논문 2014-09-23

리눅스 기반 ARINC 653 헬스 모니터

(Linux-based ARINC 653 Health Monitor)

윤 영 일, 조 현 우, 김 형 신*
(Young-il Yoon, Hyunwoo Joe, Hyungshin Kim)

Abstract : The software running on avionic system is required to be highly reliable and productive. The air transport industry has developed ARINC Specification 653(ARINC653) as a standardized software requirement of avionics computers. The document specifies the interface boundary between avionics application software and the core executive software. Dependability in ARINC 653 is provided by spatial and temporal partitioning whilst fault-tolerance is provided by health monitoring mechanism. Legacy real-time operating systems are used to support ARINC653 health monitor on integrated modular avionics(IMA). However, legacy real-time operating systems are costly and difficult to modify the kernel. In this paper, we suggest a Linux-based ARINC653 health monitor. Functionalities to support ARINC653 health monitor are implemented as a Linux kernel module and its performance is evaluated.

Keywords : ARINC 653, Linux, Health monitor, IMA, Fault tolerance

Ⅰ. 서론

항공전자 시스템은 다양한 종류의 임무를 수행하기 위해서 여러 가지 장치로 구성 되어 있으며, 지속적으로 임무가 증가함에 따라 전자 장치의 수도 증가하고 있다. 이에 따라 시스템의 복잡성은 증가하게 되고 안정성과 신뢰성을 보장하기가 어려워지고 있다. 이러한 문제를 해결하기 위해서 과거 항공전자 시스템에서는 각 장치들이 독립적으로 구성된 연방형(Federated) 시스템이 제안되었다.

연방형 시스템은 다양한 독립적인 전자장치가 사용가능하며, 각 장치의 설계와 제작이 용이하다는 장점이 있다. 하지만 독립시스템들의 일부 기능이 중복되어 비용이 증가하고 소모되는 전력과 공간이불가피하게 증가하는 문제점이 있다. 위와 같은 문제점을 개선하기 위해서 미국과 유럽에서는 연방형 시스템보다 향상된 통합 모듈 항공 전자 시스템(Integrated Modular Avionics System)이 개발되었다[1].

통합 모듈 항공 전자 시스템은 기존의 독립적으로 구성된 전자 장비를 하나로 통합한다. 이에 기존의 독립적인 장비들이 차지하던 공간을 활용하여 장비의 크기와 무게를 줄이는 효과를 가져왔다. 뿐만 아니라 전력 소비와 유지보수 비용을 줄이고 임무 효율성을 증가시키는 장점을 얻을 수 있다. 이러한 통합 모듈 항공 전자 시스템을 구성하기 위해서는 실시간 운영체제와 응용프로그램간의 독립성을 보장할 수 있는 표준이 필요하였고 이를 위해 항공 전자 관련 학계에서는 ARINC 653 이라는 항공 전자 시스템을 위한 운영체제와 응용프로그램간의 표준 인터페이스를 규정하였다[2].

통합 모듈 항공 전자 시스템은 하나의 모듈이 고장이 발생될 경우 전체 시스템에 영향을 미치게 된다. 따라서 이러한 문제를 해결하기 위해서 ARINC 653 에서는 고장 발생시 고장을 탐지하고, 적절하게 처리할 수 있는 헬스 모니터 표준을 정의하고 있다. 현재 ARINC 653 헬스 모니터는 실시간 운영체제인 VxWorks[3], POK[4],

*Corresponding Author(hyungshin@cnu.ac.kr)
Received: 3 Mar. 2014, Revised: 4 Apr. 2014, Accepted: 1 May 2014.
Y. Yoon: LIG Nexone
H.W. Joe, H.S. Kim: Chungnam National University
※ 본 연구는 한국연구재단을 통해 교육과학기술부의 우주기초원천기술개발 사업(NSL, National Space Lab)으로부터 지원받아 수행되었습니다. (2011-0020905).

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2014.9.3.183

LynxOS[5], XtratuM[6] 등에 구현되어 항공 전자 시스템 및 인공위성에서 적용되었고 헬스모니터 기능역시 포함하고 있다. 하지만 상용 실시간 운영체제들은 비싼 로열티와 유지보수 비용문제를 고려해야 하며 상용 실시간 운영체제에 적용된 연구는 기능수정과 확장이 용이하지 못하여 성능 개선이 어렵다는 단점이 있다. 실제로 VxWorks 653 기반 헬스모니터 연구[7] 에서는 ARINC 653 표준에 따라 헬스모니터를 설계하였으나, 이를 직접 구현하고 평가하지는 못했다.

최근 이러한 단점을 개선하기 위해 오픈 소스 기반인 리눅스에 ARINC 653 적용을 제안하고 구현한 연구들이 있다[8, 9]. 리눅스는 오픈소스로 다른 실시간 운영체제에 비해 비용을 절감할 수 있고, 커널의 확장과 수정이 용이함을 장점이 있다. 하지만 이들 연구에서는 시간 공간 파티셔닝 지원에만 집중했으며 헬스모니터 표준은 고려하지 않은 한계가 있다.

따라서 본 논문에서는 ARINC653 표준에 따라 안정성과 신뢰성을 만족하기 위한 헬스모니터 기능을 오픈 소스 기반의 리눅스에 구현하였다. 우리가 구현한 리눅스 기반 ARINC 653 헬스모니터는 빠른 고장 탐지를 위해서 커널 영역과 사용자 영역에서 동시에 동작하여 고장처리하고 복구하는 시간을 줄일 수 있도록 하였다. 본 연구의 결과로 헬스모니터의 기능의 정상적으로 동작하는지 확인하였고, 고장탐지 시간과 시스템 오버헤드를 각각 측정하여 우리가 개발한 헬스모니터를 평가하였다. 평가결과

본 논문은 2장에서 기존에 실시간 운영체제에 적용된 헬스 모니터의 관련연구를 알아보고 3장에서는 리눅스에서 헬스 모니터 구현과 설계를 말한다. 4장에서는 구현한 헬스 모니터의 결과와 성능을 평가한다. 마지막으로 5장에서 결론과 향후 과제에 대해서 이야기한다.

II. 관련 연구

1. ARINC 653

IMA(Integrated Modular Avionics) 시스템은 기존의 연방형 시스템의 전자장비를 하나의 컴퓨터 장비로 통합한 형태이다. 따라서 장비의 크기와 무게, 그리고 전력소모를 줄이는 효과를 가질 수 있다. 통합 모듈 항공 시스템은 서로 다른 안전 인증 레벨을 가진 응용프로그램으로 이루어진다. 이들의 안정성과 효율성을 위해서는 응용프로그램들이 실

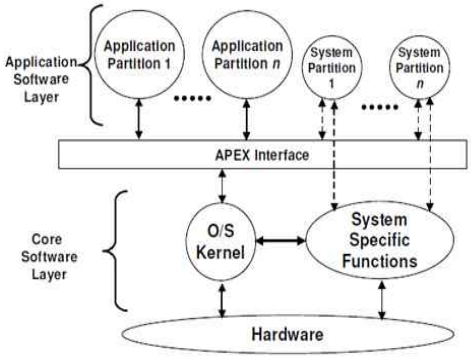

그림 1. ARINC 653 구조 [10]
Fig. 1 ARINC 653 Structure [10]

행 중에 서로의 영향을 받지 않아야 한다. 따라서 실시간 운영체제와 응용 프로그램간의 독립성을 보장할 수 있는 표준이 필요하다. 이를 위해 항공 전자 관련 학계와 업계가 모여 ARINC 653이라는 항공 전자기기를 위한 운영체제와 응용 프로그램간의 표준 인터페이스를 규정하였다. 통합 모듈 항공기반의 ARINC 653의 구조는 그림 1과 같으며 실시간 운영체제와 응용프로그램간의 인터페이스인 APEX (APplication/Executive) 를 제공한다.

ARINC 653 규격에 정의되어 있는 APEX API는 다음 6가지의 기술 조건을 만족해야 한다[11-13].

- 파티션 관리 : IMA시스템은 시간적, 공간적으로 분리하는 파티션(Partition)이라는 핵심적인 기능이 필요하다. 파티션은 물리적인 프로세서 및 메모리를 공유하기 때문에 각 파티션마다 시간과 공간을 할당해주고 관리할 수 있어야 한다.

- 프로세스 관리 : IMA시스템은 파티션내부에는 하나 이상의 프로세스들로 이루어진다. 따라서 여러 개의 프로세스들의 관계를 정확하게 구성하여 동작할 수 있도록 해야 한다. ARINC 653 표준 API에서는 파티션 내부의 프로세스를 지원 할 수 있도록 초기화, 종료, 재조정 방법을 정의하고 있다..

- 시간 관리 : 시간 관리자는 ARINC653에서 실시간 운영체제를 위해 중요한 역할을 한다. 실시간성을 보장하기 위해서 시간 관리자를 통해 주기적 스케줄링을 하는 방법을 제공하고 있다. 또한 REPLENISH 기능을 이용해서 데드라인을 이용해서 시간을 연장시켜주는 기능을 제공하고, 현재시간을 얻어오는 GET_TIME 기능 등을 제공한다.

- 파티션간 통신 (Interpartition Communication): ARINC 653은 파티션들간의 통신을 하기 위해 파티션간 통신API를 제공 한다. 파티션간 통신은 파티션간에 데이터 전송과 다른 장치와의 통신을 위해 큐잉포트와 셈플링를 사용한다. 큐잉포트의 경우에는 데이터 양이 많은 경우 사용되며, 셈플링 포트는 데이터의 양보다 속도를 중요할 때 사용 된다.

- 파티션 내부 통신 (Intrapartition Communication) : 파티션안에서 프로세스간 통신을 사용하기 위한 API로 버퍼와 블랙보드(Blackboard)가 사용된다. 버퍼방식의 특징은 사용자가 메시지 개수를 설정하여 보낼 수 있으며 다량의 데이터가 저장이 가능하다. 는 단일 메시지만 가능하며 빠른 속도가 요구되는 데이터를 처리 할 때 사용 된다.

- 헬스 모니터 API : ARINC653 APEX에는 에러가 발생되었을 사용자에게 헬스 모니터를 통해 에러 상태를 전달할 수 있는 방법이 정의되어 있다. 헬스 모니터 API는 크게 아래와 같이 3가지로 상태에서 구성해야 한다.

- 프로세스 헬스 모니터 : 프로세스상태에서 발생하는 에러를 처리하며 응용프로그램 개발자들에 의해서 에러 핸들러를 등록할 수 있다.
- 파티션 헬스 모니터 : 파티션상태에서 발생하는 에러를 처리하고 시스템 통합자(System Integrator)가 XML 설정파일을 통해 정의한다.
- 모듈 헬스 모니터 : 통합 모듈 항공시스템 또는 Core 운영체제 상태에서 발생하는 있는 에러를 식별하고 결함에 대한 처리방식을 정의한다. 이는 플랫폼 제공자가 XML 설정파일로 정의 한다

2. VxWorks 653

임베디드 시장을 선도하는 기업인 윈드리버(Wind River System)는 통합 모듈 항공 시스템을 위한 ARINC 653 기반의 실시간 운영체제인 VxWorks 653을 개발하였다[3]. VxWorks 653은 보잉 787 드림라이너를 포함하여 40가지 이상의 기체에서 전 세계 100여 곳에서 사용되고 있다[3].

VxWorks 653의 헬스 모니터는 고장이 발생되면 핸들링하는 프레임워크를 제공한다. 헬스 모니터는 이벤트를 통해 고장이 발생을 알리고, 복구 행동에 따라 복구시키는 역할을 한다. 또한 이벤트가 발생되면 시스템 헬스 모니터 테이블은 이벤트를 처리하기위해 하드 코딩된 형태로 자동적으로 처리한다.

VxWorks 653은 실제 항공기에서 많이 사용되면서 높은 안정성을 인정받았다. 하지만 비싼 로열티를 지불해야 하고, 유지보수 비용이 많이 든다. 뿐만 아니라 운영체제를 개발자에 의해 임의로 수정하거나 확장이 불가능하여 헬스 모니터의 성능개선을 할 수 없다는 단점이 있다.

3. LynxOS-178

LynuxWork. Inc가 1988년에 만든 RTOS 인 LynxOS-178 패키지[5]는 DO-178B 규격[16]의 주요 구조적 고려사항을 충족하고 있다[11]. DO-178B 규격은 항공시스템으로 안정성을 중시하는 소프트웨어의 엄격한 기술요건이다 LynxOS-178은 코어는 LynxOS 이며 UNIX와 유사한 운영체제로 설계되었다. LynxOS 와 LynxOS-178은 가장 엄격한 안정성이 요구되는 항공 시스템 표준에 기초가 되는 운영체제이다[12]. LynxOS 178에서는 파티션0 의 기능은 다른 파티션을 초기화 시키거나 실행 시키는 시스템 파티션 역할을 한다. 또한 고장이 발생되었을 때 적절하게 처리하기 위한 ARINC 653 헬스 모니터 기능도 제공하고 있다. 하지만 운영체제를 직접 수정해서 사용할 수 없고 로얄티가 비싸다는 단점을 가지고 있다[6, 7].

4. POK

POK는 안전성와 보안성에 중점을 두고 개발된 실시간 임베디드 시스템을 위한 파티셔닝 운영체제이다. POK는 마이크로 커널 구조로 개발되었으며 ARINC 653 표준뿐만 아니라 다양한 표준을 적용 가능하게 설계되었다[4]. 그리고 안정성과 보안성의 수준이 다른 응용프로그램을 실행하기 위한 파티셔닝 기능을 제공해준다. POK는 파티셔닝 기능을 제공하기 위한 ARINC 653 APEX와 스케줄링 기능을 가진 구조로 구성되어있다[13]. 만약 하나의 파티션에서 치명적인 에러가 발생되면 ARINC 653 헬스 모니터를 통해 에러를 처리할 수 있도록 API를 제공하고 있다. 하지만 아직 API가 완벽하게 지원되지는 않고 커널 레벨에서 동작하는 헬스 모니터를 제공하지 않고 있다[5].

Ⅲ. 헬스 모니터 구현

1. 전체 시스템 구조

본 연구에서 제안하는 구조는 그림 2와 같다. 리눅스에서 파티셔닝 기능을 제공하기 위해서 사용

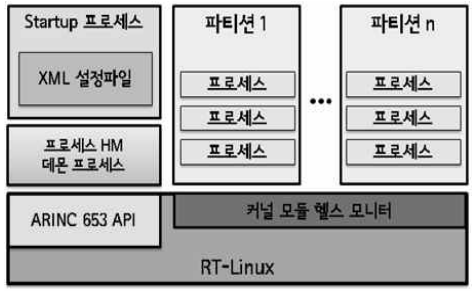

그림 2 .시스템 구조
Fig. 2 System Structure

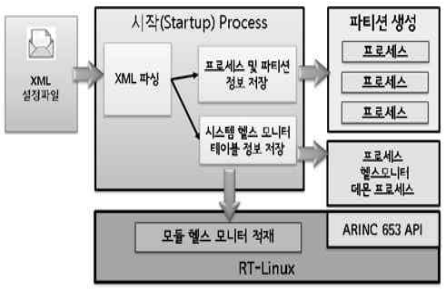

그림 3 .시작 프로세스 동작
Fig. 3 Startup process operator

자 영역의 프로세스들을 그룹으로 묶어 파티션화 할 수 있도록 구성하였다. 그리고 고장 발생을 감지 하기 위해서 사용자 영역과 커널 영역에 각각 헬스 모니터가 동작한다. 핵심이 되는 헬스 모니터는 리눅스 커널 영역에서 모듈형태로 동작하는 커널 모듈 레벨 헬스 모니터와 사용자 영역에서 데몬 프로세스 형태로 동작하는 프로세스 레벨 헬스 모니터로 구성되어있다. 또한 파티셔닝 기능을 제공하기 위한 ARINC 653 API 기능도 일부 지원한다.

시작 프로세스(Startup Process)는 각 파티션을 실행 시키고 헬스 모니터 테이블을 설정하기 위한 XML 설정파일을 가져오는 역할을 한다. 생성된 파티션들은 라운드로빈(Round Robin) 형태로 사용자 영역에서 동작한다.

2. 시작 프로세스 (Startup Process)

리눅스에서 파티셔닝 상태로 동작하기 위해서는 프로세스를 그룹화 시키며 헬스 모니터에게 테이블 정보를 전달해야만 한다. 이러한 기능은 시작 프로세스(Startup Process)를 통해 그림 3과 같이 이루어진다. 시작 프로세스는 XML 파일의 정보를 파

표 1. 시스템 상태
Table 1. System status

시스템 레벨	상태 종류
프로세스 레벨	커널에서 인터럽트가 발생된 상태
	프로세스가 수행중인 상태
파티션 레벨	파티션이 초기화된 상태
	파티션 헬스모니터가 동작중인 상태
	운영체제에서 시스템 콜이 발생된 상태
	프로세스 관리중인 상태
모듈 레벨	모듈 헬스 모니터가 동작중인 상태
	모듈 헬스 모니터 테이블 설정중인 상태
	알 수 없는 시스템 상태

싱하여 설정에 맞게 파티션과 프로세스를 생성한다. 파티션 내부의 프로세스를 생성하기 위해서는 ARINC 653 표준에 정의된 API를 호출하여 프로세스를 생성하고 같은 파티션끼리 그룹화 시켜서 하나의 파티션으로 만든다. 또한 시작 프로세스는 헬스 모니터에게 정보를 제공하기 위해 헬스 모니터 테이블 정보를 헬스 모니터에게 제공하며, 시스템 통합자에 의해서 작성된 헬스 모니터 테이블은 처리해야 하는 고장 복구 액션들이 작성되어 있다. XML정보가 저장되면 커널 모듈 헬스 모니터를 커널 영역 적재 하고 프로세스 헬스 모니터는 사용자 영역에 생성한다. 시작프로세스의 동작이 완료되면 생성된 프로세스는 라운드로빈(Round Robin) 스케줄링 방식으로 동작하게 된다.

3. 헬스 모니터 (Health Monitor)

표 1은 ARINC 653 표준에 정의된 시스템 레벨에 따른 상태에 대한 설명이다. 헬스 모니터는 각 시스템 레벨에서의 상태에 따라 고장이 발생된 것을 탐지(Detection)하고, 고장에 의해 다른 파티션에 영향을 미치지 않게 격리(Isolation) 시킨다. 그리고 XML 설정에 따라 고장을 복구(Recovery) 하게 된다.

프로세스 레벨은 어플리케이션 소프트웨어가 동작하면서 각 어플리케이션의 독립적인 작업을 처리하고 있는 상태이거나, 프로세스에 주어진 시간 중에 커널이 인터럽트를 처리할 때를 정의한다. 파티션 레벨은 파티션 초기화시 XML 설정이 초기화되는 시점과, 운영체제의 시스템 콜을 처리하는 상태를 말하며, 마지막으로 모듈레벨은 시스템 전체 또는 파티션에서 동작하는 모듈 운영체제에서 헬스모니터의 동작 상태를 의미한다.

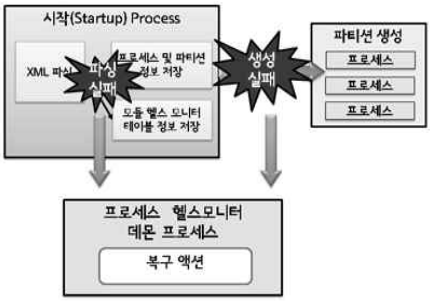

그림 4 사용자 영역 고장탐지 방법
Fig. 4 User domain fault detector method

3.1. 고장 탐지

본 연구에서는 리눅스에서 고장을 빠르게 탐지하기 위해서는 사용자 영역과 커널 영역에서 프로세스레벨, 파티션레벨, 모듈레벨의 고장을 탐지할 수 있도록 헬스모니터를 커널모듈과 사용자영역 헬스모니터 프로세스로 각각 구현하였다. 사용자 헬스모니터 프로세스는 프로세스 레벨에서 발생하는 고장을 탐지할 수 있고 파티션 레벨 발생하는 고장 중 파티션 설정이 잘못되거나 파티션 초기화가 잘못된 경우에 이를 탐지 할 수 있다. 프로세스 레벨 고장 탐지는 유저가 등록해 놓은 에러 핸들러가 있을 경우 탐지 되며, 등록되어 있지 않은 에러의 경우는 커널 영역 헬스모니터가 탐지한다. 파티션레벨에서 사용자 영역에서 발생되는 고장은 XML설정파일을 가져올 때 잘못된 문법이나 XML 파일이 아닌 다른 파일의 경우 고장으로 판단하거나 파티션 생성이 실패한 경우로 헬스모니터가 그림 4와 같이 동작한다.

커널 영역 헬스모니터는 파티션의 메이저 프레임 스케줄링 위반과 같은 파티션 고장과, 운영체제의 고장을 탐지할 수 있다. 파티션 스케줄링 시간보장을 위해 커널영역 헬스모니터는 파티션 초기화 시 XML을 통해 설정된 메이저 프레임 시간을 커널 타이머를 이용해 지난 파티션의 메이저 프레임 시간과 비교하고 파티션 상태 데이터를 이용해 파티션 고장 상태를 확인한다. 모듈레벨에서의 고장탐지를 위해 헬스모니터 커널모듈은 인터럽트 디스크립터 테이블(Interrupt Descriptor Table)을 통해 모든 트랩을 후킹하여 스택 오버플로우(StackOverflow)등 과 같은 모듈레벨에서 발생하는 고장을 감지한다. 감지된 트랩이 ARINC653 표준으로 정의된 고장인 경우 XML로 정의된 액션으로 처리하고 아닌 경우 기존의 리눅스 인터럽트 핸들러를 통해 처리하도록 그림 5와 같이 구성 하였다.

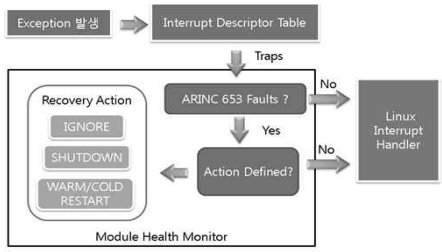

그림 5. 커널 영역 고장탐지 방법
Fig. 5 Kernel domain fault detector method

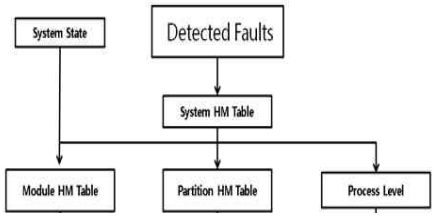

그림 6. 고장 격리 메커니즘
Fig. 6 Fault isolation mechanism

3.2. 고장 격리

통합모듈 항공 시스템에서 고장이 발생되면 다른 파티션에 영향을 미치게 되어 전체 시스템에 이상이 발생되는 경우가 있다. 이러한 문제를 방지하기 위해서 그림 6과 같이 시스템 상태 별로 격리시켜 처리한다.

모듈레벨에서의 고장을 탐지했을 때 시스템 구동 시 등록되어 있는 고장일 경우, 어느 파티션, 어느 프로세스에서 고장을 발생시켰는지 확인하고 이 파티션과 프로세스의 상태를 고장으로 변경하여 파티션과 프로세스를 격리시킬 수 있다. 시스템 하드웨어나 리눅스 커널 자체에 에러가 발생했을 경우에 대해서는 본 논문에서는 고려하지 않았다. 하지만 리눅스 커널이나 하드웨어 에러 역시 핸들러를 등록할 수 있게 설계 하였으며 등록된 에러를 감지할 경우 별도로 작성된 핸들러에 의해 격리가 가능하다.

파티션 초기화 시 에러가 발생했을 경우에는 파티션 생성을 하지 않고, 시스템을 구성한다. 모든 파티션이 생성된 후 파티션 고장이 발생되었을 경우, 헬스모니터는 파티션 실행시간 동안 수 있는 시간 동안에는 파티션을 복구하고, 고장상태로 다른 파티션 시간을 쓰지 않도록 격리 시킨다.

프로세스가 고장이 날 경우, 해당 프로세스의 상태를 고장으로 변경하고 프로세스의 고장이 해당

표 2. 파티션 복구 행동
Table 2. Partition Recovery Action

복구 액션	정의
IDLE	파티션을 Sleep 상태로 변경
COLD_START	파티션을 초기화하고 재시작
WARM_START	기존의 데이터를 두고 특정 시점에서 재시작
IGNORE	발생된 고장 무시
USER ERROR HANDLER	사용자에 의한 복구 핸들러 등록

표 3. 모듈복구 행동
Table 3. Module Recovery Action

복구 액션	정의
SHUTDOWN	전체 시스템 종료
RESET	전체 시스템 재실행
IGNORE	발생된 고장 무시

표 4. 헬스모니터를 ARINC 653 API
Table 4. Implemented ARINC 653 APIs for Health Monitor

APIS	정의
GET_PARTITION_STATUS	파티션 상태 정보를 얻어 온다
SET_PARTITION_STATUS	파티션 동작모드를 설정한다
CREATE_PROCESS	파티션 내부의 프로세스 생성
SET_PRIORITY	프로세스의 우선순위 설정
SUSPEND	프로세스를 중단
RESUME	프로세스를 다시 실행
REPORT_APPLICATION_MESSAGE	에러 발생시 사용자 영역으로 메시지 보냄
CREATE_ERROR_HANDLER	사용자에 등록된 에러 핸들러 등록
GET_ERROR_STATUS	현재 발생된 에러상태 얻어옴
RAISE_APPLICATION_ERROR	프로세스 헬스모니터로 에러 전달

파티션의 고장에 영향을 미치지 않도록 다른 프로세스들은 정상적으로 스케줄링 한다.

3.3. 고장 복구

헬스 모니터에 의해서 탐지된 고장은 각각의 모듈, 프로세스, 파티션 테이블에 정의된 복구 행동에 따라 처리 한다 파티션레벨은 표 2에 정의와 같이 처리한다. COLD_START 의 경우 고장이 발생된 파티션에 속한 모든 프로세스를 다시 생성하게 하여 기존에 데이터를 모두 제거하고 파티션을 재시작하는 방식을 말한다. WARM_START의 경우 고장이 발생되면 파티션에 속한 모든 프로세스의 자료구조를 저장하고 지정된 시점에서 다시 시작한다. 즉 기존의 데이터를 초기화 하지 않고 특정 시점에서부터 시작되는 방법이다. IGNORE의 경우 무시해도 될 정도의 소프트에러인 경우로 고장으로 탐지되었지만 아무런 처리를 하지 않고 시스템을 수행한다. 마지막 UserError Handler 의 경우 사용자가 추가로 처리하고 싶은 처리 방안을 설정하고 핸들러로 추가 등록하여 처리하는 방식이다. 본 연구에서 개발한 헬스 모니터는 개념증명(proof-of-concept)을 위한 시제품으로 WARM_START 는 구현되지 않았다.

모듈레벨은 표 3의 정의된 바와 같이 SHUTDOWN 기능을 통해 종료시키거나 RESET 을 통해 다시 시작하거나 IGNORE를 통해 무시하는 방법으로 고장을 복구한다.

4. 헬스 모니터 테이블 설정

파티션과 헬스 모니터는 XML을 통해 파티션정보를 입력 받는다. 파티션정보는 Partition Name, Duration, Period, Partition ID를 포함하며, 프로세스는 Process Name, Deadline, Base_Priority, Period, Process ID 정보등이 포함하여 XML설정을 하도록 구성하였다. 헬스 모니터 테이블은 각 상태 별로 고장을 정의하고 상태에 따라서 복구 액션에 관한 정보를 제공한다. 헬스 모니터 테이블의 상태는 크게 모듈 헬스 모니터 테이블, 파티션 헬스 모니터 테이블, 프로세스 헬스 모니터 테이블로 나누어 입력 받는다. 에러의 경우 에러 행동(Error Action)을 통해 정의하고 복구행동(Recovery Action) 을 통해 복구상태를 XML로 구성하였다.

5. ARINC 653 API

본 논문에서 ARINC 653 헬스 모니터를 통해 고장을 처리하는 것에 구현에 초점을 맞췄기 때문에 헬스 모니터에 필요한 API 와 파티션과 프로세스 관련 API만 표 4와 같이 구현하였다.

Ⅳ. 구현결과 및 성능평가

1. 동작 환경

본 논문에서 개발한 리눅스 기반의 ARINC 653 헬스 모니터는 인텔 x86 프로세서에서 동작할

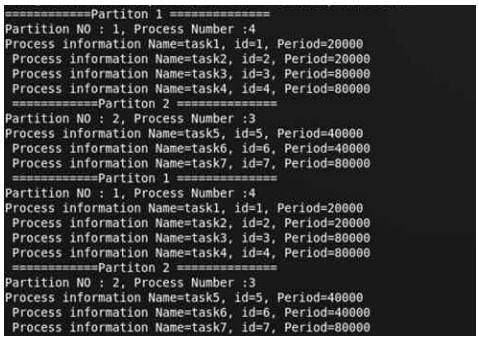

그림 7. 시작 프로세스 동작화면
Fig. 7 Startup process operating screen

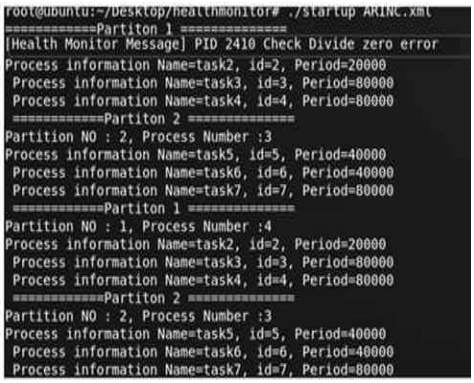

그림 8. 헬스모니터 동작 화면
Fig. 8 Health Monitor operating screen

수 있게 개발하였다. 또한 실시간성을 높이기 위해서 우분투 리눅스 11.04(커널 2.6.32) 버전에 RT 패치를 적용하여 각각의 프로세스들은 라운드로빈 스케줄러(SCHED_RR)를 통해 프로세스 스케줄링 하도록 설정하였고, 각 파티션들은 전체 Major Frame 을 가지고 동작하도록 설장하였다.

2. 구현 결과

리눅스 기반 ARINC 653 헬스모니터의 기능을 확인하기 위해 가상 파티션 두 개를 생성하고 가 파티션1 에는 4개의 프로세스를 파티션2 에는 3개의 프로세스를 실행시켰다. 그리고 프로세스들은 더미(Dummy) 프로세스들로 단순히 화면 출력만을 수행하며 라운드 로빈 형태로 실행되도록 하였다. 파티션 Major Frame 의 시간은 각각 36ms 씩 동작하도록 설정했고, 그림 7은 2개의 파티션이 라운드로빈 형태로 스케줄링 하면서 동작하는 것을 보여준다.

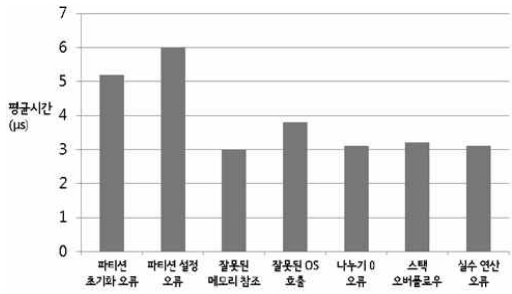

그림 9. 헬스모니터 고장탐지 시간
Fig. 9 Health Monitor fault detector time

본 연구에서 개발한 헬스모니터는 WARM_START 가 구현되어 있지 않으므로 COLD_START 의 기능만을 검증했다. 이를 위해 파티션 1에 나누기0(Divide by zero) 에러를 발생시켰을 때 헬스모니터가 처리되는지 확인해보았다. 그림 8의 첫 번째 동작에서 파티션 1의 task1 프로세스가 고장이 났을지라도 고장격리로 다른 프로세스와 파티션은 정상적으로 동작하는 것을 볼 수 있다. COLD_START 복구행동으로 본 실험에서는 SHUTDOWN 으로 task1 프로세스를 종료시켰으며 프로세스영역에 존재하는 헬스 모니터는 사용자에게 고장 정보를 알리기 위해서 사용자 영역으로 그림 8과 같이 메시지를 출력한다.

3. 성능평가

성능평가는 인텔 x86 프로세서에서 우분투 리눅스 11.04 (커널 2.6.32) 에서 동작 하였다. 그리고 헬스 모니터의 성능을 측정하기 위해서 고장을 탐지하는 시간과 실제 ARINC 653 API가 미치는 오버헤드에 대해서 측정하였다.

3.1. 헬스모니터 고장 탐지시간

헬스모니터의 고장 탐지 시간은 고장이 발생된 시간부터 헬스모니터에서 복구 핸들러가 동작하기 전까지의 시간을 측정 하였다. 실험 결과는 그림 9와 같다. 사용자영역 헬스모니터 프로세스의 파티션 초기화 오류와 파티션 설정오류를 탐지 시간을 측정하기 위하여 각각 50회 실행하여 평균 5.2μs 와 6 μs 의 시간이 소요되었다. 나머지 오류들은 커널모듈 헬스모니터에 의해서 탐지된 결과이다. 프로세스 헬스모니터가 커널모듈 헬스모니터에 비해 더 많은 시간이 소요되었는데, 이 부분은 프로세스간 통신시 커널영역 전환이 필요하기 때문이다. 평균 고장탐지시간과 마이크로초(μs) 단위로 이는 우리가 구현한

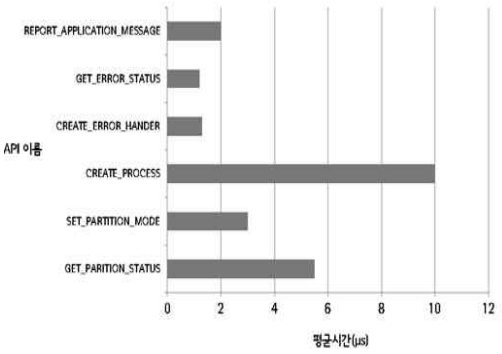

그림 10. ARINC 653 API 오버헤드
Fig. 10 ARINC 653 API Overhead

헬스모니터의 탐지시간이 일반적인 실시간 시스템이 요구하는 지연시간(latency)인 10~20 마이크로초 응답시간[16]을 만족할 수 있음으로 보여준 결과이다.

3.2. ARINC 653 API 오버헤드

ARINC 653 API 오버헤드를 측정하기 위해 API가 호출된 시점부터 API 동작이 완료되는 시간을 그림 10과 같이 측정하였다. ARINC653 표준 API중 CREATE_PROCESS 가 오버헤드의 측정결과가 가장 많은 시간인 10μs가 소요되었다. 이는 프로세스를 생성하는데 걸리는 시간이므로 헬스모니터와는 직접적인 연관이 없는 오버헤드이다. 본 논문에서 중요하게 살펴본 부분은 헬스모니터관련 API로 이는 2μs이하 정도로 측정되었다. 따라서 헬스모니터 API에서 발생되는 오버헤드 역시 마이크로초 단위로 실시간 리눅스의 지연시간 범위 내의 결과를 보여 주었다.

Ⅴ. 결론 및 향후 계획

본 논문에서는 통합 모듈 항공 시스템을 구성하기 위해 리눅스 기반에 ARINC 653 헬스모니터를 설계하고 개발하였다. 리눅스는 기존의 실시간 운영체제와 다르게 오픈 소스 기반으로 개발비용과 유지보수비용을 줄일 수 있다. 또한 확장과 수정이 용이하여 성능개선을 할 수 있었고 개발비용과 유지보수비용을 줄이는 효과를 가져올 수 있었다.

본 연구에서는 개발한 ARINC 653 헬스모니터는 고장을 탐지시간이 전체 평균 3 μs 의 빠른시간을 보였고, ARINC 653 표준에 정의된 9개의 고장 중에 7개의 고장을 처리할 수 있었다. 체크 포인트에 의한 WARM_START를 구현은 향후 계획 중에 있으며, 다양한 아키텍처에서 지원가능 하도록 할 예정이다.

References

[1] C.H. Song, “Focus on Advanced Avionics Technology,” IT SoC Magazine No. 34, pp.24-31. 2009 (in Korean).

[2] S.H. Han, J.S. Seok, H.W. Jin, “A Partition Scheduling Scheme to Support Efficient Mixed Partitioning,” Journal of KIISE : Computer Systems and Theory, Vol. 19, No. 2, pp.85-89, 2013 (in Korean).

[3] VxWorks 653, http://www.windriver.com/products/platforms/safety_critical_arinc_653/ (accessed on 20 on Feb 2014)

[4] POK, http://pok.safety-critical.net/ (accessed on 20 on Feb 2014)

[5] LynxOS-178 http://www.lynuxworks.com/rtos/rtos-178.php (accessed on 20 on Feb 2014)

[6] M. Masmano, Y. Valiente, , P. Balbastre, I. Ripoll, A. Crespo, J. Metege, "LithOS: a ARINC-653 guest Operating for XtratuM," Proceedings of Realtime Linux Workshop, 2011.

[7] Y.K. Ko, S.H. Lee, S.Y. Park, C.B. Ban, D.L. Kang, J.Y. Jeong, C.H. Lee, “HM System Design for Fault Tolerance on the IMA System,” Journal of Korea Contents Association, Vol. 12, No. 8 pp.77-86 2012 (in Korean).

[8] S.H. Han, H.W. Jin, "Kernel-Level ARINC 653 Partitioning for Linux", Proceedings of ACM Symposium on Applied Computing, pp.1632-1637, 2012.

[9] H.W. Joe, H.A. Jeong, Y.I. Yoon, H.S. Kim, S.H. Han, H.W. Jin. "Full virtualizing micro hypervisor for space flight computer," Proceedings of IEEE/AIAA Conference on Digital Avionics Systems, pp.1-24, 2012.

[10] Arinc Specification 653, Airlines Electronic Engineering Commitee "Avionics Application

Software Standard Interface," ARINC, 2010.
[11] S., Serigio, R. Jose, S. Tobias, T. Cassia, W. James, "A Portable ARINC 653 standard interface," Proceedings of IEEE/AIIA Conference on Digital Avionics System, pp.1.E.2.1-7, 2008.
[12] P. Paul, L. Kinnan, "Safety-critical Software development for Integrated modular avionics," Embedded System Engineering, Vol. 11, No. 7 pp.40-41, 2003.
[13] ARINC report 653 "Design Gudance for Integrated Modular Avionics" Aerionautical Radio Inc. Annapolis MD 1991.
[14] S. Slawomir, "ARINC Specification 653 Based Real Time Software Engineering," e-Informatica Vol. 5, No. 1, pp.39-49 2011.
[15] RCTA. DO-178-B, "Software Consideratons in Airborne Systems and Equipment Certification," Technical Report RCTA Paper No. 548-92/SC167-177, RCTA, 1140 Connecticuit Avenue, Wasgington D.C., 1992.
[16] K. Kushal, "Myths and realities of real-time linux software systems," Proceedings of Real-Time Linux Workshop, pp.13-18, 2009.

저 자 소 개

윤 영 일

2005년, 충남대 전기공학과 학사.
2011년, 충남대 컴퓨터공학과 석사.
현재, LIG 넥스원 통신연구센터 연구원.
관심분야: Fault-Tolerance, 임베디드 소프트웨어, 항공전자 소프트웨어
Email: youngil.yoon@lignex1.com

조 현 우

2005년, 충남대 정보통신공학부 컴퓨터 학사.
2007년, 충남대 컴퓨터공학과 석사.
2014년, 충남대 컴퓨터공학과 박사.
현재, 충남대 소프트웨어연구소 전임연구원.
관심분야: 임베디드 시스템 가상화, 항공우주 시스템, Resource Aware Computing, 저전력 시스템
Email: jhwzero@cnu.ac.kr

김 형 신

1990년, 한국과학기술원 전산학과 학사.
1991년, Univ. of Surrey 위성통신학과 석사.
1992년~2001년, 한국과학기술원 인공위성 연구센터 선임연구원.
2003년, 한국과학기술원 전산학과 박사.
2003년~2004년, Carnegie Mellon University 박사후연구원.
현재, 충남대 컴퓨터공학과 교수.
관심분야: 항공우주 시스템, 저전력 컴퓨팅, 임베디드 시스템 소프트웨어.
Email: hyungshin@cnu.ac.kr