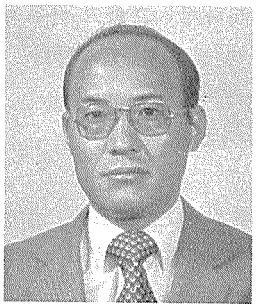

吳 吉 祿
韓国電子通信研究所
컴퓨터연구부장/工博

IPC(Interprocess Communication) Mechanism

I. 序 論

여러 computer 들이 network으로 연결되고, 이들 간의 정보교환이 활발하게 이루어질 수 있는 환경이 국내적으로 조성되고 있는 실정이다. 이러한 정보 교환을 위한 방법으로써 network operating system (NOS)을 구현하는 것과 distributed operating system (DOS)을 구현하는 두가지로 대변할 수 있다.

물론 사용하고자 하는 응용 분야에 이들 각각의 장단점이 존재하게 된다. 먼저 NOS인 경우에는 현재 있는 system을 바탕으로 하여 구현하는 것이므로 단기간 쉽게 구현이 가능하다. 반면에 DOS는 모든 connect된 system이 동일한 O.S.를 갖게 되므로 이를 위하여 많은 시간과 노력이 필요하게 된다. 그러나 실제 응용면에서 볼 때는 NOS인 경우 O. S.와 network과의 불일치로 인한 overhead가 많고 network transparency가 DOS보다 약하다는 점들을 고려하여 볼 때 앞으로 DOS가 구현되어서 user에게 service하게 될 것이 틀림없는 것이다. [4]

이러한 DOS를 구현하기 위하여 필수적으로 필요한 사항이 process간의 통신이 되는 것이다. 즉 interprocess communication (IPC)는 distributed 시스템의 구조 및 performance에 커다란 영향을 미치게 된다. 즉 우수한 IPC의 mechanism은 전체 시스템의 효율을 증가시킬 것이며, 그렇지 못한 경우 오히려 단일 시스템보다 IPC에 의한 overhead로 인하여 성능이 떨어지게 되는 경우도 존재한다. 자원이 분산되는 경우, 특히 process가 분산되는 경우에 있어서 이들 상호간의 정보 교환은 여러가지 문제를 내포하고 있다. Process의 identification이라든가, process immigration으로 인한 run

註 : 본고는 과학기술처의 국가주도 특정연구과제로 전자통신연구소에서 수행하는 분산처리형 컴퓨터개발과제중 고려대학교에 위탁하였던 내용에서 발췌한 것임을 밝힌다.

enviromnent의 변화 등, 이 각각을 잘 고려하여서 prccess의 통신 구조 및 mechanism을 설계, 구현하여야 한다.

또한 분산시스템을 위하여 분산된 process간에 통신을 할 수 있는 protocol facility 또한 중요한 요소가 된다. Protocol이 전체 시스템의 성능에 상당히 중요한 요소가 된다. 여기서 protocol은 IPC mechanism을 구현하는 기초가 되는데 이 protcol 설계시 분산되는 환경을 잘 파악하고 설계하여야 할 것이다. 즉 분산되는 정도가 근거리에서 LAN을 통하여 이루어질 것이냐 아니면 상당히 원거리까지 이루어져야 할 것이냐에 따라 이 protocol의 설계는 달라지게 된다.

본 소개에서는 process간의 통신 방법에 대하여 살펴보고, 지면 제약상 기존의 개발된 몇몇 IPC를 소개하기로 한다. 다음으로는 본연구팀이 설계 구현중인 IPC를 위한 protocol의 설계 및 구현에 대하여 설명하기로 한다.

Ⅱ. 프로세스간의 통신[4]

본 장에서는 process가 분리된 환경에 있을 경우 이들 간의 통신에 관하여 살펴본다.

Interprocess communication은 세가지 정도의 측면에서 고려하여 볼 수 있다. 첫번째는 message passing을 통신 primitive이고 다음에는 이러한 message를 buffering하기 위한 방법이다. 그리고 나머지 다른 하나는 process간에 서로 주고 받는 message의 구조이다. 즉 message가 어떤 field를 가질 것이며, 그 field는 어떻게 표현될 것인가 하는 것이다. 이들 각각에 대하여 살펴보겠다.

1. 통신 primitive

통신 primitive에는 reply-send, synchronized send, no-wait send 형태의 send mechanism이 있으며 수신하는 편을 고려하여 볼 때는 unconditional receive와 conditional receive가 있다.

먼저 reply-send는 message를 받아서 그에 관한 execution을 한 후에 이에 대한 응답으로써 어떤 값 또는 message를 return 함으로써 종료되는 mechanism이다. 한편 synchronized send는 보내진 message를 remote process 할 때까지 전송한 process는 blocking 되어져 있는 것을 말한다. 여기서 remote process가 message를 수신하였다는 것은 다시 말해서 sending process로부터 reply(ack)를 수신하였음을 의미하게 되는 것이다.

이상에서와 같이 reply-send와 synchronized send인 경우에는 만약 하나의 process가 통신 path에 대한 전송 access를 가졌다는 message의 buffering이나 이를 위한 queuing이 필요없게 된다.

다음으로는 no-wait send에 대하여 생각하면 이는 asynchronous한 전송이 되는데 통신하는 process에게 최대의 concurrency를 제공하게 된다. 이는 다시 말해서 두 process는 최대의 autonomy를 갖는다는 것이다. No-wait send는 위에서 언급한 다른 send primitive들보다도 flexible하다. Sending process는 전송할 message를 순서대로 buffering한 후 communication facility가 허용하는 범위 내에서 전송하게 된다. 그러므로 no-wait인 경우에는 buffering에 필요한 flow control, congestion control 등의 문제를 고려하여야 한다.

지금까지는 send primitive에 관하여 언급하였다. 다음에는 receive primitive에 대하여 생각하는데, 여기에는 unconditional, conditional 두 가지의 receive primitive가 존재한다.

먼저 unconditional receive에 대하여 생각하면, 이 경우에는 port에 messsage 가 queue 될 때까지 blocking되어져 있게 된다. 그러므로 process는 상대편 process가 message를 전송하여 주기 전까지는 어떠한 작업도 하지 않게 되므로 process concurrency라는 측면에서는 상당한 손실을 초래하게 되는 것이다. 여기에서 두 가지 방법이 있다. 하나는 몇몇의 port에 대하여 blocking 되어져 있는 것이다. 이 경우에는 몇몇의 port중 하나만의 port에 message가 queue 되더라도 blockihg 상태에서 변화하게 되는데 이를 restricted unconditional receive

라 한다. 이는 여러 source로부터 message를 기다려서 serving하는 process인 경우에 유용하게 이용될 것이다. 다른 하나는 blind unconditional receive인데 요구한 모든 port에 message가 queue될 때까지를 기다리게 되는 것이다.

다음으로는 conditional receive이다. 이는 polling capacity를 가지게 된다. 그러므로 원하는 정도의 process concurrency를 얻을 수 있다. 여기에서 각 지정된 port를 polling하는데 message가 존재하면 이를 return하고 그렇지 않은 경우에는 message가 존재하지 않음을 알리고 control을 return하게 된다.

이상에서 언급한 send와 receive primitive를 응용하고자 하는 process communication에 적합하도록 조합하여서 사용하여야 할 것이다.

2. Buffering of message

이 문제에 있어서 buffer의 관리는 결국 resource allocation 관계의 문제가 된다. 여기에는 4가지 정도의 방법이 있다. Port-local, process-local, system global allocation, 그리고 혼합 형태인 port-global allocation이 있다.

각각을 살펴보면 다음과 같다.

• Port-local allocation : 이는 각 port당 일정량의 buffer 공간을 할당하는 방법이다. 이러한 방법을 사용하는 경우, 할당 받은 buffer가 가득찬 경우 보내고자 하는 Process는 만약 flow control option이 있으면 buffer space를 확장하여 message의 전송을 할 수 있도록 한다는 것이다. 그리고 다른 하나는 buffer space가 생길 때까지 block된 상태로 남아 있어야 하는 경우이다. 또 다른 하나는 port의 buffer 공간이 완전히 채워져 있음을 flag을 통하여 return하는 방법이다.

• Process-local alloction : 이 방법은 앞에서 언급한 port-local allocation과 비교하여 볼 때 모든 점이 동일하나 buffer가 port에 할당되어지는 것이 아니고 process에 할당된다는 점이다. 이 경우에는 process를 위한 outstanding message의 수를 제한하는 것이 보통이다.

• System-global allocation : 전체 buffer pool로 관장하는 것으로써 시스템에 bottleneck을 초래할 수 있다.

이와 같은 점들을 고려하여 볼 때 port-local과 system-global 방법을 혼합한 port-global hybrid 형태가 나올 수 있는데 이는 각 port 마다 일정한 buffer space를 할당하고, buffer space는 system으로부터 할당받는 것이다.

3. Message 구조

Message 구조에서 가장 중요한 문제는 message에 어떤 data type이 들어가게 되는 것이다. 어떤 system에서는 strongly-typed data 구조를 갖도록 되어 있는데 이 경우 주고 받는 message상에 reliability가 증가하겠지만 분산 시스템의 경우 heterogenous node들을 연결하게 되므로 message의 변환이 필요하게 되는 단점을 지니게 된다.

다음으로의 문제는 message의 길이가 문제된다. 일반적으로 tightly coupled 시스템인 경우에는 길이가 짧고, 길이가 일정한 message를 사용하여 control message를 주고 받는 방법을 사용하며 loosely coupled 시스템을 위한 message는 변화되는 길이의 message로 구성된다. 이 경우에는 buffering 문제 등에 있어서 구현하기 어려운 점이 존재한다.

이상에서는 process 상호간에 통신법 및 그에 필요한 사항들에 관하여 생각하여 보았다. 다음으로 실제 구현되어진 기존의 몇몇 IPC들의 mechanism에 관하여 살펴보도록 한다. 이를 통하여 IPC mechanism을 생각하도록 하며, 원하는 IPC mechanism을 어떻게 설계, 구현할 수 있을까를 알 수 있을 것으로 생각된다.

Ⅲ. 기존 IPC mechanism

본 장에서는 널리 알려져 있는 IPC mechanism 중에서 CHROUS, Accent, V Kernel 시스템에 관하여 살펴 보도록 한다.

1. CHROUS[2,5]

CHROUS 시스템에서 distributed application

은 processing step이라는 수행 단위로 이루어지게 된다. 이 action은 local object에 한정되며 이러한 processing의 환경을 actor라 부른다. 이는 일반적으로 생각하는 process와 비슷한 개념으로 사용되고 있다.

이 actor들 간의 통신은 message를 교환함으로써 이루어지게 된다. Send 하고자 하는 actor는 자신의 한 port에서 receive actor의 port(destination)로 message를 전송한다. 물론 이 port는 bidirectional 이므로 send와 receive를 한 port를 통하여 이루어질 수 있다. 이 port는 system에서 unique한 name에 의하여 identify 된다.

여기서 port는 actor와 결합하였을 경우에 의미를 갖게 되며 port를 통하여 다른 actor와 통신을 할 수 있다. 여기서 message는 processing의 기본단위가 되며 actor는 message를 수신하고, processing하며, 전송하는 역할을 한다. 한 순간에 하나의 actor는 하나의 message만을 handling 할 수 있도록 구성되어 있다. 그리고 process의 parallelism은 actor 하나가 이루는 것이 아니라, 이러한 actor 몇을 결합하여서 전체적으로 parallelism을 얻을 수 있다. Po rt는 kernel에 관련 table이 존재하여서 관리하도록 되어 있다.

2. Accent[1]

Accent system에 있어서 interprocess communication을 위한 개념은 port이다. Accent에서의 IPC는 message based IPC로서 port를 통해서 한 process에서 다른 process로 message를 전송함으로써 process간의 통신이 이루어진다. Process는 port에 대하여 access right을 갖게 되는데 여기에는 send, receive access와 ownership이 있다. Port라는 것은 일정 길이의 queue로서 send process에 의하여 채워지게 되며 여기에 관한 flow control은 Ⅱ장에서 언급한 바와 같이 flow control option이 send primitive 내에 포함되어 있으면 port는 더 많은 message를 받아야 하므로 queue size를 dynamic하게 변화시킨다. 그렇지 않은 경우에는 queue에 space(send 할 수 있을 정도)가 존재할 때까지 blocking 되어 기다리게 된다.

Message send call을 수행할 때는 send하고자 하는 process의 address space로부터 전송 port로 copy하고 receive call을 수행할 때는 port로부터 receiver process의 address space로 copy하게 된다. 실제 message는 header 부분과 structured data 두 부분으로 분리되는데 header에는 flow control, priority, sequence control, reliability, maximum age, security 등과 관련있는 정보가 기록되게 되며, typed information을 communication하기 위한 standard protocol을 공급하기 위하여 message structuring을 한다.

그리고 network을 통한 IPC의 경우에는 network IPC를 관할하는 intermediate process인 network server를 둠으로써 가능하게 된다. 여기서 예를 들어서 설명하면 그림에서 hostA 에 존재하는 a process와 host B에 존재하는 b process가 어떤 방법을 이용하여 통신하는가 알 수 있다. 그림에서 보는 바와 같이 다른 host에 있는 process와 통신하고자 할 때는 communication path는 양 host의 network server를 통하여 만들어지고 host A의 port가 host B로 migrate하게 된다. HostA의 process a는 connection, 사용 protocol, network topology 등에 관한 정보를 알 필요가 없다. 이렇게 되므로써 network transparency를 제공하게 된다.

3. V Kernel[7]

V Kernel에서는 IPC primitive를 system call

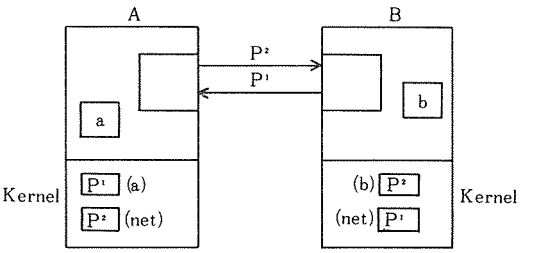

그림 1. Network communication

trap형태로 구현하였다. 각 process는 message를 통하여 통신하며 unique한 global identifier를 가지게 된다. 그러므로 message 수신자와 송신자는 pid에 의하여 지정되게 된다. Message는 32 byte로 고정하고 있으며 memory segment에 대한 access를 pass 할 수 있도록 되어 있다. 그림 2 와 같이 Ⅱ장에서 언급한 send-receive-reply 행동에 의하여 이루어진다.

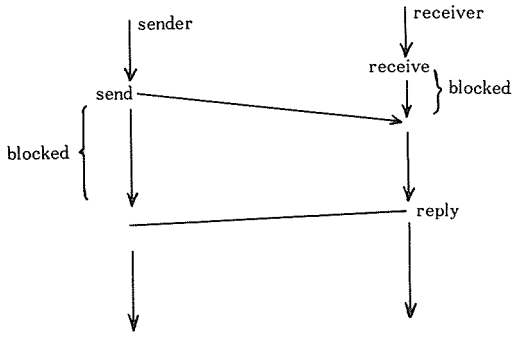

그림 2. **Message transaction**

Service를 받고자 하는 process (client)는 server process에 message를 send하고 suspend 한다. 이때 server process는 receive operation을 행함으로써 전송이 이루어진다. Client가 자신의 address space의 memory segment에 대한 access를 pass하고 server process는 copy to, copy from 등의 operation을 이용하여 실제 data를 copy하게 된다. Server process는 message에 관한 처리를 행한 후에 reply operation을 행한다. 여기서 reply 되는 message는 send에 대한 응답 신호로 기능을 하게 되며 send process와 recive process의 동기를 제공하여 준다. 또한 Ⅱ장에서 언급한 바와 같이 sender가 blocking 되어져 있으므로 buffering이 필요하지 않음을 알 수 있다. V kernel에는 process를 grouping할 수 있는 기능이 있으므로 이와 관련된 server group 등의 응용 분야에 이용되기에는 적합함을 알 수 있다.

여기서 V Kernel에서 사용되는 주된 IPC primitive를 살펴보면 다음과 같다.

• Send (message, pid)

32 byte message를 pid로 identify되는 process로 전송한다. 이 경우 sender process 는 앞에서 언급한 바와 같이 reply message를 받을 때까지 waiting 하며 reply message는 original message의 동일 영역에 싸지게 된다.

• (pid, count) = Receive with segment (message, segptr, segsize)

Message가 없으면 blocking상태로 process가 들어가게 되며 그렇지 않은 경우에는 32 byte의 message를 읽어 들인다. 이 경우 읽어 들인 message에 read access가 주어졌다면 segptr로부터 segsize 만큼의 실제 data를 읽어 들인다. 이때 실제 읽어 들인 data는 count를 통하여 byte수를 알 수 있다.

• Reply with segment (message, pid, destptr, segptr, segsize)

32 byte의 message를 pid의 process로 전송한다. 한편 segptr과 segsize로 표시된 segment를 목적지 destptr로 전송하게 된다. 이 message에는 한개의 bit가 있는데 이는 message transaction이 idempotent인지 아닌지를 나타낸다. 이에 따라 처리하는 방법들이 여러 가지로 변하게 된다.

• Copy from (scrpid, destptr, srcptr, size)

Srcpid에 의하여 표시되는 process의 영역에서 srcpt로부터 size 만큼의 data를 이 operation을행하는 process의 address space의 destptr로 copy하는 행동을 한다. 여기서 scrpid에 해당하는 process가 memory segment에 대하여 read access를 pass하지 않은 경우에는 이와 같은 copy from operatitin을 행할 수 없다.

• Copy to (destpid, destptr, srcptr, size)

이 operation을 부른 process의 address space의 srcptr로부터 size만큼의 data를 destpidprocess의 주소 영역의 destptr로 copy하라는 것이다. Destpid process 는 operation을 행하는 process로부터 reply를 기다리고 있어야 한다. 이경우 destpid process가 message

를 send할 때 write access를 주었어야만 이러한 operation이 가능하게 되는 것이다.

• Forward (message, pid 1, pid 2)

Message를 pid 1 process가 pid 2 process로 보내는 것과 같은 행동을 하는 것이다.

• Pid=Receive specifie (message, pid, segptr, segsize)

이것은 위에서 언급한 Receive with segment와 동일하나 receive하고자 하는 process를 specify 하므로써 원하는 특성 process 로부터 message를 받을 때까지 blocking 되어져 있게 된다.

이상에서 CHROUS, Accent, V Kernel system에서의 process communication에 관하여 살펴 보았다. 이는 Ⅱ장에서 언급하였던 사항들의 적용이 되는 것이다. 물론 이 외에도 UNIX[3]의 IPC mechanism인 socket 등과 같은 여러 IPC mechanism이 존재하지만 이에 관하여는 지면 관계상 본 소개에서는 언급하지 않으며, 다음으로는 본 연구팀에서 수행하고 있는 LAN environment의 IPC를 위한 protocol 설계, 구현 중에서 설계에 관하여 간략히 설명하고자 한다.

Ⅳ. IPC를 위한 Protocol 설계[8]

분산 시스템이 원거리에서 이루어지는 분산 시스템인가 아니면 근거리에 의하여 resource가 분산되어 있는가에 따라 실제 구현에 있어서는 커다란 차이를 보이게 된다. 그러나 대부분의 분산 시스템들이 근거리에 resource가 분산되어 있음에도 불구하고 원거리에서 사용하고 있는 protocol을 그대로 사용하고 있는 경우가 있다. 한편 데이타의 성격에 관계없이 동일한 protocol을 사용함으로써 protocol에 의한 overhead가 고속 전송의 주된 장해가 되어 왔던 것이다. 그리하여 본 연구팀에서는 이런한 단점을 극복하기 위하여 새로운 방법으로써, 전송하고자 하는 data의 성질에 따라, 전송 선로, 전송 상태의 변화에 따라 다른 protocol을 사용함으로써 전송 속도의 고속화를 꾀할 수 있도

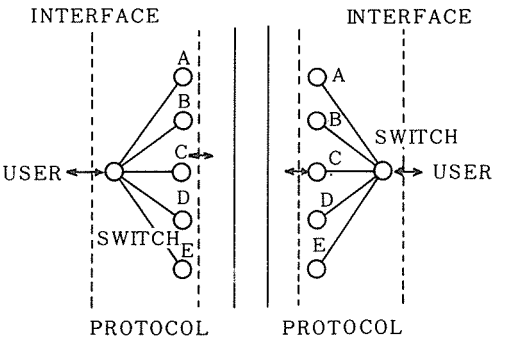

그림 3. System view

록 하였다.

먼저 system의 구조를 보면 그림 3과 같다.

여기서 A, B, C, D, E는 서로 다른 protocol entity를 가리킨다. Protocol entity 각각은 remote entity로 communication할 수 있도록 한다. Link level 이하는 multi-drop을 고려함으로써 routing은 고려하지 않도록 하였다. 여기서 사용된 protocol의 성격을 살펴보면 다음과 같다.

• User가 필요한 protocol을 선택하도록 하였다. User가 protocol을 선택함으로 인하여 자신의 bata 성격에 적합한 protocol을 사용하도록 하였다. 음성 또는 화상인 경우에는 reliability가 떨어지는 protocol을 사용하는 대신에 고속 전송의 효과를 얻을 수 있으므로 이러한 경우에는 이에 맞는 protocol을 선택한다.

• Protocol의 교환이 가능하다. 하나의 새로운 protocol의 설계, 구현할 경우 그 protocol을 하나의 module로 첨가함으로써 이루어질 수 있도록 하였다.

• Protocol 각각이 ℓ : n, n : ℓ, n : m의 통신이 가능하도록 하였다. 이렇게 함으로써 multicast 등이 가능하게 되므로 LAN environment에 적합하다. 즉 office automation 등에 이용 가능하게 될 것으로 보인다. 여기서 ℓ : n이 가장 간단하므로 ℓ : n, n : ℓ, n : m 순으로 설계, 구현하고 있다. 위에서 언급한 이러한 사항들이 본 protocol의 간략한 성격이 된다.

Protocol에 의한 overhead는 결국 flowcontrol과 error control을 위한 것인데 이러한 문

제를 본 protocol에서는 A, B, C, D, E라는 5개의 protocol로 분리함으로써 해결하고 있다. 즉 A부터 D까지는 virtual circuit을 support하며 E는 datagram service를 한다. Protocol A는 fow control과 error control을 모두 행하는 기능을 가지고 있게 되며, B는 flow control 기능만 존재하며, C같은 경우에는 flow control 기능 없이 error control 만을 행하게 된다. 한편 D는 flow, error control을 전혀 하지 않는다. 즉 이와 같은 경우에는 전송 시스템이 완전 reliable 한 경우의 protocol이 된다.

본 protocol의 설계, 및 구현의 자세한 내용은 본 소개에서는 언급하지 않으며 자세한 것은 추후 소개할 기회가 있을 것으로 생각된다. Ⅱ장과 Ⅲ장에서 언급한 IPC primitive 및 mechanism은 이와 같은 protocol에 의하여 support되게 되는데 이러한 새로운 protocol 선택 방법에 의한 IPC의 새로운 mechanism이 구축될 수 있을 것으로 생각하며, 좀더 나아가서 protocol의 새로운 selection mehanism도 설계 구현하고자 한다.

参 考 文 献

(1)Richard F. Rashid, "Accent : A Communication oriented network operating system Kernel" April, 1981.

(2) Eimmermann, etc., "CHROUS : A Communication and processing Architecture for distributed system", INRIA, September, 1984

(3) Samuel J. Leffler, etc., "A 4 2 Bsd Intprocess Comml nication primer" July, 1983.

(4) John A. Stankovic, etc. "CurrentRese arch and critributed Software Systems"

(5) J. S. Banino, etc., "Distributed Coupled Actors : A CHROUS Proposal for reliability", Oct., 1982.

(6) Peter J. Denning, etc., "Should Distributcd Systems de hidden ? ", 1983, IEEE

(7) David R. Cheriton, "The V Kernel : A Softwase Base for Distributed Systems", April, 1984, IEEE

(8) KOREA UNIV. Computer Netwo k Lab. Internal Note, Mote March, 1986. *

P.37에서 계속

沿岸諸國들이 歐洲와는 전혀 다른 규범으로부터 발생되는 「다이내믹한 에너지」를 無視, 또는 軽視해 오지나 않았나 하는 것을 생각케 하고 있다. 그러므로 歐洲共同体(EC)는 아직도 모자이크狀態이며 國際競爭力의 열쇠의 하나가 되는 規模經濟를 실현못하고 있는 실정이다.

6. 혼자서 살아가는 사람은 아무도 없다.

西歐가 어떤 戰略産業에 있어 美國이나 日本에 뒤떨어지고 있는 이유는 歐洲자신이 반성할 문제이다. 지금까지 25년간 歐洲를 觀察해 온 나에게 있어서는 이것은 매우 유감스러운 事態이다. 世界經濟의 安定成長은 創造的이며 경쟁력이 있는 歐洲없이는 기대할 수 없다. 이런 의미에서도 歐洲의 경쟁력을 높일 가능성이 있는 ESPRIT, RACE, BRITE와 같은 共同의 高度技術프로젝트는 높이 평가되어야 한다.

한마디로 말하면 우리들은 모두 상호관계 가운데서 살고 있다는 것이다. 혼자서 살고 있는 사람은 누구도 없다. 예를 들면 第3世界에의 有効한 援助도 歐洲, 日本, 美國間의 협력없이는 이룰 수가 없다.

마이크로 일렉트로닉스의 광범위한 침투에 따라 新産業革命이 일어나고 있는 오늘날, 歐洲와 日本의 電子産業의 책임은 막중하다. 따라서 우리들의 당면한 共同과제는 과거의 편견이나 対抗的인 자세를 버리기 위해서도 새로이 技術과 産業面에서 協力할 수 있는 활동을 추구해야 한다는 것에 있다.