運營体系 소프트웨어

姜錫烈

〈要　　約〉

컴퓨터의 많은 응용에 따라 분산처리,실시간처리, 고장감내처리 등에 대해 운영체계의 연구가 많이 되어왔다. 본고는 한국형 전전자교환기인 TDX-1 시스템에서 사용된 TDX OS의 실현과 그 특성에 관해 서술하였다. TDXOS는 고실시간처리(Hard real time processing)와 분산처리, 컴퓨터의 이중화, 과부하제어, 실시간 디버거(Debugger) 들을 실현하였으며 뱅크(Bank)시스템 형태의 메모리 관리기법이 사용되었다. 특히, 소형 마이크로프로세서(Z80) 및 어셈블리 언어에최적의 동작 환경을 제공하도록 설계되어 실시간의 효율이 최대화되었다.

Ⅰ. 서　　론

반도체 기술의 발달로 컴퓨터의 가격은 엄청난 속도로 저렴해지고 있으며 통신 기술의 발달로 분산처리 컴퓨터의 기술이 과거 몇년 동안 급속히 발달하여 왔다. 특히, 중앙집중처리에 대해 신뢰도(Reliability) 증진과 가용성(Availability) 증대를 위한 분산처리 구조 및 기술에 대한 연구가 80년대 들어서 활발히 전개되었다. 또한 공장의 자동화, 사무의 자동화, 로보트의 출현, 원자로 관리, 전자교환기 등 많은 부분에 컴퓨터의 응용이 증가하면서 실시간처리(Real time processing)에 대한 OS의 연구가 활발해지게 되었다.

특히, 전자교환기에서는 내부의 프로세스 수가 가입자의 수, 가입자의 평균 통화시간, 사용가능한 통화로의 수 등에 따라 결정되어지며 같은 시각에 엄청난 숫자의 프로세스가 생성, 소멸되고 고실시간성(Hard real time)이 요구되기 때문에 이러한 실시간 분산처리 운영체계(Real time distributed operating system)는 비실시간의 상용 운영체계와는 많은 부분에서 그 특성이 다르며 구조도 달

라지게 된다. 여기서 말하는 실시간성이란 어떤 외부 물리적 현상(Event)에 대해 그 반응(Response)이 일정한 짧은 시간내에 이루어져야 하며 이러한 외부 event는 일반적으로 비동기성(Asynchronous)을 갖게 되는 성질을 의미한다. 특히, 이러한 실시간의 보장 때문에 모든 프로그램과 데이터는 주 메모리(Main memory)에 상주(Resident) 하게 되며 이에 따라 메모리의 관리나 고장감내성(Fault tolerence)에 대해 기존의 OS와는 그 성격이 많이 달라지게 된다. 또한 분산시스템이 갖는 필수적 요건으로서 컴퓨터간의 통신에 있어서도 전자교환기용 OS는 그 실시간성 문제로 많은 제약을 받게 되고 이러한 제약하에서 요구조건이 만족되도록 설계되어져야 한다.

본고에서는 한국형 전전자교환기 TDX-1에서 실현한 실시간 분산처리 운영체계인 TDXOS의 그 특성과 실현에 대해 서술하고자 한다. TDXOS는 분산처리가 가능하도록 하면서 고실시간성을 보장하도록 설계 되었으며 고장감내를 위해 컴퓨터의 이중화가 가능하도록 실현되어 있다. Ⅱ장에서는 TDXOS의 구조와 컴퓨터의 기동에 대하여, 그리고 Ⅲ장에서는 TDX-1의 프로세스와 동시 수행성에 관해 언급하고 Ⅳ장에서 Ⅹ장까지 TDXOS의 실현을 논하였으며 끝으로 결론을 내렸다.

Ⅱ. TDXOS 구조

TDXOS는 TDX-1 시스템을 잘 정리된 IPC(Inter Processor Communication) 메시지에 연결되어진 독립적 기능을 수행하는 프로세서들의 군으로 보고 있다. 이러한 관점에서 각 프로세서의 고유 기능에 관계없이 공통적인 성격을 갖는 OS의 부분을 공통적인 독립 OS로 실현하고 각 프로세서 고유 기능에 관계되면서 필요한 OS의 기능을 비 독립 OS로 실현하였다. 이러한 관점은 다른 교환기용 OS에서도 찾아볼 수 있는 형태이며[4] 특히 디스크, 마그네틱 테이프, 프린터, 카트리지 드라이브 등과 같은 입·출력 장비들은 공통 OS를 기반으로 하여 하나의 server 형태로 실현되었다.[2]

TDXOS의 전체적인 구조는 〈그림 1〉 과 같으며 기동처리기에서 시스템의 초기화를 시행한후 이중화 관리기를 거쳐 active 프로세서와 standby 프로세서가 결정되고 그 결과에 따라 인터럽트 처리기와 IPC관리기의 도움을 받아 프로그램 및 데이터의 로우딩이 이루어지게 된다.

〈그림 1〉 TDXOS 구조

Ⅲ. 프로세스와 실행 동시성(Concurrency)

TDXOS에서 취급하는 프로세스는 단말 프로세스와 시스템 프로세스로 나누어진다.

시스템 프로세스는 server동작이나 OS의 프로세스 각종 관리자, 감시 프로세스 등이 이에 속한다.

단말 프로세스란 하나의 단말 행위에 의해서 발생되는 프로세스로서 대표적인것이 호

의 시도를 들 수 있으며 발신 프로세스와 착신 프로세스로 독립되어 이루어진다. 이는 또한 분산처리 시스템에 적합하여 각기 분리된 프로세서에서 그 처리가 가능해지는 특징을 가지고 있다. 단말 프로세스는 외부의 event나 signal과 같이 요구가 있을때만 생성(Create)되고 기동(Activate)되어 이 event가 처리되고 난후 제어기능을 OS에 돌려주고 그 프로세스는 다음 입력 event가 들어올 때까지 어떤 프로세스 상태에서 머물게 된다. 이러한 활동상태에 있는 event는 동시에 많이 존재하게 되며 동시처리(Concurrent Processing)가 TDXOS하에서 가능하다. 이러한 단말 프로세스라는 개념에 따라 응용 소프트웨어를 구성하는 것은 여러가지 잇점이 있게 된다.

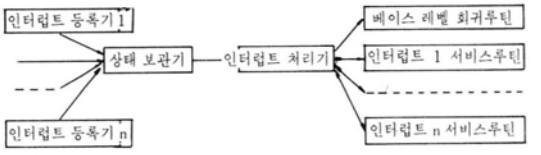
응용소프트웨어 설계자는 단계적인 외부 event를 한 프로세스내의 분리된 task에 대응시켜 프로그램함으로써 단순히 외부 변화에 대해서만 생각하고 사용자의 입장에서 보게 됨으로써 프로그램이 단순해지고 구조화(Modular)하게 된다.

Ⅳ. 인터럽트 처리

인터럽트는 현재 수행되고 있는 프로세스를 중단시키고 그 인터럽트를 수행해 줄 프로세스로 제어기능을 옮기는 방법이다. 이때 중단되는 프로세스의 모든 상태가 저장되어 인터럽트 프로세스의 수행이 끝난 후 중단된 프로세스의 수행을 지속시켜주기 위해 다시 복구되어져야 한다.

TDXOS는 NMI(Non Maskable Interrupt)와 MI(Maskable Interrupt)를 모두 처리할 수 있으며 NMI의 항목(Source)간에는 우선순위(Priority)가 없으며 polling방법에 의해 NMI 처리가 서비스 루틴으로 연결되어 진다. 다수의 MI 항목들 간에는 하아드웨어에 의한 daisy chain 방식으로 우선순위가 정해지지만 OS에서 table에 의해 우선순위를 하드웨어 구성과는 독립적으로 조정할 수 있어 유연성과 독립성을 갖으므로 편리하다.

인터럽트 처리기는 〈그림 2〉와 같이 4개의 시스템 프로세스로 구성되어져 있다.



〈그림 2〉 인터럽트 처리기

인터럽트 등록기는 해당 인터럽트 벡터에 의해 그 시작 주소가 결정되어 지며 각각의 인터럽트 발생을 인터럽트 등록표에 기록하게 된다. 이때 인터럽트 처리기는 등록표를 인덱스로 하여 인터럽트 서비스 루틴 주소를 갖고 있는 연결 테이블(Link table)을 액세스하도록 되어있다.

인터럽트 레벨은 단일 레벨화시킴으로써 높은 우선순위의 인터럽트 서비스중에도 우선순위가 낮은 인터럽트가 검출될 수 있도록 하는 특수한 구조를 갖고 있다. 또한 베이스 레벨 귀환처리 루틴이 있어 인터럽트 처리기도 베이스 레벨 귀환에 대한 특별한 주의를 필요치 않으며 이는 인터럽트 등록 테이블의 모든 bit가 0으로 되는 때가 된다. 인터럽트는 시스템 인터럽트와 사용자 인터럽트인 경우 모두 지연처리가 가능하여 베이스 레벨의 최고 우선순위 시점에서 처리된다.

Ⅴ. Job Scheduling

TDX-1에서의 job의 생성은 인터럽트, IPC메시지, time out등의 3가지 종류에 의해서 이루어 지며 이들의 처리되는 레벨에는 3가지 즉, 긴급레벨, 인터럽트레벨, 기본레벨로 나누어져 있고 다시 기본레벨은 실시간성에 따라 기본레벨 0(B0), 기본레벨 1(B1) 으로 나누어져 있다.

기본레벨은 다시 기본레벨 제어기에 의해 B0 dispatcher와 B1 dispatcher가 기동되어지고 이들은 인터럽트 레벨이나 긴급 레벨에 의해 preemption을 당할 수가 있으며 인터럽트 레벨은 긴급 레벨에 의해 preemption을 당하며 B1 dispatcher는 실시간성 때문에 B0 dispatcher에 의해 preemption을 당하고 항상 B0 job의 처리가 끝난후 B1 job의 수행이 가능하다. B1 레벨의 job들은 OS primitive에 의해 job이 기동되든지 취소될 수가 있으며 scheduling 요구에 따라 영구주기적인 job, 일정한 횟수만큼만 수행 가능한 job, 일정시간 지연 수행되는 job, 요구즉시 수행가능한 job 등으로 선택될 수 있으며 이때의 지연시간은 초, 분, 시간 단위로 등록될 수 있다.

수행되는 프로세스는 여러개의 task로 구성되어 있으며 TET(Task Entry Table)를 가지고 있고 process ready list에 의해 연결된다. B0 dispatcher와 B1 dispatcher는 각각 ready list의 job들에 하나씩 차례대로 프로세스를 할당시켜주게 된다.

Ⅵ. 메모리 관리 (Memory Management)

TDX-1의 CPU인 Z80의 논리적 주소공간은 64K이므로 가상(Virtual) 메모리의 반대개념인 뱅크(Bank) 시스팀으로 실제 메모리 주소공간을 확장하여 최대 256 Kbytes 까지의 메모리를 독립적으로 액세스 가능하도록 되어 있다. TDX-1은 실시간처리가 요구되는 시스팀이므로 일반 OS의 메모리 관리와는 달리 프로그램과 데이터를 모두 주 메모리에 위치시켜 수행시간을 단축하는 방식을 사용하게 되는데 논리적 주소공간 이상의 메모리는 mapping 기능을 사용하여 액세스한다.

MMU(Memory Mapping Unit)를 이용하여 특정 논리적 주소공간에 확장된 메모리의 페이지(4 Kbyte 단위)를 위치시키는 것을 mapping이라고 하고 그 논리적 주소 공간을 window라고 한다. 이 window의 크기는 페이지 단위이며 한번의 mapping으로 액세스가 가능한 프로그램 또는 데이터의 크기도 페이지 크기 이내로 제한된다. 따라서 특정 페이지를 액세스하기 위해서는 아래 프로그램과 같이 OS의 primitive를 이용하여야 한다. 여기서 program-A는 실제 수행하고자 하는 프로그램의 이름이며 MAP( )과 release( )는 OS primitive이다.

```
MAP(page-number);
program-A;
Release( );
```

메모리 Mapping기능 사용예

이러한 primitive의 사용은 중첩이 가능하다. 한편, 각 페이지내의 프로그램 또는 데이터의 특성에 따라 페이지 특성을 설정 또는 변경하는 primitive를 사용함으로써 메모리 사용위반(Violation)을 탐지하고 메모리 내용을 보호할 수 있다. 이러한 각 페이지에 부여될 수 있는 페이지 특성으로는 write 불능, 프로그램 수행방지, 사용하지 않는 페이지 설정 등이 있다.

Ⅶ. 프로세서간 통신

메시지 통신은 분산시스팀의 프로세스 간에 여러 바이트의 정보를 교환하는 주된 방법으로서 발신측은 OS의 통신 primitive 를 이용하여 통신용 시스팀 버퍼에 메시지를 저장한다. 이후에 IPC 토큰을 받으면 전송 primitive가 일정량의 메시지를 통신용 버스를 통해 수신측으로 전달한다. 이때 한번에 전송가능한 메시지의 량을 제한하여 한 프로세서가 통신시간을 독점하여 시스팀에 통신 불균형을 일으키는 것을 막고 있다. 프로세서간의 통신은 토큰 버스 형태에 SDLC방식을 사용하고 있다.

TDXOS는 physical layer에 대한 user layer의 통신 overhead를 극소로 하고 실시간성을 극대화하기 위하여 data link layer까지 주로 중점을 두었다.

메시지 전송은 point-to-point 방식과 broadcasting방식이 모두가 가능하다.

1. MSG 형태

교환기에서 사용되는 메시지의 크기는 아주 작아 수 바이트에서 수십 바이트내의 임의의 크기가 된다. 이러한 메시지의 요구에 맞는 최적 조건의 IPC알고리즘을 찾아 TDXOS에서 실현하였다.

일반적으로 메시지의 전송도중에 physical layer에서 여러가지 에러가 발생될 수 있으며 이중에는 bit값의 변화, 전달 메시지의 절단 등이 있는데 극단적인 경우에는 한 메시지 내에 표시되어 있는 그 메시지의 길이를 나타내는 데이터도 변할 수 있다. 이러한 경우에는 수신측의 입력 메시지큐에 대한 에러 복구가 아주 어렵고 복잡해진다. 에러의 전파를 최소화 하고 에러 복구 및 검출을 빨리 할 수 있는 방법으로 TDXOS에서는 unit-variable(UV) 메시지 형태를 취하고 있다.

일반적인 메시지의 형태는 〈그림 3〉과 같으며 메시지는 12n(n=1, 2, 3, ……) 형태를 갖고 있다. 즉, UV형태에서 1 unit는 12byte가 되어 메시지 수신기가 수신된 메시지의 크기를 측정한 후 12의 배수가 아니면 에러로 처리하도록 한다.

수신자-주소	메시지-종류	메시지-길이	
발신자-주소	일반정보	패리티	CRC

〈그림 3〉 IPC 메시지 형태

수신자의 주소에는 발신자 자신도 가능하며 TDXOS에서는 inter-processor 통신과 intra-processor 구분이 없다. 이렇게 함으로써 통신 알고리즘이 단일화 되어 유지보수가 쉬운 잇점이 있으며, 이중화된 두 프로세서 사이에 별도의 추가적인 통신 노력이 절감된다. 이는 이중화에서 데이터 일치를 위한 중요한 사항이 된다.

2. 메시지 저장

메시지는 수신자와 송신자 사이에 사전에 잘 정의되어져야 하며 발신자는 OS primitive와 다음과 같이 대화함으로써 다른 프로세서에 메시지를 전달할 수 있다.

```
TOQRQ(destination-address, message-
        length, originator-address,
        message-id)
RECEIVE(return-flag, bfr-address)
        IF return flag= 0
        THEN write message from
          bfr-address
        ELSE cancel request
```

메시지 전달 알고리즘

TOQRQ는 통신 버퍼에 대한 open request이며 이때, OS primitive는 결과 파라미터와 사용 가능한 통신 버퍼의 주소를 가르쳐준다. 사용자는 전달 메시지를 모두 기록한 후 T-OQCL을 불러 반드시 통신 버퍼를 close 해야 한다.

3. 메시지 전송

TDXOS의 kernel에서 50ms의 RTC 마다 IPC 토큰을 요구한다. 토큰은 토큰 인터럽트에 의해 전달되어 인터럽트 처리기에 의해 인식되어 지고 인터럽트 처리기가 메시지 전송기(Message transmitter)를 부르게 된다. 이때 메시지 전송기는 일정량의 메시지 만을 전송하게 되며 최대허용 전송량은

총메시지의 갯수< 5

n< 26

에 의해 제한된다. 이것은 한 프로세서가 버스점유를 독점하는 경우를 막고 또한 메시지 전송이 어느 특정 node로 집중될 경우 최대 허용량을 넘는 것을 막기 위함이다. 메시지 전송기는 메시지 전송이 끝나면 즉시 IPC토큰을 다음 프로세서로 넘겨준다.

한편, 입력 메시지는 physical layer 에서 DMA에 의해 수신된다. IPC 토큰이 접수될 때 메시지 수신기는 DMA의 상태를 읽어들여 입력 메시지량을 측정하고 기본 레벨의 입력 메시지 처리기가 처리할 수 있도록 준비를 해준다.

Ⅷ. Time Management

실시간처리용 OS에서는 다양한 종류의 시간 처리 기능을 갖게 된다. 특히 교환기용 실시간 처리가 가능한 TDXOS에서는 50ms RTC(Real Time Clock)를 기초로 하여 0.1초, 0.5초, 10초, 분, 시, 일, 월, 년도의 시간을 사용자에 제공하고 있다.

또한 이들 시간은 OS 자신도 사용하며 각종 프로세서 유지보수 및 교환기 유지보수용으로 short term timer와 long term timer에 의해 관리되어지고 해당 시간처리기를 가동시키게 된다. 각 시간처리기는 규정된 시간이 지나는 timer에 대해서 해당 프로세스를 dispatch시켜 time out에 대한 서비스가 일어나도록 한다. 또한 사용자가 요구하면 시스템이 현재의 시각을 알려주기도 한다.

Ⅸ. 프로세서 이중화 (Processor Duplication)

TDXOS는 한 node의 프로세서에 대해 이중화의 관리기능을 갖고 있는데 이러한 이중화 기본 개념은 다음과 같다. 즉, 분산처리 시스템에서 동일한 IPC 메시지가 동일한 시각에 이중화된 두 프로세서에 전달되고 또한 동시에 두 프로세서가 이 메시지들을 순차적으로 수행하면 그 처리 결과는 동일할 것이라는 것이다.[3] 그러나 이러한 기능을 실현하기 위해서는 많은 부수적인 문제가 발생하는데 이에 대한 실현 방법으로

1) User transparency를 보장하는 방법 (OS가 모든 처리를 함)
2) OS와 응용 프로그램 각각에 필요한 사항을 실현하는 방법
3) 응용 프로그램에서 모든 필요사항을 실현하는 방법

등이 있다. TDXOS에서는 두번째 방법을 택하여 실현하였으며 프로세서 이중화 운용 방식은 hot standby 방식을 취하고 있다.

이러한 처리 결과의 동일성 유지를 프로세서 이중화에 이용하기 위해서는 active 프로세서의 고장이나 기능 수행 불능시 즉시 프로세서 절체를 수행할 수 있어야 한다. 이러한 프로세서 절체 기능은 각 프로세서의 상태를 상대방이 주기적으로 감시하는 기능과 스스로 프로세서 절체를 요구하는 기능으로 이루어지는데, active 프로세서의 기능 수행이 불가능할 때 이러한 기능에 의해 active는 standby로, standby는 active로 절체되어 외부 서비스에 단절이 일어나지 않도록 하고 있다.

그리고 이중화 구조에는 한 프로세서가 기동(Restart)하면 active나 standby중 어느 하나의 상태로 동작하여야 하므로 프로세서가 기동후 정상상태가 되기까지는 이중화의 관리하에 있게 된다. 그러므로 이에 대한 체계적 관리 및 제어를 위하여 프로세서의 동작상태를 다음과 같이 정의하고 있다.

1. 각 프로세서의 상태천이

프로세서 이중화에 관련되어 각 프로세서는 다음과 같이 특별한 동작상태를 가지고 있다.

N : 정상상태로 정의되며 프로세서의 정규

기능을 수행할 수 있는 상태

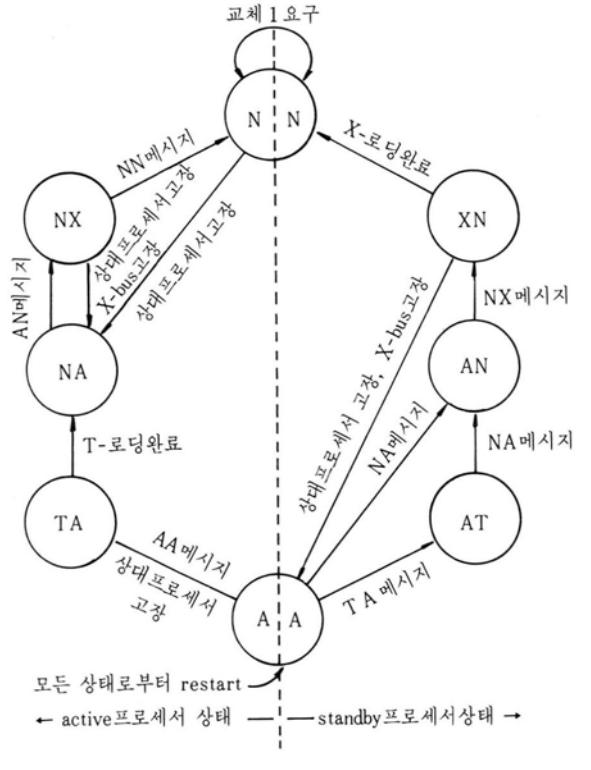
T : T-로우딩으로 정의되며 T-bus를 통하여 프로그램 및 데이터를 로우딩받는상태

X : X-로우딩으로 정의되며 X-bus를통하여 프로그램 및 데이터를 로우딩받는상태

A : 비정상상태로 정의되며 프로세서가 고장이거나 로우딩을 준비하고 있는 상태

Active 프로세서는 T 상태로서 standby 프로세서는 active 프로세서가 N 상태일때 X 상태로서 프로그램이나 데이터의 로우딩이 수행되고 정상적 동작상태는 active프로세서나 standby 프로세서 모두 N 상태이다. 그런데 양 프로세서는 자신 및 상대방의 동작상태를 알고 있어야 active/standby의 결정 및 상태의 천이를 수행할 수 있는데 이를 위해 자신의 상태와 상대방의 상태를 조합하여 갖고 있다. 이중화 상태로 허용되는 조합은 모두 8개로 NN, NX, NA, XN, TA, AT, AN, AA이며 앞의 상태는 자신을, 뒤의 상태는 상대방을 나타낸다.

이러한 이중화 상태를 상대방 프로세서에 알리기 위해서 X-bus를 통해 이중화 상태 메시지가 이중화 상태가 변화할 때마다 서로 교환된다. 이중화 상태 사이의 천이는 이중화 상태메시지, 감시기능에서 감지한 상대 프로세서의 동작 수행 불능, 자신의 주요 고장에 따른 교체요구, 프로그램이나 데이터의 로우딩완료 등에 의해 수행되며 이들 이중화 상태의 천이와 사건을 〈그림 4〉에 나타내었다.



〈그림 4〉 이중화상태와 이들의 천이 및 사건

2. 데이터 일치 (Data Consistency)

양 프로세서는 동일한 기능을 동시에 수행하므로 프로세서의 입력(IPC메시지)이 동일하고 프로세서의 상태가 동일하다면 그 처리결과도 동일하여 완전한 hot standby 방식으로 이중화를 구성할 수 있다. 그러나 active 프로세서가 서비스를 수행하면서 standby 프로세서를 로우딩시키므로 양 프로세서의 처리의 시작점이 일치하지 않으며 또 프로세서의 상태 환경이 고장 인터럽트 등으로 인해 언제나 서로 동일할 수 없다.

따라서 처리 시작점을 일치시켜도 일정 시간 후에는 고장처리 등으로 인해 active 프로세서와 standby 프로세서의 데이터가 상이하게 되는 경우가 발생한다. 그러므로 양 프로세서 간의 데이터 일치를 위하여 T-bus를 통해 데이터 synch. 메시지를 보냄으로써 중요 데이터를 일치시키는 방식을 사용한다. 즉, active 프로세서가 한 입력에 대해 처리한 후 그 결과의 중요 데이터를 standby 프로세서로 넘겨서 update시켜 active/standby 교체후에도 중요처리 상태가 계속 유지되도록 한다.

X. 프로세서 부하 관리

분산처리 시스템에서 프로세서에 걸리는 부하가 한 프로세서에 집중되면 시스템 전체의 기능 불능 상태에 이를 수도 있다. 더우기 실시간처리의 요구사항을 만족해야 하므로 일반 운영체계와는 달리 프로세서의 부하를 계속해서 측정해야 한다.

TDXOS는 이러한 부하측정 기능을 갖고 있으며 항시 읽을 수가 있다. 프로세서가 수행해야 할 task가 전혀 없을 때는 이러한 task의 발생을 기다리면서 기본적으로 돌고 있는 일정한 loop을 가지고 있으며 이 loop을 수행할 때 자신이 수행한 횟수(N)를 계산하게 된다. 한 loop을 수행하는데 필요한 시간(T)을 알 수 있으므로 휴지시간 및 프로세서의 부하시간은 다음과 같이 계산된다.

휴지시간=N×T

프로세서의 부하=50ms−휴지시간

이러한 부하 측정은 50ms마다 수행하면서 4 sec 단위로 감시를 하여 프로세서의 부하가 전체시간의 20% 미만일 때를 경과부하(Minor load), 10% 미만일 때를 중과부하(Major load)로 규정하고 호처리, 운용 유지보수 처리를 단계적으로 차단하며 다른 node의 프로세서에게 알리어 시스템 전체의 부하조절을 자동적으로 수행하고 있으며 전체 시스템이 과부하로 인해 기능 불능 상태로 이르는 것을 막고 있다.

XI. 결 론

TDXOS는 고실시간성을 위하여 redundancy를 최소로 줄인 효율적인 OS이며, 특히 높은 시스템 부하를 처리할 수 있도록 IPC 처리 속도를 최대한으로 높였다. 또한, 단말 프로세서 개념을 도입함으로써 응용 프로그램의 제작 용이성을 높이고 구조(Modular)화 시켰다. 인터럽트 레벨을 단순화 시켜 처리 속도를 빠르게 하였으며 프로세서의 부하 관리와 이중화 기능을 효율적으로 실현시킨 OS이다.

한편, 장차 계속 연구 및 실현되어야할 분야 또한 적지않다. 응용 프로그램의 고급 언어 사용이 가능하도록 하며 이중화에 대한 고장감내 및 데이터 일치(Data consistency)를 OS에서 담당하는 user transpancy를 추구하여야 한다. 또한, 보다 많은 primitive를 체계적으로 제공하여야 하며 IPC 부분에서 point-to-point 및 broadcasting 기능을 포함하는 고신뢰도 IPC 기능이 실현되어야 한다.

〈參 考 文 献〉

1. 강석열외 34명 "전전자 교환기 개발 사업중 CP S/W개발 과제 보고서", pp. 1565-1716, Dec. 1983.
2. R. H. Needham and A. J. Herbert, "The Cambridge Distributed Computing System", Addison-Wesley, 1984.
3. Samuel D. Glazer, "Fault Tolerent Mini Needs Enhanced Operating System", Computer Design, Aug. 1984.
4. R. J, G. J. Kujawinski, and E. H. Stredde, "Real Time Architecture utilizing the DMERT Operating System", B. S. T. J., Vol. 62, No. 3, part3, pp 775-826, Mar. 1983.
5. J. P. Delatore, R. J. Frank, H. Oehring, and L. C. Steeher, "Operational Software", B. S. T. J., Vol. 64, No. 6, Part2, pp, 1339-1356, Aug. 1985.
6. Jim Gray, Paul McJones, "The Recovery Manager of the System R Database Manger", Computing Surveys, Vol. 13, No, 2, June 1981.
7. Bob Snead, Frank Ho, and Bob Engram, "Operating System Features Real Time and Fault Tolerence", Computer Design, Aug. 1984.

姜 錫 烈(Kang, Seok Youl)
1949년12월 19일생
1973 : 부산대학교 전자공학과 공학사
1976 : Canada Ottawa대학 수학
1982~1986 : 한국과학기술원 재학
1976~1977 : 한국반도체(주)
1977~1980 : 삼성GTE통신연구소 근무
1980~1986 : 한국전자통신연구소
1986. 8. 현재 : 운영체계연구실장