Journal of the Industrial Technology Research Institute.
Korea Maritime University. Vol. 12. 1994.

실시간 멀티태스크 처리 기법을 이용한 SSU시스템 개발에 관한 연구

정경열* · 황동희** · 류길수***

A Study on the Development of SSU Using Multi-tasking Method in Real Time

K.Y.Chung · D.H.Hwang · K.S.Rhyu

Abstract

In general it is difficult to process object controls in real time when they are large. Therefore multi-tasking environment should be implemented in the controlling and monitoring systems in order to process in real time. This paper illusrates the use of time-slice and event-driven multi tasking methods in Safety System Unit(SSU) which is constructed with several tasks. The developed system has two merits. First, since the switching time of the task is short, the system takes small execution time. Second, the tasks which is classified into specific domain characteristics can be modified easily.

1. 서 론

종래에 사용되어 왔던 마이크로프로세서를 이용한 실시간 제어 및 감시시스템들은 대상 시스템의 특성에 맞는 기능들을 순차적으로 실행해 나가는 방식이 대부분이었다. 그러나 선박제어와 같이 대상시스템이 대형

* 한국기계연구원
** 한국해양대학교 대학원
*** 한국해양대학교 부교수

화되고 작업영역이 방대해지면 순차적인 실행방식만으로는 실시간 제어 프로그램의 작성이 대단히 복잡하게 된다. 따라서 이러한 대상영역에서는 작업내용들을 특성에 따라 여러개의 태스크로 분할한 후 인터럽트를 이용하여 필요시에만 실행해 주는 기능을 미리 준비해 두는 것이 바람직하다. 이러한 기능을 실시간 멀티태스크 기능이라고 하며,[1] 이러한 기능이 갖추어진 환경에서는 대상영역의 응용태스크를 특성에 따라 분류하여 작성해 놓는 것만으로 시스템의 구축이 완료된다.

UNIX와 같은 운영체제에서는 타임슬라이스 기법을 이용한 멀티태스크 환경이 구축되어 사용되고 있으며, PC에서도 MS-WINDOWS와 같은 멀티태스크 기능을 가진 운영체제의 개발이 활발하게 이루어지고 있다.[2] 그러나 INTEL 8086/88과 같은 저급 MPU를 이용한 시스템[3]의 경우는 프로그래머를 위한 실시간 멀티태스크 시스템만이 고가로 제공되고 있다. 이러한 시스템들은 대부분 일반화된 기능을 가지고 있어서 메모리를 필요이상으로 많이 차지하는 단점을 가지고 있기 때문에 응용력이 부족하고 시스템이 커질수록 메모리를 많이 필요로 하므로 저급 MPU를 이용한 시스템의 개발에는 사실상 적용이 어려워 지게 된다. 따라서 경제성을 고려하여 저급 MPU를 이용할 경우에는 필요불가결한 최소한의 멀티태스크 기능만을 부여하여 메모리를 적게 차지하도록 하고, 태스크의 절환시간을 줄일 수 있는 타임슬라이스 방식과 이벤트드리븐 방식을[4] 혼합한 멀티태스크 환경을 구축하는 것이 바람직하다.

선박에서 RPM, Overspeed, Shut Down 및 Slow Down Alarm등 엔진을 감시, 보호하기 위한 Safety System의 경우도 아직 INTEL 8088과 같은 저급 MPU를 이용한 시스템으로 구성되어 있어 그 기능의 보다 효율적인 수행을 위해서는 멀티태스크 기능을 가진 효율적인 시스템의 개발이 요구된다.

이상과 같은 배경으로 부터 본 논문에서는 멀티태스크 기능을 가진 SSU(Safety System Unit)시스템의 구현에 관하여 논하기로 한다. 즉 SSU의 실시간제어를 타임슬라이스 방식을 이용해 원할히 수행하면서 파라미터의 변환, 원격제어시스템과의 통신 및 특정 상황에 따른 동작 등은 이벤트드리븐 방식을 이용하여 처리되게 함으로써 프로그램작성을 용이하게 함과 동시에 태스크의 절환시간을 최소화하도록 한다. 실제 시스템의 하드웨어는 NORCON사의 Autochief 4에서 SSU[5]를 구성하는 INTEL 8088 메인보드를 대상으로 하고 구동 소프트웨어는 Turbo-C와 어셈블리언어를 이용하여 멀티태스크 기능을 가진 시스템으로 구현한다.

2. 실시간 시스템에서 요구되는 멀티태스킹

2.1 멀티태스킹을 위한 기본요소

멀티태스킹 기능을 실시간에서 처리가 가능하도록 하기 위해서는 멀티태스킹의 고급스런 기능 보다는 꼭 필요한 기능만을 가지도록 하고 무엇보다 태스크의 절환시간과 메모리의 크기를 최소화하는 것이 중요하다. 본 논문에서는 이러한 조건을 만족하는 실시간 멀티태스크 처리용 시스템 환경을 구축하기로 한다.

이러한 환경을 구축하기 위해서는 태스크 및 각 태스크의 정보를 저장할 수 있는 태스크테이블, 자원의 관리를 위한 TCB(Task Control Block)와 RCB(Resource Control Block), 그리고 메시지의 송수신시에 중간 매체가 되는 Mail box가 기본요소로 필요하다. 이들 각각의 요소들은 메모리에 상주하는 것들로서 다음과 같이 그 크기 및 구성내용을 결정할 수 있다.

1) 태스크테이블

태스크테이블은 태스크의 등록, 제거 등을 행할 때 그 정보를 저장하는 곳이다. 태스크 테이블에 저장되는 내용은 Fig.2.1과 같이 태스크의 우선순위를 결정하는 태스크 번호, 8087의 사용여부를 나타내는 플래그, 스택 세그먼트, 스택 포인터, 코드 세그먼트, 프로그램 카운터, 데이터 세그먼트, 엑스트라 세그먼트 등이 된다.

8 bit	task number
8 bit	8087 flag
16 bit	stack segment
16 bit	stack pointer
16 bit	code segment
16 bit	program counter
16 bit	data segment
16 bit	extra segment

Fig.2.1 **Format of Task Table**

여기서 8087 플래그는 태스크가 8087 보조 프로세서를 사용하는가 사용하지 않는가를 나타내는데 만일 시스템 자체가 8087 보조 프로세서를 장착하고 있지 않을 경우에는 모든 태스크 테이블의 8087 플래그를 '0'으로 설정해 두어야만 한다.

2) TCB (Task Control Block)

실시간 멀티태스크 시스템에서는 각 태스크의 상태를 TCB라고 하는 태스크 제어 블럭을 이용하여 관리하도록 하는데 그 상태는 다음 7가지로 분류할 수 있다.

① 실행상태
② 실행가능상태
③ 메시지 수신대기상태
④ 무조건대기상태

⑤ 자원해방대기상태

⑥ 시한부대기상태

⑦ 미등록상태

TCB는 한개의 태스크에 대해 반드시 하나씩 할당하여 사용하도록 하고 Fig.2.2 (a) 및 (b)와 같이 태스크의 상태, 8087의 사용유무, 스택 포인터, 스택 세그먼트, 카운터를 저장하며 특히 태스크의 상태는 (b)와 같이 상기 7개의 상태에 대한 정보를 저장하도록 한다. 태스크의 상태에서 등록상태는 태스크가 등록이 되었으면 '1'로 미등록상태이면 '0'으로 저장한다. 메시지 수신상태는 수신대기 중인지 아닌지, 자원점유 해제상태는 자원점유의 해제 대기 중인지 아닌지에 따라 '1'과 '0'을 각각 저장하도록 하고, 무조건 대기와 시한부 대기상태도 현재 태스크가 대기 중인지 아닌지에 따라 '1'과 '0'을 각각 저장한다. 카운터는 태스크가 시한부 대기상태일 때 시한부 대기해제까지의 시간이 저장되어 있어 이 카운터 값이 '0'이 되면 시한부 대기가 해제된다.

8 bit	task status
8 bit	8087 flag
16 bit	stack segment
16 bit	stack pointer
16 bit	counter

Fig.2.2 (a) **Format of TCB**

7	6	5	4	3	2	1	0

0: entry state
1: state of receiving message
2: state of unconditional wait
3: state of wait for resource free
4: state of wait with tick time
5, 6, 7: no definition

Fig.2.2 (b) **Format of Task Status**

3) RCB(Resource Control Block)

멀티태스킹에 있어서 하나의 자원을 2개 이상의 태스크가 공유하는 경우에 배타적인 자원의 관리가 필요할 경우가 있다. 즉, 프린터를 이용하여 2개의 태스크가 출력을 행하고자 할 때 배타적 자원관리를 행하지 않는다면 한 태스크의 출력 도중 다른 태스크로의 절환이 발생하여 두개의 출력 데이터가 혼합되어 버리게 된다. 따라서 하나의 태스크에서 출력을 하고자 할 때에는 프린터라는 자원을 배타적으로 점유하여 사용함으로써 작업을 마치기 전에는 다른 태스크로 절환되지 않도록 하여 다른 태스크의 프린터 사용이 금지되도록 할 필요가 있다.

본 논문에서는 이러한 경우 배타적인 자원관리의 수행을 위해 자원의 배타적 점유 및 해방을 행할 수 있는 기능을 제공하기 위해 있어서 RCB라고 하는 자원제어블럭을 Fig.2.3과 같이 정의하여 두고 이것을 이용하여 자원의 배타적 점유 및 해방이 가능하도록 하였다. RCB는 하나의 자원에 대해 반드시 하나를 작성하여야 하고 그 자원을 사용하는 태스크는 반드시 해당 자원에 대한 RCB를 지정하여야 한다.

8 bit	pointer
8 bit	number of task occupying RCB
8 bit	wait queue
8 bit	wait queue
8 bit	wait queue
8 bit	wait queue

Fig.2.3 **Format of RCB**

포인터는 현재 점유중인 태스크의 번호가 저장되어 있는 어드레스의 오프셋 값이 저장되어 있다. 점유중인 태스크의 번호 및 대기행렬에서 포인터로 표시되는 위치가 현재 점유중인 태스크 번호이며 그 다음부터 링구조로 4개의 대기행렬이 구성된다.

4) 메시지의 송수신

실시간 멀티태스크 시스템에서 사용하는 메시지의 송수신 방법에는 다음과 같이 5가지가 있다.

① 메시지의 송신에 있어서 상대측이 수신하지 않을 경우 대기행렬에 저장하고 스케쥴링을 수행한다.

② 메시지의 송신에 있어서 상대측이 수신하지 않을 경우 대기행렬에 저장하고 스케쥴링을 수행하지 않는다.

③ 메시지의 송신에 있어서 상대측이 수신하지 않을 경우 대기행렬의 메시지가 수신될 때까지 대기상태가 된다.

④ 인터럽트 처리내에서 메시지를 송신한다.

⑤ 송신된 메시지를 수신하며 상대측이 송신하지 않을 때는 송신될 때까지 대기상태가 된다.

이 5가지 중 어느 경우라도 메시지의 송수신은 반드시 Fig.2.4와 같은 형태의 Mail Box라는 공유영역을 통해서 이루어 지도록 하였다. 이 때 하나의 메시지를 2개 이상의 태스크에 송신할 때는 메시지를 수신할 태스크의 수 만큼 Mail Box를 작성하여 각각의 Mail Box에 동일 데이터를 송신하는 방법을 이용해야 한다. 만일 Mail Box의 대기행렬이 가득 찼을 경우에 메시지를 송신하면 에러를 리턴하게 되므로 일정시간 후 재송신하도록 할 필요가 있다.

대기행렬의 송신 세마포어에서 최상위 1비트는 송신상태가 비동기통신이면 '0', 동기통신이면 '1'로 지정하고, 하위 7비트는 송신한 태스크의 번호를 저장하고 있다. 수신 세마포어에서는 최상위 1비트는 사용하지 않고 하위 7비트만 수신대기 상태의 태스크 번호를 나타내도록 하였다.

8 bit	transfer pointer
8 bit	receive pointer
48 bit	wait queue 1
48 bit	wait queue 1
48 bit	wait queue 1
48 bit	wait queue 1
48 bit	wait queue 1

Wait Queue

8 bit	transfer semaphore
8 bit	receive semahore
16 bit	value of message 1
16 bit	value of message 2

Fig.2.4 **Format of Mail Box**

2.2 멀티태스킹 알고리즘

멀티 태스킹의 기능이 많을수록 메모리의 용량도 커지므로 본 논문에서는 메모리 용량을 최소화하기 위해 꼭 필요한 기능들만을 다음과 같이 정의 한다.

2.2.1 초기화 루틴 (_init_rtm)

시스템을 startup 할 때에 이용되는 기능으로서 태스크 등록 기능에 의해 태스크의 등록을 행하기 전에 반드시 실행해야 한다. 이 함수가 호출되면 아래의 step 1에서 처럼 인터럽트 테이블 영역내에 RTM 서비스의 인터럽트백터가 설정되고 step 2에서 TCB 등의 작업영역은 모두 0으로 클리어된다. 그러나 Mail Box나 RCB의 초기화는 행해지지 않으므로 멀티태스크 개시 기능을 호출하기 전에 사용자가 반드시 0으로 초기화 시켜야만 한다.

step 1. Set interrupt vector of real-time monitor

step 2. Initialize working area of real-time monitor

2.2.2 태스크 등록 (_rtm_entry)

아래 step 1에서와 같이 태스크를 스케쥴링의 대상이 되도록 등록하는 기능으로 초기에 태스크를 한꺼번에 등록할 때 사용되며 태스크테이블은 step 2에서와 같이 영역만을 확보해 두었다가 이 함수가 호출되면 필요한 정보를 태스크테이블에 설정한다.

step 1. Entrance tasks as the object of scheduling

step 2. Set task information to task table

2.2.3 메시지 송신 (_rtm_send)

아래의 알고리즘과 같이 step 1에서 상대측이 수신하고 있는 상태인지 아닌지를 검출하여 수신하고 있으면 상대측에 메시지를 넘겨주고 step 2에서 '0'을 리턴하여 종료하고 수신하고 있지 않을 경우에는 step 4로 가서 Mail Box상의 대기행렬이 비어 있는지를 조사한다. 이 때 대기행렬이 비어 있으면 메시지를 대기행렬에 저장하고 빠져나오며 대기행렬이 가득차 있을 경우에는 step 6에서 '1'을 리턴하고 비정상종료한다.

step 1. If target task is not already receiving then goto step 4

step 2. Transfer message to target task

step 3. return '0'

step 4. If wait queue is full then goto step 6

step 5. Transfer message to wait queue and goto step 3

step 6. return '1'

2.2.4 메시지 수신 (_rtm_recv)

step 1에서 대기행렬에 저장된 메시지가 있는지를 검색하여 저장된 메시지가 있으면 step 2에서 메시지를 수신하고 step 3에서 그 메시지 값을 리턴한 후 종료한다. 만일 메시지가 아직 송신되어 있지 않을 경우에는 step 4와 같이 지정한 Mail Box에 메시지가 송신될 때까지 대기상태로 들어간다.

step 1. If wait queue is empty then goto step 4

step 2. Save message to register

step 3. Return message value

step 4. Wait for transfer message

2.2.5 자원의 배타적 점유 (_rtm_enqw)

step 1에서 RCB가 다른 태스크에 의해 이미 점유되어 있는지를 조사한다. 지정한 RCB가 다른 태스크에 의해 점유되어 있지 않을 경우에는 step 2에서 RCB는 현재 작업중인 태스크에 의해 점유되며 다른 태스크에 의해 이미 점유중일 때에는 step 3에서 대기행렬에 등록하고 그 태스크는 지정한 RCB가 해방될 때까지 대기상태가 된다. 비정상종료의 경우 이외에는 스케쥴링을 행한다.

step 1. If RCB is already occupied by another task then goto step 3

step 2. RCB is occupied by working task

step 3. Set working task to wait queue

step 4. Wait for resource free

2.2.6 자원점유의 해제 (_rtm_deq)

지정한 RCB가 대상 태스크에 의해 점유되어 있는지를 step 1에서 조사하여 점유되어 있을 때는 step 2에

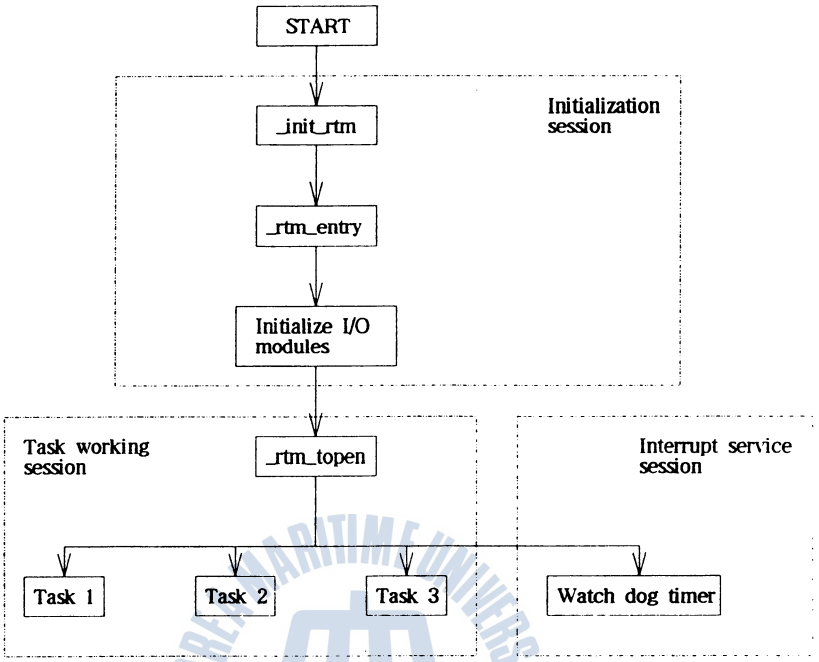

Fig.3.2 **Overall flowchart of SSU driving program**

1) 초기화부 (Initialization session)

초기화부는 Fig.3.2 에서와 같이 세 루틴으로 구성되고 각각의 기능은 다음과 같다.

① _init_rtm: RTM에서 사용하는 모든 작업영역을 초기화 한다.

② _rtm_entry: 각 태스크가 스케쥴링의 대상이 될 수 있게 등록한다.

③ 주변소자 초기화: 시스템의 구동에 필요한 주변소자들을 동작 특성에 맞게 초기화 한다.

2) 태스크 구동부 (Task working session)

일단 태스크가 구동되면 각각의 태스크는 타임슬라이스 기법에 의해 멀티태스킹되며 태스크 내부에서 일어나는 특정 상황에 대한 동작은 자원의 점유를 통한 이벤트드리븐 방식으로 구동함으로써 태스크의 절환시간을 최소화하고 있다.

각 태스크의 동작 내용은 다음과 같다.

① Task 1: Fig.3.3의 key_in()에서 8255를 통해서 키의 상태를 입력받아 그 키값이 일반 키의 값이면 일반키 데이터 변수에 저장한다. 그리고 파라미터 값의 변동을 지시하는 키가 눌려지면 그 변경된 값을 파라미터 변수에 저장하는데 이 때 변경된 파라미터 값은 변수에만 저장이 될 뿐 EEPROM상에는 저장되지 않는다. EEPROM의 영역에 저장하는 경우는 파라미터 값의 재저장을 지시하는 키가 눌려질 때이며 EEPROM

영역에 데이터를 저장하는 것은 전원의 소멸시에도 파라미터값을 안전하게 보관하기 위해서이다.

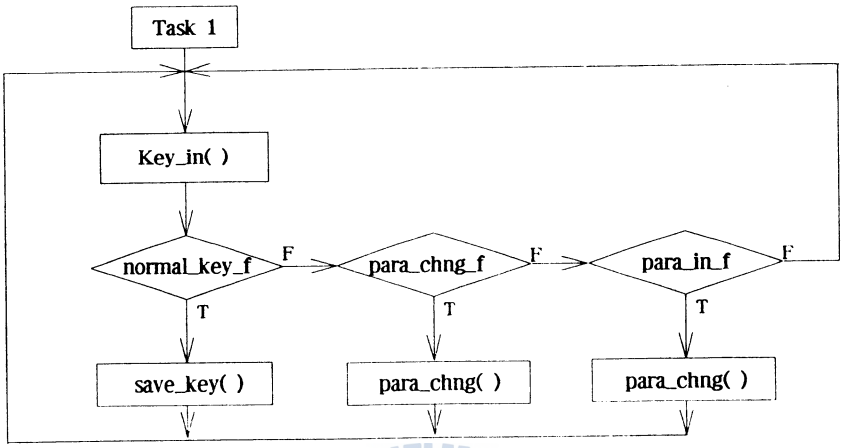

Fig.3.3 **Flowchart of task 1**

② Task 2: Fig.3.4의 rpm_in()과 adapt_card_in()에서 RPM 검출기와 어뎁터 카드를 통해 들어오는 값을 입력받아 이전의 값과 비교하여 변동이 있으면 새로운 값을 저장하고 그렇지 않을 경우에는 변동이 있을 때까지 입력값과 비교만을 행한다. 이때 RPM은 0에서 4096사이의 8비트 디지탈 값으로 변환되어 입력되며 어뎁터 카드에서는 Shut down, Slow down, Emergency stop등의 상태가 비트별로 입력된다.

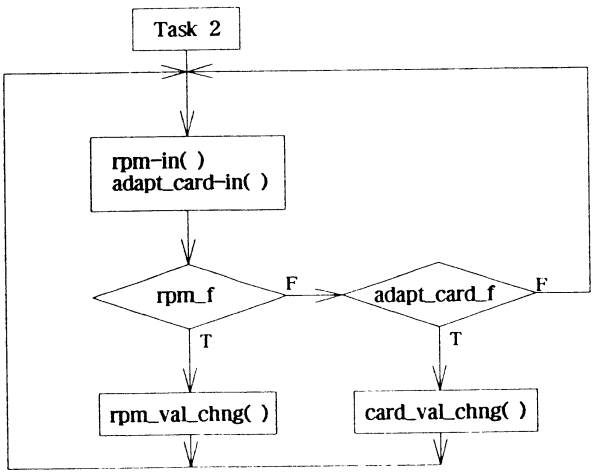

Fig.3.4 **Flowchart of task 2**

③ Task 3: 검출된 데이터 값을 바탕으로 산출되는 결과가 이전의 상태와 다를 경우에는 LED나 세븐 세그먼트를 통해 변동된 상태를 출력하거나 릴레이를 구동시키고 그렇지 않을 경우는 데이터의 비교와 연산만을 행한다. 즉 Fig.3.5에서와 같이 읽어들인 데이터와 파라미터 값을 비교하여 데이터에 변환이 있을 때는 disp_lamp()를 통해 램프를 구동시키고, 파라미터의 변동이 있을 때는 disp_para()에서 변환된 파라미터를 세븐세그먼트를 통해 디스플레이하며 솔레노이드 밸브 및 알람의 구동은 run_relay()와 alarm_relay()에서 행한다.

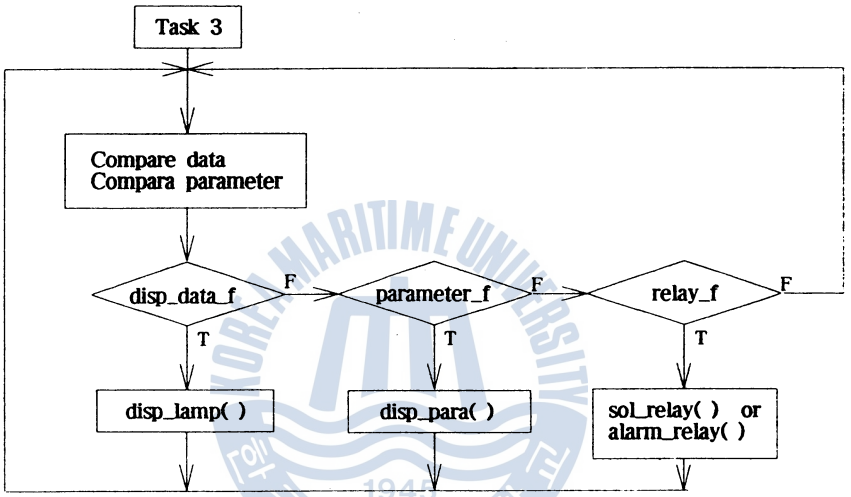

Fig.3.5 **Flowchart of task 3**

3) 인터럽트 처리부 (Interrupt service session)

8253으로 부터 클럭이 발생하면 그것이 8259의 입력이 되어 주기적으로 인터럽트 신호를 발생시킨다. 이때 프로그램상에서 인터럽트 서비스 루틴이 실행되어 매 주기마다 시스템의 이상유무를 점검하게 된다.

Fig.3.6에서 매 인터럽트 발생시 마다 count가 '10'인지를 비교하여 '10' 이 아니면 count를 '1' 증가 시키며, '10'이면 0x58번지를 엑세스하여 워치도그타이머 기능을 수행하고 count를 '0'으로 세트한 후 8259에 인터럽트의 끝을 알리고 리턴한다.

4. 결 론

본 논문에서는 실시간 처리가 가능한 멀티태스크 기법을 이용한 SSU 시스템의 구현에 대해 논하였다. 구

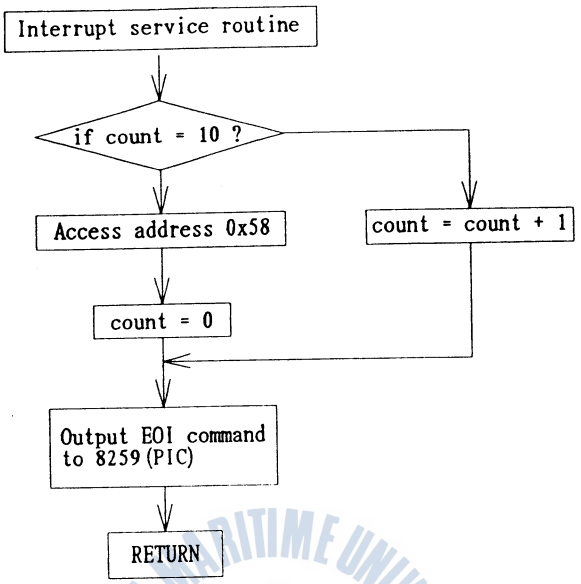

Fig.3.6 **Flowchart of interrupt service routine**

현한 시스템을 ROM화한 후 NORCON사의 Autochief 4 시스템 중에서 SSU를 대상으로 실험하였으며 RPM 값의 입력, Shut down, Slow down, Emergency stop 등의 외부입력 값은 실제 선박을 대상으로 하지는 못하였고 별도로 제작한 가상입력장치를 이용하였다.

멀티태스킹 환경하에서 각 상태 및 RPM 값의 입력, EEPROM에 대한 저장 및 삭제, 램프 및 세븐 세그먼트의 구동 등의 기본적인 동작이 실시간내에서 정상적으로 진행됨을 확인하였다. 특히 EEPROM에 데이터를 저장 할 때에는 일반적인 RAM에 저장하는 경우보다 많은 시간이 요구되므로 한번에 많은 데이터를 저장하면 오동작이 발생함을 알았다. 이를 방지하기 위해 본 시스템에서는 데이터의 변환이 발생하면 변환된 일부분만을 저장하도록 하였다. 또한 멀티태스크 기법을 이용한 시스템 구성은 다른 시스템 구성법에 비해 태스크의 수정 및 추가, 삭제가 용이함을 알 수 있었다.

앞으로 다른 유니트와의 통신기능을 추가하여 실선에서 이용가능한 시스템을 구현하고자 한다.

참고문헌

1) 實踐リアルタイム プログラミンダ技法, 大原 茂之 & 澤田 勉 & Peter Petrov, オーム社, 1991.

2) 運營體制, 金榮燦, 尙潮社, 1992.

3) Bootstrap "8086ボードコンピュータの 設計&製作 series" project 1, NO.1-6, CQ出版社, 1992.

4) Real-time Software for Control, David M. Auslander & Cheng H. Tham, Prentice hall, 1990.

5) Instruction Manual for Safety System Unit(SSU 8810), NORCONTROL Automation AS, Grafisk Design, 1993.