J. Astron. Space Sci. 24(4), 451-456 (2007) [기술논문]

장치와 태스크 간 입출력 데이터 교환의 예측성 향상 방안

구철회[1†], 양군호[1], 최성봉[2]
[1]한국항공우주연구원 통신해양기상위성사업단 통해기체계팀
[2]한국항공우주연구원 통신해양기상위성사업단

METHODOLOGY TO ENHANCE THE PREDICTABILITY OF I/O DATA EXCHANGE BETWEEN DEVICE AND TASKS

Cheol-Hea Koo[1†], Koon-Ho Yang[1], and Seong-Bong Choi[2]
[1]Communication Satellites Dept., COMS Program Office, KARI, Daejeon 305-333, Korea
[2]Communication Satellites Dept., KARI, Daejeon 305-333, Korea
E-mail: chkoo@kasi.re.kr

(Received August 20, 2007; Accepted October 31, 2007)

요 약

장치로부터 발생된 데이터는 소프트웨어에서 사용되기 위해 최대한 정확한 시간에 데이터의 완결성을 가지고 해당 태스크에 전달되어야 한다. 이 과정에서 구조적인 위험성과 성능 저하가 발생할 원인이 잠재되어 있다. 이 논문은 장치와 태스크 사이에서 I/O 데이터 교환이 발생할 시 충돌에 의해서 발생가능한 데이터 손실을 예방할 수 있는 방안을 기술한다. 이 논문에서 기술하는 방안에 대한 메커니즘의 예시도를 제안하고 그 효과 및 장단점을 고찰하였다.

ABSTRACT

Data coming from devices shall be transported to a specific task to be used in a software with the most accurate time and data integrity. During this process, a potential cause for invoking structured hazard and performance degradation is dormant. In this paper, a method which can protect the data integrity from the possible data corruption when collision has happened during I/O data exchange between device and tasks is presented. Also, an example diagram of mechanism according to the method is shown and the effect, merits and demerits of the method is evaluated.

Keywords: data processing, buffering, predictability, deterministic

1. 서 론

위성과 같은 실시간 시스템에서 장치의 I/O는 다수의 태스크(Task)에 의해서 동시 다발적으로 액세스되어 진다. 장치를 다루고 있는 소프트웨어 중 예측보다 많은 프로세싱 파워를 사용하는 것으로 해석된 소프트웨어에서 상당부분의 프로세싱 파워가 장치의 I/O를 위해서 소모되는 것으로 알려져 있다. 그만큼 장치와의 I/O 인터페이스는 전체적인 소프트웨어 성능평가에 중요한 부분을 차지하고 있다(Ball 2000).

†corresponding author

또한 우선순위 반전(Priority inversion)도 높은 우선순위를 가진 태스크가 자신을 포함한 여러 태스크가 사용하는 공유 자원(Shared resource)에 대한 사용권을 적절히 확보하지 못했을 때에 발생한다(Burns & Wellings 2000). 장치의 I/O를 사용하는데 있어 잘못된 설계로 인해 프로세싱 파워 소모 및 안정성 저하 등이 발생하므로 이것과 관련된 태스크들 간의 우선순위 할당 및 장치의 입력 처리 작업의 스케줄링의 정밀함이 요구된다. 위성 컴퓨터 벤치를 개발하는 프로젝트를 수행하던 중 MIL-STD-1553B 버스 등 장치 하드웨어의 I/O 동작이 순전히 주기적으로 동작하게 될 때에 이 자원을 액세스하는 소프트웨어 태스크들 사이에 매우 희귀하게 문제가 발생했던 경험이 있었다. 그 근본 원인은 태스크와 장치 하드웨어가 동시에 같은 자원을 액세스함으로서 야기되었고 초기 설계부터 정적 태스크 스케줄링과 정적 장치 하드웨어 스케줄링으로 충분한 마진을 확보했다고 판단했으나 타이머의 부정확성, 태스크의 예기치 못한 지연 등으로 인해 위 현상이 발생한 것으로 해석되었다.

이 논문에서는 장치의 I/O 자원을 획득하기 위해 소요되는 연산 능력의 소비를 최대한 줄이고, 충돌없이 효과적으로 다중 태스크로부터의 I/O 요구를 처리할 수 있는 방안을 제시해서 소프트웨어 동작의 예측성을 향상시키고자 한다.

2. 우선순위 기반의 실시간 시스템

만약 최대한 태스크의 반응성(Reactivity)이 요구되는 상황이라면 장치 I/O에 지연(Delay)을 의도적으로 사용하는 데이터 버퍼링 방법을 적용시키기 곤란하다. 왜냐하면 버퍼링은 I/O를 요구받는 즉시 처리하지 않고 모아두었다가 적당한 시간에 한꺼번에 처리하는 방식이기 때문이다. 반면 버퍼링의 장점으로 데이터를 모아 두었다가 충돌을 발생시키지 않는 적절한 시점에 데이터를 처리할 수 있다는 점을 손꼽을 수 있다.

위성 운용 소프트웨어의 경우 장비의 I/O를 사용하는 태스크의 프로세싱은 아래와 같이 구분되고 순차적으로 동작한다.

- 물리 입력 처리(Physical input processing) : 연산 처리를 하기 위해서 필요한 입력 데이터를 입력받는 프로세싱
- 연산 프로세싱(Numerical processing) : 물리 출력 처리를 수행하기 위해 필요한 데이터를 연산하는 프로세싱
- 물리 출력 처리(Physical output processing) : 연산이 완료된 데이터를 출력하는 프로세싱

그림 1에서 볼 수 있듯이 장치의 I/O를 위해서 기다리는 대기 시간(Waiting time)은 효율적으로 프로세서를 활용하기 위해서 최소화되어야 한다. 그러기 위해서 입력 데이터는 미리 입력 버퍼에 대기해 있어야 하며, 물리 출력 프로세싱은 출력 데이터가 장치를 통해 처리가 완료되었는지 확인을 하지 않고 종료되어야 한다. 즉 태스크는 장치의 I/O를 로컬 함수(Local function)를 호출하듯이 수행할 수 있어야 한다. 만약 입력 데이터를 처리하는 중 장치 하드웨어가 입력 데이터에 액세스하게 되면 예상할 수 없는 시간 지연과 최악의 상황으로 데이터의 완결성이 손상되는 경우가 발생하게 된다.

이를 예방하기 위해서 장치 I/O를 위해서 필요한 데이터는 우선 버퍼링이 된 후 또 다른 후속 프로세스에 의해서 처리되어야 한다. 그림 2와 같이 대기 시간 감소 및 충돌 가능성을 배제하기 위해서는 태스크의 장치 I/O 기능과 실제 장치 I/O 사이에 버퍼링 층(Buffering layer)이 존재해야 한다. 그

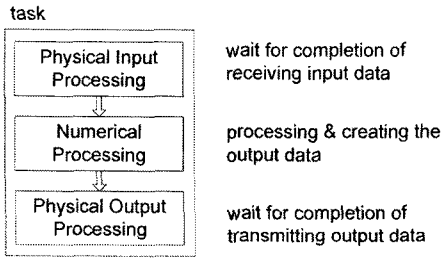

그림 1. 위성 제어 태스크의 일반 구조.

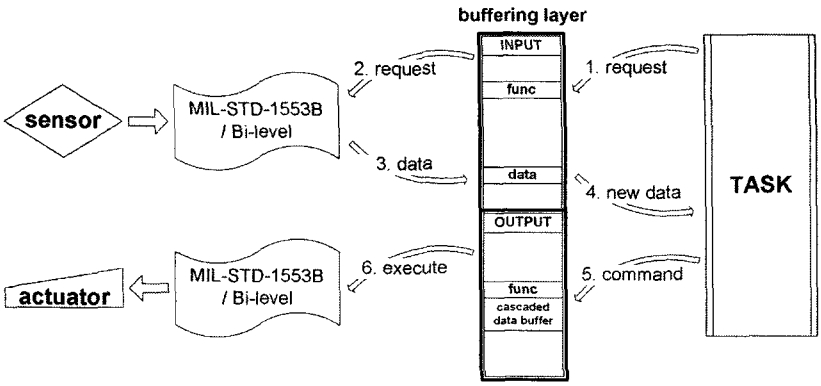

그림 2. 버퍼링 층을 통한 데이터 교환 다이어그램.

림 2는 태스크가 센서로부터 데이터를 입수하는 순서와 경로, 입수한 데이터를 바탕으로 명령을 생성하여 구동기에 명령을 보내는 과정을 나타내고 있다. 입력에 사용되는 데이터에는 단일 버퍼링이, 출력에 사용되는 데이터에는 직렬로 쌓여지는 캐스케이드 버퍼링이 필요하게 된다.

3. 데이터 교환 예측성 향상 방안

3.1 예측성 향상 스케줄링

실시간 시스템에서는 공유 자원을 다투어 사용하는 태스크(또는 태스크와 하드웨어 사이의) 간의 경쟁이 문제의 주원인이 되기 때문에 예상 가능한 잠재적인 위험은 아래와 같다.

- 공유 자원을 선점시 발생할 수 있는 데이터 손실(bad data integrity)
- 데이터 발생부터 처리까지 발생 가능한 불필요한 시간 손실(Dead time)

이러한 잠재적인 위험은 모두 태스크 사이의 CPU 또는 공유자원에 대한 선점 경쟁으로부터 야기된다. 태스크 간의 공유 자원 선점과 관련한 충돌시 해결 방법은 그동안 충분히 제시되어 왔다(Ben-Ari 1990, Burns & Wellings 1997). 그러나 사실 근본적인 해결책은 스케줄링에서 그러한 위험 가능성을 사전에 배제하는 것이다. 하지만 이렇게 되면 스케줄링은 소극적으로 설정되게 되고 반응성도 필히 나쁘게 될 것이라는 점은 역시 문제로 남는다. 다행히 위성 운영 소프트웨어는 아주 빠른 반응

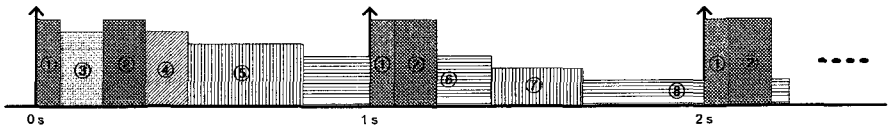

그림 3. 데이터 완결성이 보장되는 태스크 스케줄링 배치도.

성을 요구하지도 않고, 100%의 프로세싱 파워를 소모하게끔 요구하지도 않는다. 위성 운영 소프트웨어의 경우 CPU 사용 버짓(Budget) 요구사양이 40~50% 보다 적은 경우는 흔치 않을 것이다. 참고로 현재 본체 개발 완료된 통신해양기상위성의 경우 비행소프트웨어 런타임 마진은 약 50%로 평가되고 있다.

위성 운영 소프트웨어에서는 반응성 보다는 예측성(Predictability)이 더 높은 비중을 가지고 있다(Burns & Wellings 1995). 이 논문에서는 예측성을 보장하기 위해서 데이터를 생산하는 부분과 데이터를 읽어 들이는 부분을 스케줄링으로 완벽히 분리하는 방안이 제시된다.

위성 장치 관리 태스크(Device management task)는 장치와 태스크 사이에 요구되는 데이터를 생산 및 관리하는 일을 담당하는데 이 태스크는 데이터를 사용하는 태스크 보다 먼저 가동되어 데이터를 생성하고 제공하여야 한다. 이 두 태스크의 동작은 시간적으로 중첩되어서는 안된다.

데이터의 생산 및 사용과정에서 발생하는 충돌을 제거하기 위해서 그림 3과 같은 태스크 스케줄링 배치도를 사용할 수 있으며, 이 배치도에서 태스크는 대표적으로 실행주기가 빠르고 우선순위가 높은 태스크와 실행주기가 느리고 우선순위가 상대적으로 낮은 태스크 2가지로 구분되며, 그 내부적인 동작 성격은 아래와 같이 정의가 되어야 한다.

• 빠른 실행주기, 높은 우선순위 태스크(Fast execution rate, Higher priority Task)

- ①, ② : 데이터 획득과 관련이 없는 빠른 실행 주기 태스크, 또는 명령 데이터 처리와 관련된 태스크

- ③ : 태스크 ⑥~⑧가 필요로 하는 데이터 획득을 개시하는 태스크

- ④ : 태스크 ⑥~⑧가 필요로 하는 데이터 획득이 완료되기를 기다리는 태스크

• 느린 실행주기, 낮은 우선순위 태스크(Slow execution rate, Lower priority Task)

- ⑤ : 획득이 완료된 데이터를 후 그림 2의 버퍼링 층에 저장하는 태스크, 또는 명령 데이터를 처리하는 태스크

- ⑥, ⑦, ⑧ : 위성 제어 태스크, 그림 2의 버퍼링 층에서 데이터를 가져와(reading without waiting) 위성 제어 알고리즘을 수행하고 그 수행 결과물로 명령 데이터 생성

즉, 그림 3과 같이 데이터를 생산 및 관리하는 태스크는 데이터를 사용하는 태스크보다 실행 주기가 짧고, 우선순위가 높으므로 필시 데이터를 사용하는 태스크가 실행되고 있는 중간에 데이터를 사용하는 태스크를 선점하게 되므로 선점하고 있는 동안에 선점당한 태스크와 동일한 데이터를 취급하고 있어서는 안된다. 그렇지 않다면 설계자는 우선순위 선점과 같은 잠재적 위험에 대해 상세한 해석을 수행하여야 한다.

높은 우선순위 태스크가 실행시 낮은 우선순위 태스크보다 먼저 수행되므로, 실제 데이터를 사용하는 낮은 우선순위 태스크가 활성화되기 전에 데이터 획득을 수행하는 높은 우선순위 태스크가

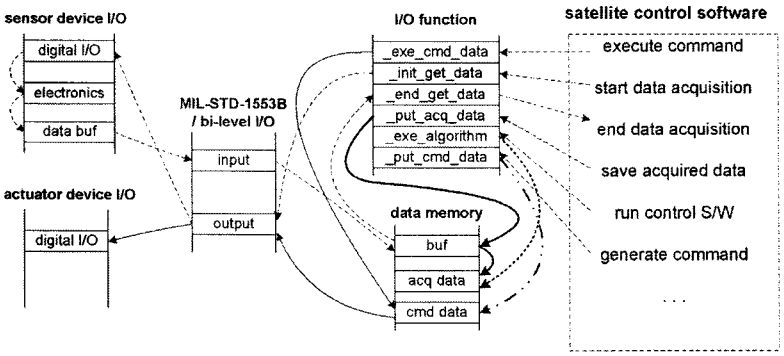

그림 4. 데이터 충돌 없는 위성 제어 소프트웨어 흐름도.

완료된다.

데이터 획득이 완료된 이후 느린 주기를 가지고 있는 데이터를 사용하는 태스크는 순차적으로 처리가 되어야 하는데 이들 태스크들이 실행되는 동안 데이터 획득(Acquisition) 동작은 발생하지 않아야 한다. 특히 빠른 주기 ③, ④와 느린 주기의 ⑤는 절대적으로 순차적 동작이 보장되어야 한다.

시스템의 목적 및 특성에 따라 태스크 구조가 변경될 필요성이 존재하지만 가급적 동일한 데이터 사용하는 태스크들이 중첩이 되도록 설계되어서는 안된다(Ganssle 2004).

3.2 I/O 데이터 프로세싱

위성 비행 소프트웨어의 자세제어 태스크는 태양센서, 지구센서, 자이로(Gyro) 등과 같은 장치들로부터 데이터를 입력받는다. 이들 센서로부터 데이터가 태스크에 도달하기 까지 어떤 경로를 거치든지 자세제어 태스크의 관점에서 봤을 때 I/O 데이터는 마치 로컬 데이터 처럼 접근이 가능해야 한다. 이 과정에서 위성 태양 센서로부터 데이터가 발생해 자세 제어 태스크에 도달하기 까지의 동작에 대한 내용은 그림 2에 나타나 있다.

그러므로 센서 등 장치로부터의 입력 데이터는 그림 3에서 볼 수 있는 바와 같이 실제 이 입력 데이터를 필요로 하는 태스크가 활성화되기 이전에 또 다른 버퍼링을 전문으로 하는 태스크에 의해서 처리되어 공유 메모리에 저장되어야 한다. 입력 처리는 이와 같은 순서가 준수되어야 한다.

효과적으로 입력 데이터가 버퍼링 되어 그 입력 데이터를 필요로 하는 태스크에서 사용되기 위해선 아래와 같은 요건을 필요로 한다.

- 입력 데이터 구조체 및 데이터 인터페이스 함수(input data structure & data interface function)
- 데이터 어드레스 맵핑 함수(data address mapping function)

출력 데이터 프로세싱의 경우 센서 등으로부터 자세 제어에 필요한 원천 데이터를 수집한 자세제어 태스크는 자세 유지 알고리즘을 사용하여 휠 또는 추력기와 같은 위성 작동기(Actuator)에 위성 자세를 제어하기 위해 명령을 보낸다.

이 명령 데이터는 MIL-STD-1553B 버스 또는 bi-level 명령 라인을 통해서 위성 작동기로 전달되게 되는데 이 MIL-STD-1553B 버스와 bi-level 명령 라인은 공유 자원이기 때문에 입력 데이터와 마

찬가지로 출력 데이터는 그림 3과 4에서 볼 수 있는 바와 같이 별도로 처리되어야 한다. 출력 데이터를 처리하기 위해 아래와 같은 기능을 필요로 한다.

- 출력 데이터 구조체(output data structure)
- 데이터 저장 함수(data store function)

그림 4에는 실제 위성 제어 소프트웨어에서 센서 및 구동기(Actuator)와 같은 장치와 어떤 방식으로 데이터를 가져오고 내보내는 지에 대한 기능 분해도를 나타내고 있다. 그림 4의 각 부분의 동작은 그림 3에서 나타난 바와 같이 데이터를 획득하는 부분, 데이터를 사용하는 부분으로 나뉜다. 즉 예를 들어 'start data acquisition'에서 'save acquired data' 부분은 빠른 실행주기, 높은 우선순위 태스크로 동작해야 하고, 나머지 부분은 느린 실행주기, 낮은 우선순위 태스크로 동작해야 한다.

4. 결 론

지금까지 이 논문에서 기술한 바와 같이 데이터를 획득 및 사용하는 태스크들 간의 충돌을 방지하는 태스크 및 하드웨어 I/O 스케줄링을 통해서 데이터의 완결성(Data integrity)을 보장할 수 있다. 데이터 입수를 담당하는 태스크와 데이터를 처리하는 태스크를 구조적으로 배치함으로써 실시간 성능과 프로세서 자원을 효율적으로 사용하는 장점을 동시에 획득할 수 있다. 하지만 이 경우 데이터를 운반하는 하드웨어 I/O 채널의 설계는 기존보다 더 고기술을 요구하게 된다. 왜냐하면 기존에는 I/O 채널은 주기적 I/O 동작으로 요구되는 성능을 만족시킬 수 있었지만 현재는 "요구 후 처리(Process after request)"로 수행되어야 하기 때문에 실시간적으로 다음 I/O 동작에 해당되는 작업을 장치 내에서 스케줄링 해야만 하기 때문이다. 하지만 이 논문에 소개된 방안을 통해서 상위 제어 소프트웨어 단계의 데이터 완결성은 완전히 보호된다.

요구 후 처리 기술은 장치 구동의 오버헤드를 상승시키므로 후속의 논문에서는 하드웨어 디바이스 I/O에서 오버헤드를 가능한 줄이면서 실시간으로 채널 I/O 작업을 스케줄링 하는 방안을 제시할 예정이다.

감사의 글: 이 논문은 과학기술부 "통신해양기상위성 시스템 및 본체 개발사업"의 일부임을 밝히며 연구지원에 감사를 드립니다.

참고문헌

Ball, S. R. 2000, Embedded Microprocessor Systems 2nd Ed (Woburn, MA: Newnes), pp.125-131

Ben-Ari, M. 1990, Principles of Concurrent and Distributed Programming (Hertfordshire: Prentice Hall), pp.27-59

Burns, A. & Wellings, A. J. 1995, Software-practice and Experience, 25, 705

Burns, A. & Wellings, A. J. 1997, Real-Time Systems and Programming Language 2nd Ed (Harlow: Addison-Wesley), pp.203-353

Burns, A. & Wellings, A. J. 2000, Concurrency in Ada 2nd Ed (Cambridge: Cambridge Univ.), pp.277-301

Ganssle, J. 2004, The Firmware Handbook (Woburn, MA: Newnes), pp.212-223