'92 韓國自動制御學術會議論文集 1992. 10. 19~21

실시간 운영체제 환경하에서 이중화된 제어시스템을 위한 소프트웨어의 구현

박세화* 황동환* 이재혁* 김병국* 변증남* 문봉채** 김은기o

* : 한국과학기술원 전기 및 전자공학과

** : 삼성데이타시스템 (주) 연구소/기술개발팀

o : 한전기술연구원 자동제어연구실

Implementation of a Software for a Control System With Dual Structure Under the Real-Time Operating System

S.-H. Park* D.-H. Hwang* J.H. Lee* B.K. Kim* Z. Bien* B.C. Moon** E.K. Kim o

* : KAIST Dept. of Elec. Eng.

** : Samsung Data Systems Co., Advisory Systems Engineer

o : Automatic Control Dept., R&D Center, KEPCO

ABSTRACT

In this paper, a method for implementing software for the control system with dual structure in processor module is proposed and implemented to enhance its reliability. In this implementation the multi-tasking function which is provided by a real-time operating system is applied. The overall software is divided into five tasks and is performed in each of the dual processor module, independently. By this, the processor module with dual structure can achieve a control objective and fault diagnostics effectively. An experimental result shows that the backup processor module can be substituted for the primary processor module immediately when it happens to fail, because data relating the failure information are exchanged continuously done via shared memories.

I. 서론

제어시스템은 외란에도 불구하고 대상 플랜트로부터, 원하는 출력을 얻도록 하는 데에 그 목적이 있으며, 지속적이고 안정되게 제어하기 위해 제어시스템 자체의 신뢰성을 높이려는 노력이 있어왔다[1]-[8]. 제어시스템의 낮은 신뢰성으로 인한 잦은 고장으로 전체시스템의 가동이 중지되기도 하는데, 실례로 울산화력 발전소의 경우 전체 고장사고의 35.1[%]가 제어시스템의 고장에 기인한 것이며[6], 최근 제어시스템의 고장으로 인해 국내 원자력 발전소의 가동 중지등도 보고되고 있다[10].

최근의 제어시스템은 반도체 기술과 마이크로 프로세서(micro-processor)의 발달에 힘입어 디지탈 기술로 구현된 여러 하드웨어 모듈로 구성이 된다. 모듈 내의 단일 소자 만의 고장으로도 모듈이 오동작을 할 수 있으며, 나아가서 전체 제어시스템이 제 기능을 다 못하는 경우가 생기기도 한다. 고장이 발생하더라도 제어시스템의 성능에는 영향을 주지않도록 시스템의 구성시부터 고장을 고려하여 설계하는 적극적인 대처 방법을 내고장성 제어(fault tolerant control) 방법이라 한다[11][12]. 일반적으로 내고장성 제어는 같은 기능을 하는 것을 여러 개 두는 중복구조(redundancy)를 갖도록 하는 것인데, 우주선 항공기의 제어나 원자력 발전소 같은 대규모 공정제어 분야 등에 많이 이용되어 왔다. 중복구조에는 하드웨어 중복구조(hardware redundancy)와 해석적 중복구조(analytical redundancy)가 있다[13]. 해석적 중복구조는 계통과 감지기를 포함한 플랜트의 고장탐지 및 고장진단에 보통 이용된다[14]. 반면에 하드웨어 중복구조는 같은 기능을 하는 여러 하드웨어를 두는 것으로, 하드웨어 모듈 내에서 고장(예로 메모리 소자의 고장)이 발생하여 제 기능을 다하지 못할 시에는 그 기능을 대신하기 위해서 하드웨어 중복구조가 요구된다. 따라서, 높은 신뢰성을 요구하는 제어시스템은 중복구조를 갖도록 할 필요가 있으며, 이중구조 만으로도 충분히 제어시스템의 신뢰도를 높일 수 있다[7][8]. 본 논문에서는 이중화된 하드웨어 모듈과 소프트웨어적으로 두 하드웨어의 고장을 진단하여 한 하드웨어의 이상시에 그 모듈을 백업할 수 있도록 하였다.

일반적으로 대규모 플랜트를 위한 제어시스템은 프로세서(processor) 모듈, I/O(input/output) 모듈과 신호 조절 모듈(signal conditioning module)로 구성되는데, [7]-[9]에서는 프로세서 모듈과 I/O 모듈을 각각 이중화하여 제어시스템의 신뢰도를 높였다. 그런데, [7]에서 언급됐듯이 프로세서 모듈 내의 MPU(micro-processor unit)가 I/O 모듈을 주변 소자처럼 진단하고 주 I/O 모듈의 고장시 백업(backup) 과정도 수행하므로, 프로세서 모듈의 이중화가 I/O 모듈의 이중화보다 더 중요함을 알 수 있다. 따라서 본 논문에서는 프로세서 모듈의 이중화에 촛점을 맞춘 내용에 대해 서술한다. 한편, [7]-[9]에서는 고장진단용 소프트웨어의 구현시에 주프로세서 모듈과 부프로세서 모듈이 동일한 하드웨어를 이용함에도 불구하고 두 하드웨어의 역할이 약간 다르기 때문에, 두 종류의 소프트웨어가 필요하게되어

소프트웨어의 상호 호환성이 없었다. 즉, 주프로세서 모듈은 소프트웨어의 변경 없이는 부프로세서 모듈로 동작이 될 수 없다. 실제로 주프로세서 모듈의 기능과 부프로세서 모듈의 그것이 다를 필요가 없으므로 본 논문에서는 [7]-[9]에서 적용된 고장탐지방법 등을 보완 및 변경하여 동일한 소프트웨어 구조를 갖도록 하는 방법도 서술한다. [7]-[9]에서는 전체 소프트웨어 중 제어를 위한 부분과 고장진단을 위한 부분의 구별이 명확하지 않았으나, 멀티태스킹(multi-tasking)이라는 유용한 기능을 갖고 있는 실시간 운영체제(real-time operating system, 이하 RTOS)[15][16] 환경하에서 전체 소프트웨어를 태스크(task) 별로 모듈화함으로써 소프트웨어 각 부분의 역할을 명확하게 할 수 있음을 보인다.

또한, 이중화된 프로세서 모듈간에 공유메모리를 이용하여 고장과 관련된 정보를 교환하는 방법[7]-[9]이 본 논문에도 적용되었다.

II. 고장진단을 위한 소프트웨어

1. RTOS 환경하에서의 멀티태스킹

대부분 기존의 프로그램들은 단순히 순차적으로 인스트럭션을 수행하는 구조이며 이미 결정된 순서만으로 수행을 하며, 수행속도가 결과에 미치는 영향이 거의 없는 '순차적인 프로그램(sequential program)'의 형태이다. 한편 계산 결과가 상대적인 수행속도에 의존하는 다수의 '순차적인 프로그램'이 병렬로 수행되는 '동시수행 프로그램(concurrent program)' 방법도 생각할 수 있는데, 이는 계산 결과의 옳고 그름이 여러 '순차적인 프로그램'의 상대적인 수행속도와는 무관하기에 주로 시간 공유(time sharing) 오퍼레이팅 시스템에 이용된다. 여기에 옳은 계산 결과를 위한 각 순차적인 프로그램에 요구한계시간을 두어, 계산 결과는 절대적인 수행속도에 크게 좌우되는 형태가 '실시간 프로그램(real-time program)'이다. 이는 보통 여러 '동시수행 프로그램'들로 이루어져 다양한 제어 목적에 적용이 가능하다[15]. 또한 이는 태스크라고 하는 프로그램 모듈로 이루어져 이들 사이에는 메시지큐나 세마포어(semaphore)를 통해 인터페이스가 가능해야 한다. 이 태스크들은 생성(spawn), 수행(execute), 대기(ready), 기다림(pend) 그리고 제거(delete) 등을 하며 각 태스크의 우선순위에 따라 태스크 스윗칭(task context switching)이 이루어지면서 번갈아 수행되는 것이다. 이러한 목적을 위해 전체 소프트웨어를 태스크 별로 모듈화하여 유지 및 보완에 잇점이 많은 RTOS가 도입되었다. 즉, 전체 소프트웨어는 여러 태스크로 구분이 되어, 그 역할이 명확히 구별될 수 있다. 본 논문에서는 RTOS의 기능 중 멀티태스킹 기능이 주로 이용되었으며 기타 기능들도 부가적으로 이용되었다.

2. 소프트웨어 구조

이중구조를 갖는 프로세서 모듈에 고장이 발생하더라도 백업프로세서 모듈이 고장정보를 처리하여 주프로세서 모듈의 고장 직전의 상태를 유지하도록 함으로써 제어시스템을 안정되게 동작시키는 것이 고장진단의 목적이다. 이를 위해서 주프로세서 모듈과 백업프로세서 모듈 사이에서 상호 고장에 관한 정보와 각각의 동작 상태에 관한 정보 교환이 신속하며 지속적으로 이루어져야 한다. 또한 위의 작업은 제어알고리즘의 수행과 병행하여 이루어진다. 한편, 프로세서 모듈은 관리제어시스템(supervisory control system)으로부터 모듈의 동작에 관련된 제반 명령(command)을 받는다. 이와 같이 제어알고리즘 수행 및 이중화를 위한 고장진단 등의 작업수행을 위해 앞에서 언급된 RTOS가 가지고 있는 멀티태스킹 기능이 이용된다. 즉, 프로세서 모듈의 고장진단, 플랜트의 제어, 관리제어시스템과의 정보교환을 위한 통신 및 관리제어시스템으로부터 받은 명령의 처리 및 수행을 위해 이들 각각 기능을 하도록 태스크를 정의하고 그 역할을 분담 모듈화하였다. 이들 태스크 사이에는 메시지큐(message queue)나 세마포어(semaphore)를 이용해 상호 정보 교환이 이루어진다. 전체적으로 태스크는 통신, 명령어 관리, 자체진단, 고장관리, 그리고 공정제어 태스크로 구분된다. 여기서 통신, 명령어 관리 및 공정제어 태스크는 프로세서 모듈의 이중화를 위한 내고장성 제어와는 무관하게 필요한 태스크들이며, 고장처리 관련 이중화를 위해 자체진단과 고장관리 태스크가 부가적으로 도입되었다.

본 논문에서는 자체진단 태스크, 고장관리 태스크, 그리고 공정제어 태스크에 대해서 프로세서 모듈 내의 고장과 관련하여 주로 다루기로 한다. 전술된 5개의 태스크들은 구조적 언어로서 산업 응용분야에 많이 적용되고 있는 C-언어로 구현되며, 이들 사이에 정보교환을 위해 세마포어 및 메시지큐는 다음과 같이 선언되었다.

```
SEM_ID      API2CMD_PROC, CMD_PROC2CNTRL;
SEM_ID      CMD_PROC2SELF_TEST, CMD_PROC2FAIL_CHK;
MSG_Q_ID    CMD_PROC2API, API2CMD_PROC;
MSG_Q_ID    CNTRL2CMD_PROC, FAIL_CHK2CMD_PROC;
MSG_Q_ID    SELF_TEST2FAIL_CHK, CMD_PROC2CNTRL;
```

여기서 메시지큐는 구체적인 정보를 보내줄 때 사용되며, 세마포어는 설정 또는 지움 두가지 정보만을 위한 것으로 태스크들의 시작 시점을 일치시키기 위해 사용된다. 태스크들은 주 프로세서 모듈뿐만 아니라 백업용 프로세서 모듈 모두에서 동일한 구조를 가지고 동시에 수행되는 것이며, 전체 태스크들과 온라인시에 이들과의 관계는 그림 1과 같다.

태스크들 사이에서 공유되는 플래그로 MAIN_PROCESSOR와 CPUB_OPERATE 있다. 이는 프로세서 모듈의 이중화를 위해 필요한 것으로 설정(set) 또는 지움(reset)의 값을 가진다. MAIN_PROCESSOR가 설정되면 해당 프로세서 모듈이 주프로세서 모듈로서 동작하게 됨을 의미하며, CPUB_OPERATE가 설정되면

이중화되었음을 의미하게 된다. 이들의 정보는 오프라인시 결정되며, 공정제어 태스크 및 고장관리 태스크에서 주로 이용된다.

III. 이중화된 제어시스템을 위한 태스크

1. 통신 태스크(Api task)

관리제어시스템과 공정제어시스템 사이에서 정보 교환을 위해 필요한 것으로 통신망(본 논문에서는 ethernet)을 통해 관리제어시스템으로부터 정보를 받아, 명령어 관리 태스크에 전달한다.

2. 명령어 관리 태스크(Cmd_Proc task)

관리제어시스템으로부터 받은 명령을 통신 태스크로부터 메시지큐를 통해 넘겨 받아 그에 따라 다른 태스크들의 동작을 관리한다. 명령어 관리 태스크가 처리하는 명령은 다음과 같다.

오프라인(off-line) 시에는 프로세서 모듈의 자체 진단을 위한 'SELFTEST', 공정제어 관련 파일을 다운로드 받기 위한 'DOWNLOAD' 그리고 모든 태스크들이 동작하게 되는 'RUN' 등이며, 온라인(on-line)시에는 모든 태스크들의 동작을 멈추게 하는 'STOP', 그외 'HOLD', 그리고 'RESTART' 등이 있다. 여기서 'DOWNLOAD' 명령 후 자체진단, 고장관리 및 공정제어 태스크를 생성(spawn) 한다. 참고로 이 태스크는 [9]의 주요 루틴(main routine)에 해당되는 대부분의 작업을 수행한다.

3. 자체진단 태스크

자체진단 태스크는 프로세서 모듈의 고장탐지의 한 방법으로서 이용된다. 수십개 이상의 칩(chip)으로 이루어져 있는 프로세서 모듈에서 모듈 내의 고장을 단일 소자 단위로 하는 것은 실제적으로 매우 어려우므로, 기능 별로 크게 몇부분으로 나누어 수행된다. 즉, MPU, 버스, 메모리, 타이머 그리고 기타 주변 디바이스들로 분류되어 온라인으로 그 기능을 테스트한다. 예로 메모리에 데이타를 썼다가 다시 읽었을 때 그 값에 차이가 있으면 이 부분을 고장으로 보는 것이다. 자체진단 태스크가 수행되기 위해 다른 태스크들(예로 공정제어 태스크)이 수행될 시간을 차지하면 안되므로, 태스크의 우선순위가 가장 낮고 부분별 테스트 시간은 가능한한 짧게 하므로써, 프로세서 모듈이 공정제어나 관련 작업들을 하고 남은 시간에 수행된다. 만일 이 태스트에서 고장이 탐지되면 고장관리 태스크에 SELF_TEST2FAIL_CHK를 통해 메시지를 보내어 백업과정을 수행토록 한 후에 'safely shutdown' 절차를 거치게 된다.

4. 고장관리 태스크

고장관리 태스크는 주 프로세서 모듈과 백업용 프로세서 모듈간에 고장에 관한 정보를 교환하며, 고장발생시 백업에 필요한 모든 조치를 취한다. CPUB_OPERATE = RESET 되었을 시에는 프로세서 모듈이 단독으로 동작됨을 의미하므로 두프로세서 모듈 사이에서의 고장 관계 정보 교환은 이루어지지 않는다. 여기서 두 프로세서 모듈 사이에서 정보교환을 위해 다음과 같이 6개의 상태 플래그를 설정했으며 이는 공유메모리에 쓰여진다. 참고로 CPUA는 해당 프로세서 모듈을 의미하는 것이며, CPUB는 상대 프로세서 모듈을 의미한다.

i) CPUA_STOP - 상대 프로세서 모듈이 해당 프로세서 모듈의 동작을 강제적으로 멈추도록 한다.

ii) CPUB_STOP - 상대 프로세서 모듈에 어떤 이상이 탐지되었을 때 이 플래그를 설정하여 상대 프로세서의 동작을 멈추도록 한다.

iii) CPUA_f_flag - 해당 프로세서 모듈내에서 고장이 탐지되었을 때 플래그를 설정하여, 상대 프로세서 모듈에 정보를 준다.

iv) CPUB_f_flag - 상대 프로세서 모듈내에서 고장이 있었음을 인지하여 백업등의 조치를 취할 수 있도록 한다.

v) CPUA_timer_cnt - 타이머 등의 고장으로 인해 프로세서 모듈이 동작하다 멈추어 버리는 등의 시간에 관계된 고장이 발생되면 위의 플래그만으로는 고장정보가 전달되거나 탐지될 수 없다. 그래서 매 샘플링 시간마다 그 값을 증가시켜 주어 멈추지 않고 동작하고 있음을 나타내게 된다.

vi) CPUB_timer_cnt - 상대 프로세서 모듈에서 매 샘플링 시간마다 증가시켜 주는 것으로, 만일 CPUA_timer_cnt와 비교하여 그 차이가 크면 상대프로세서가 고장등에 의해 동작이 멈추었음을 나타내게 된다.

위에서 정의된 6개의 플래그를 이용하여 두 프로세서 모듈간에 정보를 교환하고 고장시에는 적절히 조치를 취하게 된다. 고장시의 조치는 고장에 대한 정보를 FAIL_CHK2CMD_PROC을 통해 명령어관리 태스크에게 전해주는 것이고, 만일 해당프로세서 모듈에서 고장이 있었으면 safely shutdown을 위해 초기화 상태에 있도록 하는 것이며, 상대 프로세서 모듈에서 고장이 있었으면 CPUB_STOP = SET함으로써 강제로 동작을 멈추도록 하는 것이다. 또한 고장시의 백업과정은 단지 MAIN_PROCESSOR = SET함으로써 이루어지게 된다.

앞에서 간략히 언급됐듯이 [9]에서와 같이 프로세서 모듈의 고장탐지는 자체 테스트, 예외처리 에러 및 비교과정에 의하여 이루어진다. 전술된 바와 같이 자체테스트는 자체진단 태스크에 의해 이루어지는 것이며, 예외처리는 MPU가 하드웨어적인 고장이나 소프트웨어의 이상시에 미리 설정된 예외 벡터로부터 예외처리 루틴을 수행하는 것으로 프로세서 모듈의 고장 탐지에 효과적으로 이용될 수 있다[7] -[9]. 예로 모듈 내의 메모리에 쓰인 어드레스를 읽은 후 그 어드레스와 관계된 레지스터를 조작하려고 할 경우 메모리의 고장으로 잘못된 어드레스를 조작하게 되면 '어드레스 예외에러가 발생된다. 이 경우의 메모리의 고장이 자체테스트를 통해 발견되기도 하지만, 예외에러를 유발시키기도 한다. 예외처리를 위한 초기화 및 예외 벡터의 설정은 명

령어 관리 태스크에서 이루어지고, 예외처리 에러는 고장관리 태스크와는 실제로 무관하나, 예외처리 에러 루틴에서 CPUA_f_flag를 설정함으로써 상대프로세서 모듈에게 고장이 있었음을 알리고 'safely shutdown' 과정을 수행된다. 비교과정은 상대프로세서 모듈이 정상적으로 매샘플링 시간마다 반복적으로 플랜트를 제어하고 있는가와 상대프로세서 모듈에서 고장이 발생했을 시에 해당프로세서 모듈에서 이를 인지하는지 확인 하기 위해서 필요하다. 즉, 비교과정에서는 주로 CPUB_f_flag가 설정되었나 점검하는 것과 CPUB_timer_cnt와 CPUA_timer_cnt의 차이를 비교하여 상대 프로세서 모듈이 동작하고 있음을 점검하는 것이다. 만일 CPUB_f_flag가 설정됐거나 CPUB_timer_cnt와 CPUA_timer_cnt의 차이가 크면 고장으로 보고, CPUB_STOP 플래그를 설정한다. 또한 명령어 관리 태스크에 그 정보를 알리는 메시지를 보낸다. 그리고 해당 프로세서가 백업 모듈일 경우에는 MAIN_PROCESSOR를 설정함으로써 백업과정도 이루어진다. 그림 2는 고장관리 태스크의 흐름도(flow chart)를 보여주고 있다.

5. 공정제어 태스크

공정제어 태스크에서는 제어시스템의 주 목적인 플랜트의 제어를 위한 작업이 수행되며, 이 루틴들은 주기적으로 매 샘플링 시간마다 한번씩 수행된다.

디지탈 제어기를 이용하여 소프트웨어적으로 구현되는 과정은 일반적으로 크게 세단계로 나뉘어 이루어진다. 먼저, 플랜트로부터 A/D 변환(analog to digital conversion)을 통해 공정값들(process values)을 받아들이고, 다음에 정해진 제어 알고리즘에 의해 받아들인 공정값들을 이용하여 제어 입력값들(control input values or process values)을 계산하고, 마지막으로 계산된 값들을 D/A 변환(digital to analog conversion)을 통해 플랜트로 출력한다. 본 연구에서 백업프로세서 모듈은 핫스탠드바이(hot stand-by)로 동작되므로, 첫번째와 두번째의 과정은 이중화된 프로세서 모듈 모두에서 각각 독립적으로 이루어질 수 있으나 세번째의 과정은 주프로세서 모듈에서만 이루어져야 한다. 이는 다음과 같이 초기에 미리 설정해 둔 MAIN_PROCESSOR 플래그의 상태를 점검함으로써 가능하다.

```
if   MAIN_PROCESSOR = SET
then   data_out()
```

즉, MAIN_PROCESSOR플래그가 설정되었으면 주프로세서 모듈로서 데이타를 출력하고, 그렇지 않으면 백업프로세서 모듈로 설정된 것이므로 데이타의 출력 과정은 생략된다. 만일 백업프로세서 모듈로 동작하고 있다가 주프로세서 모듈의 고장이 탐지되면, 주프로세서 모듈의 동작을 멈추게 하고 위의 플래그를 설정하게 되므로 이 때부터 데이타를 출력하게 된다. 참고로 초기에 위의 플래그를 세우거나 지우는 것은 고장관리 태스크에서 이루어진다.

마지막으로 CPUA_timer_cnt 플래그를 하나 증가시켜 계속 정상적으로 제어를 수행하고 있음을 상대 프로세서에게 알리는 작업이 이루어진다. 공정제어 태스크에서 이루어지는 과정은 그림 3과 같다. 맨처음에 명령어 관리 태스크에서 보낸 세마포어를 기다리는 부분이 있으며, 이후의 과정은 전술한 바와 같다.

IV. 실험

1. 실험 환경

실험을 위해 UNIX와 유사한 환경으로, 비교적 사용이 용이한 Wind River Systems사의 VxWorks를 RTOS로 사용하였다[16]. 또한 이중화된 프로세서 모듈의 멀티프로세싱(multi-processing)을 위해 산업적 응용에 많이 이용되고 있는 VME bus를 채택하였다. 프로세서 모듈로는 32비트 MPU인 MC68030을 탑재하며 자체에 통신을 위한 이더넷(ethernet) 포트가 내장되어 있는 Force사의 CPU30이 사용되었다[17]. 관리제어시스템으로는 SUN시스템이 이용되었다. 제어 대상 플랜트로는 서울화력 4호기 보일러의 정상상태 모델을 갖고 있는 시뮬레이터[4]를 이용하였는데, 플랜트 모델은 소프트웨어로 구현되어 또 다른 CPU30에서 독립적으로 수행된다. 즉 그림 5와 같이 주프로세서 모듈, 백업프로세서 모듈 그리고 시뮬레이터용 프로세서 모듈로 3개의 CPU30 프로세서 모듈이 이용되었으며, 이들 간에는 공유메모리를 통해 데이타 교환이 이루어진다. 관리제어시스템과 3개의 CPU30과의 이더넷을 통한 통신은 주프로세서 모듈이 매스터(master)가 되어 백플레인 네트웍(back-plane network)[16]으로 이루어진다. 시뮬레이터의 제어를 위한 샘플링 시간은 상용 SLPC(Single Loop Programmable Controller)에서 주로 사용하는 250[msec]로 하였다. 전체적인 시스템의 구성도는 그림 4와 같다.

2. 실험 내용

실험은 정상상태에서 주프로세서 모듈의 동작을 강제로 멈추게 하였을 때 백업프로세서가 이를 탐지하고 백업과정을 수행함을 보여주기 위함이다. 실험의 수행 순서는 먼저 관리제어시스템에서 각 프로세서 모듈이 수행할 프로그램들을 컴파일한 후, 각 프로세서 모듈과 접속(예로 rlogin으로)한 후 컴파일된 프로그램들을 각각에 다운로드하고 실행 명령을 수행하는 것이다. 그림 5는 백업프로세서 모듈의 콘솔(console)로 디스플레이되는 것을 보여준 것으로 주프로세서 모듈이 동작을 멈추었을 때 비교과정에 의해 이를 탐지하고 백업 조치를 즉시 취함을 볼 수 있다. 그림 6에서는 주프로세서 모듈이 제어 동작을 멈추고 백업프로세서 모듈이 백업과정을 수행할 때 시뮬레이터로의 윈도우로 디스플레이된 것을 나타낸 것이다. 여기서 윗쪽 왼편의 윈도우는 시뮬레이터의 공기 동특성을 모니터링한 것으로 백업과정을 수행할 때 한가지 출력값에 약간의 교란이 있은 후 즉시 정상상태로 되돌아 감을 볼 수 있다. 또, 윗쪽 오른편 윈도우는 시뮬레이터용 CPU30 모듈의 콘솔 윈도우로서 CPU30의 동작 상태를 보여준다. 아래편 윈도우는 모니터링하기 위한 공정값들을

선택하기 위해 필요한 윈도우이다.

V. 결론 및 추후 연구 과제

본 논문에서 제안한 RTOS 환경하에서의 통신, 명령어 관리, 자체진단, 고장관리, 그리고 공정제어 태스크로 구분된 태스크들이 이중구조를 갖는 제어시스템에서 멀티태스킹을 하면서 제어목적과 고장진단의 목적에 적합하게 수행됨을 실험을 통해 볼 수 있었다. 또한, 멀티태스킹을 위해 전체 소프트웨어를 태스크별로 모듈화 함으로써 기능의 첨가 및 변경이 용이하다. 즉, 제어알고리즘을 PID(proportional, integral and derivative) 형태에서 적응제어나 예측제어 같은 고급제어를 적용할 시에는 공정제어 태스크만을 수정함으로써 가능하며, 제어시스템의 고장진단 이외에 플랜트 자체의 고장진단이 요구될 시에는 이를 위한 태스크를 첨가함으로써 가능하다. 제어시스템의 신뢰성을 더욱 더 높이기 위해 I/O 모듈 뿐만 아니라 통신망의 이중화시, 제안한 소프트웨어의 확장 및 주프로세서 모듈의 고장으로 백업프로세서 모듈이 백업과정을 수행할 경우 주프로세서 모듈의 수리 후 재실장과 재운전 과정을 위한 소프트웨어의 보완 등이 추후 연구 과제로 남아있다.

후 기

본 연구는 발전소 제아시스템의 국산화를 위해 이루어진 것으로 연구비를 지원해 주신 한전기술연구원과 삼성데이타시스템(주)에 감사드립니다.

참고 문헌

[1] Theordore J. Williams, "The development of reliability in industrial control system," IEEE MICRO, pp. 66-80, 1984

[2] 조영조, "가법적 증복 적응 제어기를 이용한 제어시스템의 신뢰도 향상에 관한 연구", KAIST 박사논문, 1989.

[3] 한전기술연구원, "마이크로 컴퓨터를 이용한 전자제어 시스템(최종보고서)", Feb. 1988.

[4] 한전기술연구원, "발전소 제어용 디지탈 계장제어 시스템 개발(최종보고서), Sep. 1990.

[5] 신영달, "Boiler backup control을 위한 Muliprocessor 방식에서의 신뢰도 개선에 관한 연구", KAIST 석사논문, 1987.

[6] 정광균, "화력 발전소용 보일러 제어기를 위한 백업콘트롤 시스템에 관한 연구", KAIST 석사논문, 1987.

[7] 박세화, "이중구조를 갖는 제어시스템의 구현과 신뢰도 분석에 관한 연구", KASIT 석사논문, 1990.

[8] 박세화, 문봉채, 김병국, 변증남, "이중구조를 갖는 제어시스템에 관한 연구", 전자공학회 논문지, 제27권, 제9호, pp. 41-53, Sep. 1990.

[9] Z. Bien, S.-H. Park, B. K. Kim, D.-H. Hwang, H. G. Lee and J. H. Lee, "Implementation of control system with dual structure in processor module and I/O modules", will be appeared in IEEE Symposium of Industrial Electronics, Xian, Peoples of Republic of China, May. 1992

[10] 허성광, "발전소 제어계통의 고장 진단", KAIST에서의 세미나 자료, May. 13. 1989.

[11] W. F. McGill and S. E. Smith, "Fault tolerance in continuous process control," IEEE Micro, pp. 22-33, 1984.

[12] D. A. Rennels, "Fault tolernt computing concepts and examples," IEEE Trans. on Computers vol. C33, no. 12, pp. 1116-1129, Dec. 1984.

[13] E. Y. Chow and A. S. Willsky, "Analytical redundancy and design of robust failure detection systems," IEEE Trans. on Auto. Control, vol. AC-29, no. 7, pp. 603-614, July, 1984.

[14] A. S. Wliisky, "A survey of design methods for failure detction in dynamic systems," Automatica, vol. 2, pp. 601-611, 1976.

[15] Ready Systems, "VRTX 32C user's guide", U.S.A. Dec. 1988.

[16] Wind River Systems, "VxWorks programmer's guide", U.S.A.

[17] Force Computers Inc., "SYS68K/CPU-30 User's Manual", Revision No.3, 1989

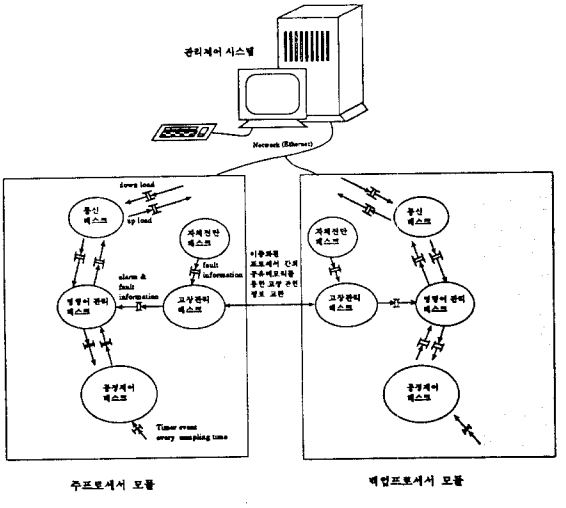

그림 1. 이중화된 프로세서 모듈과 멀티태스킹의 관계도

Fig. 1. Relationship Between the Processor Module with Dual Structure and Multi-tasking

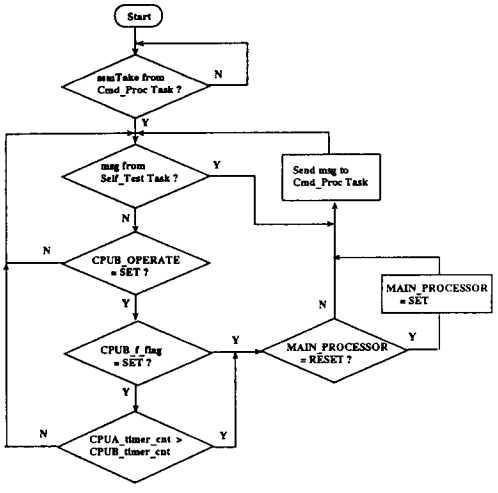

그림 2. 고장관리 태스크의 흐름도

Fig. 2. Flow Chart of Cmd_Proc Task

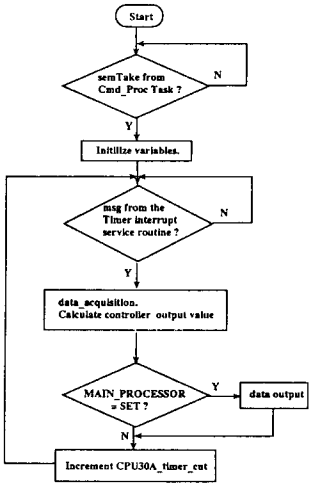

그림 3. 공정제어 태스크의 흐름도

Fig. 3. Flow Chart of Control Task

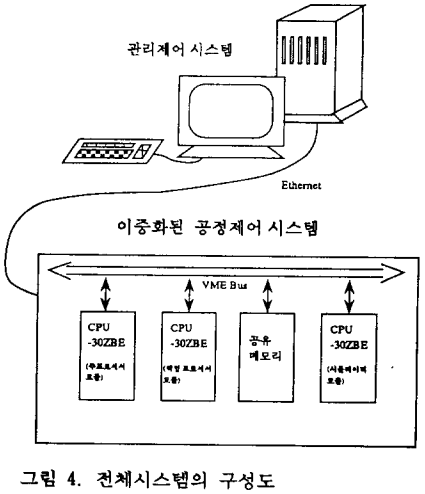

그림 4. 전체시스템의 구성도

Fig. 4. Configuration of the overall system

```
< /user/LAB/hanjeon/integ/pcs 2 > rlogin cpu30b

-> ld < main1
value = 0 = 0x0
-> main
value = 16665400 = 0xfe4b38
-> 0xfe4b38 (tCMD_PROC): exception vector set ..
0xfe4b38 (tCMD_PROC): initialize variables ..
0xfe4b38 (tCMD_PROC): signal routine initialize ..

 ==> Server socket port number : 2044
0xfe4b38 (tCMD_PROC): off_line
0xfe4b38 (tCMD_PROC): API2CMD_PROC Msg Receive !   msg=1
0xfe4b38 (tCMD_PROC): off-line self test..
0xfe4b38 (tCMD_PROC): #### CPU Test      ----> Good
0xfe4b38 (tCMD_PROC): #### FPC Test      ----> Good
0xfe4b38 (tCMD_PROC): #### Databus Test  ----> Good
0xfe4b38 (tCMD_PROC): #### Addrbus Test  ----> Good
0xfe4b38 (tCMD_PROC): #### MEMORY(DRAM) Test ----> Good
0xfe4b38 (tCMD_PROC): #### PIT2 Test     ----> Good
0xfe4b38 (tCMD_PROC): #### DUSCC1 Test   ----> Good
0xfe4b38 (tCMD_PROC): #### DUSCC2 Test   ----> Good
0xfe4b38 (tCMD_PROC): #### FGA Test      ----> Good
0xfe4b38 (tCMD_PROC): API2CMD_PROC Msg Receive !   msg=2
0xfe4b38 (tCMD_PROC): off-line down load..
0xfe4b38 (tCMD_PROC): Control, Self_Test Fail_Chk task is created !
0xfe4b38 (tCMD_PROC): Waiting for run ....
0xfe4b38 (tCMD_PROC): API2CMD_PROC Msg Receive !   msg=3
0xfe4b38 (tCMD_PROC): RUN .....
0xfe4b38 (tCMD_PROC):
 on_line
0xfc547c (tCNTRL): in t_control
0xfc547c (tCNTRL): cnt 0  A=10 B=10
0xfda0b8 (tSELF_TEST):          FGA GOOD
0xfda0b8 (tSELF_TEST):          PIT2 GOOD
0xfda0b8 (tSELF_TEST):          DUSCC1 GOOD
0xfda0b8 (tSELF_TEST):          CPU GOOD
0xfda0b8 (tSELF_TEST):          FPC GOOD
0xfda0b8 (tSELF_TEST):          Databus GOOD
0xfda0b8 (tSELF_TEST):          MEM(DRAM) GOOD
0xfc547c (tCNTRL): cnt 0  A=12 B=11
        .
        .
        .
0xfc547c (tCNTRL): cnt 0  A=428 B=418
0xfda0b8 (tSELF_TEST):          MEM(DRAM) GOOD
0xfd8bd0 (tFAIL_CHK): FAIL2CMD Send ! timer faill msg=25
0xfe4b38 (tCMD_PROC): msg from FAIL_CHK Task !  msg=25
0xfda0b8 (tSELF_TEST):          ADDRBUS GOOD
0xfe4b38 (tCMD_PROC): CMD_PROC2API Msg Send !    msg=25
0xfda0b8 (tSELF_TEST):          FGA GOOD
0xfc547c (tCNTRL): cnt 1  A=429 B=418
0xfda0b8 (tSELF_TEST):          PIT2 GOOD
0xfda0b8 (tSELF_TEST):          DUSCC1 GOOD
        .
        .
```

그림 5. 백업 프로세서 모듈의 콘솔 디스플레이

Fig. 5. The display on the console of the backup processor module

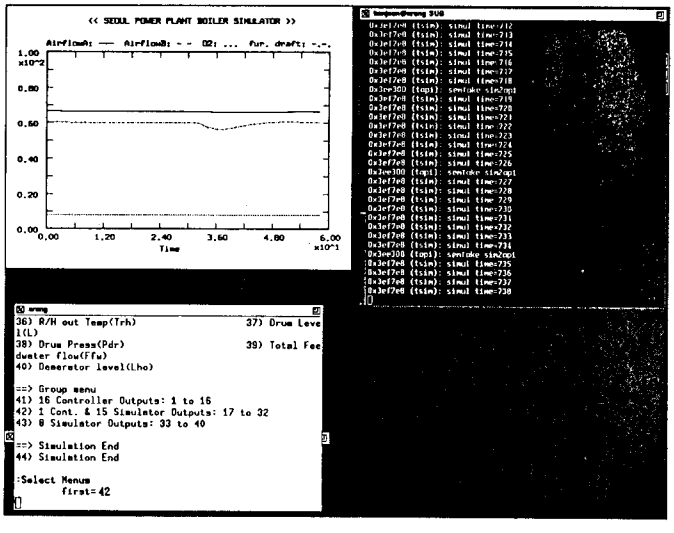

그림 6. 시뮬레이터의 모니터링 상태

Fig. 6 The monitoring status of the simulator