소 특 집

멀티미디어 시스템을 위한 실시간 운영체제 연구개발 동향

공 기 석

한국산업기술대학교 컴퓨터공학과

I. 서 론

최근 멀티미디어 응용 프로그램은 인터넷의 폭발적인 성장과 함께 대표적인 컴퓨터 응용 프로그램이 되었고 이제는 PC나 워크스테이션 등의 데스크 탑 컴퓨터에서 뿐만 아니라 PDA, 휴대전화기 등에서도 동영상을 볼 수 있는 단계에 이르렀다. 또한 전용 멀티미디어 기기 시장에서 디지털 TV를 포함한 각종 셋탑 박스, 비디오 게임기, DVD 플레이어를 비롯한 홈 시어터 등은 이들이 큰 사업분야를 이룰 것임을 예고하면서 빠른 속도로 발전하고 있다.

멀티미디어 응용 프로그램은 실시간 응용 프로그램중에서 독특한 위치를 차지하고 있다. 전통적인 의미에서의 엄격한 시간제한을 요구하는 경성 실시간(hard real-time) 응용프로그램과는 달리 멀티미디어 응용 프로그램은 시간제약이 엄격하지 않은 연성 실시간(soft real-time) 응용 프로그램에 속하며 실행되는 동안에 자원을 많이 사용하는 것이 특징이다. 여기서 자원이라고 하면 CPU, 네트워크, 디스크, 메모리 등을 의미한다. 운영체제(OS)를 이러한 자원을 효율적으로 관리하기 위한 시스템 소프트웨어라고 정의했을 때, 멀티미디어 응용 프로그램을 실행하기 위해서는 운영체제의 지원이 필요하다는 것은 당연하다.

운영체제의 측면에서 멀티미디어를 지원하기 위한 연구는 1980년대 말에 시작되어 1990년대에 본격적인 연구가 이루어지면서 많은 성과를 거두었다. 그러나 이 분야는 아직도 해결해야 될 많은 문제가 산적해 있는 것도 사실이다. 대부분의 연구는 주로 CPU 스케줄링 분야에서 이루어졌다. 멀티미디어가 갖는 연성 실시간성을 만족시키기 위해서는 기존의 경성 실시간 분야에서 주로 사용되는 RM(Rate Monotonic)이나 EDF(Earliest Deadline First)가 아닌 다른 알고리즘이 필요성이 제기되었기 때문이다. 최근의 연구는 QoS(Quality of Service)에 대한 지원에 초점이 맞추어져 있다. 이는 멀티미디어 응용 프로그램이 요구하는 다양한 서비스의 품질을 만족시키기 위해서는 CPU 만이 아닌 다양한 자원들을 효율적으로 관리해야 하고, 과부하가 걸렸을 때 이 상황에 대한 적절한 적응(adaptation)이 필요하기 때문이다.

본 고에서는 멀티미디어 운영체제의 주요 개념과 개발 동향에 대해 살펴보고자 한다. 이를 위해서 2장에서는 멀티미디어 운영체제가 갖는 특징에 대해 설명하고, 3장에서는 대표적인 연구 성과를 소개한다. 여기에서는 1990년대부터 현재에 이르기까지의 대표적인 멀티미디어 운영체제에 대해 설명하고 있다. 4장에서는 현재의 연구동향에 대해 소개하고 5장에서 결론을 맺도록 하겠다.

II. 멀티미디어 운영체제의 특징

1. 멀티미디어 데이터의 특징

멀티미디어 데이터의 주종을 이루는 오디오, 비디오 데이터는 그 속성상 연속성을 갖는다. 이런 의미에서 멀티미디어를 연속 미디어(continuous media)라고 하기도 한다. 연속성을 만족시

키기 위해서는 필연적으로 실시간성이 요구된다. 그러나 멀티미디어에서 요구되는 실시간성은 마감시간(deadline)을 반드시 지켜야 하는 전통적인 의미에서의 강한 실시간성(hard real-time, 경성 실시간)이 아니다. 가령 초당 30프레임의 화면을 보여주어야 하는 멀티미디어 플레이어 프로그램에서 비디오 데이터가 늦게 도착해서 초당 20프레임을 보여주었다면 만족스럽지는 않지만 참을 수 없는 정도는 아니다. 즉 서비스의 질(Quality of Service, QoS)이 낮아지는 것이지 치명적인 결과를 초래하지는 않는다는 것이다. 멀티미디어의 이러한 특성을 약한 실시간성(soft real-time, 연성 실시간)이라고 한다.

또한 멀티미디어 데이터는 오디오, 비디오 데이터를 주기적으로 샘플링해서 만들어진 것이기 때문에 이러한 데이터를 처리하는 연산은 주기적으로 이루어져야 한다는 특징을 갖는다.

멀티미디어 데이터는 대역폭이 가변적이라는 것도 특징이라고 할 수 있다. 압축하는 방식에 따라 다양한 대역폭이 존재할 수 있고, 필요에 따라 협상(negotiation)도 가능하다. 가령 고화질의 화상을 전송할 수 있는 대역폭을 확보할 수 없는 경우, 응용 프로그램에서는 다소 화질 떨어지는 것을 감수할 수 있다는 것을 의미한다.

2. 멀티미디어 응용 프로그램의 요구사항

멀티미디어 데이터를 처리하는 응용 프로그램의 일반적인 특성은 다음과 같다.

첫째, 높은 대역폭(bandwidth) 처리를 요구한다. 전화에서 음성 스트림은 16Kbit/s를 필요로 하지만 CD 수준은 1.5Mbit/s를 요구한다. 비디오 데이터 전송은 MPEG의 경우 대략 1.2 Mbit/s를, H.261의 경우는 64Kbit/s~2Mbit/s, 압축된 HDTV는 20Mbit/s, 비압축 HDTV의 경우 1Gbit/s를 필요로 한다.

둘째, 낮은 지연성(latency)과 높은 응답성(responsiveness)을 요구한다. 음성 스트림의 종단간(end-to-end) 지연은 특별한 하드웨어가 없는 경우 40ms 이하여야 한다. 오디오 비디오의 동기화를 위한 데이터의 왜곡(skew)은 80 ms 이하여야 한다. 음악과 특정 음표간의 동기화 왜곡은 +/− 5ms 이내여야 한다.

셋째, QoS를 보장(QoS guarantee) 해야 한다. 사용자가 요구하는 특정한 품질을 만족시키기 위하여 시스템은 협상된 QoS 매개변수(parameter)에 맞게 멀티미디어 데이터를 전송하고 처리할 수 있어야 한다.

다음 〈표 1〉에서는 멀티미디어 데이터의 종류에 따른 응용프로그램의 특징을 정리하고 있다.

〈표 1〉 멀티미디어 응용프로그램의 특징

종류	예	주기	CPU 사용율	점진적 성능저하 가능 여부	지연에 대한 민감성
오디오	MP3, AAC	약 100ms	1~10%	불가	낮음
비디오	MPEG-2, AVI	33ms	큼	가능	낮음
분산 오디오	인터넷 전화	돌발적 (bursty)	1~10%	불가	높음
분산 비디오	화상회의	돌발적	큼	가능	높음
실시간 오디오	소프트웨어 신디사이저	1~20ms	가변적	불가	매우 높음
RT simulation	가상현실, Quake	리프레시 주기까지	보통 100%	가능	높음
RT 하드웨어	소프트 모뎀, USB 스피커	3~20ms	50%까지	불가	매우 높음

3. 멀티미디어 운영체제 설계 시 고려사항

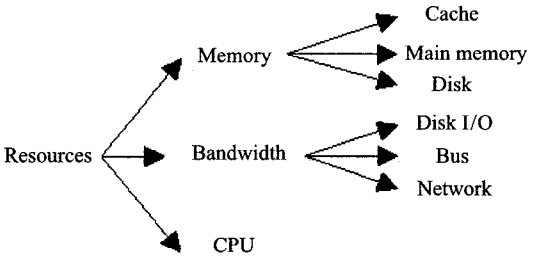
멀티미디어를 지원하는 운영체제는 다음과 같은 요소를 고려하여 설계되어야 한다.

(1) 응용 프로그램이 필요로 하는 자원과 시간 제약 요건을 운영체제에게 지정할 수 있어야 한다.
(2) 운영체제는 응용프로그램의 이러한 요청의 수용여부를 결정하는 입장제어(admission control)를 할 수 있어야 한다.
(3) 일단 입장이 허락된 응용프로그램은 스케줄러의 통제를 받아 실행된다. 스케줄러는 허용된 응용프로그램의 자원요구에 맞추어 시스템 내의 자원을 분배한다.
(4) 응용 프로그램의 쓰레드(thread)들 간에 통신을 하는 경우 또는 공유 자원에 접근하고자 하는 경우 실시간 동기화 메커니즘이 지원되어야 한다.

여기서는 위의 (1), (2) 항목을 멀티미디어 운영체제에서의 자원 관리 측면에서 살펴보고, (3)에 대해서는 CPU 스케줄링의 관점에서 주로 살펴보고자 한다.

1) 자원관리와 QoS

멀티미디어 시스템내에는 다음 〈그림 1〉에서와 같이 많은 자원이 존재한다. 운영체제의 기본적인 목표는 이러한 자원을 사용자들에게 공평하고(fairness) 효율적으로(efficiency) 분배하는 것이다. 멀티미디어 운영체제는 이러한 운영체제의 전통적인 목표이외에 시간제한성(timeliness)을 지켜줄 수 있어야 한다는 것이 추가된다.



〈그림 1〉 운영체제가 관리해야 하는 자원들

응용프로그램에서 요구하는 시간제약성은 QoS 요청으로 표현될 수 있다. 프레임 비율, 해상도, 지터(jitter), 종단간 지연시간, 동기화 왜곡 등이 QoS 매개변수의 예이다. 이러한 고수준(high-level)의 QoS 매개변수 들은 요청된 QoS를 만족시키기 위한 저수준(low-level)의 매개변수 또는 자원에 대한 표현으로 세분화 되어야 한다. 가령 주기당 CPU 시간, 메모리의 양, 평균/최대 대역폭 등이 그것이다.

QoS를 만족시키기 위해서는 해당 태스크를 수행할 때 적절한 시간에 충분한 양의 자원이 존재하도록 시스템의 자원을 관리해야 한다. 여기에는 다음과 같은 작업들이 포함된다.

- 자원에 대한 지정 및 할당 요청
- 입장 제어 : 요청을 만족시킬 만큼의 충분한 자원이 존재하는지를 테스트 한다. 테스트 방법은 자원의 지정 방식이나 할당 방식에 따라 다르다.
- 할당 및 스케줄링 : 적절한 시간에 충분한 자원이 존재할 때 시행된다. 자원의 종류에 따라 방식이 달라진다. 가령 CPU와 같은 배타적인 자원은 시간에 따라 스케줄 되어야 하고, 메모리와 같은 공유 자원은 주소 공간을 서로 다른 태스크에게 분배할 수 있다.
- 사용량 감시(usage monitoring) : 할당된 자원의 실제 사용량을 추적한다. 이것은 스케줄링을 위해서 주로 사용된다. 응용프로그램이 할당된 양 이상의 자원을 소비하지 않는다는 것을 확인하기 위해서도 필요하다. 이 정보는 또한 시스템으로 하여금 적응과정을 수행시키는 데에도 사용된다.
- 적응(adaptation) : 사용자/응용프로그램 또는 시스템에서 QoS나 해당 자원에 대한 요구수준을 높이거나 낮추는 과정을 의미한다. 적응이 필요한 이유는 우선, 멀티미디어의 경우 자원에 대한 요청이 예측하기 매우 어렵고, 둘째, 자원의 존재여부를 보장할 수 없기 때문이다. 적응을 하는 방식으로는 피드백 제어(feedback control) 방식이 많이

사용되고 있다. 피드백 방식의 경우 피드백을 얼마나 자주 수행할 것인가가 문제가 된다.

자원의 소유권을 어떻게 정할 것인가, 자원의 사용량을 어떻게 표현할 것인가에 대한 문제는 아직 해결되지 않은 분야로 현재에도 많은 연구가 이루어지고 있다.

2) CPU 스케줄링

멀티미디어의 연성 실시간성을 지원하는 CPU 스케줄러는 그 동안 가장 많은 연구가 이루어진 분야로서 대략 다음과 같은 6가지 종류로 분류할 수 있다.

(1) 정적 우선 순위 및 RM (Rate Monotonic) 방식

이것은 대부분의 범용 운영체제(Linux, Windows 2000 등)에서 사용하는 방식이다. 이들 운영체제에서는 멀티미디어 응용 프로그램에게 아주 높은 우선 순위(priority)를 부여하여 실시간성을 충족시키려고 하고 있다. 이것은 아주 단순하고 효율적인 방식이지만 많은 문제점을 갖고 있다. 대표적인 것이 우선 순위 역전(priority inversion) 문제이다. 또한 원하는 실시간성을 확보하기 위하여 우선 순위 값을 어떻게 부여할 것인가도 문제가 된다. 이러한 문제점 들은 다른 새로운 방식의 스케줄러를 탄생시킨 계기가 되었다.

(2) CPU 예약(reservation) 스케줄러

가령 매 50ms 마다 10ms의 CPU 시간을 예약하는 방식이다. 이렇게 하면 주기마다 예약된 만큼의 CPU를 할당 받는 것이 보장되기 때문에 시스템의 부하와 무관하게(부하 독립, load isolation) 지정된 응용 프로그램의 실행이 가능하다. 이 스케줄러를 구현하기 위해서 EDF나 RM 스케줄러를 사용하는 경우가 있다. 이 방식은 다양한 상황에 대응할 수 있는 유연성이 부족한 것이 문제이다. CPU 예약 방식은 주기나 CPU 사용시간이 미리 정해진 멀티미디어 응용 프로그램에 사용하기 적합하다.

(3) 비례 지분(proportional share) 스케줄러

자원에 대한 요청을, 특정 자원에 대한 상대적인 지분(relative share 또는 weight)으로 표현하는 방식이다. 가령 모든 응용 프로그램에 주어진 지분의 총합이 T라고 하고 특정 응용프로그램의 지분이 N이라고 하면, 이 응용 프로그램에게는 적어도 N/T 만큼의 CPU 시간이 할당된다. 이 방식은 패킷 스위칭 네트워크에서 사용하는 패킷 스케줄링 알고리즘에 그 근거를 두고 있다(weighted fair queueing, virtual clock, packet-by-packet generalized processor sharing 등). 비례지분 스케줄러는 동적인 상황에 적응성이 뛰어나다는 장점이 있지만 적절한 입장제어가 없으면 과부하시에는 특정 응용 프로그램에 요구되는 최소한도의 CPU 시간도 배분되지 않는다는 문제가 있다. 비례지분 방식은 점진적으로 성능저하가 가능한(graceful degradation) 멀티미디어 응용 프로그램에서 사용하기 적합하다. EEVDF(Earliest Eligible Virtual Deadline First)나 스탠포드 대학의 BVT(Borrowed Virtual Time) 스케줄러, SMART 스케줄러 등이 이 방식에 속한다.

(4) Earliest Deadline First(EDF) 스케줄러

임의의 스케줄링 알고리즘이 마감시간을 어기지 않고 어떤 태스크들을 스케줄할 수 있다면 EDF 또한 그 태스크들의 마감시간을 어기지 않고 스케줄할 수 있다는 점에서 EDF는 최적의(optimal) 알고리즘이다. 그러나 이 매력적인 알고리즘은 계산이 복잡하여 구현하기 힘들고 시스템의 오버헤드가 큰 것이 단점이다. 멀티미디어 운영체제에서는 EDF를 스케줄러 내에서 마감시간의 시급성(urgency)를 추적하는 용도로 사용한다. CPU 예약 방식의 Rialto OS나 비례지분 방식의 SMART에서 EDF를 이렇게

〈표 2〉 멀티미디어 스케줄러의 특징

스케줄링 방식	OS/스케줄러	부하 독립 (load isolation)	필요한 사전 지식	가변적인 지연에 대한 지원 여부
RM 및 기타 정적 우선순위	Linux, RTLinux, Solaris, Windows2000	낮은 우선순위로부터 독립	우선순위	가능
CPU 예약	Nemesis, Rialto, Spring	강함	주기, 계산시간	가능
비례지분	BVT, EEVDF, SMART	강함	지분, (지연)	가변적
EDF	Rialto, SMART	강함/약함	계산시간, 마감시간(deadline)	가능
피드백 제어	FC-EDF, SWiFT	가변적	metric, set point	가변적
계층적 스케줄링	CPU Inheritance, SFQ	가변적	가변적	가변적

사용하고 있다.

(5) 피드백 기반(feedback-based) 스케줄링

멀티미디어 운영체제는 총 부하의 예측이 어렵거나 개별 응용 프로그램의 실행시간이 급격하게 변하는 상황에서도 실행될 수 있어야 한다. 이러한 상황에 대응하기 위하여 제안된 것이 피드백 제어에 기반한 접근 방식이다. 피드백 제어는 입장제어 또는 스케줄링 알고리즘에 적용될 수 있다. 버지니아 대학의 FC-EDF(Feedback Control EDF)에서는 동적으로 CPU의 사용율(utilization)을 조정하는 방식을 사용하고 있다. Oregon Graduate Institute의 SwiFT에서는 응용 프로그램을 위한 CPU 시간 예약을 위해 피드백 방식을 사용하고 있다.

(6) 계층적(hierarchical) 스케줄링

계층적(또는 multi-level) 스케줄러는 전통적인 의미의 스케줄러를 일반화한 것으로서, CPU 시간을 다른 스케줄러들에게 나누어 주는 역할을 한다. 루트(root) 스케줄러는 계층상 하위에 있는 스케줄러에게 CPU 시간을 나누어 주는데, 스케줄링 트리의 리프(leaf) 노드에 이르기 까지 이 과정을 반복한다. 계층적 스케줄러를 사용하면 한 시스템에서 여러 개의 다양한 스케줄러를 동시에 사용할 수 있다는 것이 장점이다. 유타 대학의 CPU Inheritance Scheduling이나 오스틴 소재 텍사스 대학의 SFQ(Start-time Fair Queuing)에서 이 방식을 따르고 있다.

다음 〈표 2〉에서는 멀티미디어 운영체제에서 사용하고 있는 여러 CPU 스케줄링 방식의 특징을 정리하고 있다.

III. 대표적인 멀티미디어 운영체제

여기서는 그 동안 연구개발 되어온 대표적인 멀티미디어 운영체제에 대해 소개한다.

1. The Spring Kernel

멀티프로세서나 분산환경에서 실행되는 실시간 응용 프로그램을 지원하기위해 버지니아 대학에서 1991년에 개발된 운영체제이다. 이 운영체제의 특징은 반영적 구조(reflective architecture)인데 현재의 시스템의 상태를 고려하여 스케줄링을 결정하도록 되어있고 응용 프로그램의 성능을 점진적으로 저하시킬 수 있다. Spring에서는 동

적인 실시간 환경에서의 절대적인 예측가능성(absolute predictability)을 목표로 하고 있는데, 이를 위해서는 시스템에서 실행되는 모든 태스크의 최악실행시간(worst case execution time, WCET)을 알아야만 한다. Spring에서는 태스크의 중단(block)이 일어나지 않도록 자원 할당과 스케줄링을 계획 하에서 수행하고 있기 때문에 예측가능성을 확보할 수 있다.

하드웨어 관점에서 보면 Spring이 실행되는 노드는 한 개의 시스템 프로세서(SP)와 여러 개의 응용 프로세서(AP)로 구성된 멀티프로세서 시스템이다. SP는 스케줄링이나 주변장치 관리 등의 시스템 활동을 담당하고 AP는 SP에 의해서 생성된 스케줄에 의해서 응용 프로그램을 실행한다.

사용자 프로그램을 개발하기 위해서 두 개의 언어(Spring-C와 SDL)를 사용한다. Spring-C는 ANSI C의 변형된 형태로서 실시간 프로그램 개발에 사용된다. Spring-C에서는 지속시간(duration)을 예측할 수 없는 연산이 허용되지 않는다. Spring-C 컴파일러에서는 컴파일시 타이밍 분석을 수행하며 잠재적으로 중단(blocking) 가능성이 있는 곳을 여러 개의 연관성 있는 태스크들로 분해한다. Spring에서는 새로운 프로세스를 입장시킬 때 자원 사용에 대한 명시적인 계획을 수립하여 중단 가능성을 철저히 배제한다.

시스템 기술언어(SDL)는 응용 프로그램의 시간 및 자원 요청을 표현하는 데 사용된다. 이 정보는 커널과 스케줄러에 의해 실시간 응용프로그램의 요구조건을 만족시키기 위하여 사용된다.

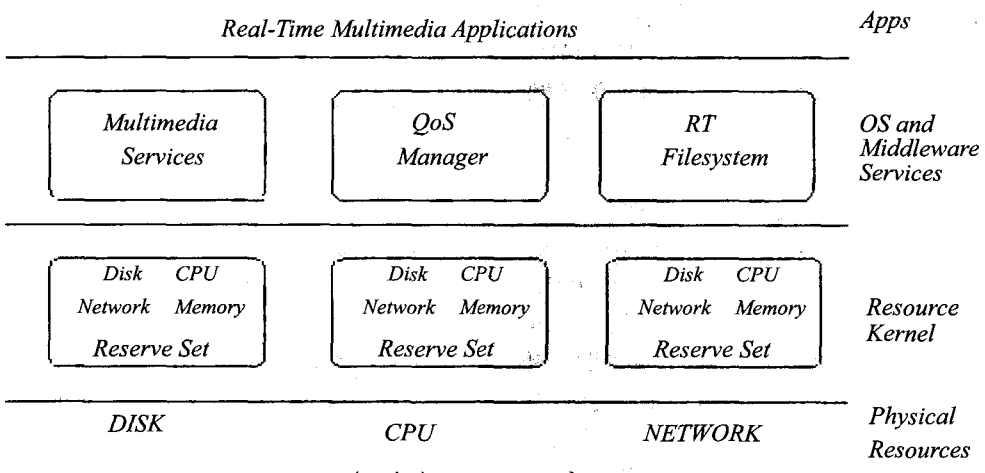
Spring 스케줄러는 SP 상에서 수행되는 사용자 수준(user-level)의 프로세스이며 AP 상에서 실행되는 사용자 프로그램들의 스케줄링을 담당한다. Spring 스케줄러는 다음 두 가지 보장을 통해서 입장제어를 수행한다.

- 모든 태스크들이 수행가능 할 때만 새로운 요청이 받아들여 진다.
- 일단 받아들여지면 모든 태스크는 미래의 다른 스케줄링 요청(같거나 낮은 우선 순위의 요청)들과 무관하게 마감시간을 준수하는 것이 보장된다.

가상 메모리(virtual memory)는 그 안에 내재된 예측 불가능성 때문에 Spring에서는 지원되지 않는다.

2. Real-Time Mach(RT-Mach)

1990년에 카네기멜론 대학에서 개발한 마이크로커널 기반의 분산 실시간 운영체제이다. RT-Mach의 구조가 다음 〈그림 2〉에 제시되어 있다. RT-Mach에서는 우선순위기반 선점형 스케줄링 방식(priority driven preemptive sche-



〈그림 2〉 RT-Mach의 구조

duling)을 사용하고 있다.

RT-Mach는 고정 우선순위 스케줄링, EDF, 라운드 로빈 등 여러 가지 다양한 스케줄링 정책을 지원하며 동적으로 선택 가능하다. 또한 RT-Mach는 프로세서 리저브(processor reserve) 개념을 지원하는데, 응용 프로그램이 매 T시간마다 C 시간 만큼의 계산을 수행하도록 요청하는 것을 의미한다. 이를 통해서 요청된 자원량에 근거해 RM 우선순위를 부여 받는다. 여러 개의 쓰레드는 동일한 프로세서 리저브에 속할 수 있다. RT-Mach는 시스템의 과부하를 막고 계산시간 한도를 지켜내기 위해서 입장제어를 수행한다. RT-Mach는 또한 실시간, 비실시간 쓰레드의 공존을 허락한다. 실시간 쓰레드는 그것이 경성인지 연성인지, 주기적인지 아닌지를 나타내는 타이밍 속성을 가진다.

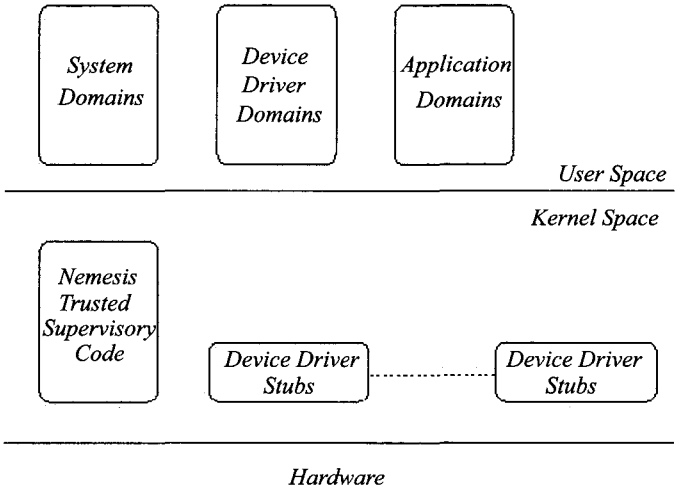
RT-Mach의 동기화 프리미티브(synchronization primitive) (mutex, condition variable 등)들은 실시간성을 지원한다. 대기중인 쓰레드의 큐잉 정책으로는 EDF가 사용된다. RT-Mach는 우선 순위 역전(priority inversion) 문제를 해결하기 위해 우선 순위 상속(priority inheritance) 방식을 사용한다. 자원을 점유하고 있는 쓰레드의 우선 순위는 그 자원을 기다리고 있는 쓰레드들이 가진 우선 순위의 최대값으로 높아진다. 이렇게 해서 높은 우선 순위를 가진 쓰레드의 정지시간이 임계영역의 길이 만큼으로 제한된다.

RT-Mach는 마이크로커널 방식의 대표적인 운영체제인 Mach를 실시간 운영체제로 변형시킨 것으로 이후의 많은 실시간 운영체제 연구의 출발점이 되었고 멀티미디어 운영체제의 연구에도 큰 영향을 미쳤다.

3. Nemesis

Nemesis는 캠브리지 대학에서 1996년에 개발된 마이크로 커널 방식의 운영체제이다. Nemesis는 CPU, 메모리, 네트워크, 디스크 등의 모든 자원에 대해 일정한 QoS를 요구하는 실시간 응용 프로그램을 지원하기위해 설계되었다. 마이크로커널은 하나의 응용 프로그램의 실행을 위해 여러 개의 서버가 동작하는 방식이기 때문에 자원의 사용량을 추정하기가 어렵다. Nemesis에서는 이러한 어려움을 해결하기 위해 도메인(domain) 개념을 도입하여 응용 프로그램을 구성하는 대부분의 기능이 하나의 도메인 내에서 실행되도록 설계되었다.

Nemesis의 커널은 스케줄러와 NTSC(Nemesis Trusted Supervisory Code)라고 부르는 작은 코드로 구성되어 있다. NTSC는 도메인 간의 통



〈그림 3〉 Nemesis의 구조

신과 스케줄러와의 상호작용을 위해 사용된다. NTSC는 도메인 별로 존재하는 DCB(Domain Control Block)를 통해서 도메인 및 커널 스케줄러와 대화한다. 다음 〈그림 3〉은 Nemesis의 구조를 보여준다.

Nemesis는 split-level 방식의 CPU 스케줄링 기법을 사용한다. 이 방식에서는 도메인 간의 스케줄링은 커널에 의해서 낮은 수준(low-level)에서 실행되고, 도메인 내의 쓰레드간의 스케줄링은 도메인 자체에 의해서 사용자 수준(user-level)에서 실행된다. 도메인은 QoS 매개변수들을 DCB에 기재한다. 도메인 간의 스케줄링은 EDF의 변형 알고리즘을 사용하는데 마감시간은 DCB에 있는 QoS 매개변수를 통해 유추된다.

Nemesis는 단일 주소 공간(single address space) 운영체제이다. 즉 임의의 물리 메모리 영역은 항상 동일한 가상 주소에 위치한다. 단일 주소 공간의 사용으로 인해 도메인 간의 문맥 전환(context switch) 시에 캐시를 청소(flush) 해야 할 필요성이 없어지게 되고 도메인 간의 통신 시에 데이터를 복사하지 않아도 된다는 것이 Nemesis의 장점이다.

Nemesis의 장치 드라이버(device driver)는 일반 도메인과 마찬가지 형태로 구현되지만 사용자 모드와 커널 모드를 전환할 수 있는 특권을 가진 점이 다르다. 커널은 장치별로 장치 드라이버 스터브(stub)를 두어 사용자 공간에 있는 특정 장치 드라이버에게 메시지를 보낼 수 있다. 이렇게 하여 응용 프로그램을 위해 커널에서 수행되는 코드를 최소화할 수 있고 응용 프로그램간에 야기될 수 있는 QoS 간섭을 줄일 수 있다. 응용 프로그램은 사전에 설정된 연결통로를 통해 장치 드라이버에게 I/O 요청을 보낸다. 이 요청은 QoS 매개변수에 따라 드라이버내의 자원 스케줄러에 의해 서비스 된다.

4. Eclipse/BSD

벨 연구소(Bell Lab.)에서 1999년 개발된 Eclipse/BSD는 FreeBSD에 기반을 둔 운영체제로서 유연하고도 세밀한(fine-grained) QoS 지원을 목적으로 하고 있다. Eclipse/BSD의 설계 목표중 하나는 응용프로그램의 구조를 바꾸지 않은 상태에서 대규모의 서버 응용프로그램들에 대한 QoS 지원하고자 하는 것이다.

Eclipse/BSD는 예약(reservation)들에 대한 계층 개념을 지원한다. 각 계층은 스케줄러 노드와 큐 노드로 구성되어 있다. 스케줄러 노드는 바로 아래 계층에 있는 자손 노드로부터의 자원 요청에 대한 스케줄링 알고리즘을 구현한다. 스케줄러 노드는 큐를 가지거나 스케줄러 노드를 가질 수 있다. 큐 노드는 계층의 맨 마지막(리프 노드)으로서 실질적인 자원 요청이 적재되는 곳이다. 이러한 방식의 큐 노드는 다양한 자원 스케줄러를 지원한다. 가령 스케줄러는 큐 노드에 우선순위가 지정되어 있을 경우 우선순위 방식의 스케줄링을 사용할 수 있고, 큐 노드에 지분이 할당되어 있을 때는 비례지분 스케줄링 알고리즘을 사용할 수 있다.

Eclipse/BSD는 계층적인 비례지분 방식의 CPU, 디스크 및 링크 스케줄러를 구현하고 있다. 계층내의 각 노드는 해당 지분을 가진 자원 예약을 나타낸다.

Eclipse/BSD는 시스템의 자원 스케줄러에 대한 접근, 사용 및 재배치를 위해서 /reserve 파일 시스템을 구현하고 있다. /reserve 파일 시스템 내의 디렉토리는 스케줄러 계층의 해당 노드에 대응되며 따라서 자원 예약을 나타낸다. 각 자원은 가령 /reserv/cpu, /reserv/wd0(disk), /reserv/fxp0(network) 등과 같이 한 개의 디렉토리를 갖는다.

Eclipse/BSD는 객체를 예약과 연관짓기 위해 태깅(tagging) 방식을 사용한다. 예약은 객체 자체가 아니라 객체에 대한 참조에 대해 연관지어진다. 가령 파일을 오픈할 경우 해당 파일 기술자(descriptor)에 예약이 태깅된다. 소켓의 경우 해당 소켓 기술자에 예약이 태깅된다. CPU의 경우에는 각 프로세스에 예약이 태깅된다.

5. Rialto

마이크로소프트에서 1996년 개발된 연구용 운

영체제로서 멀티미디어 응용프로그램의 지원을 위하여 설계되었다. Rialto에서 사용하는 세 가지 핵심 개념은 다음과 같다.

- 액티비티(activity) : 여러 개의 쓰레드를 갖는 실행 중인 프로그램을 의미한다. 자원은 액티비티에게 할당된다.
- CPU 예약(reservation) : 액티비티에 의해 "액티비티 A를 위해 매 Y 시간 마다 X 시간을 예약해라"하는 형태로 요청된다.
- 시간 제약(time constraint) : 쓰레드가 스케줄러에게 동적으로 요청하는 것으로서 특정 코드에 대한 지정된 시작시간과 마감시간을 의미한다.

스케줄링 판단은 미리 계산된 스케줄링 그래프에 근거하여 이루어진다. CPU 예약 요청이 이루어질 때마다 이 스케줄링 그래프는 새롭게 계산된다. 스케줄링 그래프내의 각 노드는 액티비티를 나타내며 각 액티비티는 CPU 예약, 주기, CPU 시간 등의 정보를 갖고 있다. 스케줄러는 그래프를 탐색하면서 EDF 순서에 따라 적절한 액티비티를 선택한다.

응용 프로그램은 사용자가 인지하는 시스템 자원 활용도를 최대한도로 높이기 위해 자원 계획 객체(resource planner object)와의 협상을 통해서 QoS 매개변수를 정한다. Rialto 운영체제의 CPU 예약 개념은 Windows NT 운영체제에 적용되었다.

6. SMART(Scheduler for Multimedia And Real-Time applications)

스탠포드 대학의 Jason Nieh에 의해 1997년 개발된 스케줄러로서 기존의 비실시간 응용 프로그램과 함께 멀티미디어 오디오, 비디오 응용프로그램의 실행을 지원하기 위해 설계되었고 Solaris 2.5.1 상에서 구현되었다. SMART의 핵심 아이디어는 실행중인 태스크의 '중요성'(importance)과 '시급성'(urgency)을 구별하는 것이다. 중요한 태스크란 높은 우선순위를 가진 태스크를 의미한다. 시급한 태스크는 즉각적인 시간 제약이 있는 태스크를 말한다. 시급한 태스크라 할지라도 자신이 가진 지분(share) 보다 많은 자원을 요청할 경우에는 즉시 실행되지 못할 수도 있다. 중요한 태스크인 경우에도 바로 실행될 필요는 없다. 중요성은 우선순위 값과 BVFT (Biased Virtual Finishing Time) 값의 쌍으로 표시된다. BVFT는 태스크가 가진 지분에 따라 실제로 자원이 그 태스크에 할당되는 정도를 나타낸다.

중요성은 모든 태스크에 부여되지만 시급성을 실시간 태스크에만 해당된다.

SMART 스케줄러는 다음과 같은 방식으로 스케줄링 판단을 내린다.

- 가장 높은 중요성을 가진 태스크가 비실시간 태스크라면 그 태스크를 스케줄한다.
- 만약 그 태스크가 실시간 태스크라면 가장 높은 중요성의 갖는 비실시간 태스크보다 높은 중요성을 갖는 모든 실시간 태스크들로 구성되는 후보자 집합(candidate set)을 만든다.
- 후보자 집합 중에서 가장 빠른 마감시간을 갖는 태스크를 스케줄한다. 선택 시 이 태스크보다 높은 중요성을 갖는 태스크의 마감시간을 위반하지 않는지 확인해야 한다.
- 만약 어떤 태스크가 마감시간을 어기게 되면 해당 응용프로그램에게 이 사실을 통보해야 한다.

SMART에는 입장제어가 구현되지 않았기 때문에 실시간성의 보장은 시스템의 부하가 낮은 상태에서만 가능하다.

7. Windows CE

마이크로소프트에서 개발한 매우 작은 크기의 내장형 실시간 운영체제이다. 마이크로소프트의 범용 데스크탑 운영체제인 윈도우즈와 동일한 사용자 인터페이스를 주된 특징으로 하고 있어 멀티미디어 응용 프로그램의 실행이 용이한 Windows CE는 PDA, 셋탑 박스 등 많은 종류의 휴대용 기기, 내장형 시스템에 채택되어 있는

대표적인 실시간 운영체제 중의 하나이다.

Windows CE는 256개의 우선순위를 가진 우선순위기반 스케줄링을 지원하고, 우선순위 역전 문제는 우선순위 상속 방식으로 처리하고 있다. 인터럽트 처리는 두 단계로 이루어 진다. 우선 인터럽트 서비스 루틴에서 인터럽트를 받아 인터럽트 서비스 쓰레드(IST)를 깨운다. 대부분의 인터럽트 처리는 IST에서 이루어진다. Windows CE에서는 1ms 해상도의 타이머를 지원한다.

2002년 7월 30일 발표된 Windows CE의 최신 버전인 Windows CE 닷넷 4.1에서는 멀티미디어 응용 프로그램 개발을 도와주는 다양한 API와 도구 등을 제공하고 있다.

Ⅳ. 최근 동향

멀티미디어를 지원하는 실시간 운영체제의 분야에서 현재 활발히 연구가 되고 있거나 앞으로 많은 주목을 받을 분야를 살펴보면 다음과 같다.

(1) 새로운 구조의 운영체제

기존의 단일구조(monolithic) 커널이나 마이크로커널 형태의 운영체제가 아닌 새로운 형태의 운영체제에 대한 연구이다. MIT의 Exokernel과 같은 라이브러리 OS가 그 예이다. 또한 운영체제의 확장성을 높이기 위한 컴포넌트 방식의 조립형 운영체제가 많이 연구되고 있다.

(2) 다양한 자원에 대한 통합적인 관리 및 스케줄링

멀티미디어 응용 프로그램에서 요청하는 QoS를 제공하려면 CPU 뿐만 아니라 디스크, 메모리, 네트워크 등의 다른 자원들도 통합적으로 관리하고 스케줄할 수 있어야 한다. 이를 위해서는 다양한 자원들의 상호작용에 대한 이해가 필요하고 시스템 내의 자원에 대한 통일적인 모델이 필요하다. 이러한 시도는 스토니 부룩 소재 뉴욕 주립대의 IRS(Integrated Real-Time Resource Scheduling)나 카네기멜론 대학의 MRMD (Multiple Resource Multiple QoS Dimensions) 모델, 일리노이 대학의 Multi-Resource Reservation Framework 등에서 이루어지고 있다.

(3) 예측불가능성(unpredictability)에 대한 스케줄링 및 측정지표

기존의 실시간 시스템에서는 모든 소프트웨어의 실행시간이 예측 가능했다. 그러나 현재의 하드웨어와 소프트웨어 개발 추세로 볼 때 이러한 실행시간 예측은 점점 불가능해지고 있다. 가령 여러 단계의 캐싱, 멀티프로세서의 보편화, 에너지 절약을 위한 프로세서 속도의 변동, 소프트웨어의 다 계층화, 최적화 등이 예측가능성을 어렵게 하는 요인들이다. 이러한 상황에서는 응용 프로그램과 시스템이 응용 프로그램의 진행상황에 적절히 대응할 수 있는 보다 적응성이 뛰어난 스케줄링이 요구된다. 또한 예측 불가능성과 응용 프로그램의 QoS 사이의 복잡한 관계에 대해 평가하고 대처할 수 있는 수단을 제공하기 위해 멀티미디어의 연성 실시간성에 대한 새로운 측정지표가 만들어져야 한다.

(4) 응용 프로그램의 다양한 스케줄링 요청을 지원하는 스케줄링 알고리즘

멀티미디어 시스템은 최소한 세가지 종류의 스케줄링을 지원해야 한다. 첫째, 점진적 성능저하를 허락하지 않는 실시간 응용프로그램에 대한 성능을 보장할 수 있는 스케줄링, 둘째, 점진적 성능저하가 가능한 응용 프로그램에 대한 최선을 다한(best-effort) 스케줄링, 셋째, 비실시간 응용 프로그램들에 대한 시분할 스케줄링을 지원할 수 있어야 한다.

(5) 효과적인 프로그래밍 모델의 제공

여기에는 프로그래머가 멀티미디어 시스템에서 실행되는 실시간 프로그램의 개발을 쉽고 빠르게 할 수 있는 다양한 요소들이 포함된다. 프로그래머들이 자원에 대한 요청을 지정할 수 있는 프로

그래밍 인터페이스의 작성 및 표준화는 많은 연구가 필요한 부분이다. 현재 나와있는 POSIX나 Win32 인터페이스는 실시간 프로그래머의 요구를 만족시켜 주기에는 크게 부족하다. 또한 Windows CE의 개발 도구 및 환경에서 볼 수 있는 것처럼 멀티미디어 시스템 개발을 위한 미들웨어의 중요성이 부각되고 있다.

V. 결 론

지금까지 멀티미디어 시스템을 지원하는 운영체제의 현황에 대해 살펴보았다. 이를 위해서 먼저 다양한 멀티미디어 데이터의 특성에 대해 설명하고 멀티미디어 시스템에서 요구하는 실시간성은 마감시간을 엄격히 지켜야 하는 전통적인 의미의 실시간성인 경성 실시간성과는 다른 연성 실시간성이기 때문에 운영체제의 설계에 있어서도 이를 고려해야 함을 지적했다.

멀티미디어 운영체제의 특징을 자원관리 및 QoS 측면과 CPU 스케줄링의 관점에서 설명하였다. 이는 현재까지 멀티미디어 운영체제에 대한 많은 연구가 이 부분에 대해 이루어 졌기 때문이다. 그러나 운영체제를 이루는 다른 서브 시스템들에 대한 고려도 아울러 필요하다는 것을 지적하고 싶다. 가령 파일 시스템, 디스크 관리, 메모리 관리, I/O 장치에 대한 관리 등도 역시 중요한 부분이며 그 동안 많은 연구가 이루어져 왔다. 본 고에서는 이 부분이 기존의 범용 운영체제에 대한 연구와 중복되는 부분이 많다고 보아 QoS 측면에 대해서만 언급하는 것으로 그쳤는데 관심 있는 분들은 참고문헌에 소개된 자료를 참조하시기 바란다.

또한 그 동안 개발되어온 대표적인 멀티미디어 운영체제 및 스케줄러에 대해 소개하였고 현재의 연구동향 및 앞으로 연구가 이루어져야 할 분야에 대해서도 간략히 소개하였다. 앞에서도 언급하였지만 인터넷과 Post PC 산업의 급격한 성장으로 인해 멀티미디어 시스템을 위한 운영체제는 현재 연구, 개발이 매우 활발히 이루어지고 있는 분야이다. 본 고가 이 분야의 연구에 대한 관심을 촉구하는데 도움이 되길 바라고, 아울러 시스템 소프트웨어 개발 인력이 취약한 우리나라의 현실에서는 이 분야의 인력양성이 매우 중요한 과제임을 강조하고 싶다.

참 고 문 헌

[1] Tatjana M. Burkow, "Operating System Support for Distributed Multimedia Applications; A Survey of Current Research," Technical Report(Pegasus Paper 94-8), Faculty of Computer Science, University of Twente, 1994.

[2] Ralf Steinmetz, Klara Nahrstedt, *Multimedia: Computing, Communications and Applications*, Prentice Hall, pp.225-310(Ch. 9: Multimedia Operating Systems), 1995.

[3] Ralf Steinmetz, "Analyzing the Multimedia Operating System," *IEEE Multimedia*, Vol. 2, No. 1, pp.68-84, 1995.

[4] Christina Aurrecoechea, Andrew T. Campbell, Linda Hauw, "A Survey of QoS Architectures," *Multimedia Systems*, Vol. 6, No. 3, pp.138-151, 1998.

[5] T. Plagemann, V. Goebel, P. Halvorsen, O. Anshus, "Operating System Support for Multimedia Systems," *The Computer Communications Journal*, Vol. 23, No. 3, pp.267-289, Feb. 2000.

[6] J. Regehr, M. B. Jones, J. A. Stankovic. "Operating System Support for Multimedia: The Programming Model Matters," Technical Report MSR-TR-2000-89, Microsoft Research, Sept. 2000.

[7] Kartik Gopalan, "Real-Time Support in General Purpose Operating System,"

ECSL Technical Report TR92, State University of New York at Stony Brook, 2001.

[8] 김선자, 김홍남, 김채규, "인터넷 정보가전용 RTOS 기술 현황," *한국정보과학회지*, Vol. 19, No. 4, pp.57-64, 2001. 4.

[9] L. Fernando Friedrich, J. Stankovic, M. Humphrey, M. Marley, J. Haskins Jr., "A Survey of Configurable, Component-Based Operating Systems for Embedded Applications," *IEEE Micro*, Vol. 21, No. 3, pp.54-68, May-Jun. 2001.

저 자 소 개

孔 基 錫

1984년 2월 서울대학교 제어계측공학과 졸업 (학사), 1986년 2월 서울대학교 전자계산기공학과 대학원 졸업 (석사), 1999년 2월 KAIST 전산학과 박사과정 졸업 (박사), 1986년 1월~1989년 10월 : 삼성전자, 1989년 11월~1992년 6월 삼보컴퓨터, 1992년 7월~1994년 4월 : EOS Technologies, Inc., 1999년 4월~2000년 8월 : 한국전자통신연구원 컴퓨터소프트웨어연구소, 2000년 9월~2001년 7월 IMDB, Inc., 2001년 9월~현재 : 한국산업기술대학교 컴퓨터공학과 조교수, 〈주관심 분야 : 운영체제, 실시간시스템, 분산시스템, 컴퓨터 네트워크〉