

실시간 운영체제 UbiFOS™에서 메모리 압축 기법 설계 및 구현

이원용⁰, 이승열, 김용희, 이철훈

충남대학교 컴퓨터공학과

{nissikr⁰, sy-lee, yonghee, chlee}@cnu.ac.kr

The Design and Implementation of Memory Compaction for Real-Time Operating System

Won-Yong Lee⁰, Soong-Yeol Lee, Yong-Hee Kim, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National University*

요약

실시간 운영체제를 탑재한 임베디드 시스템(Embedded System)은 특성상 다른 시스템에 비해 상대적으로 저 용량의 메모리를 지닌다. 따라서 제한된 메모리를 효율적으로 사용할 수 있는 기법이 적용될 필요가 있다. 외부 단편화(External Fragmentation)로 생긴 메모리 공간을 재조정하는 기법도 효율적인 메모리 사용을 위한 방안이며, 본 논문에서는 실시간 운영체제에서 사용 가능한 메모리를 효율적으로 활용할 수 있는 메모리 압축(Memory Compaction) 기법에 대해서 설계 및 구현하였다.

1. 서론

임베디드 시스템은 시스템을 동작시키는 소프트웨어를 하드웨어에 내장하여 특수한 기능만을 수행하는 시스템이다.

임베디드 시스템에 탑재되는 실시간 운영체제는 제한된 하드웨어 자원을 효율적으로 관리할 수 있어야 한다. 특히, 한정된 자원인 메모리는 프로세서와 함께 운영체제에 중요한 자원으로 효율적인 메모리 관리 기법을 통해 시스템의 성능을 향상 시킬 수 있다.

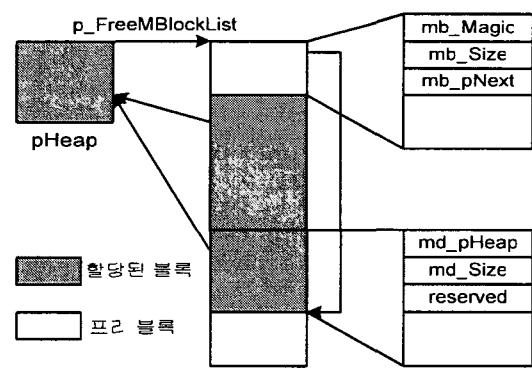
본 논문의 구성은 2 장에서는 관련 연구로서 실시간 운영체제인 UbiFOS™에서 사용하고 있는 메모리 관리 기법에 대해 설명하고 3 장에서는 메모리 단편화 현상과 효율적인 메모리 사용을 위한 압축 기법의 구현 내용에 대해 기술하고 4 장에는 테스트 환경 및 결과를 설명하고, 5 장에서는 결론 및 향후 연구과제에 대해서 논하고자 한다.

2. 관련 연구

UbiFOS™ 커널은 동적 메모리 관리를 위해 두 단계의 메모리 관리기법을 제공한다. 힙 스토리지 매니저로부터 가변 크기의 메모리를 동적으로 할당 받을 수 있는 하위단계가 있고, 고정크기의 메모리를 할당 받을 수 있는 상위단계의 메모리 풀이 있다. 메모리 풀 사용으로 가변크기의 메모리를 할당 받을 시 생길 수 있는 힙(Heap)의 외부 단편화 현상을 보안 할 수 있도록 지원하고 있다. 그러나 메모리 풀에서 고정된 크기의 베퍼를 할당 받아서 사용하는 것 역시 내부 단편화 현상을 피할 수 없다.

2.1 힙 스토리지 매니저

시스템이 부팅되면 하드웨어상의 메모리는 힙 스토리지 매니저에 의해 관리된다.



[그림 2-1] 힙 스토리지 매니저

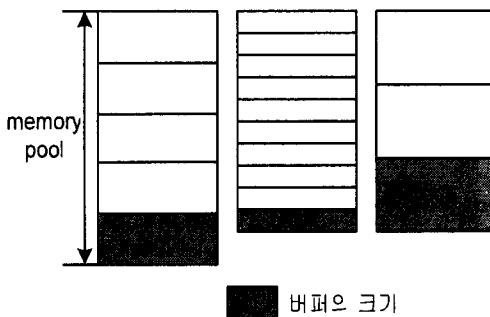
[그림 2-1]은 힙 스토리지 매니저가 힙 영역의 메모리를 리스트로 관리하는 구조를 나타낸다. 힙 스토리지 매니저에서 할당되지 않은 메모리와 할당된 메모리를 관리하기 위해 각각 구조체를 선언하고 관리한다 [3][4].

2.2. 메모리 풀

힙 스토리지 매니저는 메모리 할당을 위해 필요한 가용 메모리 블록의 검색시간 때문에 시간 결정성을

* 본 논문은 국방과학연구소의 실시간 운영체제 인터페이스용 디들웨어 연구과제의 수행결과임.

저해한다. 이 점을 보완하기 위해 고정크기 메모리 풀을 사용한다.



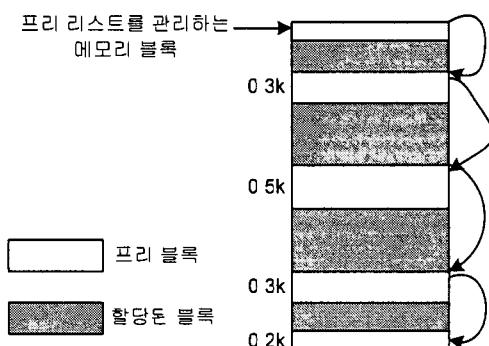
[그림 2-2] 메모리 풀

[그림 2-2]는 힙에서 할당 받은 메모리 시작주소(pAddr)와 베퍼의 개수(count), 베퍼의 크기(size)를 인자로 하여 생성된 메모리 풀로 할당 받은 메모리를 주어진 고정 크기의 베퍼 단위로 나뉘게 되며, 베퍼의 크기에 따라 여러 크기의 메모리 풀을 생성 할 수 있다. 각 베퍼는 세마포로 관리되고 베퍼의 개수는 세마포의 카운트가 된다[1][2].

3. 메모리 압축 기법 설계 및 구현

힙 스토리지 매니저에서 GetMemory()를 통하여 메모리를 할당 받는데, 퍼스트 펫(first-fit) 알고리즘에 따라 프리 블록 리스트로부터 할당 가능한 메모리 블록이 나타나면 할당해주고 남은 영역은 다시 프리 리스트에 추가한다.

3.1 메모리 단편화 현상



[그림 3-1] 메모리 단편화 예

[그림 3-1]에서와 같이 가용한 메모리 블록 크기가 크지 않은 상황에서 태스크가 $0.6k$ 를 요구한다고 가정하자. 현재 가용한 메모리의 총 합이 $1.3k$ 임에도 불구하고 메모리 단편화 때문에 태스크에서 요구하는

0.6k 를 할당 할 수 없게 된다. 따라서 테스크가 PENDING 상태로 들어가서 메모리가 반납 될 때까지 기다리게 된다. 이처럼 이용할 메모리 공간이 있음에도 불구하고 메모리 할당을 받지 못하는 상태가 발생한다.

3.2 메모리 압축 기법 설계 및 구현

본 논문에서는 이와 같은 외부 단편화 문제를 해결하기 위해 힙 스토리지 매니저에 할당된 메모리의 크기와 메모리의 어느 위치에서부터 어떤 크기로 할당되어 있는가에 대한 정보를 알기 위해 할당된 메모리에 대한 포인터와 비트맵을 두어 메모리에 대한 정보를 관리한다.

```

typedef struct mk_task_struct {
    MK_U32_t t_Magic;
    MK_U32_t t_Priority;
    struct mk_heap_dummy_struct *t_Pointer;
    // 할당된 메모리를 가리키는 포인터 추가
} MK_TASK;

typedef struct mk_heap_dummy_struct {
    struct mk_heap_struct *md_pHeap;
    MK_U32_t md_Size;
    struct mk_heap_dummy_struct *md_pNext;
    // 할당된 메모리 다음을 가리키는 포인터 추가
} MK_MBLOCK_DUMMY;

```

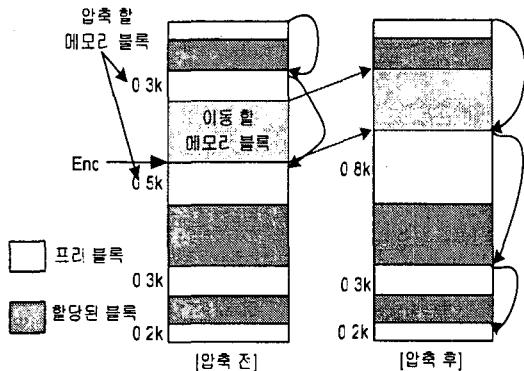
[그림 3-2] 할당된 메모리 관리를 위한 구조체

[그림 3-2]에서 메모리 압축을 하기 위해서는 할당된 메모리 위치를 알고 있어야 하므로 할당된 메모리를 관리하는 구조체에 다음 할당된 블록을 가리키는 포인터를 추가 하였다. 그리고 힙 스토리지 매니저에 al_bitmap을 통해 할당된 메모리 영역에 대한 위치와 h_pAllocBlockList 포인터를 통해서 할당된 메모리 영역에 대한 순차적 접근으로 어떤 태스크에 어떤 크기로 할당되었는지를 알 수 있다. 그리고 TCB(Task Control Block)에 태스크가 접근한 메모리 시작위치를 가리키는 포인터를 추가하여 메모리 압축 후에 변환된 메모리 위치를 태스크가 가리키게 하였다.

힙 스토리지 매니저에서 태스크가 메모리를 요구하면 가용한 메모리가 있는지를 알기 위해 프리 리스트를 검색하고 할당 가능한 메모리가 없으면 PENDING 상태로 일단 들어간다. 가용한 메모리의 합을 나타내는 `h_FreeSize`의 값이 PENDING 상태에 있는 태스크의 메모리 사이즈보다 클 경우 `MK_CompactionCopy()`를 호출해서 메모리 압축을 한다.

`MK_CompactionCopy()` 함수에서 `MK_GetFirstPendingTask()`를 통해 PENDING 상태 태스크의 메모리 크기를 얻어온다. 그런 후에 힙 스토리지 매니저에서 프리 리스트를 순차적으로 검색하여 메모리를 더하다가 PENDING 상태 태스크의 메모리 크기보다 크게 되면 그 위치를 `End`로 정하고, 가용한 메모리 시작을 가리키

는 `h_pFreeMBlockList` 와 `End` 사이에 있는 할당된 블록들을 위치 이동시킨다.



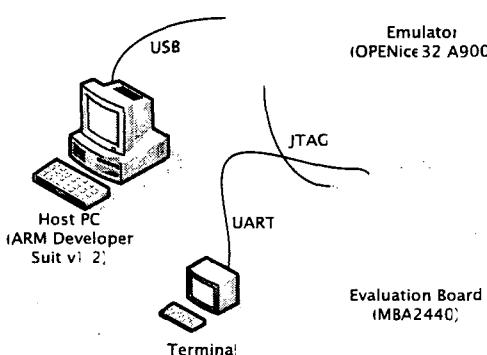
[그림 3-3] 메모리 압축 전·후 메모리 블록 위치

할당된 메모리를 위치 이동에 따른 시간을 최소화하기 위해서 [그림 3-3]에서와 같이 힙 스토리지 매니저가 관리하는 메모리 전체에 대해 하지 않고 태스크가 요구했던 크기를 만족하는 만큼만 한다.

필요한 메모리 공간만큼 메모리 압축이 일어난 후에 PENDING 상태에 있던 태스크를 READY 상태에 넣어주고 스케줄링을 해줘서 메모리를 할당 받을 수 있게 한다.

4. 테스트 환경 및 결과

본 논문에서 구현한 메모리 압축 기법은 실시간 운영체제 UbiFOS™에 구현하였고 컴파일러는 ARM ADS 1.2을 사용하였으며, ARM920T 기반의 MBA2440 보드에 테스트 하였다.



[그림 4-1] 테스트 환경

그림 4-2은 험 스토리지 매니저가 10k의 메모리 공간을 할당 받고 이 험 스토리지 매니저로부터 태스크 T1부터 T7까지 각각 메모리 블록을 할당 받은 후, 태스크 T2, T4, T6를 삭제해서 메모리를 반납한다. 그런 다음 태스크 T8이 메모리 0.6k를 요청하면 할당 가능한 메모리 공간이 없어서 PENDING 상태로 들어간다. 이후 메모리 압축을 하고 PENDING 상태에 있는 태스크 T8을 메모리 압축이 일어난 공간에 할당해서 T8이 실행되는 결과이다.

```

Tera Term - COM1 VT
File Edit Setup Control Window Help
UbiFOS [Memory Compaction] Test Program
T1 is allocated memory size-5k
T2 is allocated memory size-0.3k
T3 is allocated memory size-2k
T4 is allocated memory size-0.5k
T5 is allocated memory size-1k
T6 is allocated memory size-0.3k
T7 is allocated memory size-0.8k
T2 is deallocated memory size-0.3k !!!
T4 is deallocated memory size-0.5k !!!
T6 is deallocated memory size-0.3k !!!
T8 requests memory[0.6k] -> PENDING STATE
CompactionCopy() -> T8 is allocated memory size-0.6k

```

[그림 4-2] 실행 결과

[그림 4-2]은 험 스토리지 매니저가 10k의 메모리 공간을 할당 받고 이 험 스토리지 매니저로부터 태스크 T1부터 T7까지 각각 메모리 블록을 할당 받은 후, 태스크 T2, T4, T6를 삭제해서 메모리를 반납한다. 그런 다음 태스크 T8이 메모리 0.6k를 요청하면 할당 가능한 메모리 공간이 없어서 PENDING 상태로 들어간다. 이후 메모리 압축을 하고 PENDING 상태에 있는 태스크 T8을 메모리 압축이 일어난 공간에 할당해서 T8이 실행되는 결과이다.

5. 결론 및 향후 과제

본 논문에서는 실시간 운영체제 UbiFOS™의 메모리 관리를 보다 명확하게 하는 메모리 모니터링 기법과 이를 통해 할당된 메모리에 대한 정보를 파악하고 이를 바탕으로 메모리 단편화 현상을 대처할 수 있게끔 메모리 압축 기법에 대해 설계하고 구현한 내용을 기술하였다.

향후 연구과제로는 실시간 운영체제에서 시간 결정성을 보장하면서 보다 효율적인 메모리 관리를 할 수 있도록 메모리 모니터링 기법과 가용한 메모리 블록을 선택하는 방법 및 압축 시점에 대한 연구가 계속 되어야 하겠다.

참고 문헌

- [1] 조문행, 최인범, 정명조, 이철훈, "The Design and Implementation of Memory Monitoring Mechanism for Preventing A memory leakage on Real-Time Operating Systems", 한국정보과학회, Vol. 32. No. 1(A), pp.859-861, 2005.
- [2] Brett Hammond, "Software Quality Vs. Dynamic Memory Allocation", Embedded System Programming, June 1995.
- [3] David Lafreniere, "An Efficient Dynamic Storage Allocator", Embedded Systems Programming, Sept. 1998.
- [4] Qing Li, Caroline Yao, "RTOS를 이용한 실시간 임베디드 시스템 디자인", CMP Books, 2003