GPS 항법 컴퓨터를 위한 실시간 운영체제의 설계 및 구현

배 장 식[†] · 송 대 기[†] · 이 철 훈[††] · 송 호 준[†††]

요 약

GPS(Global Positioning System)는 시간, 기상 상태에 관계없이 지구 전역에서 사용 가능한 가장 이상적인 항법 시스템이다. GPS는 현재 건설, 측량, 환경, 통신과 지능 항체 등 다양한 분야에서 응용되고 있으며, 앞으로 그 활용은 더욱 커질 것이다. 본 논문은 GPS와 관성 항법 시스템(INS : Inertial Navigation System)을 혼합 구성한 수신보드의 항법 컴퓨터 부에서 동작하는 실시간 운영체제에 대한 설계 및 구현에 관한 것이다. 실시간 운영체제는 항법 컴퓨터 부에서 GPS의 운용 태스크(task)들을 수행하게 된다. 개발된 실시간 운영체제는 GPS 수신기 수행에 최적인 환경을 제공하도록 하며, 사용자로 하여금 고수준의 응용으로 GPS용 프로그램을 개발할 수 있도록 한다. 본 운영체제는 태스크들을 우선 순위 기반으로 처리하는 선점형(preemptive) 스케쥴링 방식을 채택한 운영체제이며, 동적 메모리 관리, 인터럽트, 타이머, IPC 등의 부분으로 구성되어 있다. 논문에는 GPS/INS 통합 보드 시스템의 구조, 운영체제의 구조, 개발 환경, 실시간 운영체제의 성능 평가에 대한 내용들을 기술하였다.

Design and Implementation of Real-Time Operating System for a GPS Navigation Computer

Jang-Sik Bae[†] · Dae-Ki Song[†] · Cheol-Hoon Lee[††] · Ho-Jun Song[†††]

ABSTRACT

GPS (Global Positioning System) is the most ideal navigation system which can be used on the earth irrespective of time and weather conditions. GPS has been used for various applications such as construction, survey, environment, communication, intelligent vehicles and airplanes and the needs of GPS are increasing in these days. This paper deals with the design and implementation of the RTOS (Real-Time Operating System) for a GPS navigation computer in the GPS/INS integrated navigation system. The RTOS provides the optimal environment for execution and the basc platform to develop GPS application programs. Thc key facilities supplied by the RTOS devcloped in this paper are priority-based preemptive scheduling policy, dynamic memory management, intelligent interrupt handling, timers and IPC, etc. We also verify the correct operations of all application tasks of the GPS navigation computer on the RTOS and evaluate the performance by measuring the overhead of using the RTOS services.

키워드 : 실시간 운영체제(Real-Time OS), 임베디드 시스템(Embedded System), 위성 항법 시스템(Global Positioning System)

1. 서 론

GPS는 미 국방성에서 군사적인 목적으로 만든 항법 시스템으로, 시간, 기상 조건에 관계없이 지구상에 있는 항체의 위치, 속도, 시간 등을 알아낼 수 있다. GPS 수신기는 군사용 정밀 측위 뿐만 아니라 선박, 자동차, 항공기, 환경, 통신 등 위치 측정 및 시각 동기 등에 폭넓게 활용되고 있고, 관성 항법 시스템과 결합되어 자세 제어 등 여러 분야에서 응용이 확산되어 가고 있다[1]. 일반의 GPS 수신기 운용 프로그램은 실시간으로 인공위성으로부터 수신되어지는 원시 데이터 획득, GPS 항법, 자세 결정, 통합 항법, 위성 추적을 수행하는 태스크들로 구성되어 지는데, 이러한 GPS 태스크들은 시간에 밀접한 작업을 수행하므로 그들 간의 스케쥴링을 위해서는 실시간 처리를 보장해 주는 컴퓨팅 환경을 필요로 한다. 또한 점차 GPS 수신기의 높은 신뢰도가 요구되고, 사용 범위가 넓어져 GPS 수신기의 운용 프로그램들도 다양해지면서 이를 위한 운영체제가 필요하게 되었다. 이 운영체제는 일반적인 운영체제와는 달리 GPS 수신기에 적합해야 하며 또한 인공위성으로부터의 메시지를 실시간으로 처리할 수 있어야 하기 때문에 실시간 운영체제가 사용되어야 한다. 많은 범용의 상용 실시간 운영체제들이 존재하지만 실제 GPS 수신기에 필요 없는 기능들로 인한 오버헤드로 성능상의 문제가 있기 때문에 시스템에 적합한 실시간 운영체제를 독자적으로 설계하고 구현하였다. 실시간 운영체제는 그 특성상 MS-DOS, Windows98, UNIX, LINUX 등과 같이 컴퓨터에

※ 본 연구는 1997년도 학술진흥재단 대학부설연구소 지원 과제에 의해 수행되었음.
† 준 회 원 : 충남대학교 대학원 컴퓨터공학과
†† 정 회 원 : 충남대학교 컴퓨터공학과 교수
††† 정 회 원 : (주)코아매직 부사장
논문접수 : 2001년 6월 13일, 심사완료 : 2001년 9월 24일

쓰이는 범용 운영체제와는 달리, 시스템이 예상하지 못한 특정 이벤트가 발생하는 악 조건에서도 태스크 수행의 데드라인을 초과하지 않도록 시간적 측면의 결정성(determinism)을 보장하는 안정된 스케줄링 기능을 갖춘 운영체제이다.

타겟(Target) 시스템에 최적화된 실시간 운영체제를 설계하기 위해서는 하드웨어의 구조적 특성과 응용프로그램의 성질을 고려해야 한다. 초기 실시간 운영체제를 선정할 때에는 인터럽트 처리, 커널 서비스 함수, 태스크 스위칭(task switching)과 관련된 지연(latency)을 최소화였으나, 오늘날에는 마이크로프로세서가 발전하면서 지연 최소화의 중요성이 점차 약해지고 있다. 이제는 이러한 요소보다 더 다양한 면에서 성능을 비교할 수 있는 기준이 필요하다. 기본적인 요구사항으로 성능을 저하시키지 않고 메모리를 많이 요구하지 않도록 코드가 간략해야 한다. 또한 사용자의 편의성을 고려하여 적절한 시스템 서비스 호출(system service call)들을 제공하고, 태스크 스케줄링 정책면에서도 다양한 응용프로그램 요구에 유연하게 적용할 수 있도록 설계해야 한다[2, 5].

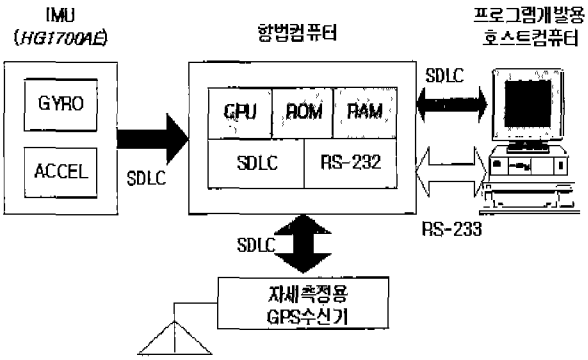
본 논문에서 구현된 실시간 운영체제는 앞서 언급한 고려사항들 중에, 운영체제로서 갖추어야 할 다양한 기능을 지원하면서도 각 기능들을 모듈별로 설계하여 최소의 메모리를 갖도록 하였다. 그리고 실시간 운영체제의 주요 특징인 결정성을 보장하기 위해 인터럽트가 disable 되는 구간을 최소화시켜 인터럽트 응답시간을 줄였다. 또한 실시간 운영체제 상에서 동작하는 응용프로그램들의 데드라인과 공유자원의 안정적 접근을 위해 통신 및 동기화 도구에 대해 중점적으로 검증하였다.

본 연구는 GPS 수신기에 적합한 실시간 운영체제를 개발하는 것으로, GPS 수신기 운용 프로그램의 실시간 특성을 분석하고, 수신기의 하드웨어적인 면을 고려하여 최적화된 운영체제를 설계하고 구현하는 것을 목적으로 하고 있다.

본 논문은 5개의 장으로 구성되어 있다. 2장에서는 실험 대상이 되는 GPS/INS 통합 보드 시스템의 구조와 특징을 언급하고, 3장에서는 실시간 운영체제의 설계 시 고려사항과 구현 내용을 기술한다. 그리고 4장에서는 실험 및 개발환경과 실시간 운영체제의 성능 평가를 보인다. 마지막으로 5장에서 이 논문의 결론을 맺는다.

2. GPS/INS 통합 보드 시스템 구조

본 논문에 대상이 되는 GPS 수신기 보드의 전체 하드웨어 구조는 크게 자세 측정용 GPS 수신기로 구성되는 센서부와 항법 컴퓨터부, 그리고 프로그램 개발을 위한 호스트 컴퓨터로 나눌 수 있다. GPS 수신기는 4 안테나 48 채널의 자세 측정용 GPS 수신기를 사용하였으며 항법 컴퓨터의 프로세서는 통합 시스템의 많은 데이터량을 처리하기 위해 최신의 RISC CPU인 Intel사의 StrongARM(SA-1100)을 사용하였다. 아래의 (그림 1)은 본 연구에서 대상 된 GPS/INS 통합 항법 시스템 보드의 구성을 보이고 있다.

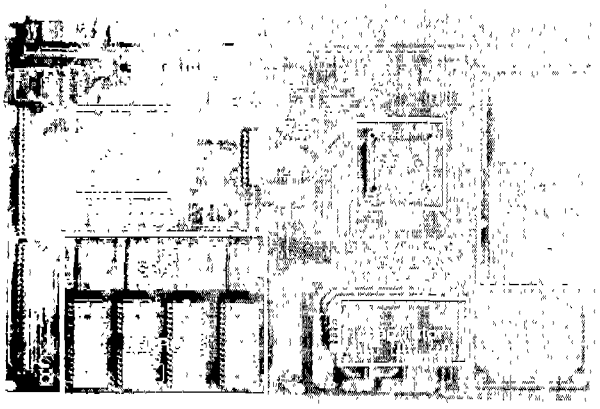


(그림 1) GPS 통합 시스템 보드 구성

(그림 1)에서 보는 바와 같이 시스템은 자세 결정용 GPS부와 IMU 그리고 각 부분으로부터 얻은 정보로써 계산을 수행하는 항법 컴퓨터부로 구성된다. 자세 결정용 GPS부는 크게 4개의 안테나 48채널 수신기와 위치, 속도 및 자세 데이터를 출력할 SDLC, UART로 구성되며, IMU부는 Honeywell사의 중저급인 HG1700AE를 사용하였다. 항법 컴퓨터는 통합 시스템의 많은 데이터량을 처리하기 위해 RISC CPU인 Intel사의 StrongARM(SA-1100)을 사용하였다. GPS수신부와 IMU부의 인터페이스는 GPS수신부는 많은 양의 데이터

〈표 1〉 각 부 시스템 특징

분 류	특 징
자세 측정용 GPS 수신기	• 4 안테나 48 채널 수신기 • 위치, 속도 및 자세 데이터 출력 • 최대 1Mbps의 Single SDLC • 최대 115Kbps의 Dual UART
항법 컴퓨터	• Intel StrongARM @220MHz (SA-1100) • 최대 1Mbps의 Dual SDLC • 최대 115Kbps의 Dual UART • 최대 100Kbps의 ARINC 429 • 실시간 운영체제 탑재
프로그램 개발용 호스트 컴퓨터	• PC(Windows 2000)
개발 환경	• Visual C++ • ARM 컴파일러 및 어셈블러



(그림 2) 항법 컴퓨터 보드

전송이 필요하고 IMU부가 SDLC 프로토콜을 사용하기 때문에 Zilog 사의 Z16C30 USC를 사용하여 인터페이스를 설계하였다[3, 4]. 이 시스템 보드는 충남대학교 전자공학과 제어연구실에서 설계 개발한 것이다. 각 부 시스템의 간략한 특징을 <표 1>에 나타내었다.

3. 실시간 운영체제 설계 및 구현

3.1 설계 및 구현의 고려사항

GPS/INS통합 시스템은 항법보드, 하드웨어들을 제어하기 위한 디바이스 드라이버(device driver)로 구성된 BSP(Board Support Package) 그리고, 항법 알고리즘을 구현하기 위한 모듈화 된 실시간 항법 소프트웨어로 구성된다. 항법 알고리즘을 구현하기 위한 모듈들은 다수의 태스크들과 인터럽트 서비스 루틴(ISR)으로 구성이 되며, 각각의 루틴들은 멀티태스킹 환경 하에서 수행된다.

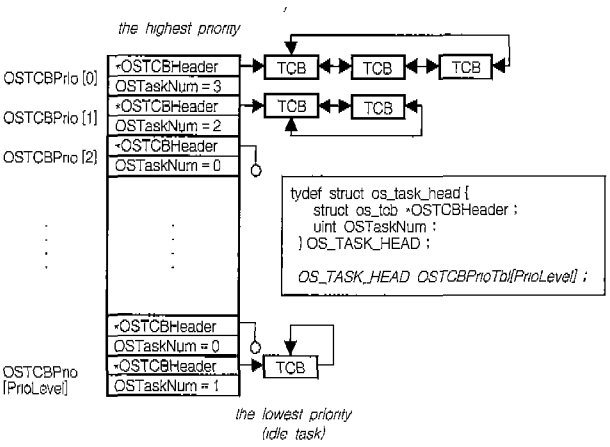
실시간 운영체제 설계시 고려되어야 할 사항은 항법 컴퓨터 보드의 하드웨어 특성들과 항법 소프트웨어의 수행 특성이다. 하드웨어의 경우 항법 컴퓨터 보드가 Intel의 StrongARM에 기반 한다는 점과 IMU부의 SDLC 프로토콜을 제어하기 위한 Zilog 칩을 사용했다는 점이다. 이와 관련하여 문맥교환(context switch)과 인터럽트 처리와 같은 운영체제의 하드웨어 의존적인 부분들을 StrongARM에 맞게 수정하였으며, Zilog 칩의 경우 구동을 위한 BSP 코드를 작성하였다. 그리고 항법 소프트웨어의 경우, 멀티태스킹 시 공유자원의 효율적인 관리 및 태스크들의 주기성을 보장하기 위한 실시간 스케쥴링 방법을 고려하였다. 또한 항법 컴퓨터에서 GPS 태스크에 ITC (Inter-Task Communication) 도구를 제공하여 태스크간 통신을 가능하게 하고 세마포어(semaphore), 뮤텍스(mutex)를 사용하여 공유자원의 효율적인 관리를 가능하게 하였으며, 각각의 태스크들의 실시간 적인 주기성을 보장하기 위해 빠른 인터럽트의 처리가 가능하게 하였다.

3.2 실시간 운영체제의 구조 및 기본 서비스 구현

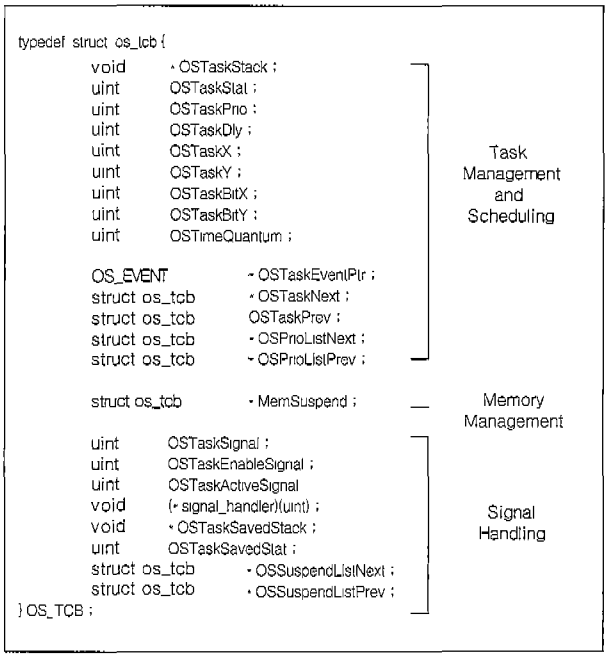
3.2.1 커널 및 태스크 구조

GPS 항법 컴퓨터에서 사용되는 각각의 태스크는 타겟의 RAM 크기와 응용프로그램에서 요구하는 크기에 따라 스택영역을 독립적으로 지정한다. 태스크들을 제어하기 위한 커널의 구조가 TCB(Task Control Block)이며, 각각의 TCB는 (그림 3)과 같은 OS_TASK_HEAD 구조체에 의해 관리된다. 이는 실시간 운영체제가 초기화 될 때 미리 정의되어야 하는 부분으로, OSTCBHeader는 생성된 태스크의 정보를 가지고 있는 TCB를 가리키는 포인터로 사용되고 OSTaskNum은 해당 우선순위에 존재하는 태스크의 개수를 나타내기 위해 사용된다. 생성된 각 태스크들의 정보를 저장하고 있는 TCB필드는 (그림 4)와 같다.

- OSTaskStack
  태스크에 할당된 스택의 탑(top)을 가리키며, 각각의 태스크는 자신의 로컬 스택을 가지고 있다. 태스크 생성시 응용프로그램 작성자의 요구에 따라 크기도 설정할 수 있다. 스케쥴링이나 인터럽트에 의한 문맥교환이 발생할 때 어셈블리 언어로 작성된 코드에서 사용한다.
- OSTaskStat
  태스크의 현재 상태를 나타내며, 이 값이 0이 되면 ready 상태로서 CPU에 의한 스케쥴링 대상에 포함된다.
- OSTaskPrio
  태스크의 우선순위를 저장하며, 숫자가 낮을수록 더 높은 우선 순위를 갖는다.
- OSTaskDly
  태스크가 지연(delay) 또는 특정 이벤트들에 대해서 펜딩(pending)될 때 얼마만큼의 타임아웃(timeout) 값을 가질 것인가를 나타낸다.



(그림 3) TCB를 위한 우선순위 테이블의 헤더(Header)구조



(그림 4) TCB의 구조

- OSTaskX(Y), OSTaskBitX(Y)
  태스크를 ready 상태에서 빠르게 스케줄링하기 위해 사용된다. 이 필드의 값들은 태스크가 생성되거나 삭제되었을 때 또는 태스크의 우선순위가 변경되었을 경우 (그림 5)와 같이 계산되어 등록된다.

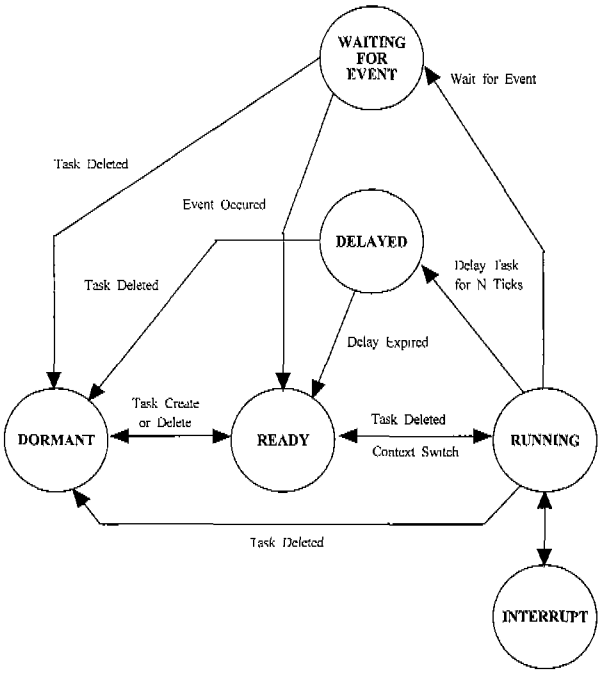
```
OSTaskX      = priority & 0x07;
OSTaskY      = OSMapTbl[priority & 0x07];
OSTaskBitX   = priority >> 3;
OSTaskBitY   = OSMapTbl[priority >> 3];
```

(그림 5) TCB의 우선순위 등록

- OSTimeQuantum
  현재 태스크의 타임 슬랏(time slot) 값이며, 같은 우선순위의 태스크들을 라운드 로빈(round robin) 시킬 때 사용된다. 전역변수로 정의된 퀀텀(quantum) 값이 태스크 생성시 부여되며 OSTimeTick( ) 함수가 호출될 때마다 현재 태스크의 OSTimeQuantum 값이 1씩 감소된다. 만약 0이 되었다면, 같은 우선순위의 태스크가 CPU를 점유한다.
- OSTaskEventPtr
  ECB(Event Control Block)을 가리키는 포인터이다.
- OSTaskNext, OSTaskPrev
  OS_TCB를 이중 연결 리스트로 만드는데 사용한다. 이 리스트는 하드웨어 틱(tick) 때마다 각 태스크의 OSTaskDly 값을 갱신하는 OSTimeTick( ) 함수에 의해 사용된다.
- OSPrioListNext, OSPrioListPrev
  ready 리스트에 있을 때 같은 우선순위의 TCB들을 이중 연결 리스트로 구성하거나 특정 이벤트를 기다리며 펜딩 상태에 있는 TCB들을 suspend 리스트에 우선순위대로 연결한다.
- MemSuspend
  동적 메모리 할당에 의해 메모리를 할당받을 경우, 더 이상 충분한 메모리가 없게 되면 메모리를 요청한 태스크들은 중단되어야 한다. 이 때 중단되는 태스크들을 단일 연결 리스트로 관리하게 되며, MemSuspend는 이를 가리키는 포인터로 쓰인다.
- Signal Handling을 위한 Optional Field
  시그널 처리를 위한 추가 필드로서 차후에 다시 언급할 것이다.

생성된 태스크를 식별하기 위해서는 응용프로그램에서 각 태스크의 TCB를 정의하고, TCB의 주소로서 태스크를 식별한다. 선점형 커널로서 항상 최상위 우선순위 태스크가 CPU를 점유하여 자신의 작업을 수행한다. 이벤트에 사용되는 ECB 역시 응용 프로그램이 수행되기 이전에 정의해야 한다. (그림 6)은 태스크들의 상태 천이를 나타낸 것이다.

(그림 6)과 같이 태스크는 dormant, ready, running, delay, wait for event, interrupt 상태에 있을 수 있다. 태스크의 생성은 문맥교환을 위해 각 CPU의 특성에 맞는 레지스터 셋(register set)의 구성 요소들을 저장할 수 있는 로컬 스택영역 확보해야 하고, 수행할 루틴과 우선 순위를 정의해야 한다. 태스크의 삭제는 TCB에서 제외시켜 더 이상 커널이 이를 이용하지 않는 상태로 만드는 것이다. 이러한 태스크의 생성과 삭제는 동적으로 가능하지만 본 구현에서는 미리 정의된 태스크를 커널에 등록시켜 운영하는데, 이는 수행도중 태스크의 우선순위를 변화시키는 것은 빈번한 문맥교환과 커널 자료구조 접근을 위해 인터럽트를 disable 시켜야 하므로 시스템 성능의 저하가 크기 때문이다.



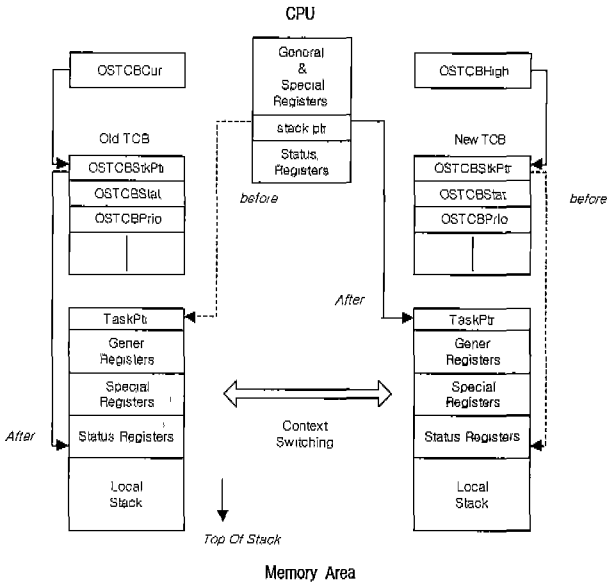
(그림 6) 태스크 상태 천이도

3.2.2 문맥 교환

CPU는 여러 태스크나 ISR에 의해 공유되는 중요한 자원이므로 현재 CPU를 점유하여 수행중인 프로그램은 자신의 우선순위보다 높은 태스크가 실행할 준비가 되거나, 외부 인터럽트에 대해 ISR을 수행시키기 위해 문맥교환을 일으킨다. CPU를 점유하고 있던 낮은 우선순위의 태스크는 자신이 수행하던 작업을 중단하고, 그 시점의 PC(program counter)와 CPU의 레지스터등과 같은 수행 환경을 로컬스택에 저장하고, CPU를 점유하게 될 태스크나 ISR의 로컬스택에서 새로운 CPU환경을 읽어와서 수행을 계속하게 된다. 아래의 (그림 7)은 두 개의 태스크가 문맥 교환을 일으킨 직후에 각 태스크의 로컬 스택과 CPU의 문맥이 어떻게 변화하는가를 나타내고 있다.

3.2.3 스케줄링(Scheduling)

본 논문에서 구현된 스케줄러(scheduler)는 라운드 로빈 알고리즘을 지원하며, 최초 태스크 생성시 부여되는 우선순위를 가지고, 테이블 구조를 이용한 바이트(byte) 연산을 통해 항



(그림 7) 문맥 교환

상 고정된 시간에 최상위 우선순위를 가진 태스크를 디스패치(dispatch) 할 수 있는 방식이다.

아래의 (그림 8)은 태스크들의 스케쥴링을 위한 커널의 자료구조를 나타낸 것으로, OSRdyTbl[8]의 64비트 중에서 1로 셋(set)된 것은 해당 위치의 태스크가 ready되었다는 것을 뜻하며, OSRdyGrp의 8비트 중에서 1로 셋(set)된 것은 해당 위치의 그룹 중에 한 개 이상의 태스크가 ready되었다는 것을 뜻한다. 동일한 그룹 내에서 여러 비트가 1로 셋(set)되었을 때는 LSB(least significant bit)쪽에 가까운 위치에 있는 태스크가 높은 우선순위를 갖는다.

$$y = OSUnMapTbl[OSRdyGrp]\ ; \quad (1)$$

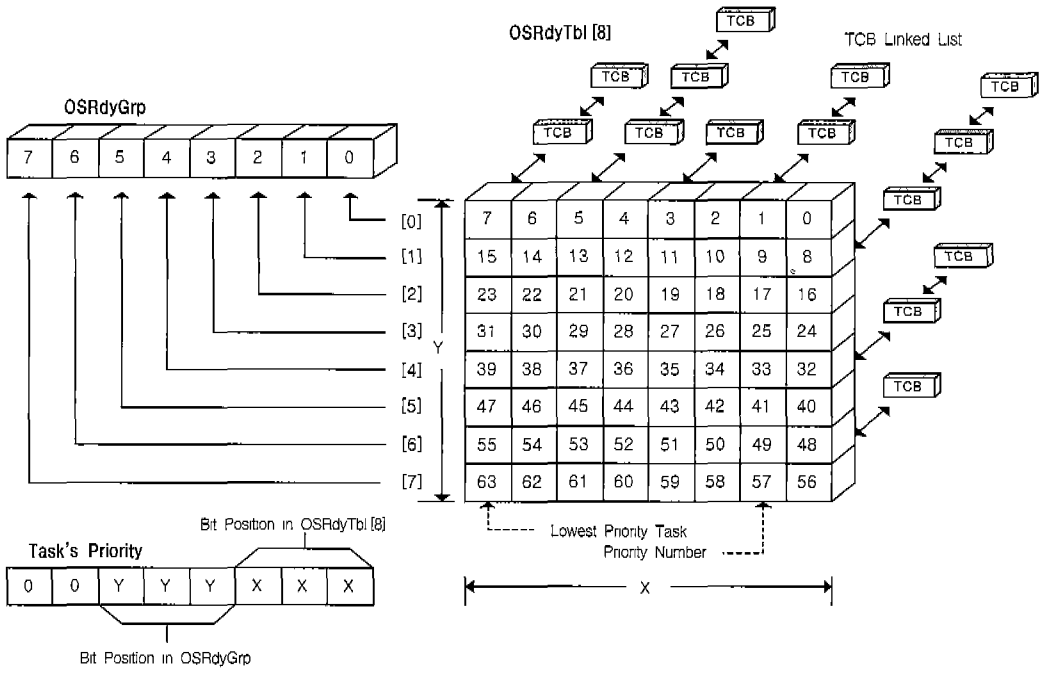
$$x = OSUnMapTbl[OSRdyTbl[y]]\ ; \quad (2)$$

$$p = (y << 3) + x\ ; \quad (3)$$

최상위 우선순위를 갖는 태스크는 위의 계산 과정을 통해 찾을 수 있다. 식 (1)은 OSRdyGrp의 값을 OSUnMapTbl[ ]에 적용하여 최상위 우선순위 태스크가 OSRdyTbl[ ]의 몇 번째 상수인가를 알아낸다. 식 (2)은 식 (1)에서 얻은 값을 이용하여 최상위 우선순위 태스크의 위치를 알아낸다. 식 (3)은 식 (1)에서 구한 값에 8을 곱하고, 식 (2)에서 구한 값을 더하여 최상위 우선순위 태스크의 우선순위 번호를 구한다. 식 (3)에서 p는 태스크의 우선순위로 OSTCBPrioTbl[ ]에서 인덱스로 사용하여 새롭게 선택된 태스크의 TCB 포인터를 얻을 수 있다.

3.2.4 임계 영역과 상호배제

임계영역의 코드는 반드시 원자 연산(atomic operation)으로 처리되어야 하므로, 일단 코드가 실행되면 수행 중에 인터럽트 등의 요인에 의해 중단되어질 수 없다. 이를 보장하기 위해 특별히 임계영역의 코드가 실행되기 전에 인터럽트가 disable 되어야 하며, 임계영역의 코드수행이 완료되었을 때 다시 인터럽트가 enable 되어진다. 이와 유사하게 여러 태스크들에 의해 공유되는 자원이나 데이터 구조에 대해서도 데이터의 충돌(corruption)을 방지하기 위해 배타적인 접근을 제공해야 한다. 이를 보장하기 위한 방법으로 Motorola의 68000 패밀리 프로세서들과 같이 TAS(Test-And-Set)라는

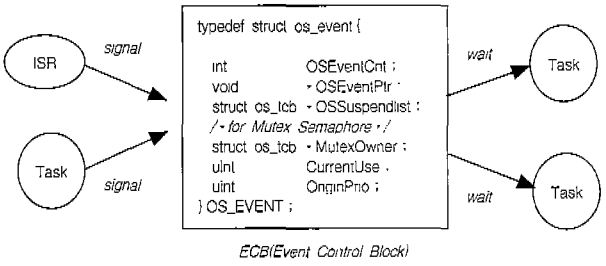


(그림 8) 스케쥴링을 위한 커널 구조

명령어를 사용하거나, 세마포어 도구를 사용하거나, 인터럽트를 disable/enable 하는 방식을 사용한다. 이들 중에 인터럽트의 disable/enable은 보통 벤더가 제공하는 CPU의 레지스터 셋 중에서 인터럽트의 수용여부를 상태 레지스터(SR)의 특정 비트를 셋 또는 리셋 함으로써 구현할 수 있다. 이러한 방법은 인터럽트 자체를 막는 것이 아니라 발생된 예외 상황을 CPU가 무시하도록 하는 것이므로 인터럽트 제어 레지스터 내에 그 정보가 남아서 나중에 지연 처리되는 것이다.

3.2.5 태스크간 동기화와 통신

본 논문에서 구현된 실시간 운영체제는 태스크들간의 동기화를 위해서 세마포어를 제공하며, 태스크들간의 통신을 위해서 메시지 메일박스, 메시지 큐를 제공한다. 또한 운영체제의 결정성을 보장하기 위해 흔히, 뮤텍스(mutex)라고 불리는 mutual exclusive 이진 세마포어(binary semaphore)를 제공하여 우선 순위 역전 시간을 최소화한다. 이러한 이벤트와 관련된 서비스를 지원하기 위해 (그림 9)와 같은 ECB(Event Control Block) 구조체가 사용된다.



(그림 9) ECB(Event Conrol Block)의 구조

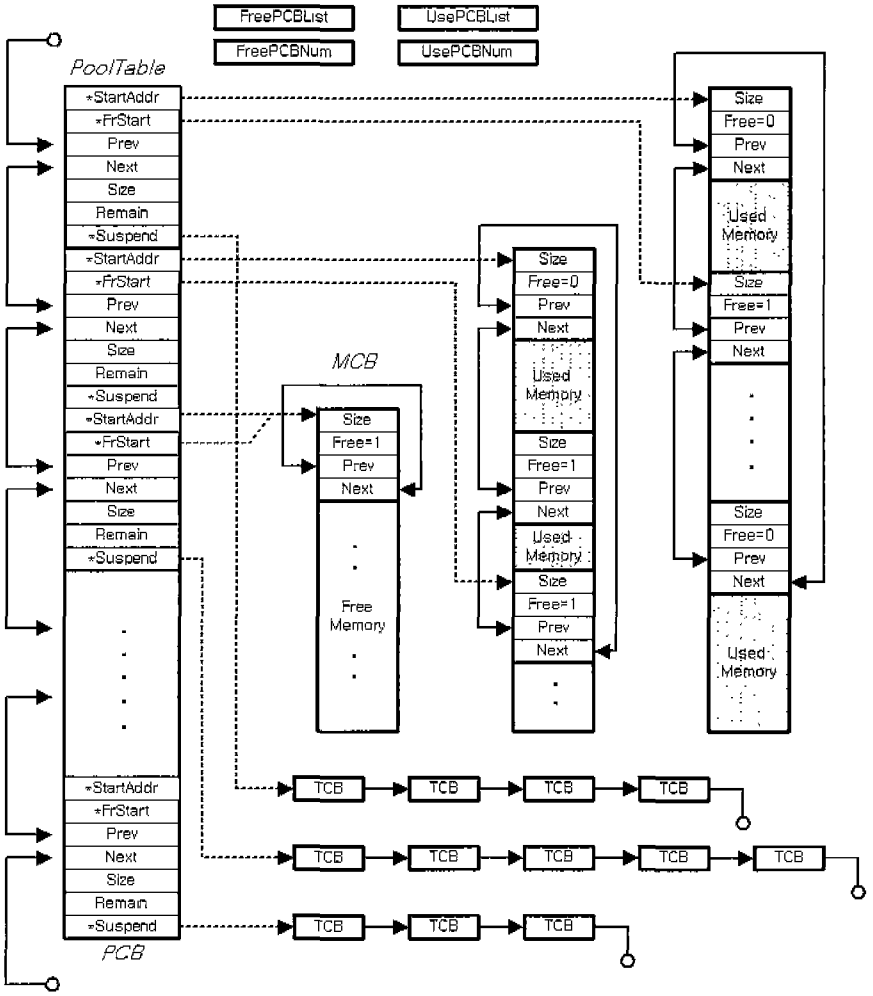
세마포어는 공유자원을 배타적으로 사용하거나 태스크간 동기를 맞출 때 사용된다. 이것은 사용되는 시스템에 따라 정수 범위의 값을 가지고 초기화된다. 만약 32비트 시스템의 경우 세마포어는 0에서 231사이의 값으로 초기화되며, 0이 되면 더 이상 사용 가능한 자원이 없음을 나타내고, 0이상일 경우 양수만큼의 자원이 가용하다는 것을 나타낸다. <표 2>는 통신과 관련한 여러 메커니즘에서 제공하는 서비스를 정리한 것이다.

3.2.6 메모리 관리

본 논문에서 구현된 실시간 운영체제는 메모리 할당 및 해제가 동적으로 이루어진다. 이는 운영체제를 사용하는 응용프로그램에서 수행도중 운영체제 초기화 과정에서 지정한 문자형 배열 공간(pool)으로부터 메모리를 할당하거나 해제하는 것이다. 풀(pool)을 관리하기 위해서 PCB(Pool Control Block)를 사용하고, 풀 내에 메모리를 관리하기 위해서 MCB (Memory Control Block)를 사용한다. 메모리 관리를 위한 전체적인 자료 구조는 (그림 10)과 (그림 11)과 같다. 풀의 개수는 헤더파일에 정의되어있는 MAX_POOL_NUM만큼 생성하여 사용할 수 있으며, 이러한 풀을 관리하기 위한 구조로 PoolTable[MAX_POOL_NUM]이 생성되는데 기본값으로 10을 사용하고 있다. 그림에서 FreePCBList는 사용 가능한 가용 풀들을 연결하는 헤더로 사용되고, FreePCBNum은 사용 가능한 풀의 개수를 나타낸다. UsePCBList는 이미 사용한 풀들을 연결하는 헤더로 사용되고, UsePCBNum은 사용중인 풀의 개수를 나타낸다. 메모리 할당과 해제의 단위는 char이며, 최초 응용프로그램에서 지정한 문자열 배열 공간에서 이뤄진다.

〈표 2〉 통신 기능을 제공하는 서비스

통신 메커니즘	함 수 명	용 도
세마포어	OSSemMake(OS_EVENT * pevent, int value)	응용프로그램에서 쓰이는 태스크가 세마포어를 접근할 수 있도록 ECB 주소와 세마포어의 최초 개수를 정의하여 생성
	OSSemGet(OS_EVENT * pevent, uint timeout, uint *err)	세마포어의 값이 0보다 클 때 세마포어의 값을 하나 감소 시키고 호출한 곳으로 복귀
	OSSemRelease(OS_EVENT * pevent)	다른 태스크나 ISR에 의해 사용을 마친 세마포어의 반환이 이루어지는 이벤트를 발생
메시지 메일박스	OSMboxMake(OS_EVENT * pevent, void * message)	ECB의 포인터와 메시지의 포인터를 넘겨받아 메일박스를 위한 ECB를 초기화
	OSMboxReceive(OS_EVENT * pevent, uint timeout, uint * err)	메일박스에 메시지가 들어있으면, 호출한 태스크에 메시지를 전달하면서 메일박스를 비움
	OSMboxSend(OS_EVENT * pevent, void * message)	메일박스에 메시지를 저장
메시지 큐	OSQMake(OS_EVENT * pevent, OS_Q * pq, void ** start, uint size)	ECB의 포인터, QCB(Queue Control Block)의 포인터, 큐 내에서 메시지가 담길 곳의 시작번지와 크기를 받아 큐에 대한 ECB를 초기화함
	OSQReceive(OS_EVENT * pevent, uint timeout, uint * err)	메시지 큐 안에 하나이상의 메시지가 들어 있을 때, 큐의 OSQOut으로부터 메시지를 꺼내 태스크에 전달
	OSQSend(OS_EVENT * pevent, void * message)	큐에 더 이상 메시지를 담을 공간이 없을 때 에러를 반환하고, 메시지를 담을 공간이 있을 때는 메시지를 큐에 넣음



(그림 10) 동적 메모리 관리를 위한 자료 구조

```
typedef struct pool_control_block
{
        char        * StartAddr ;
        MCB         * FrStart ;
        uint        Next ;
        uint        Prev ;
        uint        size ;
        uint        Remain ;
        uint        MaxNum ;
        uint        BlockSize ;
        OS_TCB      *Suspend ;
} PCB ;
```

```
typedef struct mcb
{
        uint    size ;
        uint    Free ;
        struct mcb * Prev ;
        struct mcb * Next ;
} MCB ;
```

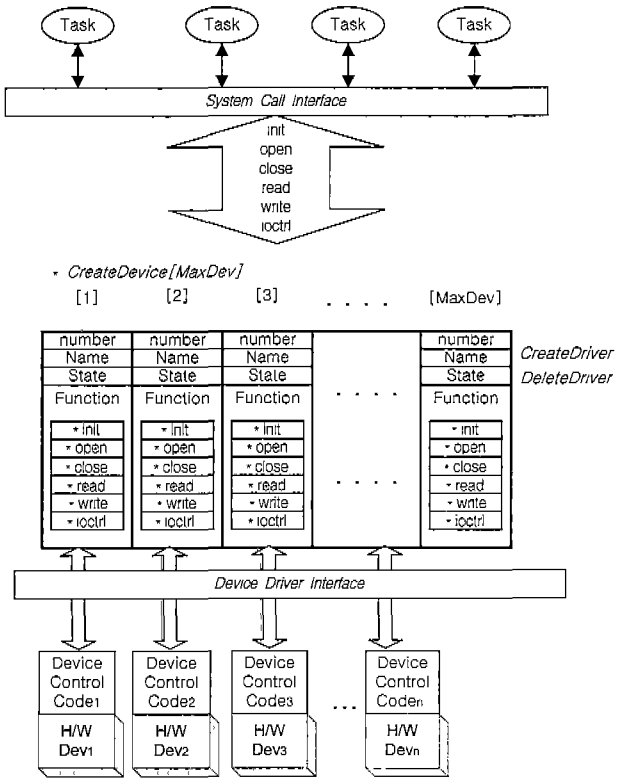
(그림 11) PCB와MCB 구조체

3.2.7 디바이스 드라이버 인터페이스(DDI)

운영 체제에서 DDI(Device Driver Interface)를 사용하는 목적은 세 가지가 있다[6]. 첫째, 대부분의 하드웨어 디바이스들은 비교적 하드웨어에 가까운 상태이므로 이를 제어하고 사용하기 위해서 복잡한 소프트웨어를 요구한다. 운영체제는 디바이스 드라이버라고 불리는 프로그램을 통해 디바이스에 데이터를 전송하고 제어하여 하드웨어에 접근하므로 복잡하고 세세한 동작 사항들을 사용자에게 감출 수 있다. 둘째, 디바이스들은 시스템에서 사용되는 공유 자원이므로 운영체제에 의해 안전하고 공평하게 접근할 수 있는 메커니즘을 제공할 수 있다. 셋째, 운영체제는 사용자가 하드웨어의 세부 설정에 대한 지식이 없어도 상위 수준 프로그래밍 언어를 통해 커널에 등록된 디바이스 이름을 가지고 일관성 있게 디바이스를 사용할 수 있는 유연한 인터페이스를 제공한다. 본 논문에서 구현된 운영체제의 디바이스 드라이버의 구조는 (그림 12)와 같다. 실제 하드웨어들을 접근하고 사용하기 위한 디바이스 제어 코드들이 운영 체제가 제공하는 DDI에 의해서 CreateDevice[] 테이블 내의 필드인 함수에 각각 등록된다. 응용 프로그램 작성자의 측면에서 실제 하드웨어에 대한 복잡한 코드들을 감추고 실제 태스크에서 사용될 디바이스에 접근할 때는 동일한 함수 이름(init, open, close, read, write, ioctrl)으로 시스템 호출하여 사용할 수 있는 유연한 인터페이스를 제공받게 되는 것이다.

3.2.8 시그널 처리(Signal Handling)

시그널은 하나 이상의 태스크들에게 비동기적인 이벤트를 알리기 위해 사용된다. 앞서 언급한 세마포어, 메일박스, 큐와 같은 태스크간의 통신 도구들은 메시지 또는 세마포어를 획득 할 때까지 태스크가 중단된 상태에서 특정 이벤트가 발



(그림 12) 디바이스 드라이버 구조

생하기를 기다려야만 하는 반면에, 태스크가 시그널을 통해 특정 서비스를 요청하면 시그널을 받은 태스크는 이벤트 플래그 변화에 관계없이 CPU를 점유하여 실행 권한이 주어졌을 때 해당 시그널에 대한 처리 루틴을 수행한다. 즉, 수행 중이던 태스크가 시그널을 받으면 바로 처리가 가능하고, 다른 태스크가 CPU를 점유 있을 때 시그널을 받으면, ready상태에서 자신의 실행 권한이 오기를 기다렸다가 최우선 순위가 되어 실행 권한을 얻었을때 받았던 시그널을 처리할 수 있다. 이는 동기화 이벤트 도구와는 달리 시그널이 비동기적 이벤트를 제공함을 의미하는 것이다. 이와 같은 특성 이외에 시그널은 한번 받은 시그널을 처리하기 전에 다시 보내진 시그널을 무시한다. 즉, 태스크는 어떤 시그널을 받았는지 만을 기억할 뿐, 어떤 시그널이 얼마나 보내졌는지는 기억하지 못한다[7]. 시그널 처리를 위한 TCB 내의 필드는 (그림 13)과 같다.

```
typedef struct os_tcb {
        /*------ Signal Handling Fields ------*/
        void        *OSTaskSavedStack;
        uint        OSTaskSavedStat;
        uint        OSTaskSignal;
        uint        OSTaskEnableSignal;
        uint        OSTaskActiveSignal;
        void        (*signal_handler)(uint);
        struct os_tcb    *OSSuspendListNext;
        struct os_tcb    *OSSuspendListPrev;
} OS_TCB;
```

(그림 13) 시그널 처리를 위한 TCB내의 필드

3.2.9 결정성을 보장하는 실시간 스케줄링

결정성의 보장은 태스크의 작업수행이 정해진 시간 안에 모두 수행될 수 있음을 보장하는 것을 의미한다. 태스크들이 정해진 주기에 따라 수행을 해야하는 항법 시스템에서는 이러한 결정성을 보장하는 것이 필수적이라 할 수 있다. 운영체제가 최상위 우선순위의 태스크를 디스패치(dispatch)하는 시간이 태스크의 수 혹은 어떠한 다른 요인에 영향을 받게 된다면 실시간 운영체제의 중요한 특성인 결정성을 보장할 수 없게 된다. 위 3.2.3절의 내용과 같이, 본 논문에서 구현된 실시간 운영체제의 스케줄러는 태스크의 수에 관계없이 고정적인 시간에 최상위 우선순위의 태스크를 디스패치 할 수 있다. 그러므로 항법시스템의 태스크들은 정해진 주기에 따라 정확한 수행을 하게 된다.

4. 실시간 운영체제의 성능평가

본 연구의 성능 평가는 GPS 소프트웨어에서 만든 독자(proprietary) 운영체제와 태스크들 수행시간을 비교하는 것으로 성능 평가를 하였다. 본 연구로 개발되어 있는 운영체제에는 스케줄러, IPC 설비, 세마포어, 타이머, 메모리 관리 기능이 탑재되어 있는 것에 반하여, GPS 소프트웨어의 독자 운영체제는 단지 스케줄러 외에 다른 운영체제로써의 기능이 없으므로 소프트웨어가 복잡해질수록 내장형 시스템(embedded system)에 버그나 오류가 더 빈번하게 나타날 수 있다. 본 성능 평가에서는 GPS 소프트웨어의 성능 및 코드상의 문제와 본 연구를 통해 그것들이 어떻게 개선되어 있는지 설명한다.

4.1 항법 소프트웨어 태스크의 기능

GPS 항법 소프트웨어는 TTakemeas, TNav, TProcSbf, TDisplay, Talloc 그리고 Tattitude 태스크로 구성된다. 각각의 태스크 역할은 <표 3>과 같다.

<표 3> GPS 항법 소프트웨어 구성 태스크

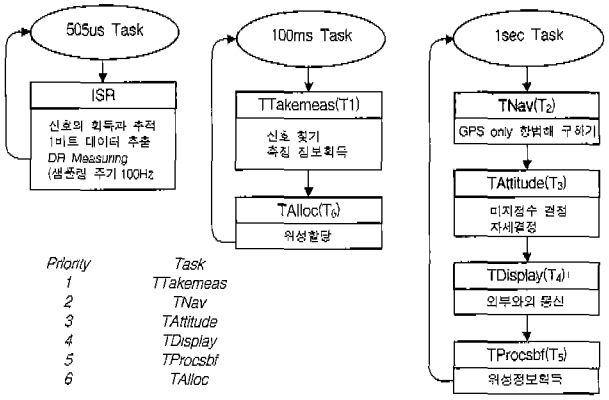
태스크명	역 할
TTakemeas 태스크	TTakemeas 태스크는 1틱(100msec)단위로 동작하며, carrier frequency search bin signal search process와 tracking channel로부터 raw measurement를 구함
TNav 태스크	TNav 태스크는 10틱(1sec)마다 수행되며, 측정치를 이용하여 위치, 속도를 구하고, 시계 모델을 결정함
TProcSbf 태스크	TprocSbf 태스크는 10틱(1sec) 주기로 동작하며 수신된 data message를 해석하고 필요한 정보(ephemeris, almanac, ionospheric, UTC correction data)를 추출
TDisplay 태스크	Tdisplay 태스크는 10틱(1sec)주기로 동작하며 수신기의 상태를 사용자에게 UART 채널을 이용하여 아스키 기반의 정보로 보여줌
Talloc 태스크	Talloc 태스크는 1틱(100sec)주기로 동작하며 수신 채널에 동작 모드에 따라 위성을 할당해주며 또한 사용자의 명령을 처리함
Tattitude 태스크	Tattitude 태스크는 10틱(1sec)주기로 동작하며 위성으로부터 수신된 데이터를 바탕으로 자세측정 초기화 및 자세측정 , 미지정수의 결정함

4.2 실험 환경

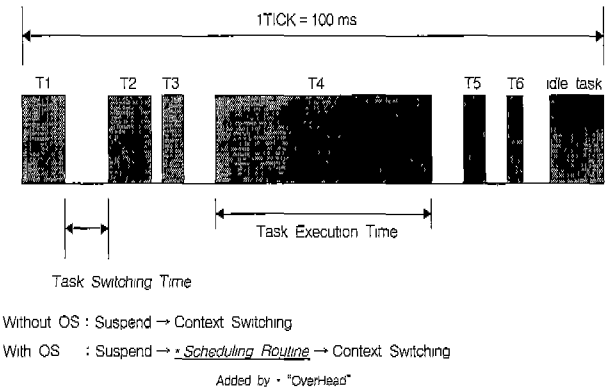
수행시간 측정을 위한 실험 환경은 개발 환경인 호스트 PC에서 GPS 항법 보드로 실시간 운영체제, 독자 운영체제 및 항법 소프트웨어 코드를 롬 에뮬레이터로 다운시켜 구동시켰다. 측정은 로직 어널라이져(logic analyzer)를 사용하여 usec 단위까지의 시간을 비교하였다.

4.3 태스크 수행 시간 비교

GPS 항법 컴퓨터 상에서 수행되는 태스크는 다음의 (그림 14)와 같이 구성되어 있다. 수행 시간 비교에서는 이러한 태스크들의 수행시간을 측정하여 독자 운영체제와 본 논문에서 구현된 실시간 운영체제 상에서의 결과를 비교하였다. (그림 15)는 태스크들의 실행과정을 보여준 것으로 오차 30ns의 로직 어널라이져를 사용하여 실시간 운영체제를 설치했을 때와 그렇지 않을 때의 태스크 스위칭 시간을 정리하였다.



(그림 14) GPS 태스크 구성

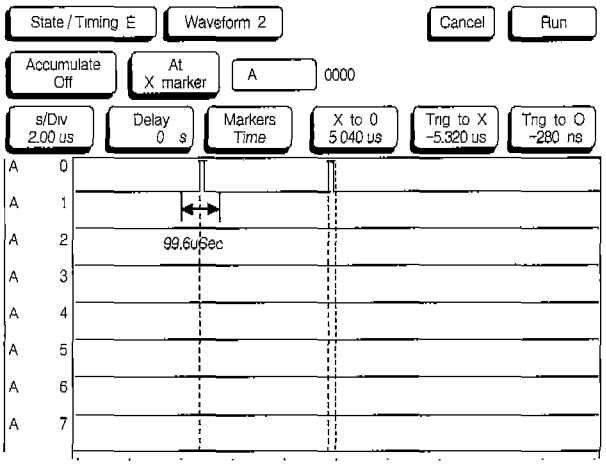


(그림 15) GPS 태스크 수행 측정

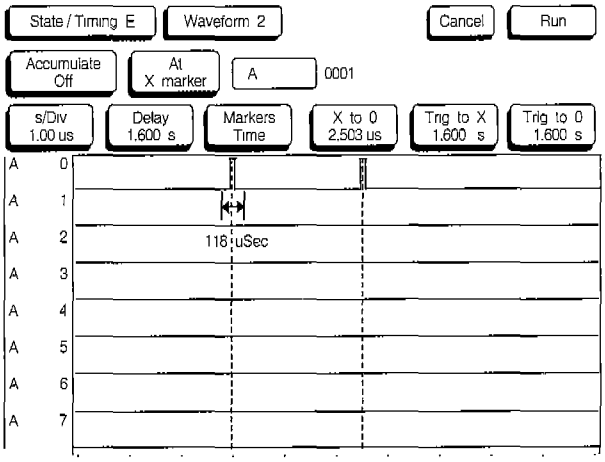
아래의 <표 4>는 수행시간 측정 결과를 도표로 정리한 것이다. 아래 (그림 16)과 (그림 17)은 태스크들중 TPROCSBF 태스크의 수행시간을 로직 어널라이져(logic analyzer)로 측정한 것이다. 두 그림을 비교해보면 실시간 운영체제를 사용하였을 경우 수행시간이 118usec로 수행 시간이 더 소요되었는데, 이는 실시간 운영체제 탑재에 따른 오버헤드 때문이다. 이는 독자(proprietary) 운영체제가 태스크의 순차적인 수행만을 제공하는 반해, 실시간 운영체제의 경우 태스크간의 동기화를 위한 ITC 제공, 실시간 스케쥴링, 인터럽트 처리 등 여러 커널 서비스를 제공하기 때문이다.

〈표 4〉 수행시간 측정 결과

테 스 크	수 행 시 간	
	Proprietary OS	RTOS
TPROCSBF	99.6 uSec	118 uSec
TATTITUDE	2.503 uSec	5.040 uSec
Tnav	288 uSec	288 uSec
TDISPLAY	47.88 mSec	76.40 mSec
TTAKEMEAS	282 uSec	282.3 uSec
TALLOC	335 uSec	348 uSEc



(그림 16) 실시간 운영체제 미사용 시 수행시간 측정



(그림 17) 실시간 운영체제 사용 시 수행시간 측정

TATTITUDE 태스크의 수행시간 비교에서 개발된 운영체제상의 수행이 독자 운영체제시 보다 더 큰 오버헤드를 갖는 이유는 커널 서비스인 세마포어의 사용과 운영체제의 커널 서비스를 사용하였기 때문이다. 또 그 이외에 태스크 스위칭 시에 약간의 차이를 보이는데, 이것은 태스크가 다른 태스크로

전이하면서 커널 서비스의 호출에 걸리는 시간과 코드 실행 후 복귀하는데 걸리는 지연시간 때문이다. 위에서의 결과를 보면 개발된 운영체제가 GPS 독자 운영체제보다 작게는 몇 uSec에서 많게는 몇 mSec까지 차이가 나는 것을 알 수 있다. 그러한 이유는 GPS 독자 운영체제에는 세마포어 같은 태스크 동기 서비스가 없기 때문에 실제 6개의 태스크들이 쓰고 읽어야 할 공유 데이터를 모두 전역 변수로 하고 어떤 태스크가 그 전역 변수를 사용할 경우 스케줄러를 잠그고 자료를 참조하게 된다. 이러한 방법은 태스크의 수가 늘어나고 프로그램이 복잡해질수록 운영체제의 선점성(preemption)에 치명적인 악영향을 미칠 수 있는 요인이 될 수 있다. 따라서 본 연구에서는 이러한 부분을 모두 세마포어를 이용하여 배타적(exclusive)으로 자료를 사용하면서도 선점성에 전혀 영향을 주지 않도록 하였다.

5. 결 론

본 논문에서 구현된 GPS 항법 컴퓨터를 위한 실시간 운영체제는 기본적으로 멀티태스킹을 위한 커널 구조와 태스크 관리 및 스케쥴링, 태스크간 통신 및 동기화 등의 서비스를 제공하고 있으며, 부가적인 기능으로 동적 메모리 관리, 디바이스 드라이버 인터페이스, 타이머, 향상된 인터럽트 처리, 시그널 등을 모듈별로 지원하고 있다. 본 논문의 목표는 특정 GPS 태스크들을 운용하는 항법 컴퓨터에 최적화된 실시간 운영체제를 설계하고 구현하는 것이었으나, 이식성을 높이기 위해 하드웨어에 의존적인 부분을 분리시키고 최소화시켜 타겟 하드웨어가 변해도 실시간 운영체제의 이식이 쉽도록 설계되어, 적은 메모리와 느린 CPU를 사용하는 소형 또는 중형 시스템에 사용하기 적합하다.

추후 진행될 연구에서는 성능 개선과 안정화에 대한 부분과 운영체제의 디버깅을 위한 기능추가와 네트웍, 파일 시스템 지원 등의 여러 디바이스를 지원하여 개발한 실시간 운영체제가 다양한 내장형 시스템들에서 사용할 수 있도록 하는 것이다.

참 고 문 헌

[1] David Wells, "Guide to GPS Positioning," Canadian GPS Associates, 1986.

[2] Khavar M. Zuberi, Padmanabhan Pillai, and Kang G. Shin, EMERALDS : a small-memory real-time microkernel, University of Michigan, 1999.

[3] M. Foss, G. J. Geier, "Integration of GPS with Other Sensors," Understanding GPS, Artech House, Inc., 1996.

[4] Zilog Inc., "Z16C30 USC User's Manual," 1997.

[5] C. M. Krishna, Kang G.shin, "Real Time Systems," The McTGraw Hill Companies INC., 1998.

[6] Douglas Comer, Timothy V.Fossum, "Operating System Design Vol.1 : The XINU APPROACH(PC EDITION)," Prentice Hall, 1998.

[7] Berny Goodheart, James Cox, "The Magic Garden Explained," Prentice Hall, 1994.

[8] Rick Grehan, Robert Moote, and Ingo Cyliax, "Real-Time Programming," Addison-Wesley, 1998.

[9] LUI SHA, RAGUNATHAN RAJKUMAR, JOHN P. LEHOCZKY, "Priority Inheritance Protocols : An Approach to Real-Time Synchronization," IEEE TRANSACTIONS ON COMPUTERS. Vol.39, September 1990.

[10] David E. Simon, "An Embedded Software Primer," Addison-Wesley, 1999.

배 장 식

e-mail : jsbae@cc.cnu.ac.kr
2000년 충남대학교 컴퓨터공학과 졸업(학사)
2000년~현재 충남대학교 컴퓨터공학과 석사과정
관심분야 : 실시간 시스템, 임베디드 시스템 등

송 대 기

e-mail : dksong@ce.cnu.ac.kr
1997년 충남대학교 컴퓨터공학과 졸업(학사)
1999년 충남대학교 컴퓨터공학과 졸업 (공학석사)
1999년~현재 충남대학교 컴퓨터공학과 박사과정
관심분야 : 고장허용 시스템, 실시간 시스템, 임베디드 시스템 등.

이 철 훈

e-mail : chlee@ce.cnu.ac.kr
1983년 서울대학교 전자공학과(학사)
1983년~1986년 삼성전자 컴퓨터개발실 연구원
1988년 한국과학기술원 전기및전자공학과 (공학석사)
1992년 한국과학기술원 전기및전자공학과 (공학박사)
1992년~1994년 삼성전자 컴퓨터사업부 선임연구원
1994년~1995년 Univ. of Michigan 객원연구원
1995년~현재 충남대학교 컴퓨터공학과 부교수
관심분야 : 운영체제, 병렬처리, 결함 허용 및 실시간 시스템 등.

송 호 준

e-mail : hjsong@coremagic.co.kr
1985년 서울대학교 제어계측공학과(학사)
1988년 한국과학기술원 전기및전자공학과 (공학석사)
1993년 한국과학기술원 전기및전자공학과 (공학박사)
1993년~1995년 현대전자 선임연구원
1995년~2000년 충남대학교 전자공학과 조교수
2000년~현재 (주)보아매직 부사장
관심분야 : 반도체 회로 설계 등