소특집 : 자동화

자동화기기용 Embedded System Software의 개발동향

임 동 진*
(*한양대 공대 제어계측공학과 조교수)

1. 서 론

Microprocessor의 발명은 그 이전에는 가능하지 않았던 많은 것들을 가능하게 하였고, 이로 인하여 인간의 생활에 미친 영향은 참으로 막대한 것이다. 이러한 것들은 단순히 민수용기기뿐이 아니라 산업용기기에 대해서도 마찬가지다. 또한 그의 발달은 기하급수적으로 이루어지고 있어서 이러한 영향은 더욱더 커질것으로 생가된다. 특히 최근 32bit microprocessor의 도입은 이러한 기기들의 새로운 차원을 열것으로 생각되어진다.

그러나, 최근의 이러한 급속한 발전에도 불구하고 장애적 요인이 드러나고 있는데, 그것은 hardware의 발달속도를 software가 미쳐 따라잡지 못하고 있는 것이다. 그 한가지 예로서 현재 많이 보급되어 있고 또한 앞으로도 많은 확산이 예상되는 IBM PC 및 그 호환기종들을 들 수 있다. Intel 8088 microprocessor를 사용한 초장기의 IBM PC와 오늘날의 386 PC 혹은 486 PC를 비교해보면 참으로 많은 발전이 있었음을 알 수 있다. 그러나 그것은 hardware적인 측면이고 software가 과연 그만큼 발전했는가 또는 지금의 software가 hardware에 걸맞는가 하는데 대해서는 많은 의심의 여지가 있다. 이러한 현상은 자동화 기기와 같은 산업용 기기에서는 더욱 두드러진다. 개인용 컴퓨터의 경우는 비교적 보급이 많이 되어있고 software의 개발에 많은 사람들이 기여를 하고 있으므로 다소 나은 편이라고 할 수 있지만, 자동화기기와 같은 특수용도의 기기의 경우는 훨씬 더 열악한 조건이라고 볼 수 있다. 이와 같은 현상은 자동화기기용 software의 개발 및 유지보수에 더욱더 많은 비용이 들게되는 결과를 초래한다. 즉 자동화기기들에게서 요구되는 기능은 더욱 복잡해지고 또한 사용되는 microprocessor hardware의 성능은 향상 되므로 이에 맞는 software의 개발은 점점 더 어려워지게 되는 것이다. 반면, 이러한 software의 개발환경은 보조를 맞추지 못해온 것이 사실이어서, 혹자는 90년대의 hardware에 70년대의 software technology를 사용하고 있다는 표현도 쓰고있다. 그런데 다행히도 1980년대에 들어서면서 이러한 embedded software의 개발에 새로운 장을 여는 개발 환경이 도입되기 시작하고있다. 본고에서는 embedded software의 특징들을 알아보고 이들을 위한 개발 환경에 대하여 서술하고자 한다.

2. Embedded System Software의 특징

일반적으로 컴퓨터는 범용컴퓨터와 embedded computer의 두가지로 분류가 가능하다. 범용컴퓨터는 keyboard, display, disk와 같은 표준 주변기기를 가지며, 그 software는 사용자에 의해서 program-

ming이 가능한 환경을 갖추고 있다. 개인용 컴퓨터가 바로 이 범주에 소속된다. 반면 embedded computer는 같은 컴퓨터 이면서 이와는 전혀 다른 형태를 갖는다. Embedded computer는 단독적으로 사용되는 컴퓨터가 아니고 큰 시스템의 한부분에 소속되어 그 시스템이 필요로하는 여러가지의 정보처리를 수행하게 된다. 예를 들면, 모터가 단독적으로는 아무 용도가 없으나 세탁기나 선풍기의 한 부분으로 구성되어서 역할을 하는데, 이때 모터는 세탁기나 선풍기에 embedded되어 있다고 볼 수 있는 것이다. 따라서 embedded computer는 범용컴퓨터와는 달리 사용자가 program을 변경할 수 없으며 사용자에게는 컴퓨터라기보다는 하나의 부품으로서 인식되어진다. 이러한 embedded computer가 사용되는 예로서는 공장 자동화기기, 사무용 통신장비, 산업용 로봇등 무수히 많은 기기들이 있을 것이다. 이러한 기기들에게는 여러종류의 sensor나 actuator들이 연결되어 있으며 user interface는 기기종류나 용도에 따라 다르므로 이에 맞는 software가 필요하다. 물론 이러한 embedded system software는 범용컴퓨터의 software와 유사한 점이 많이 있지만, 반면 특수한 목적에 사용되는 software인 만큼 특별히 요구되는 사항도 많이있다. 그러면 먼저 embedded software에서 요구되는 사항들을 알아보기로 한다.

일반적으로 embeded computer는 내장되어 있는 기기의 제어에 사용되게 되어 실제 기기의 여러 부품 및 외부 장치와 직접 연결되게 된다. 따라서 그 응답속도는 연결되어 있는 여러장치들에 의해서 결정되어야 한다. 이를 범용컴퓨터와 비교하여 보면, 범용컴퓨터의 응답속도는 컴퓨터 자신에 의해서 결정되며 응답속도가 느린경우라도 사용자가 기다리는 시간이 길어질 뿐 응답속도와 그 결과는 무관하다. 반면 embedded computer의 경우는 어떤결과나 정답을 주어진 시간 이내에 내어주어야 하며, 설령 답이 계산이나 논리적으로 맞는다 할지라도 주어진 시간내에 나온 답이 아니면 전혀 소용이 없는 답이 되는 것이다. 그러므로 embedded system software의 첫번째로 요구사항은 real-time software이어야 하는 것이다.

두번째 embedded system software에서 요구되는 사항은 컴퓨터에서 여러가지의 작업이 동시에 수행될 필요가 있다는 것이다. Process controller의 예를 들면, 어떤 process의 제어를 위해서는 여러가지의 parameter 값들의 sampling이 필요한데 각 parameter마다 요구되는 sampling주기가 다를 수가 있을 것이다. 즉, 여러개의 parameter가 서로 다른 주기로 동시에 sampling될 필요가 있게된다. 물론 CPU는 어느 한 순간에는 한가지 작업만을 하겠지만 여러가지의 작업을 교대로 하면서 전체적으로 보았을 때는 여러 가지의 작업이 동시에 이루어지는 듯이 보이도록 하게된다. 또한 여러 가지의 I/O device를 제어할 경우도 마찬가지이다. 어느 한가지의 I/O device를 제어하는 동시에 다른 I/O device의 상태를 검사해야 하게된다. 이러한 요구사항을 parallelism 혹은 concurrency라고 부르며, 이것이 real-time성 외에 embedded system software에서 요구되는 사항이다. 이러한 concurrency의 구현에 있어서 특히 어려운 문제 중의 하나는 여러개의 작업이 동시에 수행되면서 memory와 같은 resource는 공유를 해야하는 경우가 많으며 이때 공유 resource의 적절한 관리이다. 이외에도 각 작업간의 synchronization을 가능하게 하는 방법의 필요성등 embedded system의 규모가 커짐에 따라서 요구되는 사항들은 점점 많아지게 된다.

3. Embedded System Software의 설계

그러면, 앞절에서 요구되는 사항들을 만족할 수 있는 embedded system software의 설계 및 제작에는 이러한 방법이 사용될 수 있는지 알아본다. 이들을 크게 나누어 세가지로 분류한다면

1. Single Program Approach
2. Foreground/Backgroung System
3. Multi-Tasking Approach

라고 할 수 있다. [1]

첫번째인 single program approach는 가장 단순하고 쉬운 방법으로 요구되는 모든 기능들을 한개의 program module에 포함시켜서 반복적으로 수행을 시키는 것이다. 이때에 요구되는 사항은 가장 시간이 오래 걸리는 loop의 수행시간이 시스템에서 요구되는 응답시간보다 짧아야 하는 것이다. 따라서 대단히 빠른 응답을 요하는 시스템이나 큰 규모의 시스템에는 전혀 적합한 방법이 아니다. 그러나 대단히 직관적이고 단순한 방법이므로 응답시간의 조건

만 만족될 수 있으면 사용이 가능한 방법이라 볼 수 있으며, 대개는 assembly language programming environment를 사용하게 된다.

두번째의 방법인 foreground/background system은 system software에서 요구되는 기능을 빠른 응답이 요구되는 부분과 다소 시간적 여유가 허용되는 부분으로 나누어서, 빠른 응답이 요구되는 기능은 foreground에서 나머지기능은 background에서 수행이되며 foreground의 module들은 interrupt와 같은 mechanism을 사용하여 background의 module들보다 우선적으로 수행이 되도록 하는 것이다. 물론 이때에도 시간적 요구사항이 만족되어야하는데, 최악의 경우로 foreground의 작업들을 수행하기 위하여 CPU의 processing time을 모두 소모한다면 background의 작업들은 수행이되지 못하고 전체 시스템의 기능은 중단이 되고 말 것이다. 이 foreground/background approach의 개발 환경은 real-time operating system에서는 당연히 제공이 되지만, Digital Research의 CP/M이나 Microsoft의 MS-DOS와 같은 non real-time operating system에서도 사용이 가능하며, 아마도 지금 현재 사용되고 있는 embedded system 중 가장 많은 종류가 이 방법을 사용하고 있으리라고 생각된다.

세번째 방법인 multi-tasking approach는 비교적 대형의 시스템에 사용되는 방법으로 두번째의 foreground/background approach를 좀더 확장하여, 기능별로 program을 module화 하고 priority라고 하는 우선 순위를 배정하여 시간적 제약이 많은 기능은 높은 우선 순위를, 시간적 제약이 적은 기능은 낮은 우선 순위를 갖는 module에서 수행되도록한다. 이때 각각의 작업을 task 혹은 process라고 부르며, 이러한 multi-tasking system의 구현을 위해서는 real-time multi-tasking operating system environment가 필요하며 다음과 같은 사항들이 가능해야 한다.

task의 발생
task의 scheduling
task간의 data 공유
task간의 synchronization
task와 외부 사건과의 synchronization
task의 시작과 종료의 제어

여기에서, 흔히 혼동하기 쉬운 것으로 multi-user/multi-programming operating system과 multi-tasking operating system의 차이를 명확히 알아둘 필요가 있다. 그림 1에서 보는바와 같이 multi-user operating system은 여러 사용자가 동시에 컴퓨터를 사용하고 있어도 각 사용자는 컴퓨터가 자신의 program만을 수행중인 것처럼 느끼도록 해준다. 따라서 어느사용자의 program이 다른 사용자의 program을 절대로 방해할 수 없도록 해주며, 각 사용자는 자신의 보호된 환경내에 program의 수행이 가능하다. 반면 multi-tasking operating system 환경하에서는 그림 2에서 볼 수 있듯이 한사람의 사용자가 여러개의 task를 수행시키며 이 task끼리는 서로 협조하여 사용자가 원하는 작업을 하도록 해준다. 따라서 여기서 필요한 것은 각 task들을 위한 보호된 환경이 아니라, task간의 상호 통신 및 data의 공유이다. 그외에 중요한 차이점중의 한가지는 어느 순간에 CPU가 어느 task를 수행할 것인가를 결정하는 task scheduling방법이 다르다는 것이다. Multi-user operating system에서의 scheduling은 우선 순위가 높은 task에서부터 가장 낮은 task까지 CPU time이 할당이 되는데 높은 것에는 많은 시간이 낮은 것에는 적은시간이 할당 되도록 한다. 따라

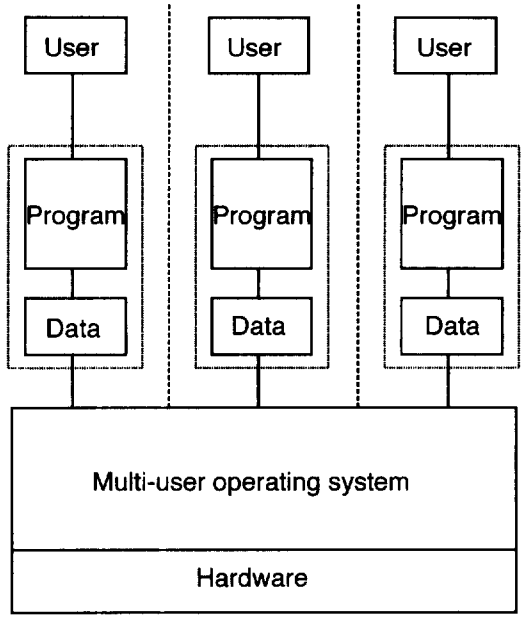

그림 1. Multi-user operating system

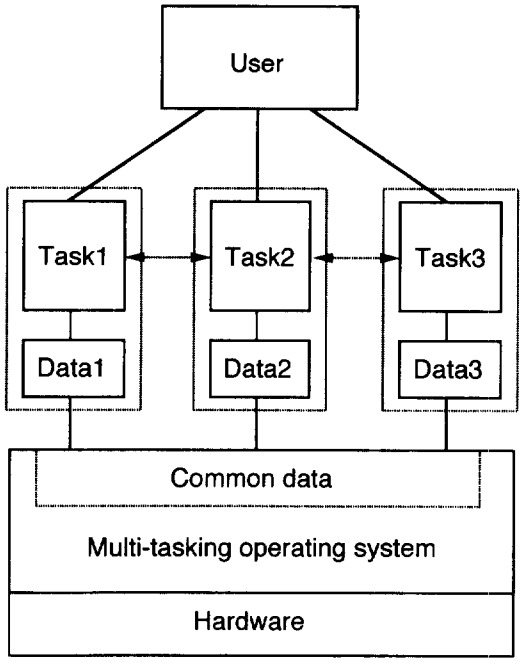

그림 2. Multi-tasking operating system

서 어느 한순간에 어느 task가 수행될지 정확한 예측은 불가능하고 단지 우선순위가 높은것 일수록 수행될 확율이 높다는 예측이 가능할 뿐이다(non-deterministic). 반면, 일반적으로 real-time multi-tasking operating system에서 요구되는 scheduling 방법은 pre-emptive scheduling이라는 것으로서 어느 순간에든지 ready 상태의 task중 항상 가장 높은 우선 순위를 갖는 task가 수행되도록 하는 방법이다. 따라서 어느 순간에 어떤 task가 수행될 것인가를 정확히 예측할 수 있다 (deterministic). 그외에도 real-time operating system 에서는 하나의 task에서 다른 task로 넘어가는 context switching time이 짧은 등 빠른 응답속도를 갖기위한 여러가지의 특성들이 있다.

이러한 real-time system software의 개발을 위해서는 범용컴퓨터와는 다소 다른 개발환경을 필요로 하는데 다음 절에서는 이에 관하여 알아본다.

4. Embedded System Software의 개발환경

Microprocessor가 등장한 이래로 소규모의 embedded system에 사용되었던 과거에는 앞절에서 기술한 사항들을 염두에 두지않고도 embedded system의 설계 및 제작에 무리가 없었다. 그러나 점차로 microprocessor가 장착된 기기들이 복잡 다양해지고, 또한 microprocessor의 성능이 기하급수적으로 향상됨에 따라 이에 걸맞는 software의 제작이 점점 어려워지고 비용이 점점 증가하게 되었다. 그뿐 아니라 일단 제작된 software의 유지보수 또한 상당히 어려운 문제로 등장하게 되었다. 따라서 이러한 embedded system software의 제작 및 유지보수의 용이성(容易性)을 고려한 개발환경이 최근에 보급이 되기 시작하고 있는데 본절에서는 이에 관하여 알아보고자 한다.

Embedded system software의 개발을 위해서는 program source의 editing, 저장, compiling에 필요한 development system과 개발된 software의 실제 수행을 위한 target system이 필요하다. Development system과 target system의 hardware architec-

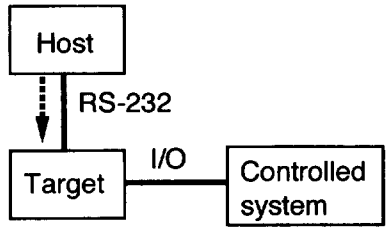

그림 3. Conventional development system configuration

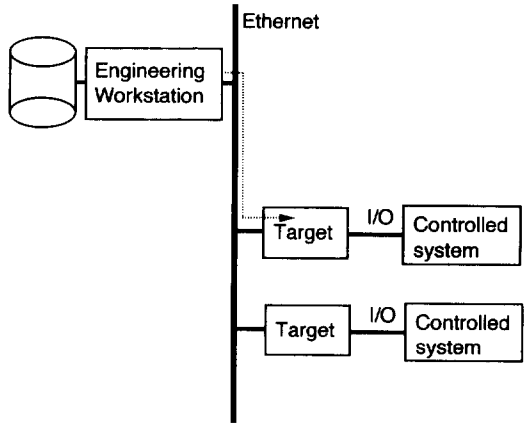

그림 4. New development system configuration

ture는 동일할 수 있고 다른 경우도 있는데, 서로 다른 경우에는 cross-development system이라고 부른다. Development system과 target system이 동일한 대표적인 경우로는 최근에 많이 적용되기 시작한 경우로서 IBM PC와 혹은 그의 호환기종을 embedded computer로서 사용하는 경우가 있겠다. 이 경우의 software는 real-time operating system을 사용하는 경우도 있지만 대부분의 경우에는 non-real time operating system인 MS-DOS상에서 interrupt를 이용한 foreground/background approach가 사용된다. 이러한 시스템은 MS-DOS용의 풍부한 software환경을 그대로 사용할 수 있는 등의 많은 장점이 있어서 널리 사용되고 있다.

Development systme과 target system이 서로 다른 경우로서 과거에 사용되어 왔던 구성은 그림 3에 보는것과 같이 VAX-11, PDP-11, IBM PC등과 같은 범용 host computer의 serial port에 target system을 연결하여 host computer상에서 개발된 software를 down load하는 형식을 가졌었다. 그러나 이러한 시스템은 전송속도가 느리고 debugging 환경이 취약한 등 규모가 큰 embedded software의 개발에 많은 한계가 있다. 그리하여 새로이 등장한 시스템으로 그림 4와 같은 구성의 개발시스템이 최근에 보급되기 시작하였다. 이 시스템의 development system은 보통 UNIX를 OS로 가진 engineering workstation이며 target system과는 ethernet과 같은 LAN을 통하거나 system bus를 통하여 접속이 된다. 따라서 software의 개발을 UNIX와 같이 개발환경이 우수한 환경에서 할 수 있게되며 target system으로의 down load또한 고속으로 가능하게 되고, debugging 환경등이 강화되었다. 최근의 변화는 단순히 시스템 구성의 변화에만 그치지 않는다. 더욱 중요한 변화는 target system에서 운영되는 real-time multi-tasking system software에 있다. Embedded system software는 적용되는 시스템마다 변경이 되어야 하며 사용자가 직접 software의 개발을 해야하는 경우도 있게됨을 고려하여 software의 구축을 용이하게 할 수 있는 software component의 개념이 도입되었다.[2, 3, 4, 5] 이것은 마치 hardware의 설계에 있어서 discrete component를 쓰는것보다는 MSI나 LSI component를 쓰는 것이 설계, 제작, 유지 보수 등 모든면에서 유리하다는 것과 흡사하다. MSI나 LSI와 같은 부품을 사용하여 hardware를 설계할 경우 설계자는 그 부품의 내부 구조 및 동작 원리를 상세히 몰라도 기능 및 사용법만 알면 이러한 부품들을 이용하여 자기가 원하는 시스템의 설계가 가능하도록 되어 있다. 이러한 원리가 software component의 개념에도 적용되는 것이다. Software설계자는 software component의 내부를 구체적으로 모르고도 이러한 software component를 사용하여 원하는 software의 제작이 가능하다. 또한 이러한 software component를 사용하여 제작한 software는 hardware를 변경할 경우도 손쉽게 적용이 가능하도록 되어있다. 이러한 종류의 개발환경을 사용할 경우 개발환경에 대한 초기투자는 다소 많을지 모르나 그것을 이용하여 제작되는 software자체에 대한 개발 노력 및 유지 보수는 많은 절약이 가능한 것으로 본다.

5. 결 언

Microprocessor의 성능은 향상되는 반면 관련 hardware의 가격은 계속 하락하고 있다. 따라서 자동화 기기와 같은 특수기기내에서 microprocessor관련 hareware의 비용이 전체기기의 가격에서 차지하는 비중은 점차로 줄어들고 있고, 이와 같은 기기류에 32bit microprocessor와 같은 고성능의 hardware를 장착하는 경우가 늘고 있다. 그러나 이에 걸맞는 software의 제작은 결코 쉬운 일이 아니다. 이와같은 고성능의 기기에 필요한 적절한 software의 제작 및 유지보수를 위해서는 반드시 적절한 개발환경이 필수적이라고 할 수 있다.

참 고 문 헌

[1] Stuart Bennet, *Real-Time Computer Control : An Introduction*, Prentice Hall, 1988

[2] J. Fiddler & J. Fogelin, "*The VxWorks Real-Time Kernel*," Wind River Systems, Inc.

[3] 江口 勉, リアルタイムOS *VxWorks*, インダ--フ

エース, Dec., 1989.
[4] Wind River Systems, Inc., *VxWorks Programmer's Guide*, 1991.
[5] J.F. Ready, "*VRTX : A Real-Time Operating System for Embedded Microprocessor Applications*" IEEE *Micro*, August 1986.

임동진(林東進)
1956년 7월 15일생. 1979년 서울대 공대 전기공학과 졸업. 1981년 동 대학원 전기공학과 졸업(석사). 1988년 미국 Iowa대 전기 및 컴퓨터공학과 졸업(공박). 1988-1991년 산업과학기술연구소 주임연구원. 현재 한양대 공대 제어계측공학과 조교수.