한국컴퓨터정보학회 동계학술대회 논문집 제20권 제1호 (2012. 1)

실시간 운영체제 CNU_RTOS에서 버디 시스템 메모리 기법 설계 및 구현

정성훈°, 권재국**, 이철훈*
°*충남대학교 컴퓨터공학과
e-mail:{shjeong, jagugy, clee}@cnu.ac.kr

Design and Implementation of Buddy System for Real-Time Operating system, CNU_RTOS

Seong-Hoon Jeong°, Jae-Guk Kwon**, Cheol-Hoon Lee*
°*Dept. of Computer Science, Chung-Nam National University

● 요 약 ●

디지털 컨버전스 시대가 도래하면서 자원이 제한된 소형기기들의 사용이 비약적으로 증가하는 추세이다. 실시간 운영체제가 탑재되는 임베디드 시스템은 특성상 제한된 메모리를 가지기 때문에 제한된 메모리를 효율적으로 사용할 수 있는 기법이 적용되어야 한다. 이러한 제한된 메모리 관리를 효율적으로 사용하도록 기존의 CNU_RTOS에서는 실시간 운영체제의 메모리 관리 기법인 메모리 풀(Pool)과 힙(Heap) 알고리즘을 사용하였다. 시간 결정성을 보장하기 위해 힙을 할당하는 방법으로 first fit 알고리즘을 사용하였지만 외부 단편화로 인한 메모리 낭비를 감수하게 되었다. 본 논문에서는 힙 스토리지 매니저에서 발생하는 외부 단편화를 최소화하기 위해 버디 시스템을 이용한 메모리 관리 기법을 설계 및 구현하였다.

키워드: 실시간 운영체제(Real-Time Operating System), 단편화(Fragmentation)

I. 서론

최근 유비쿼터스 컴퓨팅, 웨어러블 컴퓨터에 대한 관심이 늘어나면서 임베디드 시스템 분야가 급성장하고 있다. 이러한 임베디드 시스템의 자원을 효율적으로 관리해주기 위한 실시간 운영체제(Real-Time Operating System)에 대한 많은 연구들이 진행되고 있다.

실시간 운영체제는 시간 결정성을 보장해야 하는 특성을 지니고 있으며, 데드라인(Deadline)내에 수행한 결과만 신뢰하는 경성 실시간 운영체제(Hard Real-Time Operating System)와 주어진 데드라인 이후의 수행 결과도 어느 정도 신뢰하는 연성 실시간 운영체제(Soft Real-Time Operating System)로 나뉘어 진다.

임베디드 시스템의 한정된 자원으로 최대의 성능을 수행해야 하며, 이를 위해 효율적인 메모리 관리를 함으로써 시스템의 성능을 향상시킬 수 있다. 하지만 실시간 운영체제에서는 시간 결정성의 중요성 때문에 약간의 메모리 낭비를 감수하면서 시간 제약을 맞추려고 한다. 이에 따라 사용하지 않는 메모리가 충분히 있음에도 불구하고 메모리가 나뉘어져 있어 메모리를 할당할 수 없는 단편화(Fragmentation) 문제가 생겨 효율적으로 메모리를 사용할 수 없는 문제가 발생한다.

본 논문에서는 힙 스토리지 매니저에서 발생하는 외부 단편화를 최소화하기 위해 버디 시스템을 구현하였다.

본 논문의 구성은, 2 장에서는 관련 연구로서 실시간 운영체제인 CNU_RTOS에서 사용하고 있는 메모리 관리 기법인 힙 스토리지 매니저와 메모리 단편화 현상 대해 소개하고, 3 장에서는 본 논문에서 구현한 버디 시스템 기술하며, 4 장에서는 실험 환경 및 결과를 기술한다. 마지막으로, 5장에서는 결론 및 향후 연구과제에 대해서 기술한다.

II. 관련 연구

1. 관련연구

1.1 힙 스토리지 매니저(Heap Storage Manager)

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

힙 영역에서 동적 메모리 할당은 MK_GetMemory()를 통해 이루어 진다. 먼저, 퍼스트 핏(First-Fit) 알고리즘에 따라 프리 블록 리스트로부터 적당한 메모리를 선택하여 힙을 가리키는 포인터(md_pHeap)와 할당된 메모리 크기(md_size)로 구성된 strcut mk_heap_dummy 구조체인 헤더를 포함하여 할당한다. 만약, 할당 가능한 메모리가 존재하지 않으면 메모리를 요청한 태스크는 적당한 메모리가 생길 때까지 Pending된다.

메모리 해제는 MK_FreeMemory()에 의해 이루어진다. 먼저, 프리 블록 리스트에 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_struct가 포함된다.

그러나 힙 스토리지 매니저는 프리 메모리 블록 검색 시간으로 인해 시간 결정성을 저해하는 부분이 발생한다.

힙 스토리지 매니저 구조는 그림 1과 같다.

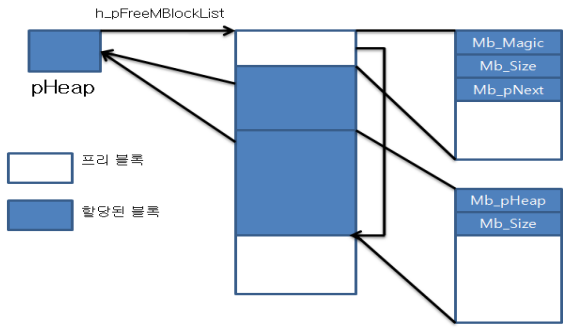

그림 1. 힙 스토리지 매니저

1.2 메모리 단편화(Memory Fragmentation)

단편화란 기본적으로 자원이 하나의 연속된 단위로 존재하는 것이 아니라 부분으로 나뉘어 파편처럼 흩어져 있는 상태를 말한다. 자원 할당 과정에서 단편화란, 자원을 배당하고 회수하는 과정에서 아무도 사용하지 않고 있는 자원이 흩어져 있어, 사용이나 배정이 불가능한 상태로 되어 있는 것을 말한다. 자원 할당 과정에서의 단편화는 크게 내부 단편화와 외부 단편화로 나눌 수 있다.

외부 단편화는 할당 가능한 자원의 합에 비해 실제 할당 가능한 단위 자원의 크기가 매우 작은 상태를 의미한다. 외부 단편화가 일어난 경우 전체 자원의 상태는 사용 중인 영역 사이에 작은 크기의 빈 공간이 흩어져 있는 것처럼 보이게 된다. 이러한 외부 단편화는 자원을 연속적으로 할당 받아야 한다는 조건과 이미 할당된 자원은 옮길 수 없다는 조건이 맞물리면서 발생하는 현상이다.

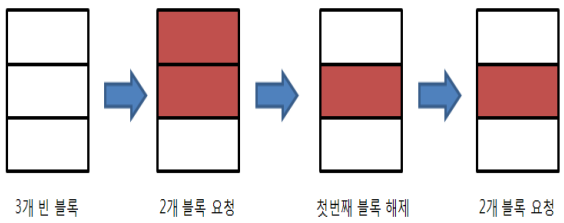

그림 2. 외부 단편화가 발생하는 과정

그림 2는 외부 단편화가 발생하는 과정이다. 연속한 3블록의 메모리가 준비되어 있을 때 1블록 크기의 자원 2개의 요청이 들어온 후 첫 번째 자원이 해체하였다. 그 다음 2블록 크기의 자원 할당을 요청하지만, 2블록의 여유 자원이 있지만 배치가 연속적이지 않기 때문에 할당할 수 없다.

내부 단편화는 할당을 요청한 크기에 비해서 실제 할당된 단위 자원의 크기가 커서, 해당 단위 자원 내에서 사용하지 않는 부분이 있을 때 발생한다. 자원 할당 과정의 단편화에 있어서 주목할 점은 이것이 여유 자원의 현재 상태뿐 아니라 자원 요청이 들어오는 패턴에 의해서 문제가 될 수도 있고 그렇지 않을 수도 있다는 점이다.

III. 본론

1. 본론

1.1 버디 시스템

버디 시스템은 고정 분할 기법과 동적 분할 기법의 단점을 보완하기 위해 제안된 기법이다. 이진 버디 시스템의 성능을 개선하기 위해 비트마스크를 이용하여 메모리를 모니터링 함으로써 좀 더 빠르고 정확하게 메모리를 할당할 수 있도록 하였다.

버디 시스템의 메커니즘을 구현하기 위해서는 할당 가능한 메모리의 최소 크기와 최대 크기를 정의해야 하기 때문에 그림 3과 같이 메모리의 최소 크기와 최대 크기를 미리 선언하였다.

```
#define MEMORY_FREE    0x01
// 메모리 해제 플래그
#define MEMORY_ALLOC   0x00
// 메모리 할당 플래그
#define MEMORY_ALLOC_MAX_SIZE ( 1 * 1024 * 1024 )
#define MEMORY_ALLOC_MIN_SIZE   64
#define MEMORY_MAX_BITMASKLISTCOUNT 15
// 비트마스크 리스트
#define MEMORY_MAX_BITMASKCOUNT ( 2 * 1024 )
```

그림 3. 버디 시스템 관련 변수 정의

메모리를 효율적으로 관리하기 위해서 비트 마스크(Bit Mask)를 이용 한다. 메모리 최소 크기의 개수를 파악하기 위해 MEMORY_MAX_BITMASKCOUNT를 사용 하였다.

```
MEMORY_MAX_BITMASKCOUNT = MEMORY_ALLOC_MAX_SIZE /
MEMORY_ALLOC_ MIN SIZE / 8
```

그림 4. 버디 시스템 관련 변수 정의

그림 4는 MEMORY_MAX_BITMASKCOUNT를 구하기 위한 공식을 나타낸다.

```
typedef struct mk_buddy_struct {
MK_U32_t    bd_Magic;
MK_S8_t     *bd_pName;
MK_U32_t    bd_StartOfBuddy;
MK_U32_t    bd_RemainSize;
MK_DWORD_t bd_SizeArray[MEMORY_ALLOC_MAX_SIZE /
                    MEMORY_ALLOC_MIN_SIZE];
MK_BYTE_t   bd_BitMask[MEMORY_MAX_BITMASKLISTCOUNT]
                    [MEMORY_MAX_BITMASKCOUNT];
struct mk_buddy_struct *bd_pNext;
struct mk_buddy_struct *bd_pPrev;
} MK_BUDDY;
```

그림 5. 버디 구조체

그림 5는 메모리 관리를 위한 구조체이다. bd_Magic은 식별자이고 bd_pName은 버디 이름을 나타낸다. bd_StartOfBuddy는 초기 메모리를 생성할 때 시작 주소를 저장하기 위해 쓰인다. bd_RemainSize는 남은 메모리 용량을 알기 위해 사용한 변수이다. bd_SizeArray는 할당한 메모리 사이즈를 저장하기 위한 배열이다. bd_BitMask는 메모리를 할당하고 해제를 할 때 세팅 해주기 위해 사용된다.

표 1. 버디 시스템 관련 함수

함수명	내용
MK_CreateBuddy()	버디 시스템 생성
MK_GetBuddy()	메모리 할당
MK_FreeBuddy()	메모리 반납
MK_AllocBuddyIn()	해당 버디 블록 크기의 메모리 블록 하나를 얻음
MK_FreeBuddyIn()	해당 인덱스의 마스크 오프셋 위치의 블록을 해제
MK_GetAlignSize()	요청한 메모리에 맞는 버디 블록 크기를 얻음
MK_FindFreeOffsetInMask()	해당 인덱스의 비트마스크를 검색하여 프리한 비트마스크를 찾음
MK_GetFlagInMask()	해당 마스크의 플래그 값을 얻음
MK_SetFlagInMask()	해당 마스크의 플래그 값을 설정
MK_GetMaskArray()	마스크의 배열 포인터를 얻음
MK_IsValidListIndex()	리스트 인덱스가 유효한지 확인
MK_GetListIndexOfMatchSize()	해당 버디 블록 크기를 이용해 리스트 인덱스를 얻음

버디 시스템은 버디 시스템을 생성하는 함수, 버디 시스템으로부터 메모리를 할당 및 해제하는 요청하는 함수, 요청한 메모리에 맞는 버디 블록 크기를 얻어오는 함수, 비트마스크를 설정하고 얻어오는 함수 등으로 구성되어 있으며, 이러한 함수를 통해서 버디 시스템으로부터 메모리를 할당 받고 해제 할 수 있다. 표 1은 버디 시스템에서 사용하는 함수들을 나타낸다.

1.2 실험 환경 및 결과

본 논문에서 구현한 메모리 관리 기법은 실시간 운영체제인 CNU_RTOS에 구현하였고 컴파일러는 ARM ADS 1.2를 사용하였으며, ARM920T 기반의 MBA2440 보드에서 테스트 하였다.

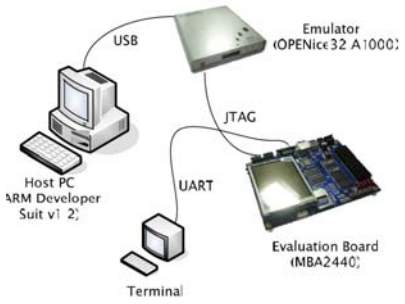

그림 6. 테스트 환경

그림 6은 힙 스토리지 매니저를 통하여 메모리를 할당하고 해제한 결과이다.

힙 스토리지 매니저에서 8Kbyte의 메모리를 생성 한 후 Task1과 Task2 각각 3Kbyte의 메모리를 할당 받은 후 Task1의 3Kbyte메모리를 할당 후 Task2가 다시 4Kbyte를 요청하였을 때 할당을 하지 못한다.

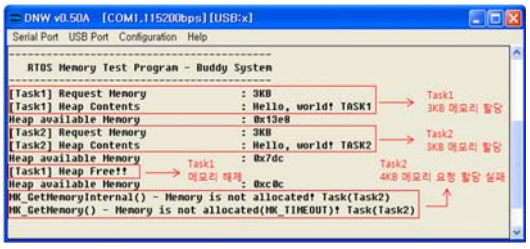

그림 7. 힙 스토리지 매니저 메모리 할당 및 해제

그림 8은 그림 7과 동일한 조건으로 버디 시스템을 이용하여 메모리 할당 및 해제 하였다.

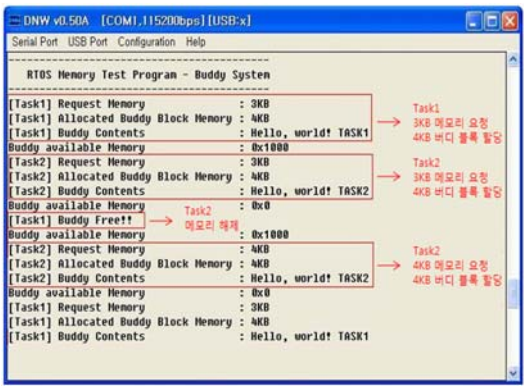

그림 8. 버디 시스템 메모리 할당 및 해제

힙 스토리지 매니저와는 다르게 메모리 할당이 가능하다는 것을 보여준다.

Ⅳ. 결론

본 논문에서는 실시간 운영체제인 CNU_RTOS에서의 동적 메모리 할당 기법인 힙 스토리지 매니저의 외부 단편화 문제를 보완하고, 정적 메모리 할당 기법인 메모리 풀의 정적 메모리 할당 문제를 보완하기 위해 메모리를 더 효율적으로 사용할 수 있도록 버디 시스템을 추가 구현하였다.

실험을 통해 설계 및 구현한 버디 시스템이 기존의 메모리 할당 방식인 First Fit에 비해 단편화 집중 현상을 방지하는 것을 확인하였다. 향후 연구 과제로는 본 논문에서 구현한 버디 시스템의 비트마스크가 배열이기 때문에 사용하지 않는 배열 공열 공간이 발생하는데, 이 문제를 해결하기 위해 배열을 사용하지 않고 비트 마스크 할 수 있는 방법에 대한 연구가 필요하다.

참고문헌

[1] Aijisystems, “Embedded hardware & software solution, UbiFOS(Ubiquitous Flexible Real-Time Operating Systems) & MCU platforms,” http://www.aijisystem.com

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

[3] Memory Fragmantation, http://support.novell.com/techcenter/articles/ana19950407.html

[4] Paul R. Wilson, Mark S. Johnstone, Michael Neely, and David Boles “Dynamic Storage Allocation : A Survey and Critical Review”

[5] Buddy System, http://beforu.egloos.com/1165105