정보 및 제어 심포지엄(ICS '05) 2005. 5. 14

위성용 명령 처리기의 명령 입수 지연 오류 정정

Correction of the delay faults of command reception in satellite command processor

구철희*, 최재동**
Cheol-Hea Koo, Jae-Dong Choi

Abstract - The command processor in satellite handles the capability of the process of command transmitted from ground station and deliver the processed data to on board computer in satellite. The command processor is consisted of redundant box to increase the reliability and availability of the capability. At each command processor, the processing time of each command processor is different, so the mismatch of processing time makes it difficult to timely synchronize the reception to on board computer and even will be became worse under the command processor's fault. To minimize the time loss induced by the command processor's fault, on board computer must analyze the time distribution of command propagation. This paper presents the logic of minimizing the delay error of command propagation the logic of analyzing the output of command processor.

Key Words : 명령처리기, 오류 정정, 결함 허용, 명령 무결성

1. 장 서 론

위성은 지상관제국으로부터 명령을 입수하여 처리하는 기능을 담당하는 명령처리기(Command Processor)를 가지고 있다. 명령처리기는 입수한 명령을 검증한 후 명령 데이터를 위성 탑재 컴퓨터(OBC)로 전송한다. 신뢰성과 가용성을 높이기 위해서 명령처리기는 예비부품으로 구성되어 있는데 동작 개념은 Hot Redundancy이다. 즉 각각의 명령처리기는 모두 동일한 명령 처리 동작을 동시에 수행하고, 처리한 명령 데이터를 각각 위성 탑재 컴퓨터에 전송한다. 위성 탑재 컴퓨터는 각각의 명령처리기에서 입수한 명령데이터를 가지고 명령 데이터의 유효성을 판단하게 되는데 각각의 명령처리기에서 처리된 명령 데이터가 위성 탑재 컴퓨터에 도달하는 시간은 명령처리기 내부의 소프트웨어 동작 상태에 따라서 불규칙적이며 비예측적이다[8, 10, 11]. 위성 탑재 컴퓨터에서는 예비부품으로 구성되어 있는 명령처리기에서 수신된 명령 데이터 중 하나를 실행하기 위해서 명령처리기로부터 전달된 명령의 저장 및 명령을 선택하는 기능을 가진다. 명령처리기의 결함 또는 명령 데이터 손실이 발생했을 때 명령 데이터의 위성 탑재 컴퓨터로의 전송은 중단되고, 이미 위성 탑재 컴퓨터에 전송된 명령 데이터는 무효화되어야 한다. 이때 다른 명령처리기의 명령 데이터를 실행하는데 데이터의 입수 시점의 차이와 결함 관리에 따라서 명령 실행의 시간 손실이 발생한다[12]. 본 논문에서는 명령처리기가 예비부품으로 구성되어 있고 명령처리기에 결함 상황이 발생했을 때 위성 탑재 컴퓨터가 명령 입수의 문제점을 처리하는 로직을 제안한다.

저자 소개
* 正 會 員 : 한국항공우주연구원 선임연구원
** 正 會 員 : 한국항공우주연구원 선임연구원

2. 장 명령 입수 로직

예비부품으로 구성된 명령처리기로부터 지상 명령 데이터를 전달받는 위성 탑재 컴퓨터는 2개의 명령처리기로부터 모두 성공적으로 명령을 수신하였을 때는 둘중 하나의 명령 데이터를 실행하면 된다. 그림 1은 명령처리기로부터 명령 데이터를 입수하고 실행하는 위성 탑재 컴퓨터의 초기 구성을 나타낸다[11].

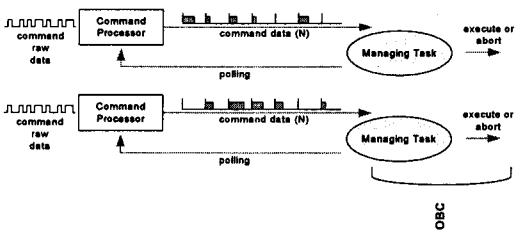

그림 1. 일반 명령 입수 과정

위성 탑재 컴퓨터는 명령처리기에서 명령 데이터를 가져오기 위해서 명령처리기를 폴링(Polling)한다[1, 4]. 폴링을 사용하는 이유는 위성 탑재 컴퓨터와 명령처리기 사이의 데이터 통신은 MIL-STD-1553B 데이터 버스를 사용한 공유 메모리 방식을 사용하기 때문이다. 전용 데이터 버스가 아닌 분산형 데이터 버스를 사용하기 때문에 인터럽트를 이용한 명령 데

이터 전송은 고려할 수 없다[10, 11, 12].

각각의 명령처리기는 지상국으로부터 동일한 원시 명령 데이터(Command Raw Data)를 수신하지만 각 명령처리기 내부의 소프트웨어 동작이 시간상으로 똑같지 않기 때문에 각각의 명령처리기가 위성 탑재 컴퓨터에 전달하는 명령 데이터의 도착 시간은 일치하지 않는다. 따라서 각각 명령처리기로부터의 명령 데이터의 도착 시간을 각각 T_1, T_2라고 했을 때, 시간 간격 $\Delta T(=|T_1-T_2|)$는 일정하지 않으며 비예측적이다. 각각의 명령처리기로부터 전달된 지상 명령은 단 1개만 선택되어 실행되어야 한다. 중복실행은 허용되지 않기 때문이다[1, 6].

그림 1에서 위성 탑재 컴퓨터의 "Managing Task"는 명령처리기로부터 전송되는 명령 데이터를 저장하고 저장이 완료되면 명령을 실행한다[12].

1개만 실행하기 위해서 "Managing Task"는 서로 정보를 교환한다. 명령 실행은 항상 제일 먼저 명령 데이터가 도착하기 시작한 명령처리기에서 수행하는 것으로 정하고 그림 2와 같은 유사 코드를 사용하였다.

```
wait until data arrive from the Command_Processor
if first arrived command then
    store & execute the command
else
    only store the command
```

그림 2. 명령 처리 유사 코드

문제는 명령을 실행하기로 설정된 "Managing Task"에서 명령처리기의 결함 또는 태스크 자체의 결함으로 인하여 데이터 전송 중단과 같은 명령 지연 오류가 발생하는 상황에서 발생한다. 이 경우 다른 "Managing Task"에서 제대로 명령 데이터를 수신한 경우에는 이 태스크에서 명령이 실행되도록 명령 처리 로직이 수정되어야 한다[12].

이러한 문제점을 해결하기 위해서는 다른 "Managing Task"의 명령 실행을 감시할 수 있는 타이머가 구현되어야 하는데 명령 지연 시간이 비예측적이기 때문에 예측할 수 있는 지연 시간의 최대값이 타이머 시간으로 설정되어야 한다. 이러한 개념이 첨가된 명령 처리 유사 코드를 그림 3에 나타내었다.

```
wait until data arrive from the Command_Processor
if first arrived command then
    store & execute the command
else
    store the command
    enable Timer at time Tc
    wait for Command Completion
    if not Command completed within time Tc then
        execute the stored command
```

그림 3. 수정된 명령 처리 유사 코드

그림 3과 같은 명령 처리 로직에서 발생할 수 있는 문제점은 명령 지연 오류가 발생하면 항상 타이머가 설정된 시간동안에는 명령 실행이 지연된다는 것이다. 타이머의 시간을 최적으로 설정하는 것은 어렵고 실수하기 쉬운 작업이며, 설정된 최대 지연 시간이 모든 형태의 명령에 대해서 성능 요구사항을 만족하는지 확인하는 것은 어려운 일이다. 또한 매우 긴급을 요하는 명령의 경우 최대 지연 시간만큼 실행이 지연되는 것은 심각한 시스템의 성능 저하를 유발할 수도 있는 문제점을 가지고 있다[5].

이런 문제점을 처리하기 위해서 본 논문에서는 "감독 태스크(Supervisor Task)"를 도입하였다. 감독 태스크는 각 "Managing Task"의 활동을 감시하고 앞에서와는 달리 먼저 데이터가 도착하기 시작한 태스크가 명령을 실행하도록 하지 않고 명령 데이터의 저장을 먼저 완료한 태스크가 명령을 실행할 권한을 갖도록 각 태스크의 구동을 제어한다.

그림 4에는 감독 태스크의 간섭을 받는 "Managing Task"의 명령 처리 유사 코드를 나타내었다.

```
wait until data arrive from the Command_Processor
report the command receiving to Supervisor Task
store the command
if command reception finished then
    report the command reception to Supervisor Task
    if got a privilege to execute the command then
        execute the stored command
    else
        abort the stored command
```

그림 4. 문제점이 수정된 명령 처리 유사 코드

이와 같이 감독 태스크의 간섭 아래에서 수행되는 명령 처리 방식은 명령 지연 오류 하에서도 불필요한 실행 시간 지연없이 처음 명령 입수가 완료된 "Managing Task"의 명령 실행을 허가할 수 있다. 명령 처리의 전반적인 과정을 그림 5에 나타내었다[1, 2, 6, 7].

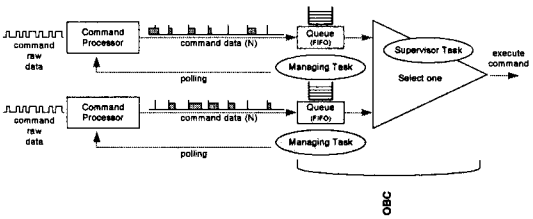

그림 5. 문제점이 해결된 명령 입수 과정

그림 5와 같이 감독 태스크에 의한 예비부품의 명령 처리기와 위성 탑재 컴퓨터의 접속 관리 방식은 목적했던 바대로 결함 상에서도 명령 실행 시간의 손해가 없었을 뿐 아니라 태스크간의 동기 신호도 크게 단순화시켰다.

3. 장 결 론

예비부품의 명령처리기의 사용은 실질적으로 명령을 실행하는 과정 중에 발생할 수 있는 결함을 허용하기 위해서 존재한다. 명령처리기와 위성 탑재 컴퓨터와의 단일 명령 처리 과정은 분명하게 결함에 취약하다. 예비부품으로 구성된 명령처리기와 위성 탑재 컴퓨터의 "Managing Task"는 다른

쪽의 결함을 허용하기 위해서 사용되었으며 또한 예상된 결함 허용효과를 제공하나, 결함의 시간 지연 오류에 따른 실행 시간의 손해가 예비부품으로 구성되면서 새롭게 대두되었다.

여러 가지 방법을 도입한 후에 2개의 "Managing Task"를 관리하는 감독 태스크를 구성하여 이 태스크들에 대한 조정을 수행하는 것이 최적이라는 결론을 내리게 되었다.

감독 태스크를 사용함으로써 명령의 실행 여부 및 명령처리기의 결함 여부도 손쉽게 판단이 가능하게 되었다. 이것은 단순히 예비부품으로 구성하고, 이러한 문제점을 "Managing Task"간의 통신을 통해 해결하려고 했을 때는 얻지 못한 장점이었다. 본 논문에서 언급한 구성으로 결함이 발생한 부품의 재구성(Reconfiguration) 로직도 쉽게 구성이 가능했다.

본 논문에서 구성한 방식은 일반적인 서버-클라이언트 구성을 따르는 것으로 볼수 있다. 이러한 개념적인 접근은 예비부품으로 구성되는 시스템의 개발 초기에 적용하는 것이 필요하다.

본 논문에서 소개된 명령처리기와 위성 탑재 컴퓨터의 인터페이스도 초기에 단일 인터페이스를 구현하고 추후에 이중화할 생각으로 개발에 착수했으나 이중화했을 당시 여러 가지 결함 상황을 실험하면서, 결함에 따른 명령 지연 오류를 기존 구성으로는 효과적으로 처리하기가 곤란하였다. 기존 구성을 유지하려고 여러 가지 다양하고 복잡한 로직을 고안하여 실험하였으나 모두 만족스러운 결과를 얻지 못하였다.

결과적으로 본 논문에서 소개한 바와 같이 이중화된 인터페이스에 대한 감독 기능을 도입하면서 로직은 매우 단순화되고 기대이상의 결과를 얻을 수 있었다.

후 기

본 논문은 과학기술부 "통신해양기상위성 1호 시스템 및 본체 개발 사업"의 일부임을 밝히며 연구지원에 감사를 드립니다.

참 고 문 헌

[1] M. Ben-Ari, "Principles of Concurrent and Distributed Programming", Prentice Hall International Series in Computer Science, 1990, pp 164~175

[2] Alan Burns, Andy Wellings, "Real-Time Systems and Programming Languages, 2nd Ed.", ADDISON-WESLEY, 1997, pp 400~432

[3] A. Burns, A. J. Wellings, "Engineering a Hard Real-time System: From Theory to Practice", SOFTWARE-PRACTICE AND EXPERIENCE, VOL. 25(7), pp. 705~726, 1995

[4] Ken Tindell, "Analysis of Hard Real-Time Communications", University of York, England

[5] Giorgio Buttazzo, "Real-Time Issues in Advanced Robotics Applications", Pisa, Italy

[6] Narain Gehani, Andrew D. McGettrick, "Concurrent Programming", International Computer Science Series, 1988, pp 312~318

[7] David E. Simon, "An Embedded Software Primer", Addison Wesley, 1999, pp 199~207

[8] 구철회, 김중표, 최재동, "명령 오류율 최소화를 위한 명령 코딩 및 해석", 02년도 추계 한국항공우주학회 학술대회, p596

[9] 구철회, "결함허용 시스템간 이상적 통신방안 연구", 03년도 대한전자공학회 하계종합학술대회, p398

[10] 구철회, "상향링크 명령 처리기의 결함 허용 설계", 03년도 한국항공우주학회지 제31권 제3호, p95

[11] 구철회, 김중표, 최재동, "위성 원격측정명령 처리기의 성능검증모델 개발", 04년도 한국항공우주학회 추계학술대회, p233

[12] 구철회, "위성 명령 처리기의 결함 처리 성능 시험 방법", 05년도 한국항공우주학회 춘계학술대회, p669