결함허용 실시간 시스템 구조에 대한 설계 및 구현
Desgin and Implementation of a Architecture For Fault-Tolerant and Real-Time System

유종상 · 김범식 · 신인철
(단국대학교 전자공학과)

ABSTRACT

A real-time operating system has focused primary on techniques to minimize processing time, with a secondary emphasis on system reliability issues. Conversely, fault-tolerant system has concentrated on using recourse and information redundancy to maximize the availability and reliability of the system, with a lesser emphasis on performance.

We have developed a fault-tolerant and real-time operationg system which support a powerful concurrent runtime environment under the above requirements.

In this paper, we present an overview of real-time systems, design and implementation of a duplex architecture using advanced concepts and technologies such as fast "fault detection","fault isolation" and "fault recovery".

Because the duplex architecture has two dentical hardware elements and has several recovery steps and hierarchy to facilitate a fast recovery which must be preceeded by a prompt fault detection and isolation. Thus it makes possible to minimize the overhead of the systems including hardware and software and guarantee the service continuity of the systems.

Ⅰ. 서 론

신뢰도 높은 컴퓨터 시스템의 필요성은 전혀 새로운 것이 아니다. 초기 컴퓨터 시대부터 컴퓨터 시스템의 신뢰도를 증가 시키기 위해 연산을 두 번 실행한 후, 그 결과를 서로 비교하여 일치할 경우에만 그 연산 결과를 사용하였고, 동일한 기능의 연산장치(ALU)를 이중으로 구성하여 연산 결과를 비교하는 기법으로 컴퓨터 시스템의 신뢰도를 증가시켜 왔다. 하드웨어의 신뢰도 향상으로 주어진 시간 이내에 응답 및 처리를 요

구하는 마이크로 프로세서 시스템 개발이 국내외에서 활발히 진행되고 있다.[1][3]

정보화 사회에서는 방대한 데이타의 전송 및 처리가 필연적이며 대량의 데이타 전송을 위한 정보의 고속도로 구축과 함께 데이타의 신뢰도 높은 처리를 위한 컴퓨터 기술 개발이 시급하며 시스템 결함에도 불구하고 지속적으로 운영되는 컴퓨터 시스템의 필요성은 다양한 분야에서 요구되고 있다. 따라서 결함(Fault), 장애(Error), 고장(Failure)이 시스템에서 발생할지라도 주어진 임무를 올바르게 수행할 수 있는 신뢰도와 가용도 높은 결함허용 시스템의 개발은 연구해야 하며, 재산과 인명 피해를 방지하기 위하여 실시간 제어에서는 반드시 필요하다.

실시간 시스템의 기본 기능들은 일반적인 컴퓨터 시스템과 달리 시간에 대한 제약을 지키는 것이 중요하다. 다시말하면, 시스템의 정확성을 요하는 계산 기능에 있어서 논리적인 결과뿐 아니라 그 결과를 만들어내는 시간의 정확성이 필요한 것이다. 특히, 실시간 응용에서 한 프로세서의 장애는 전 시스템의 정지를 유도하여 비싼 자원에 대한 치명적인 손상을 입히게 되며 인명 피해까지 유발할 수도 있다. 이런 점에서 실시간 시스템 개발에 있어서의 고신뢰 시스템의 구조의 설계는 중요한 연구 의미를 갖는다.[2][6][7]

실시간 시스템은 우리의 사회에서 중요한 역할을 담당하고 있으며, 실시간 시스템을 통하여 풀고 있는 영역은 소형에서부터 매우 복잡한 대형에 이르기까지 다양하다. 실시간 시스템의 응용은 분야별 또는 기능별로 분류되며 아래와 같다.

(1) 분야별 분류

- 자동화 및 제어 : 공장자동화, 전력, 수도, 가스, 핵발전 제어
- 통신 : 전전자 교환, EDI, 위성통신, 무선통신, 이동통신
- 교통 : 항공, 위성, 로켓트, 고속열차등의 제어
- 방위 : 미사일 발사/격추, 미확인 물체 추적/탐사
- 의료 : 의료기기, 환자 감시

(2) 기 능

- 전전자 교환 시스템
- 고속 데이터 수집 시스템

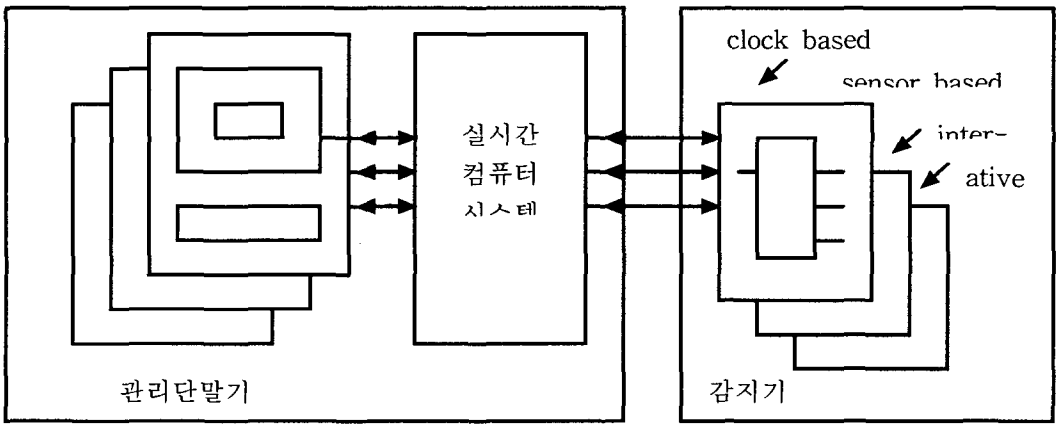
- 공정 제어 시스템

위와 같은 실시간 응용중에서 24시간 가동 제어 시스템, 통신 시스템, 그리고 방위 분야의 시스템들은 특히 결함허용 실시간 처리 기법이 요구되며 시스템 구조에 따라 여러가지 결함 허용 기법들의 연구가 활발히 진행되고 있다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 결함허용 실시간 처리 시스템을 설명하고, Ⅲ장에서는 설계된 결함허용 실시간 시스템의 결함분류, 결함검출, 결함분리, 결함복구 및 시스템 재구성을 설명한다. Ⅳ장에서 제안한 결함허용 실시간 시스템의 성능을 분석한다. 그리고 Ⅴ장에서 결론을 맺는다.

Ⅱ. 결함허용 실시간 시스템

일반적으로 실시간 시스템은 제어부와 피제어부로 이루어져 있다. <그림 1>에 표현한 것처럼 피제어부는 실제 환경을 의미하며 제어부는 피제어부의 상태를 주기적으로 감지하기 위한 감지기와 감지된 정보로부터 필요한 동작을 지시하는 실시간 내장 시스템(real-time embedded system) 그리고 피제어부의 상황을 관리자에게 알려주고 관리자로부터 명령을 받기 위한 관리 단말기로 구성된다.



<그림 1> 실시간 처리 시스템

그리고 실시간 시스템에 결함허용을 부가하는 방법으로는 하드웨어의 중복, 소프트웨어의 중복, 또는 정보의 중복이 사용될 수 있다. 실시간 시스템에 결함 허용성을 부여하기 위해서는 부가되는 하드웨어 비용 또는 시스템 성능 저하는 어느 정도 희생을 감수하여야 한다.[3][4]

앞에서 언급한 것처럼 결함 허용을 부여하는 방법에 따라서 하드웨어 기법과 소프트웨어 기법, 그리고 이들 두 기법을 혼합한 혼합 기법(hybrid mechanism)이 있다. 본 논문에서는 혼합 기법을 사용한 이중화 구성으로 고신뢰 실시간 시스템을 구현하는 방법을 제시한다. 하드웨어 중복기법은 오류 발생에 대한 신속한 대응으로 보우팅을 통하여 오류 파급을 막고 오류검출에 대하여 결함을 고립, 시스템 재구성 그리고 복구를 행하기 위한 예비 장비로 대체하고 하드웨어의 비용을 최소화한다. 오류의 파급으로 인한 시스템 파괴방지는 검사점을 이용한 소프트웨어 기법을 사용한다.

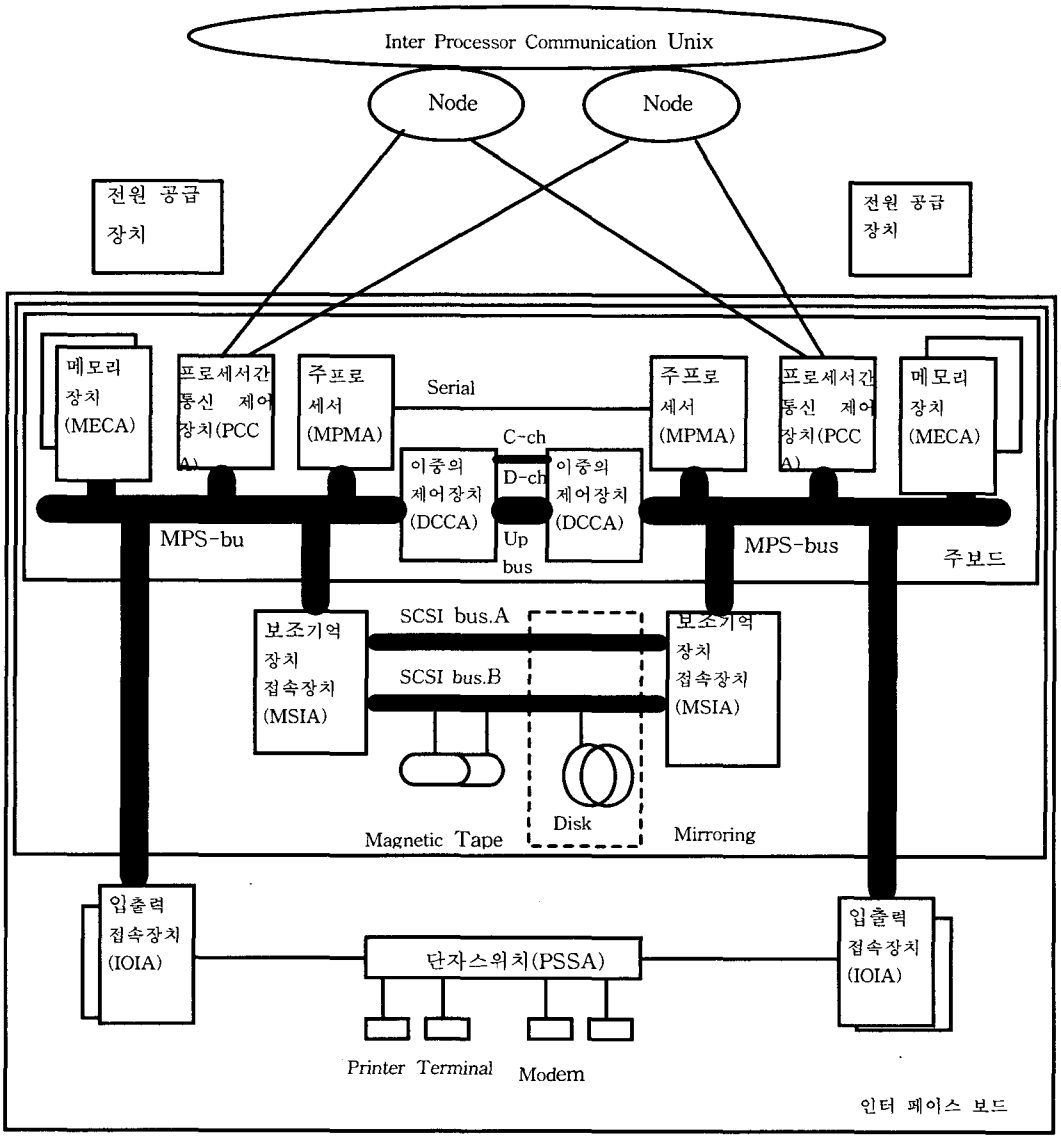
본 논문에서는 실시간 처리를 고려하여 주로 하드웨어로 결함 탐지를 수행하고 소프트웨어로 결함을 복구함으로서 하드웨어의 비용을 줄이고 최소 소프트웨어 부담으로 결함 허용을 얻고자 하였다.

2.1. 하드웨어 구성

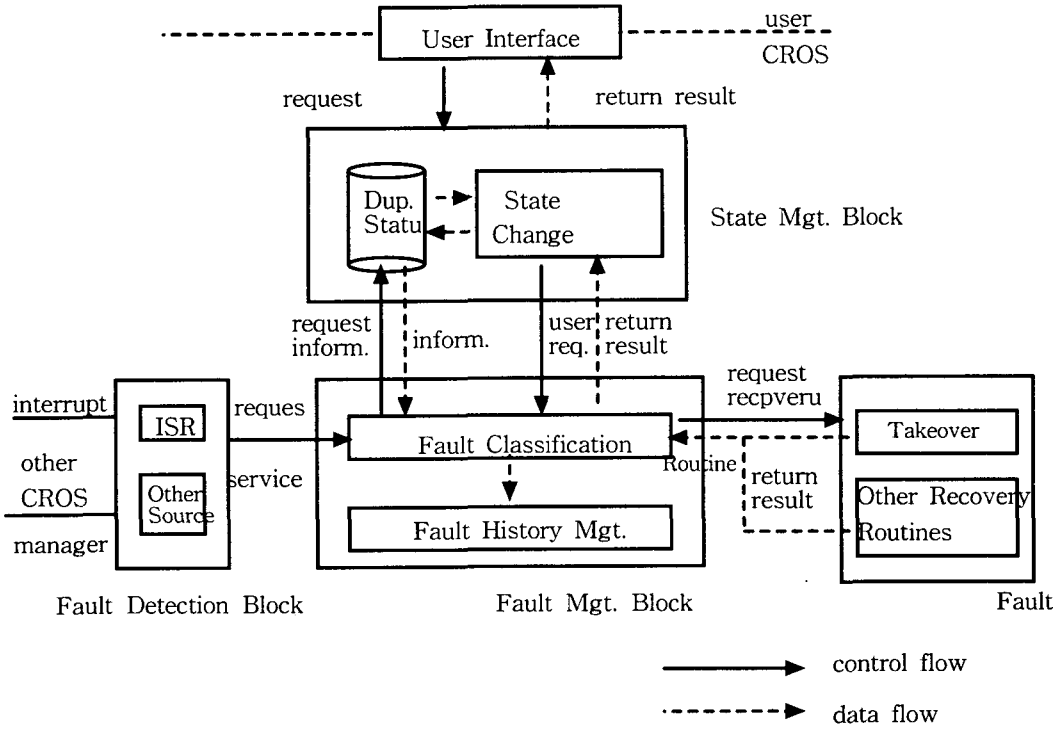
<그림 2>에서 CPU와 메모리를 가지는 주 제어 보드, 프로세서간 통신을 담당하는 통신 제어보드, 보조기억장치를 관리하는 보드, 입출력 인터페이스를 담당하는 일종의 멀티플렉서 보드, 이중화된 프로세서간 통신 채널을 제공해주는 보드가 대칭적으로 이중화 구성을 갖는다. 이중화된 양측은 개별적인 파워를 가지며, 버스도 각기 독립되어 있다. 한편 이중화된 양측 프로세서간에는 3개의 채널(D-채널, C-채널, S-채널)이 존재하며 IPC통신도 이론적으로 가능하다.

D-채널과 C-채널은 이중화 제어 보드에 의해 통제되며, S-채널은 주 제어 보드 간에 직접 연결되어 있는 시리얼 채널이다. 프로세서 간의 메모리 동기를 위해 사용하는 D-채널은 단방향의 고속 전송로로서 VME 버스의 연장이며 마스터/슬래이브(Master/Slave)의 형태로 운영된다. 만약 한 쪽 이중화 제어 보드를 마스터로, 다른 쪽 이중화 제어 보드를 슬래이브로 세트하면 마스터 측에서 슬래이브 측으로의 단방향 채널이 형성된다. C-채널은 프로세서 간의 통신 및 동기를 위해 사용하는 채널로서 양방향 통신이 가능하며 최대 1K byte의 정보를 전달할 수 있고 인터럽트 드리븐(Interrupt-driven)으로 운영된다. S-채널은 주 제어 보드간에 직접 연결된 시리얼 채널

로서 전송속도가 다소 늦은 단점이 있으나 비교적 안정하며, 이중화 제어 보드의 오동작이나, 시스템이 버스를 사용하지 못할 정도의 큰 손상을 입은 경우에도 사용할 수 있다.



<그림 2> 실시간 시스템 하드웨어 구조



<그림 3> 이중화 소프트웨어 구성

2.2. 소프트웨어 구성

(1) 이중화 초기화 및 상태 관리기능

이중화를 위한 이중화 제어 보드의 초기화 및 이와 관련된 상태 관리 기능을 수행하는 것의 주요 역할은 실행/대기(Active/Standby) 프로세서 결정 기능, 이중화 제어보드의 인터럽트 관련 레지스터 초기화 및 인터럽트 서비스 루틴(이하 ISR)의 등록기능, 대기(Standby) 프로세서의 초기화 및 운용에 관련된 기능이 이에 해당된다.

(2) 이중화 상태 천이 제어 기능

이중화 절체, 스탠바이 로딩(Standby loading), 프로세서 재시동 등 이중화 상태 변화를 유 발시키는 기능을 수행하며 그 결과를 관리하고 장애 메시지를 송신한다. 또한 양 프로세서 메모리 상태 동일화를 위한 대기 프로세서로의 메모리 복사 기능도 이에 해당된다.

(3) 장애 감지 기능

장애 감지를 위한 인터럽트의 정리, ISR의 등록에 관련된 기능 및 주기적인 감시에 의한 더티 폴트(Dirty Fault)의 검출 기능이 이에 포함된다.

(4) 장애 분류/처리 기능

검출된 장애를 장애 정도와 시스템에 미치는 영향 등을 고려하여 Minor, Major, Critical 장애로 구분하고 각각에 대한 장애 처리 루틴을 등록하여 운영하며, 적절한 장애 메시지를 송신한다. 치명적인 장애 발생시 이중화 절체를 하는 기능은 이 기능의 일부분으로 볼 수 있지만 그 중요도를 감안하여 별개의 기능으로 취급한다.

(5) 사용자 인터페이스

사용자 프리미티브를 입력받아 적절한 조치를 취하거나 관련 정보를 보여주는 기능으로서 확장성을 고려하여 추후 사용자의 요구에 대처하기 쉽도록 구성하였다.

(6) 이중화된 프로세서간 통신기능

이중화된 프로세서간의 동기 및 정보 전달을 위해서 제공되는 채널의 하드웨어 초기화, 메시지 구성, 송수신 방법등에 관련된 기능을 포함한다.

이중화 시스템 구현에 관련한 상태는 크게 실행/대기(Active/Standby) 결정상태, 운영체계 로딩 상태, 시스템 초기화 상태, 동기화 상태, 정상 상태, 이중화 절체 상태로 구분되며 실제의 각 상태 천이는 특정 사건의 조건에 따라 해당하는 행위를 완료함으로서 이루어진다.

III. 이중화 구조의 설계 및 구현

3.1. 이중화 구성을 위한 고려사상

이중화 구성의 실시간 시스템에 있어서 특별히 고려하여야 할 사항은 다음과 같은 것이 있다.

첫째, 신뢰성 향상을 가져오는 이중화 운용과 실시간 시스템의 기능의 연속성을 최대한 보장해야 한다. 둘째, 결함 허용에 중점을 두면서 실시간성을 보장해야 한다. 즉, 시

간 제약성을 고려하여 이중화 관련 소프트웨어가 응용 소프트웨어의 수행에 큰 영향을 미치지 않도록 이중화 운용으로 인터럽트가 막는 시간을 최소화해야 한다. 셋째, 이중화 운용으로 수반되는 하드웨어 및 소프트웨어의 부담을 최소화하여 성능에 큰 영향을 미치지 않아야 한다. 그 밖에 개발 중 또는 개발 후의 경우라도 추후 하드웨어의 변경이나 소프트웨어의 기능 추가 요구가 있을 수 있으며, 이를 위해 하드웨어 관련 부분 및 프리미티브 인터페이스 부분을 모듈화 하여 하드웨어 변경시나 소프트웨어의 기능 추가 욕구가 있을 수 있으며, 이를 위해 하드웨어 관련 부분 및 프리미티브 인터페이스 부분을 모듈화하여 하드웨어 변경시나 소프트웨어 기능 추가시 최소의 노력으로 소프트웨어 변경이 가능하여야 한다.

이중화 구성의 실시간 시스템 설계시 사전에 결정해야 할 사항들에 다음과 같은 것이 있다.

- 중복에 대한 운용을 어떻게 할 것인가?
- 이중화 구성의 양측 데이타의 동일성을 어떻게 보장할 것인가?
- 시스템의 결함이 발행했을 때 이를 어떻게 검출할 것인가?
- 시스템의 결함 검출에 대하여 재구성 또는 복구를 어떻게 할 것인가?
- 이중화 운용시 성능은 어느 정도인가?

본 논문에서 구현 및 분석하는 실시간 시스템에서 이중화 구조는 실행/대기(Active/Standby) 구조를 기본으로한 웜 스탠드바이(Warm Standby)로 운용하되, 양측 데이터의 동질성 유지를 위하여는 이중화 제어 보드를 통한 인스트럭션 레벨(Instruction Level) 동기화를 사용하여 검출하며 오류 복구는 소프트웨어 기법으로 설계하였다.

3.2. 결함 허용을 위한 이중화 관리의 흐름

(1) 결함 분류(Fault Classification)

결함(Fault)은 에러를 발생시킬 가능성이 있는 기계적 논리적 원인으로 고장(Failure)를 유발 시키는 하드웨어, 소프트웨어의 에러의 상태를 의미한다. 그리고 결함은 그 특성 및 지속시간에 따라 복구불능(Permanent), 복구가능(Transient)한 결함으로 분류한다.

(2) 결함 검출(Fault Detection)

대부분의 자체 결함복구(Self-recoverable Fault)와 그레이스 폴트(Graceful Fault)는 하드웨어 인터럽트등으로 신속히 검출하고 있으며 더티 폴트(Dirty Fault)의 경우는 다음과 같은 방법으로 완벽을 도모했다. 현재 주 제어 보드의 감시 인터럽트가 신속한 결함 감지를 위한 워치독(Watch Dog Timer)의 역할을 하고 있으며, 주기적인 감시를 통해 실행 프로세서 모듈의 오동작을 감시하고 있다.

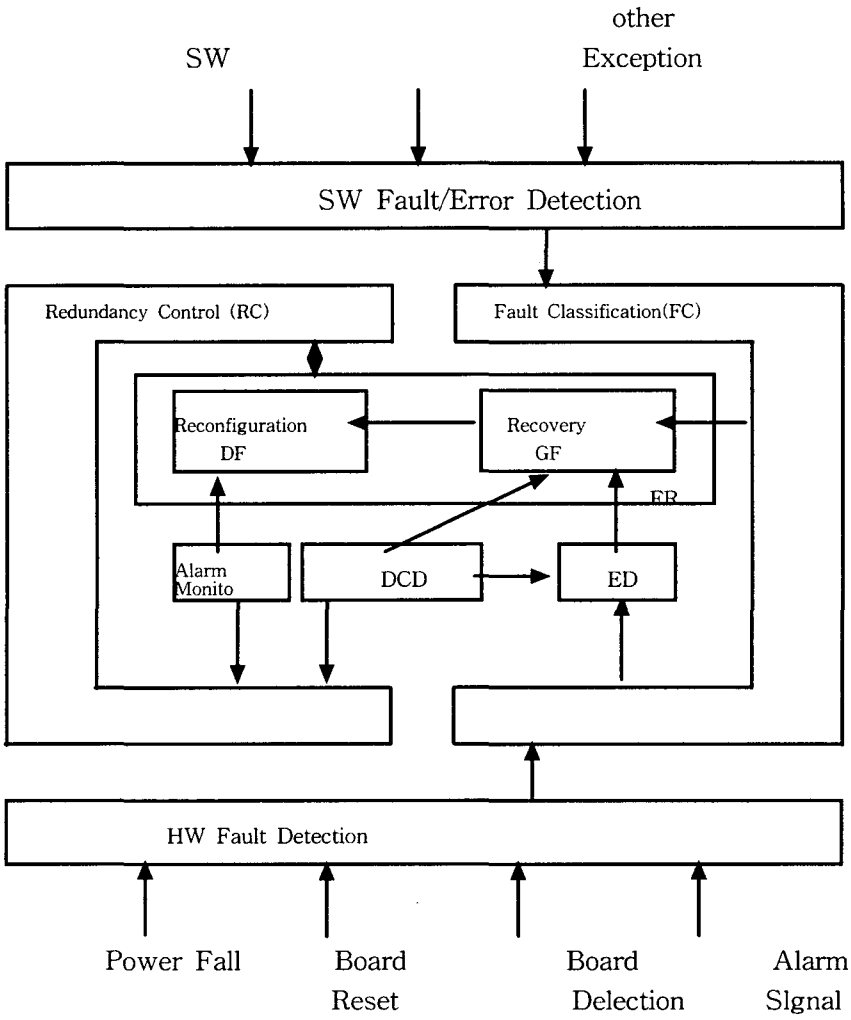
(3) 결함 복구와 재구성(Fault Recovery and Reconfiguration)

검출 분류된 Fault들에 대한 복구 행위는 결함 복구 블럭(Fault Recover Block)에서 수행된다. 복구행위의 하나인 절체 기능은 대기 프로세서가 결함이 검출되어 제어를 잃었거나 치명적으로 손상을 입은 실행 프로세서의 프로그램을 수행하는 것을 의미한다. 실행 프로세서에서 수행되던 프로그램 기능이 연속성을 보장할 수 있을지 아닐지는 그 발생 결함의 성격에 좌우된다. 이중화 절체, 프로세서 재시동(Restart), 소프트웨어 대체를 위한 스탠드바이 로딩(Standby loading)등이 포함된다. 특별히 여기서 중요한 기능은 이중화 절체 기능으로서 그 정의와 의미는 다음과 같다.

이중화로 운용되는 시스템에서 실행(Active) 프로세서의 하드웨어적인 치명적인 고장 발생시 현재 수행되고 있는 사용자 프로그램에 영향을 주지 않는 방법이 요구된다. 아울러 실행 프로세서의 하드웨어 상태를 점검하고자 하는 경우 등에 제어를 대기 프로세서로 넘겨 실행 프로세서를 동작하게 하고 자신은 대기 프로세서 상태로 천이되거나 혹은 자가 진단 상태로 천이되어 원하는 작업을 수행할 수 있도록 하는 기능이 반드시 필요하다. 실행 프로세서의 요구 또는 고장시 대기 프로세서로 하여금 실행 프로세서가 되어 프로그램을 수행할 수 있도록 제어를 넘기는 기능을 이중화 절체 기능이라고 한다. 이중화 절체 기능은 다음과 같이 대별된다.

- 그레이스 폴트등과 같은 경우에는 연속성을 보장하는 이중화 절체.
- 더티 폴트의 경우 재시동을 요구하는 이중화 절체.

구현한 결함 허용 실시간 시스템은 하드웨어의 중복을 근간으로 이중화 운용을 통하여 결함 허용의 요구 조건을 충족시키고 있으며, 이는 결함의 발생시 하드웨어의 인터럽트를 통하여 감지하여 발생 위치를 찾아내고, 다른 부분에 영향을 주는 것을 막는 결함 분리, 결함 발생 부분을 적절히 복구하거나 그 오류가 시스템에 영향을 미치지 않도록 하는 결함 복구는 소프트웨어로 구현한다.



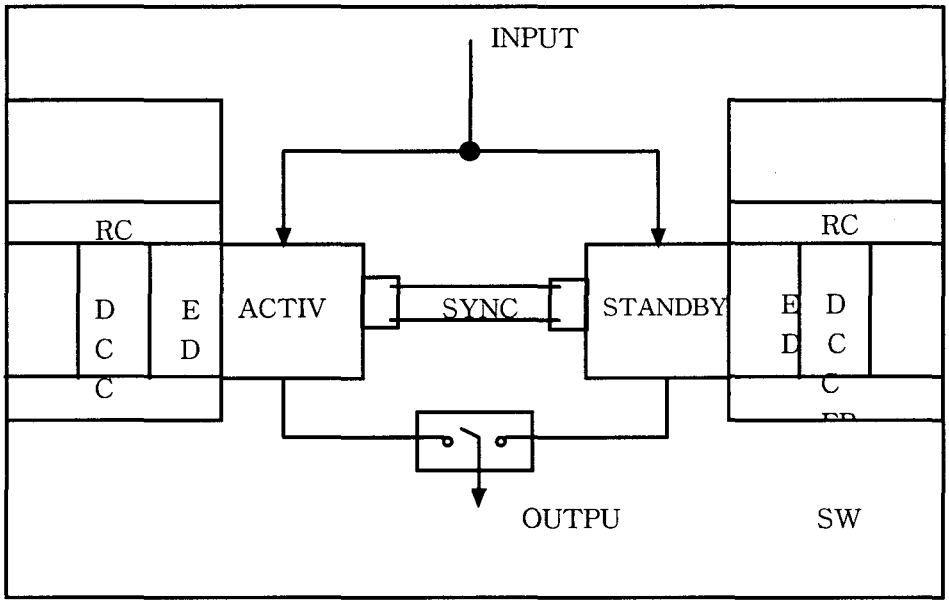
DCD : Duplication Comparision and Diagnosis
ED : Error Diagnosis
DF : Dirty Faults
GF : Graceful Faults

<그림 4> 결함 허용을 위한 이중화 관리의 흐름도

Ⅳ. 신뢰성 평가

4.1. 이중화 구성에 대한 Markov Models'

시스템 분석 기법에는 분석적 방법과 해석학적인 방법이 있다. 해석학적인 방법은 실행 모형을상태로 분류하고 천이되는 과정으로 시스템을 표현한다. 그 예로 Petri-Nets Markov Modeling등이 있으며 본 논문에서는 결함 복상점(Fault Coverage)을 가지는 Markov Modeling을 통하여 이중화 구조의 신뢰도 및 가용도를 분석 하고자 한다.



RC : Redundancy Control
ED : Error Diagnosis
DCC: Duplication Comparison Check
FR : Fault Recovery

<그림 5> Active/standby 이중화 시스템의 블럭도

(그림 5)는 본 논문에서 제안하고 구현한 Active/Standby 이중화 구성의 블록도로서

(그림 6)를 해석하기 위하여 시스템 모듈 중심으로 그린 것이다. 입력은 Active/Standby에 동일하게 들어오며 출력은 Active 프로세서의 결과가 출력된다.
만일 Active Side가 바뀌는 구조이다.

앞에서 설명한 바와 같이, 결함 허용을 구현하기 위해서 DCC, ED, FR 및 스위치등을 시스템에 내장하여 구성했다.

(그림 5)에 대한 상태의 천이와 천이 확률을 표기한 Markov model은 (그림 6)과 같다. 시스템은 초기에 결함 프리(Fault Free)상태에서 출발한다고 가정하며 상태 3으로 나타낸다. 상태 2로의 천이는 대기 모듈(Standby Module)의 자체 다운으로 실행 모듈(Active Module)에는 영향이 없고 자체 복구 기능에 의해서 상태 3으로 복구되거나 상태 2에 머물러 있는 것을 나타낸다. 상태 1로의 천이는 실행 모듈이 장애가 발생하는 경우로 결함 허용에 의해서 이중화 절체를 행하여 대기 프로세서가 실행 프로세서로 되어 하나의 프로세서만 살아 있는 경우에 해당한다. 이때에 결함 검출 보상점을 C로 나타낸다.

다음으로 상태 UD(UnDetect)는 상태 3에서 자동절체가 불가능하거나 실행 프로세서의 결함에 대한 검출 및 복구가 되지 않아서 복구 불능에 빠져 있는 것을 나타낸다. 서비스 연속성이 중요시되는 시스템에서는 이 상태에 있을 확률과 두 프로세서 다운(Dual Downed) 상태에 있지 않을 확률이 곧 시스템의 가용도이기 때문에 UD 상태를 분리하여 정의한다. 다음은 두 개의 모듈이 모두 다운된 경우를 상태 0로 나타낸다.

$$P3(1+dt) = (1-2\lambda\, dt)P3(t) + \mu\, dtP2(t) + \mu\, dtP1(t)$$

$$P2(t+dt) = (1-\lambda\, dt)P2(t) + \lambda\, dtP3(t)$$

$$P1(t+dt) = (1-\lambda\, dt)P1(t) + c\lambda\, dtP3(t)$$

$$P0(t+dt) = \lambda\, dtP2(t) + \lambda\, dtP1(t) + \lambda\, dtPud(t) + dtP0(t)$$

$$Pud(t+dt) = (1-c) + \lambda\, dtPud(t) + (1-\lambda\, dt)Pud(t)$$

실행/대기(Active/Standby) 운용에서는 (그림 6)에서 상태 3, 상태 2, 상태 1중 하나에 있을 경우에 시스템이 정상 동작을 할 것이다. 따라서 시스템의 신뢰도 Rss(t)는 다음과 같이 구할 수 있다.

Rss(t) = P3(t) + P2(t) + P1(t)

Repair Rate(μ)를 Zero로 가정하여 Rss(t)를 구하면 아래와 같다.

Rss(t) = (1+C)*e- λt - C*e-2λt

Coverage Factor를 Perfect라 가정할 때(C=1),

Rss(t) = 2*e- λt - 1*e-2λt
= 2*R(t) - R(t)2 = 1 - (1-R(t))2

따라서 Rss(t)는 Duplex Parallel Sysetm과 같아짐을 알 수 있다.
Recovery Coverage Factor를 Zero라 가정할 때 (C=0),

Rss(t) = R(t)
따라서 Rss(t)는 Non-redundant System과 같아짐을 알 수 있다.

신뢰도는 Failure Rate(λ), Coverage Factor (C) 그리고 시간을 포함한 다양한 요인에 좌우된다. 다른 접근 방식과 비교하기 위해서 Failure Rate(λ)를 2.5 * 10-3, Coverage Factor를 Perfect하고 Repair rate(μ)를 zero로 가정한다.
그리고 실행/대기 이중화 구조 대 TMR 구조의 신뢰도에 대한 Coverage Factor의 영향을 보면 표 1과 같다. 여기서 실행/대기 구조의 신뢰도는 다음과 같이 수식화할 수 있다.

Rss(t) = (1+C)e- λt - Ce- λt

시스템의 신뢰도 측정하는 또 다른 방법으로 MTTF(Mean Time To Failure)과 MT(Mission Time)이 있다. 이 두 방법도 앞에서 언급한 것처럼 시간과 복구범위 그리고 장애율이 시스템의 신뢰도 측정의 요소가 된다. 그리고 각 시스템에 대한 MFFT와 MT(허용시간)에 대한 식은 아래와 같다.

Rsimplex(t) = e-$\lambda\tau$
RTMR(t) = 3 * e-2$\lambda\tau$- 2 * e-3$\lambda\tau$
RAS(t) = (1 + 0.5) * e-$\lambda\tau$ - 0.5 * e-2$\lambda\tau$

RSIMPLEX(t), RTMR(T), RAS(T)을 적분하면 각 시스템의 MTTF는 다음과 같다.

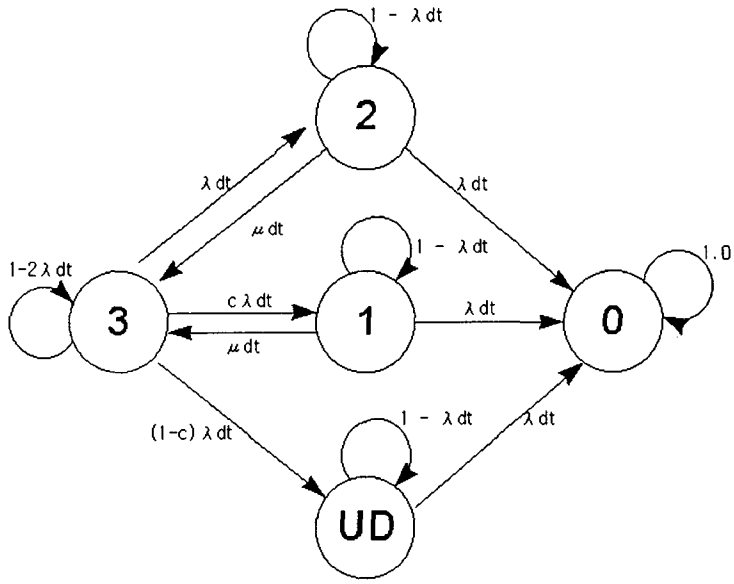
$$MTTFSIMPLEX = e^{-\lambda\tau}\, dt = 1/\lambda$$

$$MTTFTMR = (\, 3 * e^{-2\lambda\tau} - 2 * e^{-3\lambda\tau}\,)\, dt = 5/6\lambda$$

$$MTTFAS = (\, (1 + 0.5) * e^{-\lambda\tau} - 0.5 * e^{-2\lambda\tau})\, dt = 5/4\lambda$$

여기서, 이중화 시스템이 두 시스템 보다 신뢰도가 높음을 알 수 있다.

그리고 각 시스템의 허용시간을 측정하는 MT를 산출하기 위해서 λ = 0.01, 신뢰도는 0.8이라 가정하여, TMR의 허용 시간은 $(3 * e^{-2*0.01\tau} - 2 * e^{-3*0.01\tau}) = 0.8$에서 $MTTMR[\gamma]$ = 33시간을 산출했으며, 이중화 시스템의 허용 시간은 $(1.5 * e^{-0.01\tau} - 0.5 * e^{-2*0.01\tau}) = 0.8$에서 $MTAS[\gamma]$ = 36시간을 산출했다. MTR의 $MT[\gamma]$은 33시간이고 Active/Standby 이중화 시스템은 36시간 허용한다. 따라서 Active/Standby 이중화 시스템이 TMR보다 1.1% 수행 시간(Mission Time)이 향상되었다.



<그림 7> Active /standby 이중화 운용에 대한 Markov model

<표1> Active/Standby 이중화 구조 vs. TMR구조의 신뢰도에 대한 Coverage 영향

Coverage	Act/Sby	TMR
0.000000	0.99750	0.97460
0.100000	0.99775	0.97484
0.200000	0.99800	0.97558
0.300000	0.99825	0.97680
0.400000	0.99850	0.97852
0.500000	0.99875	0.98343
0.600000	0.99900	0.99602
0.700000	0.99925	0.99030
0.800000	0.99950	0.99448
0.900000	0.99974	0.99914

V. 결 론

본 논문에서 제안하고 구현한 결함 허용 실시간 시스템은 하드웨어의 중복에 바탕을 둔 실행/대기 이중화 운용으로서 결함 허용의 요구 조건을 충족시키고 있으며, 이는 결함의 발생시 그를 감지하여 발생 위치를 찾아내고, 다른 부분에 영향을 주는 것을 막는 결함 분리, 결함 발생부분을 적절히 복구하거나 그 오류가 시스템에 영향을 미치지 않도록 하는 결함 복구 기능을 제공하고 있다.

결함 허용를 위해서는 필연적으로 하드웨어의 성능 감소나 소프트웨어의 부담을 수반할 수밖에 없는데 본 논문에서는 최소한의 하드웨어 성능 감소로서 소프트웨어 부담을 줄이는 방향이 채택 되었다. 이중화 기능을 응용 소프트웨어에 부담시키지 않고 하부구조(하드웨어 및 운영체제)에서 완벽하게 담당하는 이러한 방식은 대형화되고 복잡해지는 응용 소프트웨어에 있어서는 필수적으로 추구해야할 방향이다.

본 논문에서 기술된 이중화 방식의 실시간 시스템의 특징은 먼저 하드웨어적인 측면에서는 이중화를 위한 하드웨어를 이중화 양측에서 독립적으로 운용하여 유사시 결함 분리가 쉽도록 하고 이중화 운용으로 인한 부담을 최소화시키고자 했다는 점을 들 수 있다. 소프트웨어적인 측면에서는 신속한 장애 감지에 뒤따르는 결함 진단과 복구를 실

현하여 장애 전파(Error Propagation)을 최대한 방지하고자 하였다. 부분적인 재시동에 의해 이중화 하드웨어가 Control Sanity를 잃을 경우에도 서비스 연속성을 최대한 보장할 수 있도록 하였으며, 스탠바이 로딩기능을 제공하여 실시간 프로그램 운용중 소프트웨어의 변경이 가능하도록 하였다.

그리고 장애 검출에 있어서는 자체 검사(Self-checking) 개념에 기초를 하여 다양한 인터럽트를 구현할 수 있게 됨에 따라 고장 부위를 결정하기 위한 진단 프로그램을 따로 수행시킬 필요없이 고장 즉시 고장 복구 및 재구성이 가능하다는데에 그 장점이 있다.

Markov model을 통한 분석의 결과 실행/대기 이중화 구조가 TMR 구조보다 같은 Failure Rate에서 더 높은 신뢰성을 가지며, 결함 허용을 위한 기법들(ED, DCC, FR)을 사용하고 이중화 절체기능을 구현함으로써 결함 보상점(Fault Coverage)를 완벽에 가깝도록 했다. 따라서 시스템의 가용도를 높였다. 추후 더 세분화된 고장 감지 장치 및 방법의 실현, 장애 관리자 기능의 보완, 롤백 포인트(Rollback Point)선정의 세분화 등에 많은 연구가 필요하리라 생각한다.

참 고 문 헌

[1] S. I. Jun, J. C. Park, J. M. Kim, J. H. Cho, "Concurrent Realtime Operating System for Telecommunication System", 일본 전자 정보통신학회, Jul. 1991.

[2] 이 헌, "Fault Tolerant Computing System", 전자교환기술, Vol. 1, 1985.

[3] H. Y. YOUN, "High Performance Fault-Tolerant Computing in Parallel and Distributed Systems", CSE, Arlington, Jun. 1992.

[4] 윤용익, 강석열, 조주현, "Development of the Concurrent Processing Kernel on Real Time System", 한국정보과학회, Oct. 1987.

[5] Y. I. Yoon, S. Y. Kang, J. H. Cho, "Development of the Concurrent Realtime Operating Systems for Large Capacity Electronic switching System", C/ESS, Vol. 12, No. 1, 1988.

[6] 김대석, "TDX-1B 이중화 절체시 Recovery 수준", 삼성전자통신 기술문서, Oct.

1990.

[7] R. T. Anderson, Reliability Design Handbook, IIT Research Institute, 1976.

[8] J. O. Becker, "The 3B20D Processor 8 DMERT Operating System", BSTJ, Vol. 62, Jan, 1983.

[9] Bengt E. Ossfeldt, Ingmar Jonsson, "Recovery and diagnostics in the Central Control of the AXE Switching System", IEEE Trans. Computers, Vol C-29, Jun. 1980.

[10] G. Barigazzi, L. Strigini, "Application transparent Setting of Recovery Points", IEEE 0731-3071, 1983.

[11] J. E. Allers, "No, 5 ESS- Strategies for Reliability in a Distributed Processing Environment", IEEE 0731-3071, 1983.

[12] Daniel P. Siewiorek, Robert S. Swarz, "The Theory and Practice of Reliable System Design, 1982, U. S. A.

[13] Daniel P. Siewiorek, "Architecture of Fault Tolerant Computers", COMPUTER, August, 1984.

[14] Amond D. Inselberg, "Multiprocessor Architecture Ensures Fault Tolerant Transaction Processing", MINMICRO SYSTEMS April, 1983.

[15] Dave Johnson, "The Intel 432: A VLSI Architecture for Fault Tolerant Computer System", Computer, Aug., 1984.

[16] R. A. Maxion, D. P. Siewiorek, and S. A. Elkind, "Techniques and architectures for fault-tolerant computing", Annual Review of Computer Science, vol. 2, pp. 469-520, Annual Reviews, Inc., 1987.

[17] S. Weber, "The Stratus architecture", in Reliable Computer Systems: Design and Evaluation. Bedford, MA: Digital Press, 1992.