2003 정보 및 제어 학술회의 (2003.11.21-22)

# Single Address Space(SAS) Architecture를 이용한 Embedded Operating System

## Embedded Operating System using the Single Address Space(SAS) Architecture

안 광 혁*

* 엠노스 (전화:(043) 872-2418, 팩스:(043) 872-1418, 핸드폰:(019) 466-2418, E-mail : ponytail@emnos.com)

**Abstract** : A large part of the embedded system, compared with the PC, have low performance CPU and small memory. So the embedded operating system fits the condition of that hardware system.

A Single Address Space (SAS) OS has the operating system and all applications in the single address space. The SAS architecture enhances sharing and co-operation, because addresses have a unique interpretation. Thus, pointer-based date structures can be directly communicated and shared between programs at any time, and can be stored directly on storage. The key point of the SAS OS on the embedded system is the low overhead inter-action between programs in process and usage. So SAS OS can be ported on the low performance CPU.

In this paper, we design the SAS OS (named emNOS, Embedded Network Operating System) on the ARM7TDMI processor. Finally we show the benefits of the SAS OS on the embedded system.

**Keywords** : Embedded System, Single Address Space(SAS), Operating System(OS), ARM7TDMI

### Ⅰ. 서론

최근 Embedded system 분야에서는 32 Bit급 고성능 CPU를 채택한 제품이 많이 개발되고 있다. 그럼에도 불구하고 아직도 많은 수의 Embedded system은 8 Bit급이나 16 Bit급 CPU를 중앙처리 장치로 채택하고 있다. 또한 현재 개발되어지고 있는 32 Bit급 Embedded system의 경우도 일반적으로 사용되고 있는 PC와 비교하면 처리 속도나 Memory의 Size 등의 사용 환경이 매우 열악하다. 즉, 대부분의 Embedded system은 저성능 CPU와 적은 용량의 Memory 환경에서의 개발이 요구되어 진다는 것을 쉽게 알 수 있다. [1]

이처럼 상대적으로 열악한 환경에도 불구하고 최근의 Embedded system에 요구되어지는 기능이 바로 Networking 기능이다. Networking 환경의 가장 큰 특징은 많은 용량의 Memory를 요구한다는 점이다. 이러한 요구는 Network를 통하여 System에 송/수신되는 Data를 저장하기 위해 사용하는 Memory 공간으로 송/수신을 위한 충분한 공간이 필요하다. Networking을 위한 많은 량의 Memory의 요구는 Embedded system이 갖춘 적은 용량의 Memory와는 상충되는 점이다. 적은 용량의 Memory를 갖는 Embedded system에서 Networking과 같이 많은 Memory를 요구하는 문제를 해결하기 위한 방법 중 가장 좋은 방법은 적은 용량의 Memory라도 빠르고 효율적으로 운용하는 것이다.

Embedded system이 가지고 있는 적은 용량의 Memory를 빠르고 효율적으로 관리하기 위하여 본 논문에서는 Single Address Space(SAS) architecture 방식의 Embedded Operating System(OS)을 제안하고자 한다.

### Ⅱ. Single Address Space(SAS) Architecture

SAS architecture는 원래 CAD/CAM과 같이 복잡한 응용 프로그램이 수행 되어지는 고성능 Distributed System를 위한 OS로 제안 되어졌다. CAD/CAM에서 사용되어지는 복잡한 응용 프로그램의 특징은 거대한 용량의 Database에 각각의 응용 프로그램들이 빠르고 효율적으로 접근하도록 하는데 있다. SAS architecture의 가장 큰 특징은 프로그램 코드와 데이터를 하나의 거대한 Shared memory내에 저장하며 이를 단일화 된 주소로 접근함으로써 Sharing과 Cooperation을 향상시킨다는 점이다. 그림-1에서처럼 Huge shared memory 구조를 이용함으로써 프로그램은 System의 전체 Memory를 마치 자신의 Local data에 접근하듯이 접근할 수 있게 된다.

이처럼 System이 가지고 있는 memory의 전 영역을 빠르고 쉽게 Access할 수 있다는 특징은 Networking을 위해 요구되어지는 Memory management 특성과 같다.

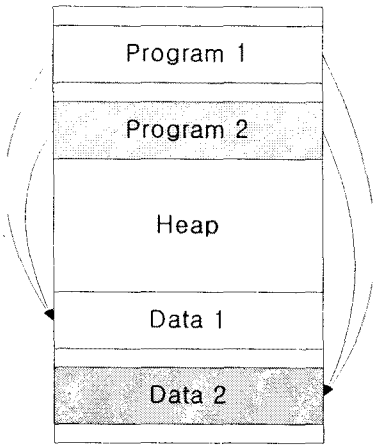


그림 1. Shared Memory의 구조
Fig 1. Architecture of shared memory

SAS architecture는 어떠한 프로그램이던지 Memory의 전 영역을 Access할 수 있기 때문에, Memory protection이 가장 중요한 문제로 주어진다. SAS OS에서 Memory protection의 문제는 하나의 프로그램이 자신의 Local memory 영역을 넘어서 다른 프로그램의 Local memory를 Access하는 경우에 발생한다. [2][3]

본 논문에서는 Memory protection을 구현하기 위하여 Boundary check algorithm을 사용하였다. Boundary check algorithm은 Memory와 관련된 작업을 수행할 경우, 할당받은 Memory의 Boundary를 check함으로써 할당받은 Memory 영역을 넘지 않았는지를 확인하는 Algorithm이다.

## III. 연구 내용

본 논문에서는 Embedded system에서 효율적으로 Networking을 지원하기 위한 SAS architecture 방식의 OS를 구현하였으며 embedded Network Operating System(emNOS)라 명명하였다. SAS OS의 특성을 살려 각각의 프로그램들은 Thread의 개념으로 동작하며, 향후 연구되어질 Network에 대한 지원을 목적으로 설계되어졌다.

### 1. Context Switching

대부분의 Non-SAS OS에서는 그림-2에서와 같이 Context Switching을 하기 위한 Event로 Interrupt를 이용한다. 이러한 예는 Embedded System에서 많이 사용하는 Real Time Operating System(RTOS)에서 두드러지게 나타나는 현상이다. 왜냐하면 RTOS는 Rela time requirement를 충족시켜주어야 하기 때문에 더 높은 우선순위를 갖는 Process가 더 빨리 수행되어야 한다. 우선순위가 높은 Process가 Sleep state에서 Ready state로 전환되는 Event는 주로 Interrupt에 의해서 발생함으로 Process간의 Context switching도 Interrupt에 의하여 발생하게끔 설계되어진다. [4]

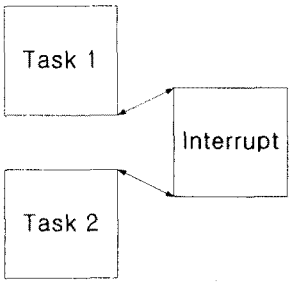


그림 2. RTOS에서의 Context Switching
Fig 2. Context Switching in the RTOS

emNOS의 경우, Context switching은 그림-3에서와 같이 현재 수행되고 있는 Thread에 의하여 발생하게 된다. 즉, Context switching과 Interrupt와는 무관하다는 특성을 갖는다. 이러한 구조의 특징은 Context switching의 Overhead가 적지만 Real time requirement를 충족시키기 어렵다는 점이다.

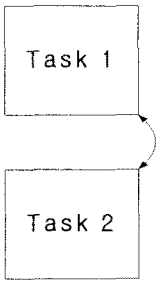


그림 3. emNOS에서의 Context Switching
Fig 3. Context Switching in the emNOS

Network 프로그램의 경우, 전송에 따른 Network의 Delay를 예측하기가 어렵기 때문에 Network상에서 Real time requirement를 충족하기란 매우 어렵다. 즉, Networking을 목적으로 하는 System에서 Real time requirement를 충족시키기 보다는 Context switching의 Overhead를 줄임으로써 System의 Performance를 높이는 방법을 선택하였다.

emNOS에서는 Cooperative multi-threading 방식의 Scheduling algorithm을 채택함으로써 Interrupt가 발생하더라도 Context switching이 발생하지 않고, 현재

실행중인 Thread가 작업을 마치면 Context switching이 발생하는 구조로 구현되어졌다. 이렇게 함으로써 Context Switching에 따른 Overhead를 최소화하였다.

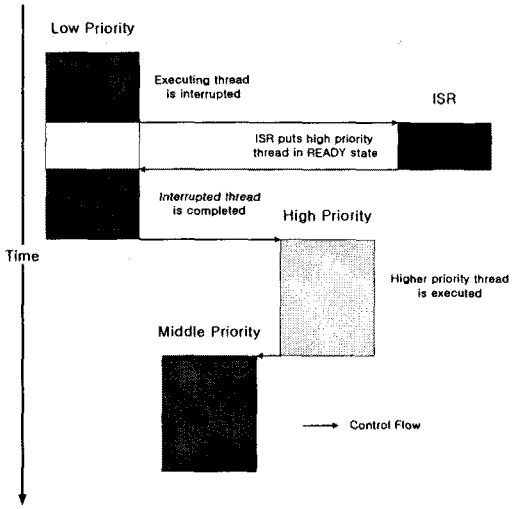


그림 4. Cooperative multi-threading
Fig 4. Cooperative multi-threading

2. Memory management

SAS OS는 다른 구조의 OS에서 보다 Memory protection이 중요한 문제로 대두된다. 왜냐하면 어떠한 Process라도 System memory의 전 영역을 다 Access할 수 있기 때문이다. 즉, A Thread의 Local memory를 B Thread가 침범할 수 있기 때문에 각 Thread의 Local memory를 보호하는 것이 중요한 문제로 대두되는 것이다.

Embedded system은 General purpose system과는 달리 정해진 작업만을 수행하도록 개발되어진다는 점이다. 즉, System 개발자는 자신의 Embedded system이 수행하여야할 작업을 정확히 알고 있으며 주어진 작업만을 수행하도록 System을 개발한다. 이러한 Embedded system의 특성은 SAS OS의 주요 문제점인 Memory protection에 손쉬운 해답을 제공해 준다.

Embedded system의 개발과정에서 Memory protection이 이루어진다면, 새로이 추가되는 프로그램이 없기 때문에 Embedded system을 사용할 때에도 Memory protection이 이루어진다는 점이다.

emNOS에서는 System의 사용가능한 Memory를 Heap pool로 정의하고, alloc함수와 free함수를 통하여 Thread에서 사용할 수 있도록 설계하였다. alloc함수는 Heap pool에서 프로그램이 요구한 만큼의 Memory를 얻어오며, 그 Heap memory의 끝부분에 End of Heap을 Making한다. free함수는 사용한 Memory를 System에 돌려주기 전에 End of Heap을 Checking함으로써 Thread에서 사용되어진 Heap이 보호되었는지를 Check한다.

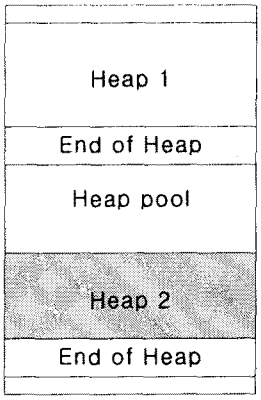


그림 5. Heap pool
Fig 5. Heap pool

3. Inter Process Communication(IPC)

Nos-SAS OS의 경우, Process간의 통신은 Semaphore와 Mailbox 등을 통하여 이루어진다. System의 Semaphore와 Mailbox는 System 초기에 생성되어 Process의 요구에 의해 사용 되어진다. 실제 이러한 IPC를 사용함에 있어서 Process는 여러 개의 IPC 가운데 자신이 사용하는 IPC를 찾아야 하는 Overhead를 갖게 된다. 한 예로 uC/OS의 경우, OS 내부에 OS_EVENT라는 자료구조를 사용한다. [4]

SAS OS는 그 개발 목적이 Memory의 효율적인 Access에 있다. 그러므로 IPC 검색에 따른 Overhead가 거의 없도록 설계되어질 수 있다. emNOS의 경우, IPC는 검색을 통하여 이루어지지 않고, Memory pointer를 통하여 이루어진다. 그러므로 Thread는 자신이 원하는 Memory에 직접 Access가 가능하다.

이러한 특징은 Thread간에 주고 받아야할 Data가 많은 Networking 환경에서 더욱 유리하다. 해당 Layer의 Thread는 자신의 Layer에 속하는 Data를 직접 Access하여 처리하며, 상위 Layer나 하위 Layer에는 단순히 Memory를 Pointer로 전달함으로써 빠르게 Data를 처리할 수 있다.

4. Target System

| 항 목 | 내 용 |
|---|---|
| Processor | Samsung S3C44B0X(ARM7TDMI) |
| ROM | 2Mbytes (16Bit Data Bus) |
| RAM | 8Mbytes (16Bit Data Bus) |

표 1. Target System의 Hardware Spec.
Table 1. Hardware Spec. of the target system

emNOS의 개발은 표-1과 같은 성능을 갖춘

ARM7TDMI board상에서 이루어 졌다. CPU는 Samsung S3C44B0X ARM7TDMI Processor를 사용하였으며, 동작 주파수는 66MHz이다. System의 Memory는 ROM 2Mbytes, RAM 8Mbytes를 갖추었는데, emNOS에서 실제 사용하는 메모리의 양은 아주 적다. 그림-6은 emNOS의 개발에 사용되어진 ARM7TDMI Board이다.

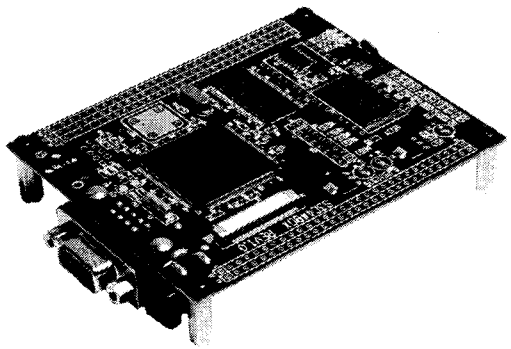

그림 6. S3C44B0X(ARM7TDMI) Board
Fig 6. S3C44B0X(ARM7TDMI) Board

실제 ARM7TDMI상에서 구현되어진 emNOS는 최소 ROM 7Kbytes, RAM 12Kbytes만을 사용하며 동작되었고, ROM 25Kbytes, RAM 15Kbytes를 사용하면 OS의 모든 기능이 사용가능하다.

## Ⅳ. 결론

본 논문에서는 SAS architecture를 이용한 Embedded Operating System을 ARM7TDMI processor 상에 직접 구현하였다. SAS architecture의 핵심이라고 할 수 있는 Memory의 보호를 위하여 Heap을 이용한 Memory management 방법을 사용함으로써 각각의 Memory를 보호하는 방법을 사용하였다.

본 논문에서 제안한 SAS architecture의 OS는 작은 크기의 Memory를 가진 임베디드 시스템에서 빠르고 효율적으로 Memory를 관리함으로써 Networking과 같이 복잡한 Application을 임베디드 시스템에 도입하기 쉽도록 설계하였다.

향후 본 논문에서 제안한 emNOS의 Networking을 위한 TCP/IP suite를 개발하려 한다. TCP/IP suite는 SAS OS의 빠르고 효율적인 Memory management 특성을 충분히 활용할 수 있도록 구현할 계획이다.

## 참고문헌


[1] Karim Yaghmour, "Building Embedded Linux System", O'reilly & Associates, Inc, pp. 5-11, Apr. 2003.

[2] Jeffrey S. Chase, Henny M. Levy, Michael J. Feeley, and Edward D. Lazowska, "Sharing and Protection in s Single-Address -Space Operating System", ACM Transaction on Computer Systems, Nov, 1994.

[3] http://www.cse.unsw.edu.au/~disy/Mungi/

[4] Jean J. Labrosse, "MicroC/OS-II", R&D Books, 1999.