GPS 향법 컴퓨터를 위한 실시간 운영체제의 설계 및 구현

배 장 식†·송 대 기†이 젤 문**·송 호 준***

요 약

GPS(Global Positioning System)는 시간, 기상 상태에 관계없이 지구 전역에서 사용 가능한 가장 이상적인 헬로 시스템이다. GPS는 현재 진행, 속행, 귀환, 통신과 지능 할당 등 다양한 분야에서 활용되고 있으며, 앞으로도 사용량이 더욱 커질 것으로 보인다. 본 논문은 GPS와 관련 헬로 시스템(INS : Inertial Navigation System)을 협업 구성한 수신포트의 향법 컴퓨터 부터 통제하는 실시간 운영체제에 대한 설계 및 구현에 관한 것이다. 실시간 운영체제는 향법 컴퓨터 부에서 GPS의 송신 데이터(task)를 수신하게 된다. 개발된 실시간 운영체제는 GPS 수신기 수하에 최적화된 운영을 제공하도록 하였다. 사용자가 하루간 고도와의 상관없이 GPS 용 프로그램을 개발할 수 있도록 한다. 본 운영체제는 향법 컴퓨터를 수신 순위 기반으로 처리하는 전략(reactive) 스케줄링 방식을 적용한 운영체제이며, 프로그램, 데이터, 하드웨어, 하드웨어, 하드웨어를 분산으로 구성되어 있다. 본 논문은 GPS/INS 협업 보드 시스템의 구조, 운영체제의 구조, 개발 환경, 실시간 운영체제의 성능 평가에 대한 내용을 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 base platform to develop GPS application programs. The key facilities supplied by the RTOS developed in this paper are priority-based preemptive scheduling policy, dynamic memory management, intelligent interrupt handling, timers and IFC, 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), 위치 향법 시스템(Global Positioning System)

1. 서 론

GPS는 무국경적에서 군사적인 목적으로 만든 향법 시스템으로, 시간, 기상 조건에 관계없이 지구상에 있는 항체의 위치, 속도, 시간 등을 알아낼 수 있다. GPS 수신기는 군사용 정밀 측위, 농업, 헬기, 착륙, 항공기, 환경, 통신 등 위치 측정 및 시각 정보 등을 복합적으로 사용하고 있으며, 빌리 향법 시스템과 결합되어 자세 제어 등 여러 분야에서 쓰이게 되어 가고 있다. 일반의 GPS 수신기 운영 프로그램은 실시간으로 인공위성으로부터 수신되어지는 위성 데 이터를, GPS 향법, 자세 측정, 통합 향법, 위성 추적을 수행하는 향법 컴퓨터로 구성되어 있는데, 이러한 GPS 향법 컴퓨터는 시간에 따른 작업을 수행함으로 그 결과에서 송신 데이터를 처리하는 실시간 컴퓨터로 하여야 한다. 또한 경보 GPS 수신기의 높은 신뢰도가 요구되고, 사용 방법이 필요하여 GPS 수신기의 송신 알고리즘도 다양해지면서 이를 위한 운영체제가 필요하게 되었다. 이 운영체제는 일반적인 운영체제와는 달리 GPS 수신기에서 작업하며, 이러한 인증성으로부터의 메시지를 실시간으로 처리할 수 있어야 하며 시간에 따른 실시간 코어가 사용되어야 한다. 많은 범용화의 구성을 실시간 운영체제가 존재하지만 실제 GPS 수신기에 필요 없는 기능들로 인한 오버헤드로 상호상의 문제를ッ기 때문에 시스템에 적합한 실시간 운영체제를 특화적으로 설계하고 구현하였다. 실시간 운영체제는 그 특성상 MS-DOS, Windows98, UNIX, LINUX 등과 같이 컴퓨터에
쓰이는 발용 운영체계와는 달리, 시스템이 예상치 못한 특 정 이벤트가 발생하는 약 30% 정도에서 태스크 수행의 메무리 인을 초과하지 않도록 시간적 측면의 결정성(observationalism)을 보장하는 안정된 스케줄링 기능을 갖춘 운영체계이다.

타겟(Target) 시스템의 최적화된 실시간 운영체계를 설계하기 위해서는 하드웨어의 구조적 특성과 응용프로그램의 성 글을 고려해야 한다. 초기 실시간 운영체계를 설계할 때에는 인 터럽트 처리, 커널 서비스 함수, 태스크 스위칭(task switching)과 관련된 지연(latency)을 측정하였으며, 오늘날에는 마 이크로프로세서가 발전하면서 지연 최소화의 중요성이 점차 압도적으로 임해. 이러한 요구보다 더 다양한 면에서 성능을 비교할 수 있는 기준이 필요하다. 기본적인 요구사항으 로 성능을 저하시키지 않고 메모리를 많이 요구하지 않도록 코드가 간결해야 한다. 또한 사용자의 권익성을 고려하여 적절한 시스템 서비스 호출(system service call)을 제공하고, 태스크 스케줄링 훈련에서도 다양한 응용프로그램 요구에 유연하게 적용할 수 있도록 설계해야 한다[2,5].

본 논문에서 구현된 실시간 운영체계는 앞서 언급한 고려 사항들 중에, 운영체계로서 갖추어야 할 다양한 기능을 지원 하면서도 각 기능들 를 모듈별로 설계하여 최소의 메모리를 갖도록 하였다. 그리고 실시간 운영체계의 주요 특성인 결정 성을 보장하기 위해 인터럽트가 disable 되는 구간을 최소화 시켜 인터럽트 오염시간을 줄였다. 또한 실시간 운영체계 상에서 동작하는 응용프로그램들의 메모리공유와 공유권자의 안 정적 접근을 위해 통신 및 동기화 도구에 대해 품질적으로 검증하였다.

본 연구는 GPS 수신기에 적합한 실시간 운영체계를 개발 하는 것으로, GPS 수신기 응용 프로그램의 실시간 특성을 분 석하고, 수신기의 하드웨어적인 면을 고려하여 최적화된 운영 체계를 설계하고 구현하는 것을 목표로 하고 있다.

본 논문은 5개의 장으로 구성되어 있다. 2장에서는 실험 대상이 되는 GPS/INS 통합 보드 시스템의 구조와 특성을 언급하고, 3장에서는 실시간 운영체계의 설계 사례에 관한 사항을 요약한 내용을 기술한다. 그리고 4장에서는 설계 및 개발환경과 실시간 운영체계의 성능 평가를 보인다. 마지막으로 5장에서 이 논문의 결론을 끝낸다.

2. GPS/INS 통합 보드 시스템 구조

본 논문에 대상이 되는 GPS 수신기 보드의 전체 하드웨 어 구조는 크게 자세 측정용 GPS 수신기로 구성되는 셀러 부와 항법 컴퓨터부, 그리고 프로그램 개발을 위한 보드 컴퓨터로 나눌 수 있다. GPS 수신기는 4 안테나 4채널의 자세 측정용 GPS 수신기를 사용하였으며 항법 컴퓨터의 프 로세서는 통합 시스템의 많은 데이터를 처리하기 위해 최 신의 RISC CPU인 Intel 사의 StrongARM(SA-1100)을 사용하였다. 아래의 (그림 1)은 본 연구에서 대상 된 GPS/INS 통합 항법 시스템 보드의 구조를 보이고 있다.
전송이 필요하고 IMU부가 SDLC 프로토콜을 사용하기 때문에 Zilog 사의 Z16C30 USC를 사용하여 인터페이스를 설계하였다[3, 4]. 이 시스템 보트는 중남대학교 전자공학과과 재학생 구실에서 개발된 것이다. 각 부 시스템의 간략한 특징을 표 1에 나타내었다.

3. 실시간 운영체제 설계 및 구현

3.1 설계 및 구현의 고려사항
GPS/INS통합 시스템은 항법보도, 하드웨어를 제어하기 위한 하드웨어 드라이버(device driver)로 구성된 BSP(Board Support Package) 그리고, 항법 알고리즘을 구현하기 위한 모듈화된 실시간 항법 소프트웨어로 구성된다. 항법 알고리즘을 구현하기 위한 모듈들은 다수의 데스크톱과 인터프리트 서비스를 의미(Interactive service)으로 구성되어, 각각의 데스크톱은 멀티태스크링 총합 하에서 수행된다.

실시간 운영체제 설계시 고려되어야 할 사항은 항법 컴퓨터 보드의 하드웨어 특성들이 항법 소프트웨어의 수행 특성이 있다. 하드웨어의 경우 항법 컴퓨터 보드가 Intel의 StrongARM에 기반한 인텔에서 IMU로부터 SDLC 프로토콜을 제어하기 위한 Zilog 패널을 사용하였다. 이와 관련하여 멀티프로세스(context switch)의 인터럽트 처리와 같은 운영체제의 시스템 의존적인 부분들을 StrongARM에 맞게 수정했으며, Zilog 칩의 경우 구동을 위한 BSP 코드를 작성하였다. 그리고 항법 소프트웨어의 경우, 멀티태스크 시 동응용의 효율적인 관리 및 데스크톱의 주기성을 조정하기 위한 실시간 스케줄링 방향을 고려하였다. 또한 항법 컴퓨터에서 GPS 테스토에 TTC(Inter-Task Communication) 구성을 제공하여 데스크톱 통신을 가능하게 하고 신호널(semaphore), 뮤태кс(mutex)을 사용하여 동응용의 효율적인 관리를 가능하게 하였으며, 각각의 데스크톱의 실행에 관련된 주기를 설정하기 위해 멀티태스크의 처리가 가능하게 하였다.

3.2 실시간 운영체제의 구조 및 기본 서비스 구현
3.2.1 커널 및 테스크 구조
GPS 항법 컴퓨터에서 사용되는 각각의 테스크는 타겟의 RAM 크기와 운영프로그램에서 요구하는 크기에 따라 스크립팅 동적적으로 지정된다. 테스크 코드를 제어하기 위한 커널의 구조 TCB(Task Control Block)이며, 각각의 TCB는 그림 3과 같은 OS_TASK_HEAD 구조체에 의해 관리된다. 이는 실시간 운영체제가 초기화 될 때 미리 정의되어야 하는 부분으로, OSTCBHeader는 생성된 데스크의 정보를 가지고 있는 TCB를 가리키는 포인터로 사용하고 OSTCBNum이 해당 우선순위에 존재하는 데스크의 개수를 나타내기 위해 사용된다. 생성된 각 데스크의 정보를 저장하고 있는 TCBPtr은 그림 4와 같다.

- OSTaskStack
  데스크에 할당된 스택의 탑(top)을 가리키며, 각각의 데스크 크는 자신의 돌릴 수 있도록 가지고 있다. 테스크 생성시 업무 프로그램 작성자의 요구에 따라 크기를 설정할 수 있다. 스케줄링이나 인터프리트에 의한 멀티프로세스가 발생할 때 이 셀프러어 언어로 작성된 코드로 사용한다.

- OSTaskStat
  테스크의 현재 상태를 나타내며, 이것에 의하면 ready 상태로서 CPU에 의한 스케줄링 대상에 포함된다.

- OSTaskPrex
  테스크의 우선순위를 저장하며, 숫자가 낮을수록 더 높은 우선 순위를 갖는다.

- OSTaskDly
  테스크가 지연(延期) 또는 특정 이벤트에 대해서 청담(pending)될 때 얼마나동안의 틀림없는 시동시간을 가질 것인가를 나타낸다.
- OSTaskX(Y), OSTaskBitX(Y)
  테스크를 ready 상태에서 따르게 스케줄링하기 위해 사용
  된다. 이 필드의 값들은 테스크가 생성되거나 삭제되었을 때 또는 테스크의 우선순위가 변경되었을 경우 (그림 5)
  와 같이 계산되어 동록된다.

  

<table>
<thead>
<tr>
<th>필드</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSTaskX</td>
<td>priority &amp; 0x07</td>
</tr>
<tr>
<td>OSTaskY</td>
<td>OSMapTbl[priority &amp; 0x07]</td>
</tr>
<tr>
<td>OSTaskBitX</td>
<td>priority &gt;&gt; 3</td>
</tr>
<tr>
<td>OSTaskBitY</td>
<td>OSMapTbl[priority &gt;&gt; 3]</td>
</tr>
</tbody>
</table>

(그림 5) TCB의 우선순위 등록

- OSTimeQuantum
  현재 테스크의 타임 슬롯(time slot) 값이며, 같은 우선순위의 테스크들은 라운드 로빈(round robin) 시스템에 사용
 된다. 정밀 변수로 정의된 원형(quantum) 값이 테스크 생성시 부여되며 OSTMtimeTick( ) 함수가 호출될 때마다 현재
  테스크의 OSTimeQuantum 값이 1씩 감소된다. 만약 이것이 되었다면, 같은 우선순위의 테스크가 CPU를 정지한다.

- OSTaskEventPtr
  ECB(Event Control Block) 목록 가리키는 포인터이다.

- OSTaskNext, OSTaskPrev
  OSM_TCB를 이용 연결 리스트로 만드는데 사용한다. 이 리스트는 하드웨어 타입의 메시지 브로커의 각 테스크의 OSTaskBit
  값을 기반으로 하는 OSTMtimeTick( ) 함수에 의해 사용된다.

- OSPrioListNext, OSPrioListPrev
  리스트에 있을 때 같은 우선순위의 TCB들을 이용 연결 리스트로 구성하거나 특정 이벤트를 기다리며 행동 상태에서 있는 TCB
  들을 suspend 리스트에 우선순위대로 연결한다.

- MemSuspend
  동적 메모리 할당에 의해 메모리 할당부를 경유할 경우, 더 이상 동적 메모리가 없게 되면 메모리가 필요한 테스크들
  을 중단되어야 한다. 이 때 중단되는 테스크들을 단일 연결 리스트로 관리하게 되며, MemSuspend는 이를 가리키는
  포인터로 쓰인다.

- Signal Handling을 위한 Optional Field
  시그널 처리를 위한 추가 필드로서 차후에 다시 연결할 것

생성된 테스크를 삭제하기 위해서는 응용프로그램에서 각 테스크의 TCB를 정지하고, TCB의 주소로서 테스크를 삭제
한다. 전형적으로 컨트롤러는 항상 최상위 우선순위의 TCB가 CPU
를 정지하여 자신의 작업을 수행한다. 이때에 사용되는 ECB
우선, 응용 프로그램이 수행되기 전에 정의해야 한다. (그
림 6)은 테스크들에 상태 체널을 나타낸다.

(그림 6) 테스크 상태 체널

3.2.2 문맥 교환
CPU는 여러 테스크나 ISR에 의해 공유되는 중요한 자원이므로 현재 CPU를 정지하기 위해 수행중인 프로그램은 자신의 우선순위보다 높은 테스크가 실행할 준비가 되거나, 외부 인터럽트에 대해 ISR를 수행시키기 위해 문맥교환을 일으킨다.
CPU를 정지하고 있던 낮은 우선순위의 테스크는 처리가 수행되던 작업을 중단하고, 그 시점의 PC, 즉 프로그램 코트량 (program counter)과 CPU의 탐지스토리지 손상 수행 환경을 로전문을 저장하고 CPU를 정지하게 된 테스크나 ISR의 로전문에서 새로운 CPU 환경을 읽어서 수행을 계속하게 된다. 아래의 (그
림 7)은 두 개의 프로세스가 문맥 교환을 일으킨 직후에 각 테
스크의 로전문 상태와 CPU의 문맥이 어떻게 변화하는지를 나
타내고 있다.

3.2.3 스케줄링(Scheduling)
본 논문에서 구현된 스케줄링(scheduler)는 라운드 로빈 알고리즘을 지원하며, 최초 테스크 생성시 부여되는 우선순위를
거진다. 이때 할당량을 이용한 바이트(byte) 연산을 통해 양
스케줄러가 높은 우선순위를 갖는다.

\[ y = \text{OSUnMapTbl}(\text{OSRdyGrp}) \]  
\[ x = \text{OSUnMapTbl}(\text{OSRdyTbl}[y]) \]  
\[ p = (y \ll 3) + x \]

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

3.2.4 임계 영역과 상호작용

임계영역 코드는 반드시 원자 연산(atomic operation)으로 처리되어야 하며, 일반 코드가 실행되던 수행 중에 인터럽트 등의 요인에 의해 중단될 수 없다. 이를 보장하기 위해 특별히 임계영역의 코드가 실행되기 전에 인터럽트가 disable 되어야 하며, 임계영역의 코드 수행이 완료되었을 때 다시 인터럽트를 enable 되어진다. 이와 유사하게 여러 데스크들에 의해 공유되는 자원이나 데이터 구조에 대해서도 데이터의 촉촉(corruption)을 방지하기 위해 버티컬한 접근을 제공해야 한다. 이를 보장하기 위한 방법으로 Motorola의 68000 패밀리 프로세서들 중 일부는 TASTest-And-Set라는
영향을 사용하거나, 세마포어 도구를 사용하거나, 인터럽트를 disable/enable 하는 방식을 사용한다. 이들 중에 인터럽트의 disable/enable는 보통 릴리스 연산 제공하는 CPU의 레지스터 셋에서 인터럽트의 수용여부를 상태 레지스터(SR)의 특정 비트로 셋 또는 런트 함으로 구현할 수 있다. 이러한 방법은 인터럽트 자체를 막는 것이 아니라 발생된 예외 상황을 CPU가 무시하도록 하는 것이므로 인터럽트 셋 레지스터 내에 그 정보가 남아있지 않게 처리되는 것이다.

3.2.5 태스크간 동기화와 통신

본 논문에서 구현한 실시간 운영체계는 태스크들간의 동기를 위해서 세마포어를 제공하며, 태스크들간의 통신을 위해서 메시지 메일리스, 메시지 큐를 제공한다. 또한 운영체계의 결정성을 보장하기 위해 흔히, 뮬렉스-(mutex)라고 불리는 mutual exclusive 이란 세마포어(binary semaphore)를 제공하여 우선 순위 역전 시간을 최소화한다. 이러한 이벤트와 관련된 서비스를 지원하기 위해 (그림 9)와 같은 ECB(Event Control Block) 구조가 사용된다.

세마포어

세마포어는 공유자원을 배타적으로 사용하거나 태스크간 동기를 맞춤 때 사용한다. 이것은 사용자는 시스템에 따라 정수 범위의 값을 가지고 조작한다. 만약 32비트 시스템의 경우 세마포어는 0에서 232사이의 값으로 조작되며, 0이 되면 더 이상 사용 가능한 자원이 없음을 나타내고, 0이상일 경우 양수일반의 자원이 가용하다는 것을 나타낸다. 이로는 동시에 관련된 여러 메커니즘에서 제공하는 서비스를 정리한 것이다.

3.2.6 메모리 관리

본 논문에서 구현한 실시간 운영체계는 메모리 할당 및 해제가 동적으로 이루어진다. 이는 운영체계를 사용하는 응용 프로그램에서 수행토중 운영체계 초기화 과정에서 지정한 메모리 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 관리에는 전역적인 지도 구조는 (그림 10)과 (그림 11)과 같다. 풀의 개수는 덤프파일에 정의되어있는 MAX_POOL_NUM만큼 생성하여 사용할 수 있으며, 이러한 풀은 관리하기 위해 구조로 PoolTable(MAX_POOL_NUM)이 생성되며 기본값으로 100을 사용하고 있다. 그림에서 FreePCBList는 사용 가능한 풀들을 연결하는 해더로 사무되고, FreePCBNum은 사용 가능한 풀의 개수를 나타낸다. 메모리 할당과 해제의 단계는 char이며, 최초 응용프로그램에서 지정한 메모리 배열 공간에서 이뤄진다.

(표 2) 통신 기능을 제공하는 서비스

<table>
<thead>
<tr>
<th>통신 메커니즘</th>
<th>함수 형식</th>
<th>용도</th>
</tr>
</thead>
<tbody>
<tr>
<td>세마포어</td>
<td>OSSemOpen(OS_EVENT = printf, int value)</td>
<td>통신을 수행하기 위해서는 테스크가 세마포어 사용할 수 있도록 ECB 주소와 세마포어의 최초 개수를 정의하기 위하여</td>
</tr>
<tr>
<td></td>
<td>OSSemGet(OS_EVENT = printf, uint timeout, uint *err)</td>
<td>세마포어의 값이 0보다 크며 세마포어의 값을 하나 감소시키고 호출된 개수로 표기</td>
</tr>
<tr>
<td></td>
<td>OSSemRelease(OS_EVENT = printf)</td>
<td>다른 테스크 또는 ISR에 의한 사용을 가진 세마포어의 번호가 이어지는 이벤트값 방식</td>
</tr>
<tr>
<td>메시지 메일리스</td>
<td>OSMboxMake(OS_EVENT = printf, void = message)</td>
<td>ECB의 원격 메시지의 메세지 텍스트를 받아 메일리스를 위한 ECB를 생성</td>
</tr>
<tr>
<td></td>
<td>OSMboxReceive(OS_EVENT = printf, uint timeout, uint *err)</td>
<td>메일리스에 메시지가 들어오면, 오호출된 테스크에 메시지를 전달하면서 메일리스를 비움</td>
</tr>
<tr>
<td></td>
<td>OSMboxSend(OS_EVENT = printf, void = message)</td>
<td>메일리스에 메시지 전달</td>
</tr>
<tr>
<td>메시지 큐</td>
<td>OMQMake(OS_EVENT = printf, OS_Q * pq, void = *start, uint size)</td>
<td>ECB의 원격, QCB(Queue Control Block)의 원격이, 매 네트워크 메시지가 담길 수 있는 시작번호와 크기를 받아 담아 있는 ECB의 초기화</td>
</tr>
<tr>
<td></td>
<td>OMQReceive(OS_EVENT = printf, uint timeout, uint *err)</td>
<td>메시지 큐 안에 메시지를 담은 메지지가 들어올 때, 큐의 OQMExit으로부터 메시지가 들어온 테스크 전달</td>
</tr>
<tr>
<td></td>
<td>OMSend(OS_EVENT = printf, void = message)</td>
<td>큐에 더 이상 메시지가 남아있지 않음 때 메시지를 발행하고, 메시지를 담은 공간이 있는 메시지를 큐에 남음</td>
</tr>
</tbody>
</table>
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 레이어로 이를 제어하고 사용하기 위해서 복잡한 소프트웨어를 요구한다. 운영체제는 디바이스 드라이버라고 불리는 프로그램을 통해 디바이스에 데이터를 전송하고 제어하여 하드웨어에 접근함으로써 복잡하고 세세한 동작 사항들을 사용자에게 맡길 수 있다. 둘째, 디바이스들은 시스템에서 사용되는 공간 자원으로, 운영체제에 의해 안전하고 공평하게 접근할 수 있는 메모리공간을 제공할 수 있다. 셋째, 운영체제는 사용자가 하드웨어의 세부 설정에 대한 지식이 없어도 상위 수준 프로그래밍 언어를 통해 키널에서 동작하는 디바이스 이룹을 가지고 일반적으로 디바이스를 사용할 수 있는 유연한 인터페이스를 제공한다. 또는 문제로 구현된 운영체제의 디바이스 드라이버 구조는 (그림 12)와 같다. 실제 하드웨어들에 접근하고 사용하기 위한 디바이스 패브로 코드들이 운영체계가 제공하는 DDI에서 CreateDevice() 테이블 내의 정보를 참조하여 각각 동작된다. 응용 프로그램 작성자의 측면에서 설계 하드웨어에 대한 복잡한 코드들은 간직하고 실제 테스크에서 사용될 디바이스에 접근할 때는 동일한 함수 이름 (init, open, close, read, write, ioctl)으로 시스템 호출하여 사용할 수 있는 유연한 인터페이스를 제공받게 되는 것이다.

3.2.8 시그널 처리 (Signal Handling)

시그널은 하나 이상의 테스크들에 비동기적인 이벤트를 알리기 위해 사용된다. 앞서 언급한 세마포어, 메일리스, 큐와 같은 테스크간의 통신 도구들은 메시지 또는 세마포어를 획득 할 때까지 테스크가 중단된 상태에서 특정 이벤트가 발생할 때, 또는 다른 이벤트가 발생할 때, 시그널에 의해 작업이 종료되거나, 상황이 변경될 때, 시그널이 발생한다. 시그널은 테스크의 종료, 이벤트의 발생, 특정 조건의 충족 등 다양한 상황에서 발생할 수 있으며, 시그널의 발생을 통해 시스템의 테스크들이 동기화되고, 종료되거나, 이벤트가 발생하는 등의 동작이 수행된다.
3.2.9 결정성을 보장하는 실시간 스케일링
결정성을 보장하는 데, 태스크의 작업수행이 정해진 시간 안에 모두 수행될 수 있음을 보장하는 것이 필요하다. 태스크들 간 주기에 따라 수행을 해야하는 탱블리스 시스템에서는 이러한 결정성을 보장하는 것이 필수적이라 할 수 있다. 운영체제가 최상위 우선순위의 태스크를 디스패치하는 시기가 태스크의 수 혹은 이러한 다른 요인에 영향을 받게 된다면 실시간 응용체제에 중요한 특성인 결정성을 보장할 수 있게 된다. 위 3.2.3절의 내용과 같이, 본 논문에서 구현한 실시간 응용체제의 스케줄링은 태스크의 수에 관계없이 고정적인 시간에 최상위 우선순위의 태스크를 디스패치 할 수 있다. 그러므로 탱블리스 시스템의 태스크들은 정해진 주기에 따라 정확한 수행을 하게 된다.

4. 실시간 응용체제의 성능평가
본 연구의 성능 평가는 GPS 소프트웨어에서 만든 응용(proprietary) 응용체제와 태스크를 수행시간 비교하는 것으로 성능 평가를 하였다. 본 연구로 개발되어 있는 응용체제는 스케줄러, IPC, 설계, 셀파로디, 태스크 관리기능이 태스템에 민감하게 되는 것에 반하여, GPS 소프트웨어의 태스크는 단지 스케줄러 외에 다른 응용체제에선 기능은 없으므로 소프트웨어가 탐색해관계 내장형 체제(embedded system)에 비터나 오류가 더 변화하게 나타날 수 있다. 본 연구에서는 GPS 소프트웨어의 성능 및 요소변수의 문제와 본 연구를 통해 그것들이 어떻게 개선되어 있는지 설명한다.

4.1 항법 소프트웨어 태스크의 기능
GPS 항법 소프트웨어는 TTrack, TNav, TProcSbf, TDisplay, Talloc 그리고 T attitude 태스크로 구성된다. 각각의 태스크 역할은 표 3과 같다.

<table>
<thead>
<tr>
<th>표 3</th>
<th>GPS 항법 소프트웨어 구성 태스크</th>
</tr>
</thead>
<tbody>
<tr>
<td>이름</td>
<td>역 할</td>
</tr>
<tr>
<td>TTrack</td>
<td>태스크는 1회(100msec) 단위로 동작하며, carrier frequency 싸운 signal signal process와 tracking channel로부터 raw measurement을 구함</td>
</tr>
<tr>
<td>TNav</td>
<td>TNav 태스크는 1회(1sec) 단위로 동작하며, 셀프로디, 운동량, 위치를 제공하고, 제어 모델을 결정함</td>
</tr>
<tr>
<td>TProcSbf</td>
<td>TProcSbf 태스크는 1회(1sec) 단위로 동작하며, sensor data message를 해석하고, 필요로 하는 정보(ephemeris, almanac, ionospheric, UTC correction data)를 수집</td>
</tr>
<tr>
<td>TDisplay</td>
<td>TDisplay 태스크는 1회(10sec) 단위로 동작하며, sensor의 개체를 상호작용하여 화면에 표시함</td>
</tr>
<tr>
<td>Talloc</td>
<td>Talloc 태스크는 1회(1sec) 단위로 동작하며, sensor의 개체를 상호작용하여 화면에 표시함</td>
</tr>
<tr>
<td>T attitude</td>
<td>T attitude 태스크는 1회(1sec) 단위로 동작하며, sensor의 개체를 상호작용하여 화면에 표시함</td>
</tr>
</tbody>
</table>

(표 3) GPS 항법 소프트웨어 구성 태스크

(그림 12) 다이어그램의 구성
생하기를 기다려야만 하는 반면에, 대스크가 시그널을 통해 특정 서비스를 요청하면 시그널을 받은 대스크는 이벤트를 레그 변화에 관계없이 CPU를 절충하여 실행 권한을 주어지게 됨. 이 때 해당 시그널에 대한 처리 루틴을 수행한다. 즉, 수행 중인 대스크가 시그널 받으면 바로 처리가 가능하고, 다른 대스크가 CPU를 절충할 때 시그널을 받으면, ready상태에서 자신이 실행 권한이 오기만 기다리면 최우선 순위가 되어 실행 권한을 얻으면서 받았던 시그널을 처리할 수 있다. 이는 둔기로 이벤트 도구에서 담당 시그널이 비동기적 이벤트로 제공함을 의미하는 것이다. 이와 같은 특성 덕분에 시그널은 필요한 때 시그널을 처리하기 전에 다시 보내진 시그널을 부수한다. 즉, 대스크는 어떤 시그널을 받았는지 모를 임목할 뿐, 어떤 시그널이 얼마나 보내졌는지는 기억하지 못한다 [7]. 시그널 처리를 위한 TCB 내의 필드는 (그림 13)과 같다.

typedef struct_tcb { /* ---------------- Signal Handling Fields ---------------- */
    void *OSTaskSavetStack;
    uint OSTaskSavestat;
    uint OSTaskSignal;
    uint OSTaskEnableSignal;
    uint OSTaskActivatesignal;
    void (*signal_handler)(uint);
    struct_tcb *OSsuspendedNext;
    struct_tcb *OSsuspendedPrev;
} TCB;

(그림 13) 시그널 처리를 위한 TCB내의 필드
42 실험 환경

수행시간 측정을 위한 실험 환경은 개발 환경인 호스트 PC에서 GPS 헬름 보드로 실시간 운영체제, 독자 운영체제 및 헬험 소프트웨어 코드를 옮 애플레이타로 다음서기 구동 시켰다. 측정은 로직 어널라이저(logic analyzer)를 사용하여 usec 단위까지의 시간을 비교하였다.

4.3 태스크 수행 시간 비교

GPS 헬럼 컴퓨터 산에서 수행되는 태스크는 다음의 (그림 14)와 같이 구성되어 있다. 수행 시간 비교에서는 이러한 태스크들의 수행시간을 측정하여 독자 운영체제와 본 논문에서 구현한 실시간 운영체제 상에서의 결과를 비교하였다. (그림 15)는 태스크들의 실행정도를 보여준 것으로 오차 30ns의 로직 어널라이저를 사용하여 실험 운영체제를 설치했을 때 외 그렇지 않은 때의 태스크 수행시간을 정리하였다.

아래의 <표 4>는 수행시간 측정 결과를 도표로 정리한 것이 다. 아래 (그림 16)과 (그림 17)은 태스크들 중 TPROCBSF, TATTITUDE의 수행시간을 로직 어널라이저(logic analyzer)로 측정한 것이다. 두 그림을 비교해 보면 실시간 운영체제를 사용 했을 경우 수행시간이 118usec로 수행 시간이 더 소요되었는데, 이는 실시간 운영체제 밑에 다른 오버헤드 때문이었다. 이러한 독자(proprietary) 운영체제가 태스크의 순차적인 수행

만을 제공하는 반면, 실시간 운영체제의 경우 태스크간의 동

기화를 위한 ITC 제공, 실시간 스케줄링, 인터럽트 처리 등

여러 시스템 서비스를 제공하기 때문이다.

>表 4) 수행시간 측정 결과

<table>
<thead>
<tr>
<th>태스크</th>
<th>수행 시간</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPROCBSF</td>
<td>99.6 usec</td>
</tr>
<tr>
<td>TATTITUDE</td>
<td>2,913 usec</td>
</tr>
<tr>
<td>Tnuav</td>
<td>228 usec</td>
</tr>
<tr>
<td>TDISPLAY</td>
<td>47.848 usec</td>
</tr>
<tr>
<td>TTAKEMEAS</td>
<td>282 usec</td>
</tr>
<tr>
<td>TALLOC</td>
<td>335 usec</td>
</tr>
</tbody>
</table>

(그림 16) 실시간 운영체제 미사용 시 수행시간 측정

(그림 17) 실시간 운영체제 사용 시 수행시간 측정

TATTITUDE 태스크의 수행시간 비교에서 개발된 운영체제 많은 수행이 독자 운영체제보다 다른 오버헤드를 갖는 이유는 커널 서비스인 셀프 포어의 사용과 운영체제 커널 서비스를 사용하였기 때문이다. 또 그 외에도 태스크 스위칭 시에 약간의 차이를 보이며, 이것은 태스크가 다른 태스크로

ToK = 100 ms

Task Switching Time
Without OS: Suspend → Context Switching
With OS: Suspend → Swapping Process → Context Switching
 Added by “Overhead”

(그림 15) GPS 태스크 수행 측정
전이하면서 커널 서비스의 호출에 걸리는 시간과 로드 실행 후 복귀하는데 걸리는 시간 사이의 대조가 있다. 위의 결과를 보면 개발된 운영체계가 GPS 독자 운영체계보다 작게는 10μsec에서 밝게는 몇 μsec까지 차이가 나는 것을 알 수 있다. 이러한 이유는 GPS 독자 운영체계에는 새롭고, 새로운 태스크 동기 서비스가 없기 때문에 실제 접근 Orders/�� tarafından 전용 태스크가 그 전용 변수를 사용할 경우 스케줄링을 참고하고 자료를 참조하게 된다. 이러한 방법은 태스크의 수가 늘어나고 프로세스가 복잡해질수록 운영체계의 성능(preemption)에 영향을 미칠 수 있는 요인이 될 수 있다. 따라서 본 연구에서는 이러한 부분을 모두 세밀히 이용하여 태스크(exclusive)으로 자료를 사용하면서도 성능에 전반 영향을 주지 않도록 하였다.

5. 결 론

본 논문에서 구현한 GPS 항법 컴퓨터를 위한 실시간 운영체계는 기본적으로 달리티어링을 위한 커널 구조와 태스크 관리 및 스케줄링, 태스크간 통신 및 동기화 등의 서비스를 제공하고 있으며, 부가적인 기능으로 동적 메모리 관리, 마이크로드로 이터 네트워크, 타이어, 향상된 인터럽트 처리, 시그널 등을 모두 정리하여 제공한다. 본 논문의 목표는 특화 GPS 태스크들을 운용하는 항법 컴퓨터에 적합한 실시간 운영체계를 설계하고 구현하는 것이었으니, 이전에 높아지기 위해 하드웨어와 의존적인 부품을 분리시켜 최소화시켜 타겟 하드웨어가 변해도 실시간 운영체계의 이전을 실현할 수 있게 하여, 적은 흐름과 느린 CPU를 사용하는 소형 또는 중형 시스템에 사용하기 적합하다.

추후 진행될 연구에서는 성능 개선과 안정성에 대한 부분과 운영체계의 디버그를 위한 기능추가와 네트워크, 태스크 시스템 지원 등의 여러 하드웨어를 지원하여 개발한 실시간 운영체계가 다양한 내장형 시스템들에 사용할 수 있도록 하는 것이다.

참고 문헌


배 장 식
e-mail: jabe@occm.ac.kr
2000년 졸업 및 컴퓨터공학과 졸업(학사)
2000년 - 현재 졸업 및 컴퓨터공학과 화학

송 대 기
e-mail: dssong@occm.ac.kr
1997년 졸업 및 컴퓨터공학과 졸업(학사)
1999년 졸업 및 컴퓨터공학과 졸업(학사)
1999년 - 현재 졸업 및 컴퓨터공학과 화학

참임부: 장학부, 실시간 시스템, 병렬시스템 등

이 철 봉
e-mail: chle@occm.ac.kr
1983년 서울대학교 컴퓨터공학과 졸업(학사)
1983년 - 1985년 삼성전자 컴퓨터개발연구원
1988년 한국과학기술원 전기전자공학과 (공학석사)
1992년 한국과학기술원 전기전자공학과 (공학박사)
1992년 - 1994년 삼성전자 컴퓨터사업부 실장연구원
1994년 - 1995년 Univ. of Michigan 전문연구원
1995년 - 현재 졸업 및 컴퓨터공학과 부교수

참임부: 운영체계, 병렬처리, 컴퓨터 하드웨어 및 실시간 시스템 등

송 호 준
e-mail: hjsong@coroemic.co.kr
1985년 서울대학교 컴퓨터공학과 졸업(학사)
1988년 한국과학기술원 전기전자공학과 (공학석사)
1993년 한국과학기술원 전기전자공학과 (공학박사)
1993년 - 1995년 한국전자 신학연구원
1995년 - 2000년 졸업 및 컴퓨터공학과 조교수
2000년 - 현재 (주)아이메직 부사장

참임부: 반도체 회로 설계 등