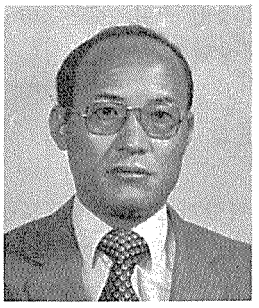
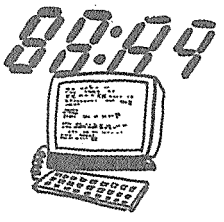


吳 吉 祿
韓國電子通信研究所
컴퓨터연구부장/工博

분산처리 운영체제의 소개



Ⅰ. 서 론

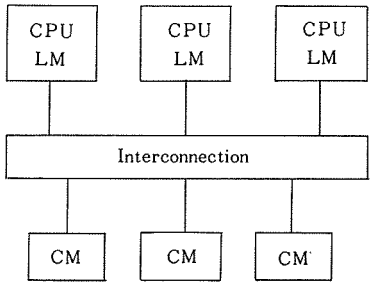
70년대 중반이후로 초집적회로 기술이 성숙하여 가고 CPU, 기억장치 등의 가격이 급격히 하락함으로써 여러가지 측면에 그 파급효과를 나타내고 있다. 그 중의 하나는 컴퓨터 시스템을 구성하는 방법에 대한 것이다. 16비트 내지는 32비트 짜리의 컴퓨터들이 소형 칩으로 생산됨에 따라 사람들은 이들 칩들을 다수 연결하여 보다 큰 처리능력을 가지는 시스템을 구축하도록 노력하게 되었다. 연산처리 능력을 가진 칩들을 여러개 동시에 연계하여 운용함으로써 전체 단위시간당 처리능력은 당연히 향상될 수가 있는 것이다. 그러면 작업의 처리능력을 병행성을 이용하여 향상시키는 데에는 어떠한 구체적인 방법들이 가능한가? 한가지 분류방법은 병행처리를 하되 어느 레벨에서 병행적으로 수행을 해나갈 것인가 하는 측면에서 분류할 수가 있다. 맨 밑으로는 논리적 계층에서 병행 수행이 가능하고, 그 위로는 레지스터 레벨에서 병행 수행을 함으로써 전체적인 성능향상을 꾀할 수가 있다. 그보다 위의 계층에서는 기계어 레벨에서, 또는 한개의 소프트웨어 프로세스 레벨에서도 병렬수행이 가능하다. 하부구조에서 병행성을 이용하고자 하는 시스템들의 예로서는 Array processor, Dataflow machine 등이 있다. 최상부 계층에서 병행성을 이루는 시스템의 예로서는 네트워크 시스템, 분산처리 시스템 등이 있고, 멀티 프로세서 시스템은 그 중간정도의 계층에서 병행성을 모색하는 것이라고 보는 것이 적절하다.[1]

네트워크 시스템, 멀티 프로세서 시스템 그리고 분산처리 시스템은 비교적 비슷한 계층에서 병행성을 모색하는 시스템이므로 서로 연관

성도 많고, 자주 혼동되기도 쉬우므로, 우선 이들 세가지 시스템들에 관하여 간략한 소개를 한다.

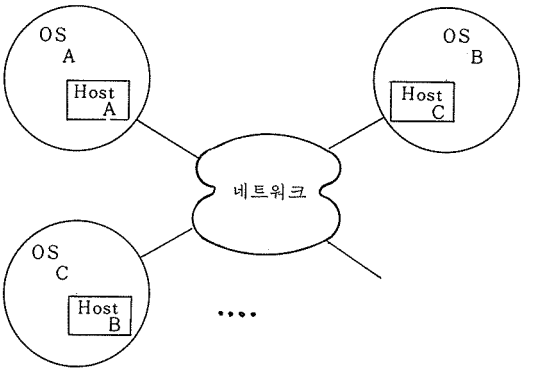
첫번째로 멀티프로세서 시스템을 분산처리나 네트워크 시스템과 구분되게 하는 가장 뚜렷한 것은 멀티프로세서 시스템에서는 여러 CPU들이 clock과 기억장치를 공유하는데 반하여, 분산처리나 네트워크 시스템에서는 CPU들이 기억장치를 공유하지 않고 단지 네트워크로 상호 연결되어 있다는 점이다. 이 공유되는 기억장치는 Common memory 또는 Shared memory라고 부르고, 멀티프로세서 시스템에서의 각 CPU는 각기 그 자신만이 액세스할 수 있는 고유의 기억장치(Local memory 또는 Private cache)를 갖는 외에도 여러 CPU 상호간에 공동적으로 필요한 정보를 공동기억장치를 통하여 저장시키고 교환하게 된다. 그림 1은 전형적인 멀티프로세서 시스템을 나타내고, 그림에서 공동기억장치와 CPU들을 묶는 것은 버스나 크로스-바 스위치를 사용하든가, 또는 직접 공동기억장치에 여러개의 포트(port)를 만들어 이를 통하여 한 기억장치를 여러 CPU에 연결시키어 주는 방법 등이 있다. 어쨌든 멀티프로세서 시스템에서는 여러 CPU들이 기억장치와 clock을 공유한다는 사실이 운영체제 등의 설계에 대단히 큰 의미를 갖는다.

분산처리 시스템과 네트워크 시스템은 공통적으로 여러 프로세서들이 네트워크 등으로 연결되어 있다는 점이 유사하다. 그러나 두 시스템은 그 운영되는 방법에 큰 차이가 있다. 네트워크 시스템에서는 네트워크에 연결된 호스트(host)들이 각자 네트워크와 관련없는 독자적인 운영체제를 가지고 있고, 이따금 네트워크를 통하여 다른 지역의 프로세서와 교신을 하기 위한 부가적 기능만을 갖춘 시스템이다(그림 2). 따라서 네트워크 운영체제(NOS : Network Operating System)라는 것은 이와 같은 매우 독립적인 시스템의 집합을 지칭하는 것이고, NOS는 그 자체가 전체 시스템을 제어하는 것이 아니고, 각 호스트내에서의 제어는 그 호스트의 독자적인 운영체제가 맡아서 하는 것이다. 각 호스트의 운영체제는 네트워크와는 무관한 별개의 운영체제라는 사실이 중요하다.



(주) CPU : Central Processing Unit LM : Local Memory
CM : Common Memory

그림 1. 멀티프로세서 시스템



(주) OS : Operating System

그림 2. 네트워크 운영체제

그러면 분산처리 시스템이란 무엇인가? 여기서 한가지 지적할 사실은 "분산처리"라는 용어는 과거(아직까지도) 매우 혼동되고 무질서하게 사용되어 왔다는 사실이다. 광의적으로는 CPU, 기억장치 및 주변기기들이 지역적으로 널리 분산이 되어 있으면, 그것을 분산처리 시스템이라고 불러왔고 또 일부에서는 아직도 그렇게 사용하고 있다. (특히 컴퓨터나 정보처리 산업계에서는 분산처리를 이와 같은 광의의 의미에서 사용하고 있다.[6]) 그러나 최근 학계에서는 분산처리 시스템이라는 용어를 매우 정확히 그리고 좁은 의미로 사용하고 있다.[2,3] 즉 분산처리 시스템이란 위에서 설명한 네트워크 시스템과 마찬가지로, 여러 장비들이 지역적으

로 분산되어 있고 또 그것들이 상호 네트워크로 연결되어 있지만, 그 전체 시스템은 단일 운영체제(분산 운영체제 또는 Distributed Operating System)로 제어되고 있는 것이다. 네트워크 시스템이 각 호스트의 독자적인 운영체제를 인정하고 있는 것과는 대조적으로 협의의 분산처리 시스템에서는 전체시스템이 논리적으로는 단 한개의 운영체제 밑에서 제어되고 운용된다(그림 3). 따라서 이러한 시스템에서는 논리적으로는 한개의 운영체제가 존재하더라도 그 데이터 및 제어는 실제적으로 여러 지역으로 분산이 되어 운영되기 마련이다. 본고에서는 여기서 소개한 협의의 의미에서의 분산처리 시스템에 대해서만 논하기로 한다.

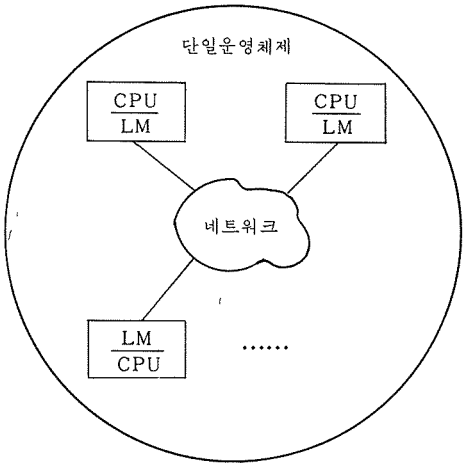


그림 3. 분산처리 시스템

최근 분산처리 시스템에 대한 연구는 매우 활발히 진행되고 있다. 그러면 왜 분산처리 시스템에 대한 연구가 필요한가? 우선 응용에 따라서는 필연적으로 전체시스템이 한 장소에 집중될 수가 없고 지역적으로 널리 분산이 되지 않으면 안되는 시스템들이 있다. 그리고 이들 분산된 시스템을 한개의 단일 운영체제로 제어할 필요가 있는 시스템들이 많다. 예를 들어 데이터가 발생되고 처리되는 장소가 필연적으로 원격 위치에 있다던가, 여러 위치에 분산되어 운용되는 공장설비 같은 것이 유기적으로 연결된 한 시스템에 의해 제어되어야 한다든가, 그 예는 얼마든지 있다.

분산처리 시스템에 대한 연구는 네트워크 시스템이나, 멀티프로세서 시스템에 대한 연구보다 비교적 덜 성숙한 분야이다. 멀티프로세서 시스템에 대해서는 과거 주로 master/slave 방식에 대해 집중적인 연구가 행하여져 왔고, 최근에는 symmetric한 시스템에 대한 연구가 중요한 문제의 하나가 되고 있다. 이에 비해 분산처리 시스템에 대한 연구는 비교적 새로운 것이다. 그러면 왜 분산처리 시스템에 대한 연구가 중요한가? 물론 앞에서도 언급한 바와 같이, 사회의 요구가 그 첫째이다. 그러나 기술적으로도 분산처리 시스템에 대한 연구는 그 중요성을 갖는다. 병행성을 통한 성능의 향상은 오늘, 내일의 노력이 아니라 벌써 꽤 오래 전부터 관심을 받아온 분야이다. 그러나 컴퓨터 내부에서 병렬성을 실현시키기 위해서는 항상 그 시스템마다 고유하고 적절한 기술을 습득하여야 하는 것이다. 최초에는 멀티-프로그래밍에 의한 병렬성에 대한 노력이 기울여졌다. 이러한 시스템에서는 여러 프로세스들이 "논리적"으로 병렬 수행을 하면서, 동일 기억장치 내에 있는 이 프로세스들이 공동으로 협력하여 작업을 수행하게 되었다. 그 후에 출현한 것이 멀티프로세서이다. 여기에서는 프로세스들이 각각 다른 CPU에 수행되면서 공동작업을 위해 협동을 할 필요가 있을 때에는, 공동 기억장치를 이용할 수 있었다. 멀티프로그래밍 시스템에서는 한 기억장치 안에서 여러 프로세스들이 협동할 수 있었고, 멀티프로세서 시스템에서는 여러 프로세스들간의 협동을 위하여 공동기억장치를 사용할 수 있었던 것에 반하여, 분산처리 시스템에서는 여러 프로세스들이 협동을 하여야 할 경우에는 네트워크를 사용할 수밖에 없게 된다. 결론적으로 멀티-프로그래밍, 멀티 프로세싱, 분산처리 시스템들이 모두 공통적으로는 다수 프로세스의 병렬수행을 지원하지만, 그 프로세스들이 협동을 해야할 때 사용되는 방법은 각기 완연히 다르다. 특히 분산처리 시스템에서는 프로세스들간의 협동이 네트워크를 통한 메

시지에 의존할 수밖에 없어진다. 이와 같이 상호 비교적 독립적인 프로세스간의 협동을 가능케 해주는 방법을 "glue"라고 부른다면, 이들 시스템들은 프로세스들의 병렬수행을 허용한다는 사실에는 동일하지만, 그 glue만큼은 완전히 다른 것을 사용하고 있는 셈이다. 멀티-프로그래밍과 멀티프로세서 시스템을 위한 glue 기술이 비교적 널리 알려지고, 많이 개발된 것에 비하면(예를 들면 linking, relocation, circular buffer, monitor 등) 분산처리 시스템을 위한 glue기법은 매우 생소한 것이다. 물론 이러한 glue만이 다른 것이 아니고, 그와 연관하여 시스템의 제어, 관리, 스케쥴링 등 여러가지 측면이 많이 다르게 된다.

결론적으로, 병렬성은 향후 갈수록 보편화되어질 추세이고, 병렬수행을 하게 되는 프로세스들이 때로는 한 기억장치 안에, 또는 공동기억장치를 가운데 두고 서로 다른 프로세서에, 또는 네트워크를 가운데 두고 지역적으로 원격하게 떨어진 프로세서들에 위치하여 수행되어질 필요가 있음은 자명한 사실이 된다. 전자 두가지 방법에 대한 연구 및 기술이 보다 일찌기 주목받기 시작하고 사용되기 시작한 것에 비하여, 분산처리 시스템에 대한 기법은 비교적 덜 알려진 것이다. 이러한 의미에서 분산처리 시스템에 대한 연구는 그 중요성이 있다할 것이다.

따라서 한 특정한 시스템을 두고, 그것이 네트워크 시스템인가, 또는 분산처리 시스템인가를 논하는 것은 별 의미가 없는 질문이 될 것이다. 80년대 후반의 시스템들은 거의 모두 이들 여러 시스템의 요소들을 함께 갖추게 될 것이기 때문이다. 이미 상용화되어 나오는 시스템에도 이와 같이 세가지 시스템의 성질을 모두 가지고 있는 그림 4와 같은 시스템은 얼마든지 그 예를 찾아 볼 수 있다. 한 시스템내에서 그때 그때 필요에 따라 여러 가지 형태의 병렬성을 최대한 활용하는 시스템이 그림 4와 같은 시스템의 목적이라 하겠다.

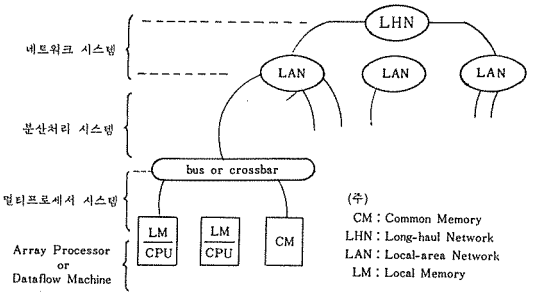


그림 4. 복합형 시스템

Ⅱ. 분산처리 시스템의 장점

분산처리 시스템은 아래와 같은 장점을 가질 수가 있다.

1. 높은 성능

한개의 칩보다 여러개의 칩이 공동으로 병렬수행하면서 과제를 풀어나가면 성능 향상을 꾀할 수가 있다.

2. 고 신뢰도

CPU나 다른 자원들이 다수 상호 연결되어 있으므로 그 중 한두개가 고장이 나더라도 전체 시스템은 가동을 계속하므로 availability가 향상된다.

3. 확장성이 용이하다.

분산처리 시스템에서는 제어나 관리기능 등이 완전히 분산되어 있어서 새로운 모듈들을 단계적으로 더하기가 쉬워진다(incremental growth).

4. 자원공유의 용이성

분산처리 시스템에서는 모든 자원들이 한 운영체제의 제어 및 관리하에 운용되므로, 그 시스템내의 어떠한 자원이라도 쉽게 공유할 수가 있게 된다. 이에 비하여 네트워크 시스템에서는 원격지역의 자원 공유가 비교적 제한이 되어 있고, 사용하기도 까다로우며, 또 비효율적일 수가 있다. 이는 각 호스트들의 운영체제가 상이하다는 데에 기인한다.

5. 기 타

이외에도 고장시 점진적으로 성능이 저하된

다든가(graceful degradation), 부하를 고르게 분배시킴으로써 시스템 전체적으로 효율성을 고르게 할 수 있다든가(load balancing)하는 장점들이 있다.

그러나 실제 시스템은 사실상 이와 같은 장점을 모두 함께 갖기가 어려울 수가 있다. 왜냐하면 위의 장점들은(달리 표현하면 위의 장점들이라고 소개한 것은 그 시스템을 설계하는 과정에서는 goal이 되는 것이다.) 그들 자체가 상호 상충이 되는 것들이 많기 때문이다.

실제 사용중인, 또는 개발중인 시스템들의 예로는 Accent,[6] Locus,[7] S/F UNIX,[8] Chorus,[9] HXDP,[10] DCS[11] 등 다수가 있다. 이들 시스템이 모두 위에서 규정한 것과 같이 완전히 제어기능과 데이터가 분산된 것은 아직 없고, 또 이들 시스템들 중에는 네트워크 시스템인지 또는 분산처리 시스템인지 그 경계를 뚜렷이 긋기가 곤란한 시스템들도 있다. 한가지 공통적인 것은 이들 시스템들이 각기 고유한 목표를 설정하고, 그 목표를 달성시키기 위한 기술개발에 주력하여 개발되었다는(또는 개발되고 있는 중이라는) 사실이다. 예를 들어 DCS시스템은 고신뢰도, Modular growth, Dynamic restructuring of use of resources 등을 주 기능으로 하고 있다. HXDP는 주로 실시간 제어를 목표로 개발된 분산처리 시스템이고, Chorus는 분산처리 시스템을 모델링하고 설계 및 구현기법을 혁신적으로 개선키 위한 시도로서 만들어졌다.

이와 같이 비교적 제한된 목표를 위하여 시스템을 설계하는 경향은 어제, 오늘의 일이 아니고, 특히 VLSI 칩들이 나오기 시작한 70년대 중반 이후로는 세계적인 추세라 할만하다.[5] 분산처리 시스템도 모든 종류의 일과 목적을 다 이루는 범용시스템의 길을 택하기 보다는, 보다 제한되고 구체적인 목표를, 그러나 보다 확실히 달성시키는 전문시스템으로 가는 추세인 듯하다.

Ⅲ. 분산처리 시스템의 과제들

분산처리 시스템을 구현하려면 여러 가지 문제들이 있다. 하드웨어 문제로서는 가장 뚜렷한 과제가 상호연결기법 (Interconnection structure)이 될 것이다. 소프트웨어 문제로서는 프로그래밍 언어, 운영체제의 설계, 분산데이터베이스, 화일 시스템 등의 다수가 있다. 여기에서는 주로 소프트웨어 문제들, 그 중에서도 운영체제상의 문제를 주로 다루기로 한다. 분산처리 시스템의 소프트웨어 문제들을 비교적 포괄적으로 다룬 논문들은 〔1〕과 〔2〕를 참조하기 바란다.

우선 분산처리 시스템의 설계 전반적으로 공통되는 어려움이 있다면 그것은 당연히 "분산" 그 자체에 기인한다. 분산처리 시스템에서는 가급적 시스템 상태 데이터와 제어권들을 한곳에 집중하여 저장하지 말고 여러장소로 분산하여 처리하고 저장하여야 한다. 그래야만 각 지역의 자주권이 향상되고, 따라서 신뢰도가 향상되고, Incremental growth가 가능하게 되는 등 분산처리 시스템의 장점이자 목표들을 달성할 수가 있게 되는 것이다. 분산처리 시스템에서는 멀티프로세서 시스템처럼 전 시스템의 상태가 한 곳에 모아져 있지 않다. 이에 더하여 분산처리 시스템의 개발을 어렵게 만드는 요인은 정보가 한 장소로부터 다른 곳으로 이동하기 위해서는 네트워크를 거쳐야 하므로 "지연"의 문제가 생기게 된다는 사실이다. 즉 어느 한 지역에서 의사를 결정하기 위하여 참조해야 할 데이터가 지연문제 때문에 원격히 떨어진 다른 장소의 상태를 늘 정확히 반영하지 못하는 것이다. 소위 현재 사용 가능한 Constructed system state table과, 임의의 시각에서의 시스템 상태를 정확히 반영시키는 Real system state table(개념적으로만 존재함)과의 사이에 늘 거리가 존재하게 된다. 이러한 상황에서 늘 정확한 의사결정을 하기란 매우 어려워지는 것이 사실이다. 또 분산처리 시스템에서의 많은 문제는 그 복잡도가 (complexity) 매우 높다. 프로세스 및 자원의 수가 많고, dynamic한 것도 사실이고, 최적화를 시켜야 할 자원의 종류도 CPU, memory, disk, channel, network 등으로 늘

어나기 때문이다.

아래에서는 분산처리 소프트웨어, 특히 운용체제의 문제들에 대해 간략히 소개를 한다.

1. 프로그램 언어

분산처리 시스템의 효율적인 이용과 개발을 위해서는 전문적인 프로그램 언어가 필요하다. 이 언어는 분산처리 시스템에서의 여러 프로세스들의 병렬수행을 원활히 지원하여야 하고 그 프로세스들간의 협동을 지원하여야 한다. 최근 개발되고 있는 ADA, CSP, SR, MOD 등은 이러한 부류의 언어의 예이다.

2. 운영체제

분산처리 운영체제의 설계시 대두되는 주요 문제들은 표 1과 같이 열거를 할 수가 있다.

표 1. 분산처리 운영체제의 기술적 문제점들

a. partitioning : 제어기능과 데이터들을 어떻게 분산시키는가?
b. scheduling : 어느 프로세스를 어느 곳에 시작시키며 이것을 누가 어떻게 결정하는가?
c. deadlock : 여러지역이 연관된 deadlock 문제의처리
d. synchronization : 상호 협동이 필요한 프로세스들간의 protocol 등 상호 교신 방법
e. fault tolerancy : error의 처리, atomicity, 네크워크 신뢰도 등 문제
f. file system : 여러지역이 공유하는 화일들의 관리 방법
g. Design & Development Methology
h. 기 타

제한된 지면에서 이들 모두를 상세히 기술하는 것은 어렵고, 이에 대한 보다 자세한 기술은 Stankovic의 논문을 참조하거나, Kleinrock의 논문을 참조하기 바란다. 아래에서는 분산처리 운영체제에 공통적인 특징을 열거한다.

첫째, 분산처리 시스템은 재래의 시스템보다 더욱 복잡한 내부구조를 가지게 된다. 우선, 병렬성이 여러가지 문제를 야기시키고, 또 네트워크라는 기능이 추가됨에 따라, 시스템의 복잡도가 몇 layer 더 추가되기 마련이다. 네트워크 운영체제에서는 네트워크 layer가 최상부층에 있게 되므로 시스템 자체의 복잡도가 더해지는 현상이 덜하지만, 분산처리 시스템에서는 네트워크 layer가 시스템 내부로 흡수가 되므로 더욱 복잡성을 더하게 된다. 덕택에 분산처리 시스템의 사용자는 network transparency, location transparency 등의 편의를 제공받게 된다.

둘째, balance와 tradeoff를 하여야 할 요소들이 더욱 늘어난다. 앞서도 언급한 바와 같이 종래 시스템에서는 CPU, 기억장치, 채널 등이 tradeoff를 해야하는 요소들이었으나, 분산처리 시스템에서는 네트워크를 통한 교신량이 쉽게 병목현상이 될 수가 있고, 이것이 최적화를 시키어야만 하는 중요한 대상으로 대두가 된다.

세째로 의사결정을 위한 계산량이 늘어난다. 분산처리 시스템에서는 프로세스의 수, 자원의 수가 매우 많고 또 dynamic하며, 위에서 언급한 것처럼 의사결정시 최적화의 고려대상이 되는 자원들의 종류도 늘어난다. 분산처리 시스템에서의 스케쥴링 문제, 교착상태(deadlock) 처리문제, 제어기능과 데이터의 분산할당 문제 등은 거의 모두가 다 NP-hard problem들이 되고, 따라서 시스템을 운용하는 중에는 제약조건을 완화시키든가 아니면 heuristic approach를 택하지 않을 수가 없게 된다.

네째로 지연문제가 있다. 앞에서도 언급한 것과 같이 분산처리 시스템에서는 타지역의 상태와 요구를 나타내는 정보들이 네트워크를 통하여 전달되어 지연문제가 발생하므로, 항상 global state에 대한 정보는 정확한 것이 아님을 염두에 두어야 한다. 지역간 clock이 공유되는 것이 아니므로 레지스터 레벨의 동기화도 어렵고, 일단 고장 등 유사시에는 atomicity의 보장이 매우 절실하게 필요하여진다. 또 여러 메시지가 한 지역에 전송되다 보면 메시지가 발생한 순서와 도착한 순서가 틀려져서, 이들 메시지에 기초한 판단이 오류를 범하기도 쉽다. 이와 같은 어려움을 극복할 수 있는 기술의 개발이 필요하다.

Ⅳ. 결 론

본고는 전자공학회 산하 전자계산연구회의 단기 강좌 활동을 위하여 분산처리 시스템중 운영

〈P. 49로 계속〉

6. Support System

S. M. T에 의해 생산할 경우 mounter의 導入은 물론 필요하나 周辺의 Support System 導入도 중요한 要素가 된다.

1) CAD System

Computer로 Print基板을 設計하는 System으로 精度의 確保와 生産資料(Chip part 位置情報, check 治具情報, Screen 印刷情報 등)을 出力한다. Sharp에서는 더욱 設計技術者가 직접 Computer와 対話를 하면서 print基板을設計하는 direct設計 System을 확립하여 더욱 高精度와 設計効率을 높이고 있다.

Headphone stereo의 S. M. T에 대해 설명했는데 앞으로 더욱더 각방면에서 print基板上의 parts는 S. M. T化될 경향이며 mounter나 납땜裝置 등도 더욱 새로운 手法이 전개될 것으로 생각된다.

P.45에서 계속

체제에 관하여 서울대 고 건 연구회 간사께서 정리하신 것이다. 포괄적인 소개가 안된 것은 지면상 제약도 있지만, 이 분야가 아직도 부단한 변화를 거듭하고 있다는 데에도 기인한다.

参 考 文 献

[1] L. Kleinrock, "Distributed Systems", IEEE Computer, vol. 18, no. 11, Nov., 1985.

[2] J. A. Stankovic & A. V. Dam, "Research Directions in Distributed Processing,"

[3] H. M. Deitel, "An Introduction to Operating Systems," Addison-Wesley.

[4] Enslow, P. H., Jr, "Multiprocessor Organization-A Survey," CACM, vol. 9, no. 1, March, 1977, pp. 103-129.

[5] H. Lorin, "Aspects of Distributed Computer System," Wiley Interscience.

[6] R. F. Rashid & G. G. Robertson, "Accent : A Communication Oriented Network Operating System Kernel," Proc. 8th Symp. on Operating System Principles Dec., 1981.

[7] G. Popek, "LOCUS, A Network Trans. Parent, High Reliability Distributed System," Proc. 8th Symp. on Operating System Principles Dec., 1986.

[8] Bell Lab. Memorandum.

[9] M. Guillemont, "The Chorus Distributed Operating System : Design and Implementation," Int'l Symp. on Local Computer Networks, Florence, Italy, April, 1982.

[10] E. D. Jensen "The Honeywell Experimental Distributed Processor — An Overview of its Objective, Philosophy and Architectural Facilities," IEEE Computer, vol. 11, no. 1, Jan., 1978.

[11] D. J. Farber, "The Distributed Computer System," Proc. 7th Annual IEEE Computer Society Int'l Conf., Feb., 1973. *