실시간 운영체제에서 분산된 프리 메모리 할당 기법의 설계 및 구현

이승열°, 이원용, 조문행, 이철훈
충남대학교 컴퓨터공학과
{sy-lee°, nissikr, mhcho, clee}@cnu.ac.kr

Design and Implementation of Allocation of the distributed free memory for Real-Time Operating Systems

Soong-Yeol Lee°, Won-Yong Lee, Moon-Haeng Cho, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National University

요 약

자원이 한정적인 시스템을 위해 실시간 운영체제는 자원을 효율적으로 관리해야 한다. 대부분 실시간 운영체제는 효율적인 메모리 관리를 위해 동적으로 메모리를 할당한다. 하지만 동적 메모리의 할당과 해제의 반복은 비연속적인 메모리 공간을 생성하고 이런 비연속적인 메모리의 생성은 단편화와 같은 성능을 저하시키는 요인이 된다. 논문은 실시간 운영체제에서 시간결정성을 높여줄 수 있도록 분산된 프리 메모리 블록을 할당하는 기법을 설계 및 구현하였다.*

1. 서 론

실시간 운영체제는 주로 시스템의 자원이 제한된 임베디드 시스템에 탑재되기 때문에 자원을 효율적으로 관리할 필요가 있다. 특히, 메모리는 시스템에서 가장 중요한 자원의 하나로써 효과적인 메모리 관리 기법을 통해 시스템의 성능을 향상시키고 메모리 낭비를 막을 수 있다[1][2].

본 논문은 실시간 운영체제의 가변길이 메모리 할당을 위한 효율적인 메모리 관리를 통해 메모리의 낭비를 최소화하여 보다 높은 실시간성을 보장할 수 있도록 분산된 프리메모리 블록 할당기법을 설계 및 구현하였다. 그리고 이 기법을 상용실시간 운영체제인 UbiFOS™ 에 적용하여 검증을 하였다[2][3].

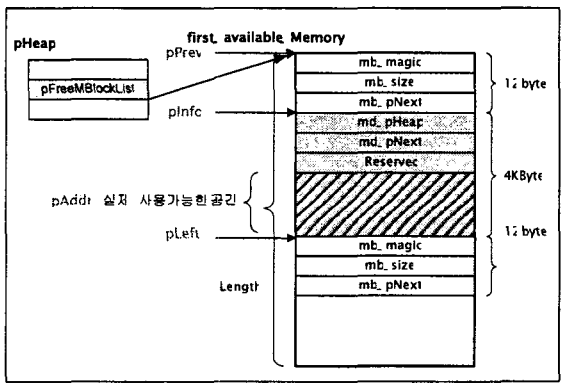
본 논문의 구성은 2 장에서 관련연구로 UbiFOS™에서 사용하고 있는 메모리 관리기법을 소개하고, 3 장에서는 본 본문에서 제안한 분산된 프리 메모리 블록의 할당 기법의 설계 및 구현을, 4 장에서는 테스트 환경 및 결과를 설명한다. 마지막으로, 5 장에서는 결론 및 향후 연구과제를 기술한다.

2. 관련 연구

UbiFOS™ 의 커널은 메모리 관리를 위해 가변 크기의 메모리를 할당, 해제할 수 있는 힙 스토리지 매니저(Heap Storage Manager)와 고정 크기의 메모리를 할당, 해제할 수 있는 메모리 풀(Memory Pools)을 사용한다[1][3].

2.1 힙 스토리지 매니저

아래 그림은 힙 스토리지 매니저의 구조를 보여주고 있다.



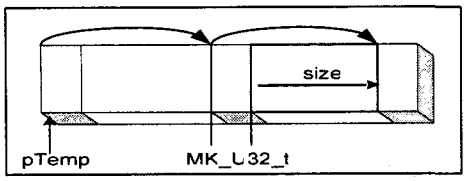
[그림 2-1] 힙 스토리지 매니저

힙 스토리지 매니저의 구성은 할당되지 않은 프리 메모리 블록 앞에 프리 블록을 관리하기 위한 12 바이트 크기의 구조체와 사용자가 원하는 메모리 크기에 추가된 8 바이트의 구조체로 이루어지며 이를 통해 할

* 본 논문은 국방과학연구소의 실시간 운영체제 인터페이스용 미들웨어 연구과제의 수행결과임.

당된 블록을 관리한다. 할당된 블록에 선언된 구조체는 자신의 힙 스토리지 매니저와 할당된 크기를 보관하며, 메모리의 해제 시에 이 구조체를 사용하여 쉽게 메모리를 해제할 수 있다[2][4].

2-2. 메모리 풀(Memory Pool)



[그림 2-2] 메모리 풀

메모리 풀은 UbiFOS™에서 메모리를 할당하는 또 다른 방법으로 힙 스토리지 매니저에서는 메모리를 할당하기 위해 가용 메모리 검색시간에 따른 시간결정성과 메모리 단편화(External Fragmentation)문제가 있기 때문에 고정길이 메모리 풀을 사용하여 이를 해결한다.

메모리 풀은 시스템 초기화 시에 미리 태스크 수행에 필요한 메모리를 결정하고, 필요한 메모리를 힙 스토리지 매니저에서 할당 받아 고정 크기의 버퍼로 구성하고, 메모리 할당 요청이 오면 버퍼 하나를 할당해 주고 해제 시는 버퍼를 해제한다. 고정 크기의 버퍼로 관리됨으로 필요한 메모리보다 큰 메모리를 받을 수 있으므로 메모리의 내부 단편화 문제가 발생하여 메모리의 낭비가 있지만, 버퍼의 크기를 다르게 하여 여러 개의 메모리 풀을 생성하고 적당한 메모리 풀에서 메모리를 할당 받음으로써 메모리 낭비를 줄일 수 있다[2][4].

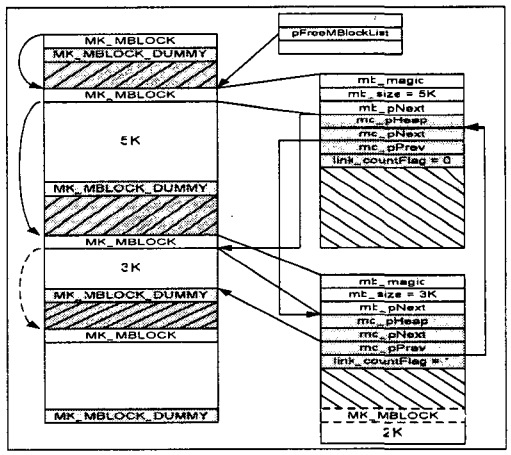
3. 메모리 관리 체계 설계 및 구현

3-1. 동적 메모리 할당

태스크가 메모리를 요청하면 실시간 운영체제는 힙 스토리지 매니저나 메모리 풀에서 적당한 크기의 메모리 공간을 할당해준다. 하지만 동적으로 메모리를 할당해주는 힙의 경우 메모리의 할당과 해제가 반복되게 되면 가용 메모리 공간이 분산된다. 이런 조건에서 메모리를 요청할 경우 퍼스트 핏 정책은 요청하는 메모리 크기를 프리 메모리 블록 리스트에서 순차적으로 검색함으로써 검색시간의 오버헤드를 갖게 된다. 기존 UbiFOS™에서는 MK_GetMemoryInternal() 함수를 통해 프리 블록 리스트에서 순차적으로 메모리를 할당해 준다. 하지만 힙에서 메모리를 할당할 경우 환경에 따라서 동적 메모리 할당 시간이 오래 걸리고 시간결정성을 저해하게 된다. 이유는 적당한 크기의 프리 메모리 블록을 찾기 위해서는 프리 메모리 블록 리스트를 순차적으로 검색해야 하기 때문이다.

3-2. 분산된 메모리 공간의 할당

위에서 언급했듯이 동적 메모리 할당은 프리 블록 리스트를 검색해야 하는 단점과 단편화 문제점이 있다. 이를 보완한 것이 메모리 풀이지만 이 역시 요청하는 메모리보다 크기가 큰 고정 크기의 메모리 공간을 할당해 줌으로써 메모리 낭비를 가져올 수 있다. 힙을 통한 동적 메모리 할당과 메모리 풀을 통한 메모리 할당은 단편화라는 문제점을 가지고 있다. 이 문제를 해결하는 한 방법으로 요청하는 메모리 크기이상의 메모리를 프리 블록 리스트에서 검색하는 방법대신 경우에 따라서 근접해 있지 않는 프리 블록 조각들에 나누어 할당해 주는 방법을 설계 및 구현하였다.



[그림 3-1] 6K 할당의 예

위 그림[3-1]은 프리 블록보다 큰 메모리를 요청했을 경우 사이즈가 작은 두 개의 프리 메모리 블록에 나누어 할당해주는 것을 보여주고 있다. 먼저 메모리를 요청하게 되면 사용자가 지정한 Max_SearchCount 값만큼 프리 메모리 블록 리스트를 검색하고 일치하는 프리 메모리 블록이 존재하면 할당해 준다. 그렇지 않을 경우 요청한 메모리 크기 이상이 될 때까지 MK_MBLOCK 을 따라가면서 pNext->mb_size 의 값을 더해 준다. 그 명령을 수행하면서 Link_countFlag 를 1 씩 증가시킨 후 저장한다. 변수 Link_countFlag 가 양수일 경우는 하나의 태스크에 몇 개의 프리 메모리 공간이 할당되었다는 것을 나타내주며 후에 메모리를 해제할 경우에도 사용된다. 그리고 긴 리스트에 따른 성능 저하를 막기 위해 Link_countFlag 값은 적용시스템에 따라 개발자가 제한을 둘 필요가 있다.

그림[3-2]는 할당 받은 블록을 관리하는 API 함수이다.

```
MK_MemoryWrite(MK_U32_t *pAddr, MK_U32_t *pData)
        // 분산된 메모리에 기록하는 함수
MK_MemoryRead(MK_U32_t *pAddr)
        //분산된 메모리에서 읽어오는 함수
```

[그림 3-2] 할당 받은 블록을 관리하기 API 함수

```
typedef struct mk_heap_dummy_struct {
        struct    mk_heap_struct *md_pHeap;
        MK_U32_t   md_Size;

        /* 분산된 메모리를 나타내주는 플래그
        MK_U32_t Link_countFlag;
        /* 같은 태스크에 할당된 메모리를 연결 */
        struct mk_heap_dummy_struct *md_pNext;
        struct mk_heap_dummy_struct *md_pPrev
} MK_MBLOCK_DUMMY;
```

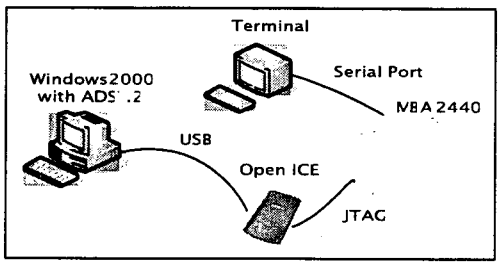
[그림 3-3] 할당 받은 블록을 관리하기 위한 구조체

분산된 메모리가 할당되어 접근할 경우 각각의 분산된 메모리 시작 주소를 포함해야 하기 때문에 MK_MemoryWrite()함수와 MK_MemoryRead()함수를 통해 기록과 읽기가 이루어진다.

메모리는 FreeMemoryInternal() 함수를 통해 삭제되며 Link_countFlag 값을 참조해서 0 이상의 값을 가지고 있을 경우 리스트를 통해 해제가 이루어진다.

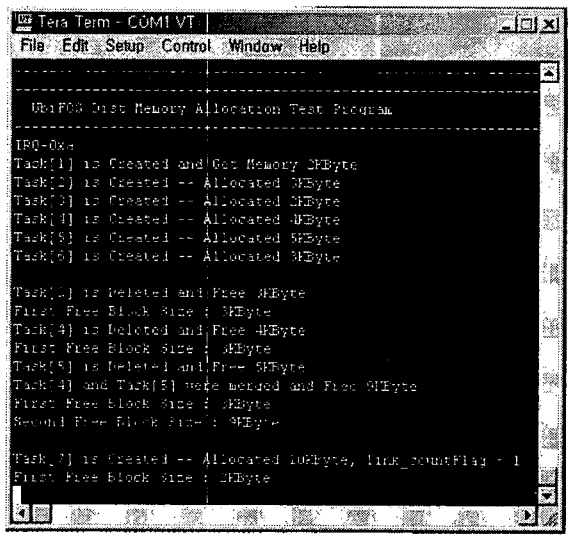
하지만 프리 블록이 연속될 경우 하나의 프리 블록으로 병합되어야 하는데 반환된 블록이 다른 프리 블록과 이웃할 경우, 반환된 블록이 두 개의 프리 블록 사이에 있을 경우, 반환된 블록이 다른 프리 블록들과 떨어져있을 경우를 고려해야 한다.

4. 테스트 환경 및 결과



본 논문에서 기술하고 있는 실시간 운영체제인 UbiFOS™ 는 32-bit Processor 를 대상으로 설계되었으며 컴파일러는 ARM ADS 1.2 을 사용하였으며, 삼성에서 제작한 ARM920T 기반의 MBA2440 Evaluation Board 에 다운로드 해서 테스트 하였다. [그림 4-2]는 시스템에서 태스크를 생성하고 삭제하는 예제를 수행한 결과로 6 개의 태스크를 생성, 3 개의 태스크를 삭제한 후 비연속적인 메모리 환경에서 태스크를 생성해주기 위해 프리 메모리 블록 리스트의 첫 번째 공간보다 큰 사이즈의 태스크를 할당해준 결과이다

[그림 4-1] 테스트 환경



[그림 4-2] 테스트 결과

5. 결론 및 향후 과제

실시간 운영체제에서 메모리를 동적으로 할당할 때 프리 메모리 블록 리스트를 검색하는 시간과 발견하지 못한 경우 대기 상태로 전환되어 메모리를 기다리는 시간을 고려한다면 분산된 메모리 할당 기법은 실시간 운영체제의 시간결정성을 높여 줄 수 있다.

향후 연구 과제로는 실시간 시스템에서 안정성과 시간결정성을 더욱 강화하기 위해 분산된 메모리를 수집하는 방법과 그 기법을 사용하는 시점을 정확히 알 수 있도록 연구가 진행되어야 할 것이다.

6. 참고 문헌

[1] Jean, J. Labrosse, *"μ C/OS-II The Real-Time Kernel"*, R&D Books, 1999

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

[3] http://www.aiji.co.kr/

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