실시간 시스템인 승강기 제어기 프로그램 개발

Programming Development Environment for the Elevator Controller of Real-Time Systems

최 병 욱, 임 계 영, 고 경 철
(Byung-Wook Choi, Kye-Young Lim, and Kyoung-Chul Koh)

Abstract : This paper discusses a real time multi-tasking system model and a development environment for an elevator control system. Recently, as the elevator systems become large-scaled and operate with high speed, there are lots of software tasks to be processed with time constraints. Thus, the control systems are designed with distributed control structure and characteristics of typical real time systems. For structural design of such real time system, we introduce a multi-tasking model based on a real time operating system model and an software development environment based on virtual prototyping which simulates real system operation in the cross development environment. The developed system model and the development environment are successfully applied to development of a new elevator system with distributed control structure and its system reliability can be verified through numerous field tests.

Keywords : real-time system, embedded system, control, development environment, elevator

I. 서론

실시간 시스템은 출력의 정확성 뿐만 아니라, 반응시간 및 처리 시간이 중요한 시스템으로서 산업용, 민수용 및 군사용 응용 분야에서 많이 찾을 수 있다. 특히 산업 분야에서 사용되는 실시간 시스템으로는 산업체의 공장 제어 시스템에서부터 승강기 제어시스템, 항공기 제어 시스템, 자동 입출금 자동화 시스템 등이 포함되며, 일상 생활에서도 자동 세탁기, 지능형 공조 시스템 등 그 규모는 작지만 널리 사용되어지고 있다. 최근 실시간 시스템이 점점 복잡화, 다양화, 고기능화 됨에 따라, 산업계 연구개발 부문에서는 실시간성과 신뢰성 향상을 위한 실시간 시스템의 구조적 설계에 관한 연구가 새로운 관심의 대상이 되고 있다. 시스템이 대형화됨에 따라, 다중 입력을 독립적으로 처리하고, 다중 출력을 신속히 발생시켜야 한다는 점에서 실시간 시스템 설계의 복잡성이 커진다. 따라서 실시간 시스템은 일반적으로 많은 다중 작업들이 시간적 제약을 갖고 병렬 처리되어지는 동시 시스템(concurrent system)으로 볼 수 있다

실시간 시스템의 프로그래밍은 일반적인 소프트웨어 설계와는 여러 가지 면에서 구별되어 진다. 우선 실시간 시스템이 실시간 제어기로서 역할을 수행하는 경우, 하드웨어 및 소프트웨어가 결합되어 기능을 갖는 내장형 시스템(embedded systems)으로 설계된다[1][2]. 이 경우, 하드웨어와 소프트웨어의 구분이 명확하지 않으며 설계 및 보완이 동시에 이루어 진다. 또한 실시간 시스템은 시스템과 외부 환경 사이에 많은 입출력 신호가 교환되는 다중의 대형 입출력 시스템이다.

따라서 많은 센서정보를 처리하고, 외부 환경에 대해 다양한 제어신호로 반응한다. 세번째 특징으로는 정보처리와 제어에 시간적 제약이 있다는 점이다. 상호 작용 시스템에서의 응답 지연은 인간 작업자에게 지루함을 줄 뿐 아니라, 예측하지 못한 시스템의 오동작, 때로는 치명적인 위험을 유발하기도 한다. 따라서 출력의 정확성과 동시에 적절한 응답성이 중요시 되는 것이다. 네번째로 실시간 시스템은 대부분 상호 반응 시스템(reactive system)으로서 사건 구동(event-driven)방식의 특징을 갖는다. 각 작업(task)들은 실시간 다중작업(real time multi-tasking)으로 수행되며, 그러므로 적절한 실시간 운영체제를 필요로 한다. 마지막으로 실시간 시스템은 하드웨어의 제약 등으로 인해, 개발환경(development environment)이 복잡하고 개발 생산성이 떨어진다. 예를 들어 종래의 개발환경에서는 모의실험이 완료되면 교차 컴파일(cross-compile)을 수행하여, 원하는 내장형 시스템에 실행파일을 다운 로딩(down loading)하고 프로그램 수행을 통해, 기능을 확인하는 과정을 반복한다. 이와 같은 개발 방법은 순차적으로 진행되기 때문에, 개발 시간의 지연이 발생하고 각 단계에서 발생되는 시스템오류를 사전에 모두 예측하기 어려우며, 시스템 설치 후에도 심각한 오류가 발생되는 등의 문제가 있다

한편 승강기 제어시스템은 다중 입출력 신호들을 갖고, 외부환경과 반응하는 실시간 특성이 강한 시스템의 전형적인 예이다. 본 논문에서는 실제 분산형 승강기 시스템 설계에 있어, 위와 같은 실시간 시스템이 갖는 특성들을 고려하여 설계된 실시간 다중작업 모델과 개발시간의 단축 및 효과적인 시스템 검증을 위해 구축된 개발환경(development environment)을 소개하고자 한다. 시스템의 규모가 커질 수록, 개발시간 단축을 위해 여러 사람이 효과적으로 공동 작업을 할 수 있는 개발환경이나, 사전 개발과 모의 실험환경 등이 필수적이다. 그러나 이와 같은 실시간 시스템의 개발 환경은 내부분 어려한

접수일자 : 1998. 4. 20., 수정완료 : 1999. 4. 20
최병욱 : LG산전 중앙연구소 Embedded System연구팀
임계영 : LG 산전 중앙연구소장
고경철 : 선문대학교 기계및제어 공학부

실시간 운영체제(real-time operating system)를 채택하느냐에 따라 그 운영체제가 제공하는 개발도구에 종속화되기 쉬우며, 개발 환경의 용이성만을 고려하여 선정한 경우, 종종 과도한 시스템사양을 요구하게 되어, 전체적인 개발비용의 증가와 원가 상승의 주요 요인이 되기도 한다[3]. 따라서 내장형 시스템의 개발시간을 단축하고, 보다 체계적인 시스템 프로그램 개발을 위해, 개발 환경을 목표 시스템(target system)과 동일하게 구축하는 가상 프로토타이핑 환경을 제안한다.

본 논문은 다음과 같이 구성된다. II장에서는 다양한 입출력을 갖고 실시간 제어를 수행하는 승강기 제어시스템의 실시간 시스템 모델을 설명하고, III장에서는 다중작업(multi-tasking) 모델을 기반으로 설계되는 승강기 제어 시스템 프로그램의 구조를 소개한다. IV장 및 V장에서는 교차 개발환경에서 실제 시스템 운전을 모의 실험하는 가상 프로토타이핑에 기반을 둔 프로그램 개발 환경에 대하여 논의하고 결론을 정리한다.

II. 실시간 시스템 모델

최근 건물이 대형, 고층화 됨에 따라, 승강기 시스템도 점점 대형화 되고 고속화 되고 있다. 따라서 승강기의 제어구조 또한 기존의 단일제어 구조로 부터 계층화 되고, 분산되는 구조로 발전하고 있다[4]. 분산제어 구조의 승강기 제어시스템에서는 승강장 제어기와 승강기 호기 제어기가 승강장 및 호기마다 별도로 설계되고, 각 제어기간의 정보는 통신망(network)을 통해 전달된다. 일반적으로 승강기 제어 시스템은 호기 제어부, 군 관리 제어부, 안전 제어부, 원격 감시 및 진단 시스템 등으로 구성된다. 승강기 호기 제어기는 승강기의 모든 운행과 외부 기기와의 정보 교환을 총괄한다.

승강장 제어기는 각 층에 설치되고 승강기를 이용할 때 사용하는 상승, 하강 부름 버튼과 요구된 방향을 표시하는 한 쌍의 램프의 입출력 장치를 갖는다. 승강장에서의 부름(call) 명령은 통신망을 통해 호기 제어기로 전달되고, 호기제어기에서는 서비스 가능한 부름인가를 검증한 뒤, 모터를 제어하여 승강기를 부름 층으로 이동시킨다. 승강기가 부름 층으로 운행되는 동안에도 승강장의 방향 표시등과 자동운전 표시등과 같은 호기제어기의 운전에 관련된 각종정보는 승강장 제어기와 승강기 호기 제어기 사이에서 통신을 통해 계속 교환된다. 승강기가 원하는 층에 정확하게 도착하기 위해, 호기 제어기는 구동모터를 엔코더와 승강기 통로 내에 설치된 절대위치 센서를 이용하여 위치제어 된다. 여기서 승객의 승차감과 안전을 위해 부드러운 운동계획 알고리즘이 적용된다. 또한 호기제어기는 승강기가 정지 후 승객의 승하차를 위한 승강기 문의 개폐제어와 승객의 안전을 위한 안전관련 센서 신호처리 등을 수행한다. 이밖에 외부 기기와의 통신으로, 군 관리 제어부, 감시 제어부, 원격 진단 시스템 등과 승강기 운전관련 정보를 교환한다. 군 관리 제어부는 승객들에게 보다 효율적인 승강기 도착 서비스를 제공하기 위하여 여러 대의 승강기 움직임을 총괄적으로 제어하고, 호기 제어부에 명령을 최적으로 할당하고 분할하는 기능을 수행한다. 감시 제어부는 승강기의 운행상황을 안전 계통의 신호등을 처리하여 감시함으로써 고장 발생시 신속한 대처 및 고장 징후에 대한 데이터를 수집하여 고장을 미연에 방지하는 역할을 한다. 원격 감시 진단 시스템은 건물 내에 상주하여 승강기 관리자나 보수자가 없는 경우에도 원격지에서 고장감시 및 원인 분석을 할 수 있는 원격 서비스 환경을 제공한다.

이 분산구조의 승강기 제어시스템의 실시간 특성은 승객의 편리성과 안전성에 직접적인 영향을 미친다[5]. 승강기 제어 시스템의 계층적 구조 특징 및 그 운용을 알아 보기 위하여, 시스템에 접속된 외부 장치들과 이들 간에 교환되는 정보 흐름을 보여주는 컨텍스트 다이어그램(context diagram)을 그림 1에 나타내었다. 승강장 제어기의 부름 단추들 그리고 승강기 통로에서의 센서 입력들은 모두 비동기식으로 처리된다. 이러한 비동기적인 입력이 원하는 시간 내에 처리되지 않으면, 승객은 승강기가 고장인지, 정상 운행 중인지 또는 부름 버튼이 고장인지를 알 수 없게 된다. 특히 승강기 통로에서 감지되는 안전 센서에 대한 처리시간은 승강기의 안전과 직결된다. 이와 같은 분산제어 구조 및 실시간 시스템 특성이 강한 승강기 제어 시스템 개발에 있어서, 구조적인 시스템 프로그램의 설계는 시스템 신뢰성 향상에 필수적이다. 일반적으로 실시간 시스템 프로그래밍을 위하여, 다중 작업(muti-tasking) 구조의 실시간 운영체제가 채택된다. 이러한 구조는 여러 동작이 동시에 복합적으로 진행되고 각각의 작업에 일정한 시간적 제약이 존재하는 경우에 자주 사용되어 진다[6][7].

그림 1은 승강기의 제어를 위한 다중 작업모델을 보여준다. 여기서 대부분의 작업들은 일정한 순서가 없이, 다양한 형태로 그리고 비동기적으로 진행된다. 외부 장치들의 입력은 인터럽트 방식에 의해 처리되거나, 우선순위가 부여된 작업에 의해 관리된다. 우선 순위와 다중 작업 구조를 이용한 방식은 복잡한 형태의 입력을 중요

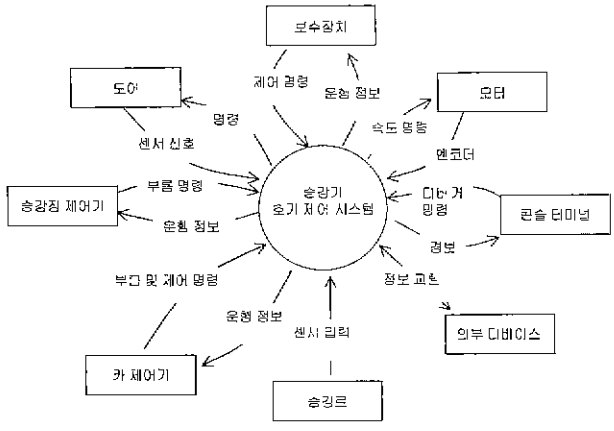


그림 1. 엘리베이터 호기 제어 시스템의 컨텍스트 다이어그램.

Fig 1. Context diagram of the elevator car control system.

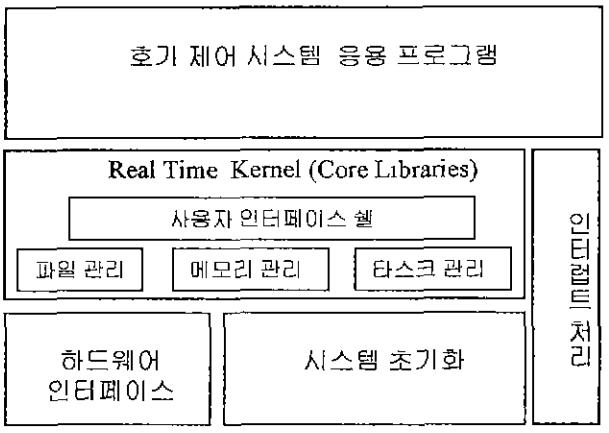


그림 2. 실시간 운영체제를 이용한 시스템 구성.
Fig. 2. System architecture using real time operating system.

도에 의하여 원하는 시간 내에 처리하는데 효과적이다. 만약 이러한 상황에서 실시간 운영체제를 사용하지 않는다면, 이러한 문제들을 모두 수동적 시간관리를 해야 한다. 그러나 건물이 고층화, 대형화 됨에 따라 승강기의 제어시스템에서 처리되어야 할 작업의 양은 늘어나며, 따라서 작업간 통신, 특히 운전 정보관련 입출력 장치들의 처리에 있어, 수동적 시간관리는 한계가 있다. 또한 제어시스템의 응답지연으로 인한 오동작, 승객과 상호작용을 하는 정보표시부의 시간지연 등을 유발하기 쉽다. 실시간 운영체제하에서의 시간관리는 이러한 문제를 방지하며, 다중작업을 각각의 원하는 응답시간 내에 처리할 수 있는 장점을 지닌다. 작업에 대한 우선 순위나 주기는 사용자의 안전과 편리성을 고려하여 정하여진다. 일부 입출력의 경우는 모든 작업에 우선하여 처리되어야 함으로 예외적으로 하드웨어 인터럽트를 사용한다. 그림 1에 표시한 승강기 통로에서의 센서 정보처리, 안전 계통의 입력 처리나 회로 및 시스템의 이상 상태에 따른 진단 처리 등이 이러한 예외적인 경우에 속한다.

그림 2는 실시간 시스템 모델에 사용되어지는 운영체제의 기본적인 구조를 보여준다. 실시간 운영체제에서는 하드웨어와 관련된 부분은 기기 구동기(device driver)로써 하드웨어의 초기화 및 응용 프로그램의 수행을 위한 시스템 초기화를 담당한다 시스템 초기화에는 크게 승강기 호기 제어 시스템의 초기화와 실시간 운영 체제인 커널의 운용을 위한 시스템 초기화 등이 있다. 실시간 운영체제를 사용하기 위한 초기화에서는 먼저 오류 처리를 위한 에러(error)처리기 초기화, 다중 작업 운용을 위한 쓰레드 초기화(thread initialization), 시스템 통신 기능인 메일 박스(mailbox), 파이프(pipe), 세마포(semaphore) 및 이벤트 그룹(event group)의 초기화 등이 진행된다. 또한 실시간 운영체제를 위한 타이머 초기화가 진행되며, 터미널 등의 입출력을 위한 논리적 입출력 기기가 초기화 된다. 이와 같은 커널 초기화를 통하여, 그림 2에 표시된 시스템 프로그램이 다중 작업으로 수행되어 진다. 실시간 커널(kernel)부는 운영체제의 핵심부로서 작업의 생성 및 스케쥴링 그리고 작업 간의 정보 교환 및 동기화 등을 관리한다. 또한 메모리 관리를 수행함으로써 공동 자원의 관리 및 운용을 지원하게 된다. 따라서 이와 같은 실시간 운영체제를 기반으로 함으로써 시스템 프로그램의 효율성과 신뢰성을 높일 수 있다. 한편 승강기의 고유기능을 수행하는 승강기 제어시스템의 초기화는 커널의 초기화 이후에 이루어 진다. 각종 메모리의 초기화, 센서의 입력 및 출력을 위한 기기의 초기화, 그리고 승강기 제어기와 호기 제어기와의 통신을 위한 초기화 등이 이에 해당한다. 이와 같은 시스템 초기화 후에는 다중 작업 모델에 의하여 실시간 시스템이 운영된다. 다중 작업의 모델 설계 및 시간관리 등은 실시간 시스템의 성능에 중요하며 주어진 제한 조건을 만족하면서 최적의 효율을 얻기 위한 기준으로 설계되어진다[8].

III. 다중 작업 모델

실시간 시스템에서 수행되는 시스템 프로그램의 최소 단위를 작업(task)으로 정의하며, 일반적으로 이는 사건 구동(event-driven) 방식과 시간 구동(time-driven) 방식으로 구분된다 사건 구동방식은 인터럽트에 의해 구동되며, 작업을 구동하는 외부 하드웨어에 의해 동기 된다. 반면 시간 구동 방식은 주기적인 타이머를 이용하여 작업을 구동한다. 시간 구동 방식은 타이머 신호를 하나의 사건으로 보아, 사건구동방식으로 분류될 수 있으나, 일반적으로는 외부의 사건을 처리를 하는 사건 구동 방식 프로세스와 대별된다[9] 승강기제어 시스템에 사용되는 작업 모델(task model)은 다음의 3가지 경우로 분류된다.

1) 사건구동방식 작업 - 인터럽트 및 오류처리 작업
2) 시간구동방식 작업 - 주기적구동의 일반응용 작업
3) 초기화 작업 - 다중 작업 이전의 초기화 작업

표 1은 이러한 작업 모델의 예와 각각의 작업내용을 보여준다. 승강기 호기제어를 위한 작업 모델은 작업의 비동기적 행위에 따라서 설계되어야 한다[2]. 예를 들면 표 1의 *Alarm*작업과 *Err_Handle* 작업은 사건 구동 방식으로 설계 된다 *Alarm*작업은 하드웨어에서의 안전계통의 이상을 감지하는 회로에 대한 처리 작업으로써, 오류 사건 발생시 동작된다. *Err_Handle*작업은 오류 발생시에 큐(queue)를 이용하여 처리하는 사건 구동 방식의 작업이다. 사건 구동 방식의 작업은 사건이 구동 되기 전에는 중앙처리장치(CPU)를 점유하지 않는다. 또한 오류 발생시에는 인터럽트와 동일한 수준으로 오류 처리 작업을 할 수 있어, 시스템의 신뢰성을 향상시킨다. 또한 정상적인 상태에서는 사건 구동 방식의 작업은 사건을 기다리는 상태로 존재하여, 전체 시스템의 연산시간에 부담을 주지 않게 된다. 따라서 작업의 우선순위는 높으나 주기적 수행이 필요 없는 작업의 설계에 사건 구동 방식이 적용되어진다.

시스템 오류감시 및 처리방식에 있어, 시스템 오류의 발생은 *Error*작업에서 주기적으로 감시하는 반면, 처리

표 1. 다중 작업 모델 및 작업 내용.
Table 1. Multi-tasking model and contents.

TASK 명칭	작업 내용	설 명
Task_0	초기화 작업	승강기 상태 초기화/다중 작업 생성/실행
DBC_Debug	시간 구동 작업	각종 시스템 변수 및 작업 상태 관찰
Alarm	사건 구동 작업	하드웨어 아림 처리
Err_Handle	사건 구동 작업	발생된 오류에 대한 사건 구동 작업 오류의 경중에 따른 시스템 조치
Event	시간 구동 작업	작업의 사건 플레그 관리 소프트웨어 감시타이머(WDT)오류 발생
Servo	시간 구동 작업	속도프로파일 생성 및 제어/정지시 위치제어
Timer	시간 구동 작업	소프트웨어 타이머의 시간 관리 WDT(Watch-dog Timer)관리
Mode	시간 구동 작업	운전시퀀스 제어/운전방향, 출발 및 정지 제어
C_Net	시간 구동 작업	카 제어기 통신처리 카 부착 센서의 입력 처리
H_Net	시간 구동 작업	승강장 제어기 통신처리 승강장 직접 입력 스위치 처리(주차, 화재경보)
Error	시간 구동 작업	오류 조건을 체크하여 오류 발생 하드웨어, 소프트웨어 및 논리적 오류 체크
Switch	시간 구동 작업	안전 계통 신호 감시/시스템 입력 감시
Ann	시간 구동 작업	보수 장치 입력 처리/승강기 상태 표시 각종 운전 제어 파리미터 측정 및 변경
CRT	시간 구동 작업	건물내 CRT 감시기로 승강기 상태 전송 제어 명령의 수행
SPEC	시간 구동 작업	사양 초기화/중고 테이블 검시
Nctl	시간 구동 작업	호기 제어기간 정보 통신/최적 호기 할당
Dispatch	시간 구동 작업	서비스콜 관리
Traffic	시간 구동 작업	가상의 부름을 발생시켜 외부 장치가 없이도 처리되는 결과를 관찰 가능
IO_Gen	시간 구동 작업	가상의 입출력 신호를 발생시켜 외부 장치가 없이도 운전 프로그램의 구동을 가능하게 함

는 사건 구동 방식의 *Err_Handle*작업에서 수행된다. 만일 *Error*작업에 오류 처리 코드가 포함된다면 대부분의 시간을 아무런 동작도 하지 않으면서 중앙처리장치(CPU)를 점유하게 될 것이다. 이와 같이 사건의 감시와 처리를 별도의 작업으로 분리하는 것을 작업 분해(task decomposition)라 한다. 이러한 작업 분해는 다중 작업 구조에서 작업 실행 효율을 높이는 데 매우 중요하다. 사건 구동방식의 사건 발생은 커널의 기본 주기에 동작하는 *Event* 작업에 의해 감시되며, 사건 플래그(event flag)에 발생한 사건을 기록한다. 그러면 각 사건 구동 작업은 사건플래그를 제공 받아서 구동되며, 작업 완료 후에는 다시 사건 플래그를 기다리는 상태가 된다.

그림 3은 승강기의 여러 가지 입력정보가 각각의 해당 작업에 의해 응집 처리되는 작업 모델을 보여준다. 이 때 각 작업모델은 작업 응집(task cohesion)의 기준에 따라 설계된다[8]. 예를 들면, 승강장의 입력 및 출력을 처리하는 작업은 *H_Net* 작업으로 응집되고, 승강기 입력 및 출력을 처리하는 작업은 *C_Net* 작업으로 응집되어 처리된다. 그리고 승강로 내부에 안전 계통의 센서 입력과 위치 센서정보 등에 관한 작업은 *Switch* 작업으로 응집되어 진다. 모터의 구동 및 위치 제어 등의 기능은 *Servo* 작업으로 응집된다. 이와 같이 수많은 입출력

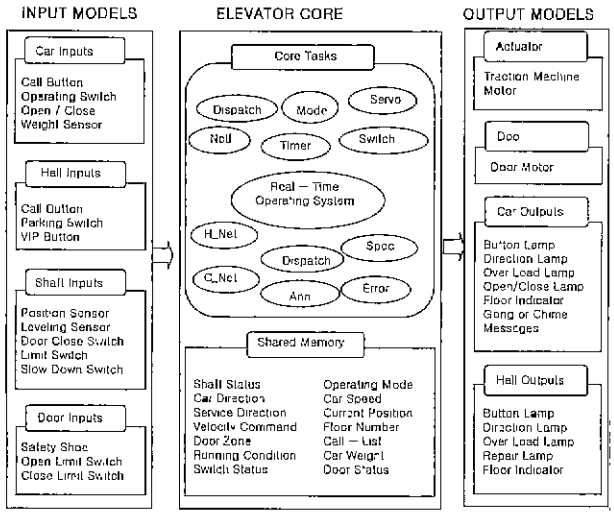


그림 3. 엘리베이터 소프트웨어 시스템 모델.
Fig. 3. System model for the elevator software.

들에 관련된 작업들을 몇 개의 작업으로 응집함으로써, 비동기적으로 발생하는 각종 입력들을 처리하는 작업들의 구조를 간략화하고 효율적인 시간관리를 통해 각각의 작업처리를 주어진 제한시간 내에 처리할 수 있다. 이 밖에 외부 기기와의 입출력을 위한 작업으로, 승강기의 설치조건에 따른 여러 가지 운전 방식의 결정이나 이상 상황 발생시에 승강기를 안전하게 정지 위치까지 이동시키는 mode작업이 있다.

시스템 프로그래밍에서 위와 같은 작업 응집 및 구조화 설계 다음으로 중요한 것이 작업 우선 순위의 결정이다. 일반적으로 실시간 운영체제는 우선 선점 시간관리(preemptive scheduling)방식을 사용한다[2][8]. 이 방식에서는 시스템의 효율 및 성능을 고려하여, 시스템 설계자가 적절한 우선 순위를 작업에 미리 배부한다. 본 연구에서는 시스템의 안전 및 기능의 중요성을 고려하여 작업모델에 대한 주기와 우선 순위를 정한다. 즉 승강기 제어에 있어서 가장 중요하고 수행 주기가 빠른 *Mode*, *Servo* 등의 작업들의 경우, 우선순위를 높게 배정한다. 승강장의 입력 및 출력을 처리하는 *H_Net* 작업의 경우, 승강장의 부름에 대한 처리작업의 지연은 승강기의 안전성 측면에서는 문제가 없으나, 위치 제어를 담당하는 *Servo* 작업이 승강장을 관리하는 *H_Net* 작업에 의하여, 작업 지연이 온다면, 이는 시스템 안전과 전체 성능에 막대한 영향을 미치게 된다. 작업들의 우선순위는 안전, 성능, 효율 등을 고려하여 정하여진다.

그림 3에서 보듯이, 작업간의 정보교환은 공유 메모리를 통하여 이루어지며, 데이터 통신의 동기를 위해, 커널 시스템내의 세마포어(semaphore)가 사용되어 진다. 공유메모리와 각 작업과의 관계는 매우 복잡하게 얽혀 있으며 시스템의 수행의 효율과 작업간의 통신의 분리라는 두 가지 관점에서 본 시스템의 실시간 시스템의 효율을 먼저 고려한 것이다. 따라서 공유메모리에 있는 승강기의 운행에 관련된 상태 정보는 거의 모든 작업이 사용하게 된다.

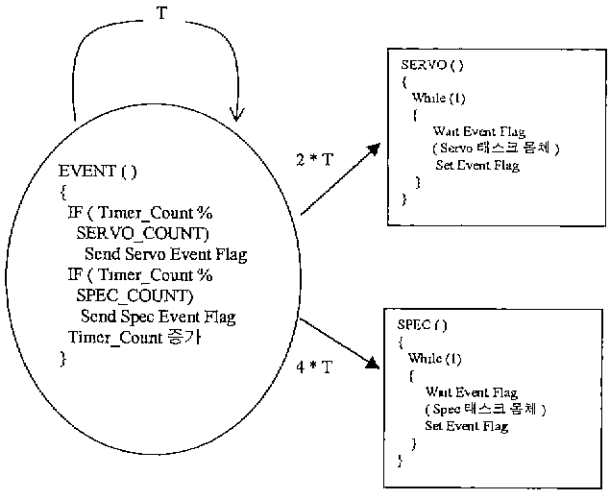


그림 4. 타이머를 이용한 다중 태스크 스케줄링.
Fig. 4. Multi-tasking scheduling using timer interrupt.

다중 작업 구현에서 우선 순위에 의한 작업 모델이 결정되면 효율적인 시간관리를 설계하여야 한다. 실시간 시스템 설계분야에 있어서 작업 시간관리(task scheduling)는 매우 관심이 높은 연구 주제이다[10]-[12]. 그림 4는 다중 작업의 동기화 및 주기적 작업 구동을 위한 알고리즘의 한 예를 보여준다. 그림 4에서 *Event* 작업은 시간구동 타이머(timer)로서 실시간 운영 체계에서 제공되는 기능이다. 실시간 운영체제상의 모든 작업은 기본적인 시간의 단위인 틱(tick)의 배수에 의하여 구동된다. 이러한 타이머는 하드웨어 타이머에 의하여 구동되며, 다른 작업보다 우선 순위가 높은 소프트웨어 인터럽트를 발생한다. 타이머를 이용한 작업의 시간관리는 우선순위의 조정에 유연성이 떨어지는 단점이 있다. 이러한 점을 보완하기 위해, 이벤트 플래그를 이용한다. 그림 4에서 보면, 타이머 구동 주기가 T 인 경우 *Event* 작업은 T의 주기로 구동된다. 이 경우에 *SERVO_COUNT* = 2이고 *SPEC_COUNT* = 4라면 EVENT에 2 * T의 주기로 *Servo* 작업에 이벤트 플래그를 제공하며, *Spec* 작업에는 4 * T의 주기로 이벤트 플래그를 제공한다. 따라서 Servo 작업과 *Spec* 작업은 각각 2 * T 및 의 4 * T의 주기로 구동된다. 작업들의 구동주기는 서로 소수로 구현되어, 한 순간에 많은 작업이 동시에 구동되지 않도록 한다. 한 주기에 여러 개의 작업이 동시에 구동을 요구하면, 각 작업간의 동작은 우선 순위에 기초하여 운영체제에서 관리하는 컨텍스트 스위칭(context switching)에 의해 이루어 진다[6][7].

IV. 실시간 시스템 개발

지금까지 실시간 시스템인 승강기 호기 제어기의 특성과 다중 작업 모델에 기반으로 한 시스템 프로그램 설계에 관하여 논하였다. 이와 같이 다양한 입력과 출력이 조합적으로 운영되는 다중 작업 실시간 시스템에 있어서, 프로그래밍 개발 환경은 개발 시간의 단축, 신뢰성의 향상, 그리고 다양한 조건에서의 운전을 모의실험을 위

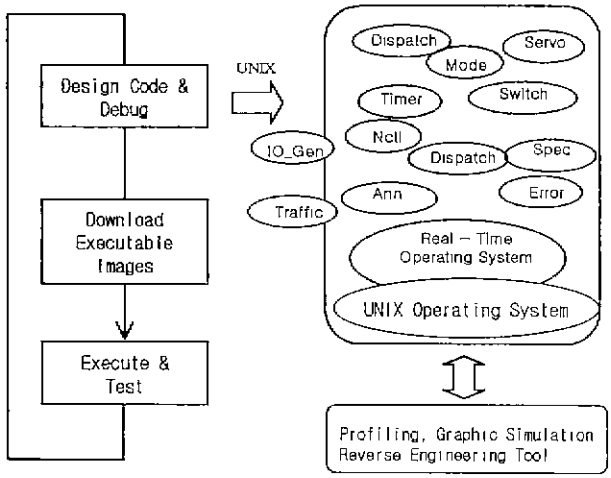


그림 5 실시간 시스템 프로그램 개발 환경.
Fig. 5. Programming development environment for real time systems.

해 필수적이다. 특히 시스템 프로그램의 규모가 커지고 복잡해 짐에 따라, 프로그램의 변경, 재사용(reuse), 관리적 측면에서, 통합 개발 환경의 의미는 더욱 중요해진다. 그림 5는 표 1의 승강기 호기 제어기 프로그램의 개발을 위한 실시간 시스템 개발 환경을 보여준다. 그림 5의 좌측에 도시한 부분은 개발 플랫폼 상에서 이루어지는 코딩 및 디버깅 작업으로, 종래의 내장형 시스템의 개발 환경을 보여준다. 일반적으로 개발 플랫폼의 하드웨어가 목표 시스템(target system)의 하드웨어와 다른 경우, 교차 컴파일(cross compile)을 수행하여 목표 시스템에서 수행 가능한 실행 파일을 만들어, 목표 시스템에 다운로딩한다. 그리고 목표 시스템에서 실행을 시킴으로써 시스템 프로그램을 검증한다. 그리고 오류수정이 완료될 때까지 이와 같은 과정을 순차적으로 반복한다.

그러나 이와 같은 교차개발 방법은 개발환경에서는 최종 응용에 적합한 메모리 용량 을 만족시키기 어려우며, 목표시스템에서는 프로그램 디버깅을 위한 다양한 분석 도구를 사용할 수 없게 된다. 따라서 개발 플랫폼과 목표 시스템의 프로그램 동작 환경을 동일하게 구축하는 것이 중요하다. 이와 같은 동일 개발환경에서는 매번 다운로딩 없이 개발 플랫폼에서 다양한 도구를 이용하여 프로그램을 개발할 수 있으며, 완성 단계에서 목표 시스템으로 다운 로딩 함으로서 프로그램을 최종 시험하게 된다. 따라서 개발 시간 단축 및 시스템의 신뢰성 제고 등의 효과를 얻을 수 있다. 그림 5의 오른쪽 부분은 이와 같은 개념에 의해 구축된 실시간 개발환경을 보여준다 즉 개발 플랫폼에서의 프로그램 수행이 목표 시스템에서 프로그램 수행과 동일한 효과를 거두도록 개발환경을 구축한다. 사용된 개발 도구는 개발 플랫폼인 *UNIX* 워크스테이션(workstation)과 *UNIX*의 다중 쓰레드(multi-thread) 프로그램으로 구현된 *UNIX*용 실시간 운영체제이다. 따라서 모든 응용프로그램은 *UNIX*용 C컴파일러로 컴파일되며, *UNIX* 환경에서 수행된다. 그림 5는 하드웨어와 관련된 작업을 제외한 승강기 시스템

프로그램을 구성하는 작업들을 보여준다.

개발플랫폼에서 완성된 프로그램이 목표시스템에서 그대로 수행될 수 있도록 최종 컴파일 단계에서 전처리를 위한 매크로(macro)가 선언된다. 개발환경에서 제외된 하드웨어부의 모의실험을 위해, 가상 프로토타이핑 환경을 구축한다. 그림 5에서 *Traffic* 작업은 승강장 입력 및 호기 입력을 대신하는 계획된 부름의 발생이나 무작위(random) 형태의 부름 발생을 담당하고, 모든 부름의 조합을 시험한다. *IO_Gen* 작업은 승강장 입력과 각종 센서(모터의 엔코더 및 도어 센서) 등을 원하는 위치에서 생성 시켜주는 기능을 수행한다. 즉 승강기가 이동시 실제 승강로에서 발생하는 각종 센서를 마치 실제 시스템과 동일하게 발생 시키는 역할을 수행한다. 이와 같이 발생된 입력들은 *Switch* 작업이 처리하여, 모든 프로그램 환경을 실제 목표시스템과 동일하게 한다. 그림 5의 개발환경은 실제 시스템에 내재된 동적특성, 즉 시간 지연이나 관성의 효과를 무시한다면, 승강기 시스템의 거의 모든 제어 및 감시 상태를 실시간 모의 실험할 수 있게 하여 준다.

본 연구를 통해 구축된 가상 프로토타이핑 개발환경을 통해 여러 사람의 공동 프로그래밍 및 디버깅 작업이 가능하며 다양한 소프트웨어 개발 도구를 이용하여 실시간 검증을 할 수 있다. 본 연구에서는 *Quantify*라는 프로그램을 이용하여 시스템 프로그램내의 각 작업의 수행시간 및 함수의 수행 횟수를 검증하였다. *Qunatify*는 *UNIX* 프로그램으로서 그림 5에 나타낸 작업의 수행 시간을 분석하는데 사용된다. 이러한 도구의 사용은 목표시스템에서는 불가능하며, *UNIX*를 기반으로 한 개발환경에서만 가능하다. 이러한 검증을 통하여, 작업의 우선순위와 주기를 조정하고, 작업들을 원하는 시간 내에 재배치하고 프로그램의 최적화를 수행한다[14]. 프로그램의 논리적 오류 등은 역 공학 도구(reverse engineering tool)인 *Logiscope*를 이용하여 관찰할 수 있다[13]. 이러한 역 공학 도구는 다양한 작업에 의한 고유 메모리의 접속 및 함수 구조 그리고 논리적 오류를 개발환경에서 디버깅할 수 있다. 이러한 환경은 실제 시스템에서는 실험하기 힘든 여러 조건을 가상적으로 발생시켜 실험해 봄으로써, 많은 입력의 조합에 따른 시스템 오류를 사전에 제거하여 신뢰성을 확보하게 된다. 그러나 그림 5와 같은 개발환경에서의 실험은 단지 실시간을 모방한 모의실험이며, 보다 정확한 시간은 실제 목표 시스템에서 오실로스코프(oscilloscope) 등과 같은 계측장비를 이용하여 얻을 수 있다. 표 2는 제안된 개발환경에서 수행되는 각 작업의 주기 및 CPU 점유율의 결과를 보여준다 표 2는 개발된 시스템이 CPU 점유율 면에서 안전하게 다중 작업이 수행되고 있음을 보여주며, 이러한 결과로부터, 개발된 시스템 프로그램과 구축된 개발 환경이 효율적인 작업 시간관리의 검증뿐 아니라 실시간 시스템 개발 시 발생하는 많은 시행 착오를 사전 예방하고 개발시간을 단축 및 시스템 신뢰성을 개선하는 데 효과적임을 보여준다

표 2. 작업 수행 주기 및 CPU 수행 비율.
Table 2. Task execution time and CPU processing rate

TASK	주기	평균 실행 시간	990ms 내 수행 횟수	CPU 점유율
Position Servo Task (SERVO)	30ms	3.15ms	33	평균 10.0%
Software Time Task (TIMER)	30ms	1.98ms	33	평균 6.6%
Swith Input Task (SWITCH)	30ms	3.54ms	33	평균 11.8%
Mode Service Task (MODE)	120ms	5.92ms	8.25	평균 4.94%
Car Network Task (C_NET)	90ms	0.696ms	11	평균 0.774%
Error Check Task (ERROE)	180ms	1.3ms	5.5	평균 0.73%
Hall Network Task (H_NET)	60ms	2.2ms	16.5	평균 3.7%
Management Task (NCTL)	180ms	3.92ms	5.5	평균 2.18%
Annunciator Task (ANN)	300ms	18.3ms	3.3	평균 6.1%
Dispatch Task (DISPATCH)	540ms	5.7ms	1.8	평균 1.04%
Specification Task (SPEC)	990ms	3.56ms	1	평균 0.36%
Montoring Task (CRT)	90ms	4.778ms	11	평균 4.4%
승강장 게이기 통신 ISR		0.01ms	33	평균 0.34%
절대 위치 센서 진입 ISR		1.1ms	1	평균 0.1%
절대 위치 센서 탈출 ISR		0.22ms	1	평균 0.20%
Total	-			평균 53.084%

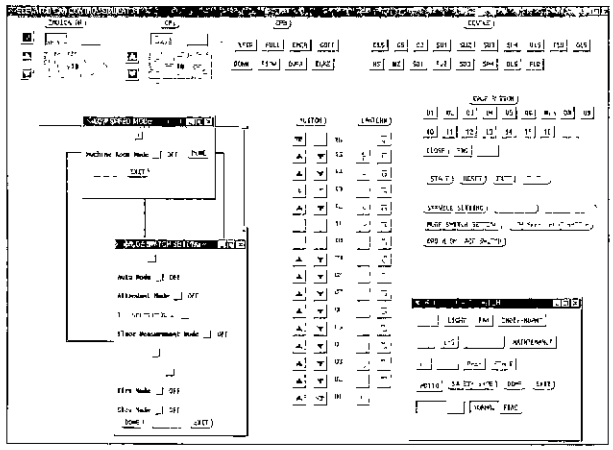


그림 6. 승강기 그래픽 시뮬레이터.
Fig. 6. Elevator graphic simulator.

그림 6은 개발 환경의 동작 검증을 위한 그래픽 시뮬레이터(graphic simulator) 화면을 보여준다. 그래픽 기능은 *UNIX*의 *Motif* 그래픽 라이브러리를 이용하여 구현된다[15] 일반적으로 그래픽 시뮬레이터는 입출력 관점에서 소프트웨어의 행위를 검증하는데 사용되어 진다. 이와 같은 관점에서 많은 입력 및 출력으로 구성된 승강기 제어 프로그램의 검증을 위하여서는 그래픽 시뮬레이터가 매우 적절한 방법이다. 그림을 보면 승강기의 제어 입력을 나타내는 *MODE SWITCH SETTING* 조작판 부분이 버튼으로 구현되어 있어서, 각 제어 모드에 따른 승강기의 운행을 검증할 수 있다. 승강기의 운행은 간단한 애니메이션으로 묘사되며 *IDICATOR* 및 *DEVICE*에 운행 상태가 표시되어 진다 이와 같은 조작 버튼 및 센서 출력 그리고 표시기 등이 승강장 및 승강기 내부에 설치되어 있으며 각각의 구동 부분을 그래픽 시뮬레이터

와 연동을 함으로써 모든 제어 프로그램의 논리적 검증이 가능하다. 또한 각 동작 모드에 대한 실제 승강기의 동작을 묘사함으로써, 개발환경뿐 아니라 승강기 보수를 위한 교육용으로도 유용하다. 새로운 알고리즘의 검증 및 보수를 위한 실제 시스템에서의 실험 및 교육은 자원적인 면에서 제한이 있고 위험성이 내포된다. 반면 시뮬레이터는 안전하여 통상 위험한 시스템의 운행시험 및 점검, 보수교육에 널리 이용될 것으로 기대된다.

그래픽 시뮬레이터의 또 하나의 장점은 실제 시스템에서는 실험하기 힘든 상황에 대하여 각종 운전 제어 스위치 입력을 *GUI*(Graphic User Interface)를 이용하여 구현함으로써, 다양한 조합의 입력 상태에 대한 검증이 가능하다는 점이다. 모델러(modeler)와 같은 프로토타이핑(prototyping)의 기능의 추가 및 보완이 이루어진다면 향후 보다 완전한 실시간 그래픽 시뮬레이터로서의 기능을 수행할 수 있을 것으로 예측된다. 또한 논리 검증기와 연동이 되면 다중 작업모델에 대한 상태 검증 및 행위 검증 등을 단계별로 수행할 수 있을 것이다.

VI. 결론

지금까지 제어시스템에 관한 연구는 주로 하드웨어 구조와 제어 알고리즘에 주로 집중되어 왔다. 그러나 실시간 제어 시스템 개발에 있어 소프트웨어가 차지하는 비중은 날로 높아지고 있다. 이러한 관점에서 본 연구에서 제시된 실시간 시스템을 위한 통합화된 개발 환경은 개발 시간의 단축이나 실시간 소프트웨어의 신뢰성 향상을 위하여 필수적이라 할 수 있다. 상용화된 프로토타이핑 도구나 통합 환경은 하드웨어의 모델링 및 원형 제작을 위한 모델링이 부가적으로 필요하며, 각 프로그램간의 데이터 공유와 목표 시스템으로의 완전한 이행을 위하여 많은 작업들이 부가적으로 수반되어야 한다. 따라서 기존의 개발 도구는 비용이 비싸지고, 도구 자체를 습득하는데도 상당한 시간이 필요하게 된다.

본 논문에서는 산업체에서 실제로 설계되는 실시간 시스템의 소프트웨어 개발 과정을 승강기 제어시스템의 예를 통해 소개하고, 효과적인 개발을 위해 동일환경에서 구축된 실시간 시스템 개발 환경에 대하여 기술하였다. 본 연구에서 구현된 개발 환경은 시스템 클럭(system clock)에 의한 시간적인 단위만이 상이하다는 것을 제외하면 실제 목표 시스템과 동일하게 설계된다는 특징을 갖는다. 이러한 개발환경은 실시간 시스템의 효율성과 안정성, 그리고 유지 및 보수성 등의 측면에서 효과적이며, 일반적인 실시간 제어 시스템 개발에 공통적으로 사용 가능하다. 특히 다양한 입력과 수많은 출력 조합을 갖는 대형 실시간 시스템 개발에 있어서 보다 완벽한 사전 모의 실험 개발환경의 구축이 필수적이다. 향후 이와 같은 개발 환경은 설계 단계에서부터 일관성을 갖는 통합화된 실시간 개발 도구(integrated real time CASE tool)의 기반이 되며, 개발기간 단축이 중요시 되는 산업체에서 좀 더 향상된 품질과 안정된 시스템 개발에 유용할 것으로 기대된다.

향후 과제로서 요구 분석의 가시화(visualization)와 행위의 검증 및 실험을 위한 논리 검증기, 각 작업 중심의 실시간 검증을 위한 실시간 특성의 분석 기능 부여, 그리고 이러한 개발 과정을 프로토타이핑할 수 있는 기능이 포함된 도구에 관한 연구가 필요하다. 이러한 도구는 매우 광범위하여 현재 단계적으로 개발 중에 있다. 이러한 통합된 도구를 통한 체계적 개발 만이 복잡하고 신뢰성이 요구되는 실시간 제어 시스템의 개발에 대한 시장 요구를 만족할 수 있을 것이다.

참고문헌

[1] P. A. Laplante, *Real-time Systems Design and Analysis · An Engineer's Handbook, 2nd Edition,* IEEE Press, pp. 154-156, 1997.

[2] H. Gomaa, *Software Design Methods for Con-Current and Real-time Systems, SEI Series in Software Engineering,* Addison Wesley, 1993.

[3] N. Cravotta, "Real-time operating systems," *Embedded Systems Programming,* pp. 97-104. March, 1997.

[4] 박종헌, 임계영, "승강기 원격 감시 진단 시스템," 제어·자동화·시스템 공학회 제1권, 제1호, 7월호, 1995.

[5] 최병욱, "분산제어형 승강기 개발," 승강기 기술. 창간호, 10월호, 1997.

[6] Nucleus PLUS Reference Manual, "Nucleus real-time software," *Accelerated Technology Inc.,* 1996.

[7] VxWorks Programmer's Guide, *Wind River Systems Inc.* 1996.

[8] E. Klein, "RTOS design - how your application is affected," *Embedded Systems Conference East,* pp. 533-558, March, 1997.

[9] W. A. Halang and A. D. Stoyenko, *Constructing Predictable Real Time Systems,* Kluwer Academic Publisher, 1991.

[10] Liu and Layland, "Scheduling algorithm for multiprogramming in a hard real time environment," *J. ACM,* 20(1), pp. 46-61, 1973.

[11] M. Dertouzos, "Control robotics : the procedual control of physical process," *Proc of the IFIP Congress,* 1974.

[12] B. Cheng, A. D. Stoyenkp and T. J. Marlowe, "Least-space-time-first scheduling algorithm : a policiy for complex real-time task in multiple processor systems," *IFAC Workshop on Real Time Programming,* pp. 33-38, 1994.

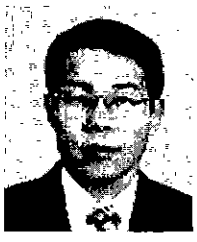
[13] LOGISCOPE Manuals, *VERILOG,* 1996.

[14] QUANTIFY Maunals, *Rational Software Corp.,* 1997.

[15] Motif, Programmer's Guide(1), (2), Sunsoft, 1993.

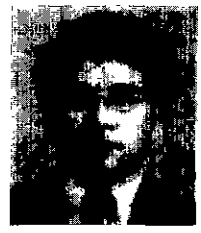
최 병 욱

1963년 2월 13일생. 1986년 항공대 전자공학과 졸업. 1988년 한국과학기술원 전기 및 전자공학과 석사. 1992년 동 대학원 박사. 1988년-현재, LG산전 중앙연구소, Embedded System연구팀 책임연구원. 관심분야는 내장형 실시간 시스템, ASIC, 소프트웨어 엔지니어링, 모션 제어.



임 계 영

1975년 서울대 전기공학과 졸업. 1983년 뉴욕주립대 전기공학과 석사. 1985년 동대학원 박사. 1978년-1981년 국방과학연구소 연구원. 1986년-현재 LG산전 중앙연구소장. 상무이사. 제어·자동화·시스템공학회 이사. 관심분야는 Real-time control, 적응제어 등.



고 경 철

1959년 12월 8일생. 1982년 연세대학교 기계공학과 졸업. 1984년 한국과학기술원 기계공학과 석사. 1994년 동 대학원 정밀공학과 박사. 1984년-1988년 금성사 중앙연구소 로봇개발팀장. 1994년-1996년 LG산전 NC/로봇연구실장. 1997년 LG종합기술원 지능제어팀 책임연구원. 1998년 3월-현재 선문대학교 기계 및 제어공학부 조교수. 관심분야는 로봇공학, 지능제어, 실시간 시스템 설계, Vision응용.