기법은 삼성에서 제작한 ARM920T 기반의 S3C2800 Evaluation Board 에 *iRTOS™* 를 탑재하여 구현하였으며, 컴파일러는 ARM 사의 SDT2.51 을 사용하였다. 그 테스트 결과, 많은 Task 를 생성하였을 때 메모리 부족으로 인한 시스템 장애가 일어나지 않았다.



[그림 6] 테스트 결과

5. 결론 및 향후 연구 과제

본 논문에서는 실시간 운영체제의 메모리 관리에 있어서 태스크가 종료된 이후 메모리가 누수되는 현상을 방지할 수 있는 기법을 설계하고 구현한 내용을 기술하였다.

향후 연구과제로 실시간 운영체제에서 누수되는 메모리를 회수하는 MK_GarbageMemFree()의 실행 시점과 메모리 풀에서 누수되는 메모리를 효과적으로 제어할 수 있는 기법이 있다.

참고문헌

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

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

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

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

[5] WindRiver, " *VxWorks Progammer's Guide*" , 1997.

[6] Accelerated Technology, " *Nucleus PLUS Internals Manual*" , 1996.

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