GP-COMPASS/DR 항법 소프트웨어를 위한 실시간 운영체제의 설계 및 구현*

°편현범, 이재호, 이철훈
충남대학교 컴퓨터공학과

Design and Implementation of A Real-Time Operating System for the GP-COMPASS/DR Navigation Software

Hyun-Bum Pyun°, Jae-Ho Lee, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
{hbpyun, jhlee,chlee}@comeng.chungnam.ac.kr

요 약 문

본 논문에서는 GPS(Global Positioning System)와 추측 항법 시스템인 DR(Dead Reckoning)을 혼합 구성한 수신보드의 운영을 위한 Embedded 운영체제를 설계하고 구현 하였다. 이 운영체제는 실시간으로 인공위성으로부터 수신되어지는 Raw Measurement 획득, GPS 항법, 자세 결정, 통합항법, 위성 추적을 수행하는 태스크들을 우선순위 기반으로 처리하는 선점형(Preemptive) 스케줄링 방식을 채택한 실시간 운영체제 이다. 본 논문에서는 자세 결정용 GPS 와 DR 센서를 이용한 통합시스템보드를 위한 실시간 운영체제의 개발 환경, 운영체제의 구조와 개발 내용에 대해 언급하였다.

1. 서론

GPS 는 미 국방성에서 군사적인 목적으로 만든 항법 시스템으로, 시간,기상 조건에 관계없이 지구상에 있는 항체의 위치, 속도, 시간 등을 알아낼 수 있다. GPS 를 단독으로 사용할 경우 위성 신호가 단절되면 연속적으로 항법 해를 계산할 수 없다. 추측 항법 시스템인 DR 은 차량의 속도와 회전각을 알아내 차량의 상대적인 위치를 알아낼 수 있는 자립형 항법 장치이다. 그러나 시간이 지날수록 오차가 누적되는데 GPS 와 DR 을 함께 사용하면 오차가 누적되는 것을 막을 수 있다[1]. 특히 GPS 의 태스크들은 시간에 밀접한 작업을 수행하므로 그들 간의 스케줄링을 위해서는 실시간 운영체제를 필요로 하게 된다.

일반적으로 실시간 운영체제라는 것은 주어진 시간 내에 요구한 작업을 처리할 수 있도록 시스템의 리소스를 분배하는 역할을 하는 운영 체제라 할 수 있다. 실시간 태스크의 특징으로는 인터럽트에 대한 응답시간이나 스케줄링에서 많은 변동폭을 갖지 않는다는 것과 빠른 응답 시간이다.

본 논문은 서로간의 단점을 보완한 자세 결정용 GPS 와 DR 을 이용한 통합시스템을 위한 실시간 운영체제를 설계 및 포팅한 결과를 기반으로 한다.

2 장에서 자세 결정용 GPS 와 DR 을 이용한 시스템구성 3 장에서는 실시간 운영체제의 구현 내용을 기술 하였다. 그리고 4 장에서는 결론 및 향후 연구 과제를 기술 하였다.

2. 자세 결정용 GPS 와 DR 을 이용한 통합시스템 구성

그림 1 에서 보는 바와 같이 시스템은 자세 결정용 GPS 부분과 DR 부분으로 나뉘어 진다. 자세 결정용 GPS 는 크게

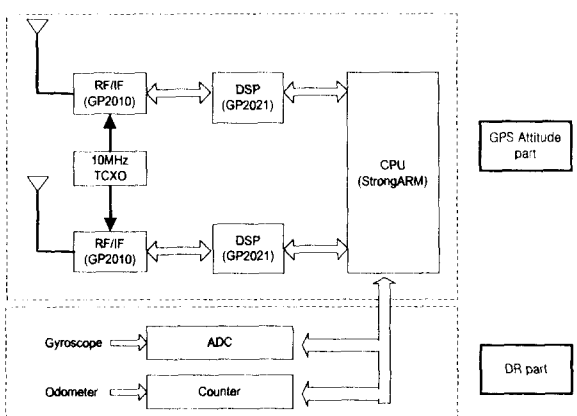

그림 1. GPS/DR 통합 시스템의 구성

안테나, 수신된 신호를 상관기부로 전달하는 RF 초단부,

12 개의 채널을 갖는 상관기부(GP2021), 상관기에서 생성된 값을 획득하여 GPS 항법 해와 자세결정 통합항법을 수행하는 중앙 처리부로 구성된다. CPU 는 통합 항법과 자세 결정할 때 속도를 빠르게 하기위해 StrongARM 을 사용한다.

DR 부분은 차속 센서인 Odometer 와 차량의 회전각을 측정하는 Gyroscope 로 구성된다[1][6].

참고로 이 시스템 보드는 충남대학교 전자공학과 제어연구실에서 설계 개발한 것이다.

3. 운영체제 설계 및 구현

3.1 구현 환경

구현 환경은 PC 를 사용하여 운영체제를 구현한후 ARM 사의 컴파일러(v 2.11)를 사용하여 이미지를 만든 후 롬라이터를 사용하여 직접 롬에 프로그래밍하여 수행시켰다. 다운로드 및 디버깅을 위한 포트가 없기 때문에 디버깅시 많은 노력이 필요하였다. 또한 실행 결과를 보기 위해서 winmon 을 사용하였다.

일단 보드에 전원이 들어오게 되면 ARM 은 ROM 이 있는 0 번지를 수행하여 이미지를 수행 할 수 있도록 기본적인 환경을 제공하게 된다. 이후 운영체제는 운영체제 초기화 작업에 들어가게 되고 초기화 작업 후, 응용프로그램이 수행 된다

3.2 구현 내용

본 절에서는 스케줄링 정책, 상태 전이도, Task 의 구조, Clock Tick, ITC (InterTask Communication), ISR 의 구조를 중심으로 한 구현 내용에 대해 기술한다.

3.2.1 스케줄링 정책

개발된 실시간 운영체제는 64 개의 태스크를 운영 관리 할 수 있으며 태스크 각각은 0 부터 63 까지의 고유의 우선순위를 갖게 된다. 63 의 우선순위를 갖는 태스크는 Idle 태스크로 지정 되여 있으며 생성된 모든 태스크들이 delayed 상태나 이벤트를 대기중일 때 이 태스크가 수행된다. 또한 높은 우선순위를 갖는 태스크가 먼저 수행하는 선점형 방식을 채택하고 있다. 가장 높은 우선순위를 찾는 방식은 64 개 태스크들의 ready 된 상태를 저장하는 1 비트씩을 갖는 테이블을 두어 태스크가 ready 되였을 때 그 중 가장 높은 우선순위를 갖는 태스크를 일정한 시간 내에 선택할 수 있도록 설계 하여 실시간 운영체제가 가져야 하는 Determinism 을 갖게 한다[3].

3.2.2 태스크의 상태

개발된 실시간 운영체제는 그림 3 처럼 다음과 같은 상태들을 가진다.

1)Dormant: 메모리에는 존재하나 스케줄링의 대상이 아닌 TCB 만을 가지고 있는 것을 상태.

2)Ready: 태스크가 생성되거나 CPU 의 제어 권을 사용하기위해 기다리고 있는 상태.

3)Running: 현재 CPU 점유하고 사용중인 상태.

4)Delayed: 운영체제의 API 중 자기 스스로를 몇 Clock Tick 동안 스케줄링 대상에서 제외되며 Delayed 되는 상태.

5)Wait For Event: 다른 태스크나 interrupt 에 의해 이벤트의 생성을 기다리는 상태로 스케줄링 대상에서 제외.

6)Interrupted: 운영체제에 Interrupt 가 요청된 경우 이를 처리하는 ISR (Interrupt Service Routine)으로 CPU 사용권을 넘기고 루틴을 종료 후 선점형 우선 순위 운영 체제이므로 ISR 이 끝난 현재 Ready 상태에서 우선순위가 가장 높은 것으로 CPU 사용권을 넘기는 선점 방식이 사용된다[3][5].

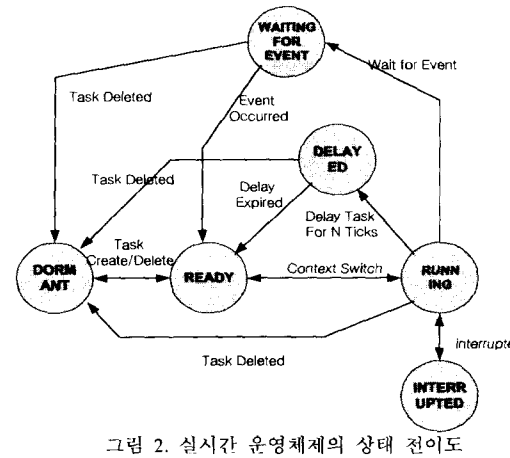

그림 2. 실시간 운영체제의 상태 전이도

3.2.3 Task 의 TCB(Task Control Block)구조

```
typedef struct os_tcb
{
    void (*OSTCBStart)(void);   /*태스크 시작위치 */
    void *OSTCBStkLimit;        /* 스택의 한계 */
    unsigned long OSTCBStkPtr;  /* 스택 포인터 */
    char OSTCBName[10]          /* 태스크의 이름 */
    unsigned long OSTCBStackSize    /*스택의 크기 */
    uint    OSTCBStat;  /* 태스크의 상태 */
    uint    OSTCBPrio;  /* 우선순위 */
    uint    OSTCBDly;   /* suspend되는 틱값 */
    uint    OSTCBX;
    uint    OSTCBY;
    uint    OSTCBBitX;
    uint    OSTCBBitY;
    OS_EVENT    *OSTCBEventPtr;
    struct os_tcb   *OSTCBNext;
    struct os_tcb   *OSTCBPrev;

} OS_TCB;
```

그림 3. Task 의 TCB 구성

- OSTCBEventPtr 는 현재 Task 가 Event Task list 에 포함되어 있을 경우 다음 Event Task 의 Pointer 에 대한 정보를 가지고 있다
- OSTCBX,Y,BitX,BitY 는 이차원 배열로 되어있는 Task list 에서 Task 가 어디에 위치하고 있는지에 대한 위치 정보를 가지고 있다. 이것을 이용해 고정된 시간 내에 원하는 우선순위의 Task 를 찾아 낼 수 있다.
- OSTCBNext,OSTCBPrev 는 현재 Task 가 Ready List 에 포함되어 있을 경우 다음 Ready Task 의 위치를 가리키는 Pointer 이다.

3.2.4 Clock Tick

클럭 틱은 delayed 된 상태의 GPS 태스크를 깨우고 Waiting for Event 상태에서 timeout 값을 감소시키는 역할을 하며 시스템의 심장이라고도 불린다. GP 2021 에서 505us 마다 StrongARM 으로 입력되어 위성 신호의 획득과 분석을 수행하는 ACCUM_INT 에서는 틱이 발생하였는지는 매번 검사하여 틱이 발생하였다면 전역 변수인 TIC 값을 1 증가 시키고 OSTimeTick()함수를 실행하여 GPS 태스크들의 대기 틱수를 감소시키며 이때

OSTimeDly 값이 0 이 된 태스크가 발생하면 ready 상태로 만들어 스케줄링 대상으로 만든다. 이 Tick 의 주기는 StrongARM 에서 Handling 할 수 있고 구현된 실시간 운영체제에서는 하드웨어를 초기화 하는 부분에서 약 100msec 로 고정 하였다.

3.2.5 GPS Task 구성

모든 GPS 태스크들의 특징은 그림 4 처럼 진입 할 때 현재의 틱값을 저장하며 코드를 수행 후 OSTickDly(ticks, start_tic)을 실행하여 Delayed 상태가 된다. 또 하나의 특징은 그림 5와 같이 모든 태스크의 수행 해야 할 시점이 결정되어있다는 것이다.

```
Void Tnav(void)
{
        while(1)
    {
        CurrentTIC(&start_tic);
        /*현재의 틱값을 얻어옴*/
          .
          .
        OSTickDly(10,start_tic);
        /* Suspend for 10TIC. (1sec) */
    }
}
```

그림 4. GPS 태스크의 구성

이것은 suspend 되는 시간이 현재부터 주어진 틱값 동안 delayed 상태로 있는 것이 아니라, 주어진 현재 ticks 값에서 전역변수 TIC 에서 태스크의 start_tic 값을 뺀값 즉, ticks 에서 태스크가 수행하는데 걸린 틱값을 뺀 나머지 틱동안 delayed 상태로 있다는 것이다. 결국 태스크가 연산시간이 가변적이더라도 태스크 수행시점부터 다음 번 다시 실행될 시점까지의 틱 간격이 항상 일정하다는 것이다. 위의 예를 들자면 태스크가 OSTickDly 를 호출한 시점이 어디라도 태스크의 시작한 시점부터 1 초가 지나면 태스크가 깨어져 Ready 상태로 된다.

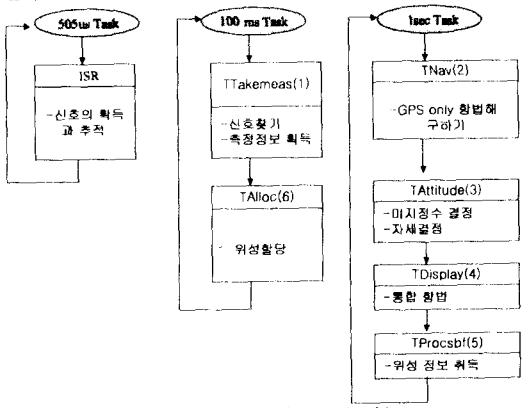

그림 5. GPS 태스크의 기능

3.2.6 ITC (InterTask Communication)

개발된 운영체제는 ITC 를 위하여 세마포, 메일박스, 메시지큐를 제공한다.

3.2.7 ISR

Interrupt Handling 을 하기 위하여 다음과 같은 요소들을 고려 하여 ISR routine 을 작성하였다.

- ISR Source
  자세 결정용 GPS 와 DR 센서를 이용한 시스템 보드의 ISR Source 는 GP2021 에서 StrongARM 으로 보내는 ACCUM_INT 가 유일한 Interrupt Source 이다. 따라서 이 Interrupt Handler 에는 Interrupt Vector 가 없는 것이 더 나은 시간상의 성능을 발휘 할 수 있으므로 개발된 운영체제에는 interrupt Vector 가 없이 Interrupt 가 발생하면 직접 ISR 을 실행하게 된다.
- Interrupt Enable/Disable
  일반적으로 태스크가 임계영역에서 수행 해야 할 코드의 앞뒤에는 Interrupt 의 Enable 과 Disable 한다. 이 인터럽트의 enable/disable 은 Micro-processor 의 Status register 에 있는 Interrupt Bit 를 Set/Reset 함으로써 Enable/Disable 을 하는 방법을 사용하나 ARM 은 Interrupt 를 enable/disable 하기 위해서 CPU Mode 가 User Mode 가 아닌 다른 모드여야 한다. 따라서 User mode 에서 Interrupt 를 Enable/Disable 하려면 복잡한 절차를 거쳐야 Interrupt 를 Enable/Disable 할 수 있다. 이러한 복잡한 절차를 생략하기 위하여 그림 6 처럼 SA-1100(StrongArm)의 인터럽트 컨트롤러에 있는 인터럽트 마스크 레지스터에 1 또는 0 을 넣어서 Interrupt Bit 를 Enable/Disable 하지 않고 아래그림 6 과 같이 Interrupt 를 Enable/Disable 간단하게 구현할 수 있다[4].

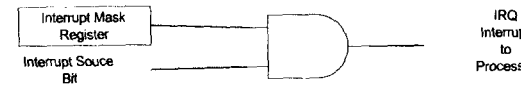

그림 6. GPS/DR 시스템 보드의 Interrupt 구조

4. 결론 및 향후 연구 과제

본 논문에서는 자세 결정용 GPS 와 DR 센서를 이용한 시스템 보드를 위한 우선순위 기반의 실시간 운영체제를 보드에 최적화 되도록 설계하고 구현하여 보드에 두개의 위성안테나를 장착하여 태스크들이 원활히 동작하는 것을 확인하였다.

앞으로 수행되어야 할 연구과제는 신뢰성 향상을 위한 실시간 운영체제의 성능 평가가 수반 되어야 하며 차량에 시스템을 장착하여 필드 테스트를 통하여 설계한 운영체제가 적합한 지에 대한 검증을 할 것 이다.

5. 참고문헌

[1]이재호, "자세 결정용 GPS 와 DR 센서를 이용한 통합 시스템 설계, 충남대학교 석사학위 논문,2000.

[2]Advanced RISC Machine Ltd(ARM), "ARM Software development Toolkit User Guide", 1996.

[3]Jean, J. Labrosse, "uC/OS The Real-Time Kernel", R&D Publications, 1992.

[4]Intel, "StrongARM™ SA-1100 Microprocessor Technical reference Manual" ,Dec. 1998.

[5]GEC Plessey Semiconductors, "GPS Architect Software Design Manual", Volume1,2, Dec. 1997.

[6]MITEL semiconductor, "12 Channel GPS Development System", Mar. 1997.