실시간을 지원하는 리눅스 인터페이스용 미들웨어 설계 및 구현

김명선°, 유인선, 최 훈
충남대학교
{mskim05°, isyou, hc }@cnu.ac.kr

Design and Implementation of the Linux Interface Middleware to Support Time Determinism

Myoungsun Kim° , Inseon You Hoon Choi
Mobile Distributed Computing Lab, Department of Computer Engineering,
Chungnam National University, KOREA

요 약

응용프로그램의 개발 비용이 높아지고 개발 기간이 점차 단축되면서, 응용프로그램의 재사용에 대한 연구가 다양하게 진행되고 있다. 특히 실시간 응용프로그램들은 플랫폼 종속성에 의해 다른 운영체제에서 새로 개발해야하므로 개발 비용과 시간상의 문제가 심각하다. 이러한 문제점을 극복하기 위해 OS Changer, Xenomai, Leagcy2Linux와 같은 다양한 인터페이스 미들웨어들이 개발되었으나 미들웨어에서 제공되는 API가 특정 운영체제에 종속되거나, API의 확장성을 제공하지 못한다는 문제점이 존재한다. 본 연구에서는 기존 미들웨어들의 문제점을 극복하고, 더 나아가 API를 동적으로 재구성할 수 있는 실시간 운영체제 인터페이스용 미들웨어를 구현하였다.

1. 서 론

응용프로그램은 과거에나 지금이나 마찬가지로 소프트웨어의 생산성과 품질을 주된 이슈로 하고 있지만, 현시점에서는 적시성과 유연성을 추가적으로 요구하고 있다. 최근 소프트웨어 개발에 대한 연구결과들을 보면 기존의 프로그램을 최대한 재사용하거나 이미 검증된 컴포넌트를 재구성하여 고품질의 시스템을 적시에 구축하는데 초점을 맞추고 있다[1].

이러한 관점은 일반적인 범용 응용프로그램뿐만아니라 실시간 응용프로그램에도 적용된다. 특히 실시간 응용프로그램은 플랫폼 종속성으로 인해 동일한 기능의 응용프로그램을 중복 개발함으로써 개발 및 유지 관리 비용이 많이 들어가게 된다. 중복 개발의 문제점을 극복하기 위해 응용프로그램의 운영체제에 대한 의존성을 줄임으로써, 운영체제가 변경되더라도 응용프로그램이 정상적으로 동작할 수 있게 지원할 수 있는 미들웨어가 필요하게 되었다.

이런 미들웨어로는 OS Changer, Xenomai, Leagcy2Linux 등이 있으나, 미들웨어에서 제공하는 API가 특정 운영체제에 종속적이고, 다양한 운영체제에 대한 이식성을 제공하지 못한다는 점과 API의 확장성을 제공하지 못한다는 문제점 등을 가지고 있다[2][3][4].

본 연구에서는 기존 미들웨어의 문제점을 보완하고 다양한 운영체제에서의 이식성을 제공할 수 있는 실시간 운영체제 인터페이스용 미들웨어를 설계하였으며, 본 논문에서는 실시간 운영체제 인터페이스용 미들웨어에서 지원하는 다양한 운영체제 중 리눅스에 초점을 맞추어 리눅스 운영체제를 지원하는 미들웨어의 설계 및 구현에 대해 기술한다[5].

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 관해 기술하고, 3장에서는 본 연구에서 개발한 미들웨어의 내부구조 및 동작 과정을 기술한다. 4장에서는 미들웨어내의 각 컴포넌트들의 기능 테스트에 대해 기술하고, 5장에서는 결론 및 향후 연구 방향에 대해 제시한다.

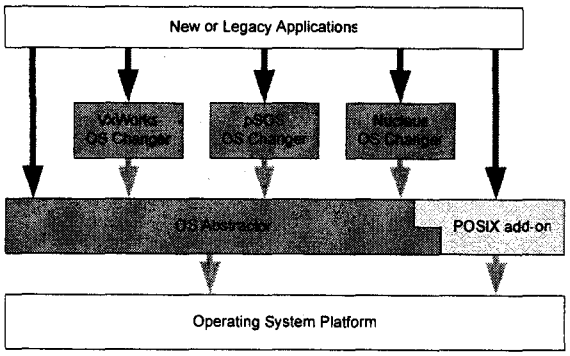
2. 관련연구

OS Changer는 여러 운영체제 상에서 동작하기 위해 통합된 호환성(compatibility)과 다양한 추상화(abstraction) 솔루

* 본 논문은 국방과학연구소의 대학 기초 과제인 " 실시간 운영체제 인터페이스용 미들웨어 연구" 의 수행 결과임.

션을 제공한다. OS Changer는 개발자들이 기존 임베디드 응용프로그램을 새로운 운영체제에서 쉽게 이식할 수 있게 한다. 임베디드 응용프로그램이 특정 운영체제에서 동작할 때, OS Changer는 해당 운영체제의 가상 API를 이용하여 응용프로그램과 운영체제를 연결한다.

[그림 1]에서는 OS Changer의 구조를 나타낸다. OS Changer는 응용프로그램의 이식성을 제공하는 OS Changer 인터페이스와 새로운 공통된 API 제공하는 OS Abstractor 인터페이스를 제공한다[2].



[그림 1] OS Changer 구조

Xenomai는 리눅스를 기반으로 기존 실시간 운영체제의 API(Application Programming Interface)를 에뮬레이트(emulate)하여 응용프로그램의 중복개발을 줄이고, 기존 실시간 운영체제 환경에 익숙한 개발자들에게 유사한 개발환경을 조성하여 리눅스 개발 환경에 쉽게 적응하는 것을 목적으로 한다[3].

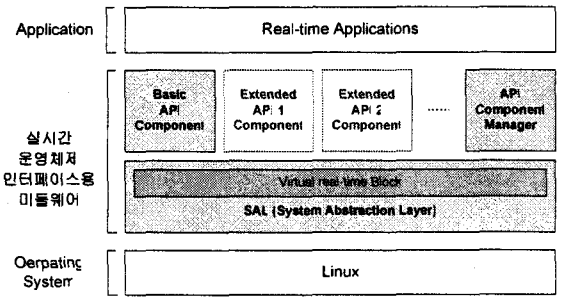
Legacy2linux는 오픈 소스를 지원하며, 기존 실시간 운영체제 커널을 에뮬레이트하여 기존 실시간 운영체제에서 개발된 응용프로그램을 리눅스 환경에서 재사용하기 위해 개발되었다[4]. [표 1]에서는 각 미들웨어를 비교하였다.

[표 1] 실시간 미들웨어간의 비교

	OS Changer	Xenomai	Legacy2linux
표준 API 제공	O	O	X
실시간성 보장	O	O	X
시스템 추상화	O	X	X
API 확장성	X	X	X
API 동적 재구성	X	X	X

3. 리눅스 인터페이스용 미들웨어 설계 및 구현

기존 실시간 응용프로그램을 위한 미들웨어의 API는 특정 운영체제에 종속적이다. 응용프로그램에게 제공되는 API는 특정 운영체제에 종속되지 않고, 운영체제에 독립적으로 제공되어야 한다. 운영체제에 독립적인 API를 제공하기 위해서는 실시간 운영체제에 독립적인 표준 API를 정의하여 하부 운영체제로의 정합을 미들웨어 계층에서 제공해야 한다. 미들웨어 역시 운영체제에 종속되지 않아야 하며, 재사용성을 보장해야 한다. 또한 실시간 응용프로그램을 지원하기 위해서는 범용 운영체제와는 달리 실시간 응용프로그램이 요청하는 이벤트에 대해 특정 시간 안에 수행을 완료해야 한다. 실시간 응용프로그램을 지원하기 위해 시간 결정성을 보장하는 안정된 스케줄링 기능을 갖추어야 하며, 미들웨어 계층 추가로 인한 오버헤드(overhead)를 최소화 해야 한다[6][7]. 이러한 요구사항을 만족하기 위해 설계된 미들웨어의 구조는 [그림 2]와 같다.



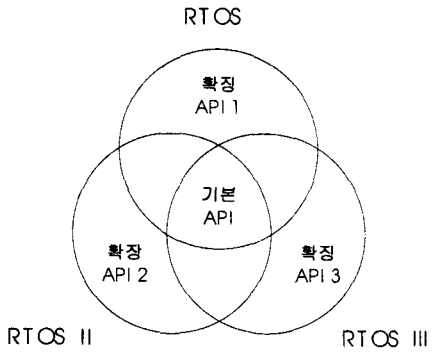
[그림 2] 리눅스 인터페이스용 미들웨어의 구조

운영체제에 독립적인 기본 API(Basic API Component)를 정의하고, 정의한 API 이외에 운영체제에 따라 확장되는 확장 API(Extended API Component)를 제공하여 API의 확장성을 제공하였다. 또한 API의 동적 재구성을 지원하기 위한 API 컴포넌트 관리자(API Component Manager)를 제공하였으며, 다양한 운영체제에서 미들웨어를 제공하기 위해 운영체제별로 SAL(System Abstraction Layer) 계층을 두어 운영체제를 추상화하였다[5].

본 논문에서는 리눅스를 지원하는 리눅스 SAL에 초점을 맞추어, 실시간성을 지원하는 리눅스 인터페이스용 미들웨어에 대해 기술하였다.

3.1 기본 API

기본 API는 보편적으로 이용되는 실시간 운영체제인 VxWorks, pSOS, UbiFOS, QNX에서 공통적으로 존재하는 API 및 실시간 응용 프로그램을 지원하기 위해 추가하는 것이 바람직한 API를 기본 API로 선정하였다.



[그림 3] 기본 API 및 확장 API 개념도

기본 API는 태스크 관리 API 23개, 메모리 관리 API 12개, 세마포 관리 API 6개, 메시지큐 관리 API 4개, 메시지 포트 관리 API 4개, 타이머 관리 API 6개, 와치독(Watchdog) 타이머 관리 API 4개, 시그널 관리 API 5개, 인터럽트 관리 API 5개로 구성되어 있으며, 그 중에서 태스크 관리 API는 [표 2]와 같은 API들로 구성된다.

[표 2] Task Management API

표준 API	함수 기능
TaskInit()	Task의 구조체를 초기화한다.
TaskSpawn()	Task를 생성한다.
TaskDelete()	생성된 Task를 삭제한다.
TaskActivate()	초기화된 Task를 활성화시킨다.
TaskResume()	Task를 ready 상태로 변경시킨다.
TaskSuspend()	Task의 Suspend 상태로 변경시킨다.
TaskDelay()	Task를 delay 시킨다.
TaskGetPriority()	Task의 현재 우선순위 값을 가져온다.
TaskSetPriority()	Task의 우선순위 값을 변경시킨다.
TaskLock()	Task를 lock 시킨다.
TaskUnlock()	Task를 unlock 시킨다.

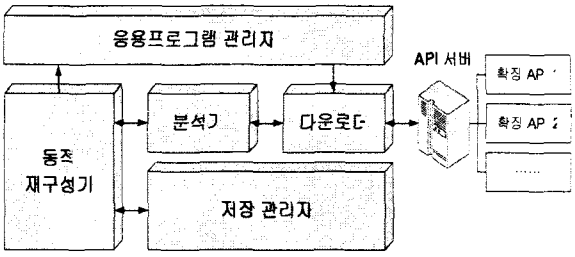
태스크 관리 API 에서는 태스크를 생성 후 즉시 스케줄링되는 방법과 태스크 생성과 스케줄링의 시작이 분리되는 방법 모두를 지원하며, 태스크 관리 모듈에서는 태스크를 생성, 삭제, 제어, 스케줄링할 수 있다. 이외 다른 기본 API들도 태스크 관리 API와 유사한 형태로 구성된다.

3.2 확장 API

확장 API는 각 실시간 응용프로그램의 특성에 따라 확장될 수 있는 API 집합으로서 동적으로 구성된다. 확장 API는 API 관리자에 의해 관리되며, 실시간 응용프로그램에서 특정 API를 필요로 하면, API 관리자에 의해 추가될 수 있다.

3.3 API 컴포넌트 관리자

API 컴포넌트 관리자는 응용프로그램의 실행 시 필요한 API 컴포넌트를 동적으로 추가, 삭제, 갱신 및 저장하는 기능을 제공한다. API 컴포넌트 관리자의 내부 구조는 [그림 4]와 같다.

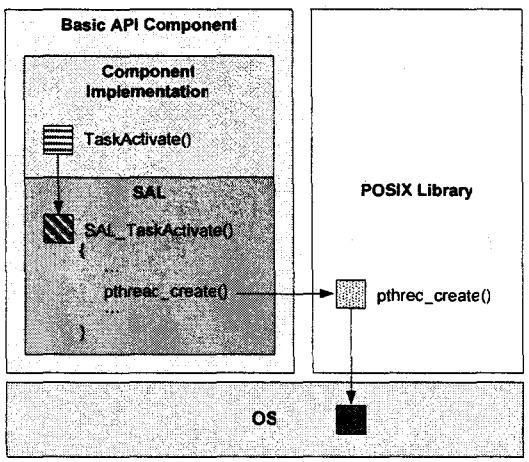


[그림 4] API 컴포넌트 관리자 구조

[그림 4]에서 응용프로그램 관리자(Application Manager)는 응용프로그램의 설치, 삭제, 실행 및 정지 기능을 제공하는 모듈이다. 동적 재구성기(Dynamic Reconfigurator)는 응용프로그램에서 요청하는 API 컴포넌트에 대해 설치 여부를 검사하여, 설치되어 있고 메모리에 로드되어 있지 않으면 해당 API 컴포넌트를 메모리에 로드하고, 사용 후 더 이상 사용하지 않는 API 컴포넌트는 메모리에서 해제 또는 삭제한다. 분석기(Analyer)는 API 컴포넌트간의 참조 여부 및 버전 정보를 관리한다. 다운로더는 설치되어 있지 않는 API 컴포넌트를 API 서버로부터 다운로드한다. 저장 관리자(Storage Managerment)는 새로운 API 컴포넌트를 시스템에 설치할 때, 기존 API 컴포넌트에 대한 백업 기능을 제공한다[8][9].

3.4 SAL

SAL은 기본 API 및 확장 API에서 정의된 API에 대한 시스템 의존적인 부분이다. 리눅스를 위한 SAL에서는 리눅스 시스템에서 기본 API를 지원하기 위해 기본 API인 태스크, 메모리, 세마포, 메시지큐, 메시지포트, 타이머, 와치독 타이머, 시그널, 인터럽트 등의 내부 구조를 구현하였다. 기본 API는 SAL 계층을 거쳐 리눅스의 API를 호출하며, 이 과정에서 API의 반환 값이나 매개변수 같은 리눅스에 의존적인 부분이 추상화된다. 그러나 대상 API가 리눅스에 존재하지 않을 경우, 대상 API를 리눅스에서도 제공할 수 있도록 하기 위해 SAL 계층에서 POSIX 라이브러리를 이용하여 구현하였다. 리눅스 SAL에서는 리눅스가 실시간 운영체제가 아니기 때문에 실시간 응용프로그램을 지원하기 위해 기본 API의 대부분을 구현하였다.



[그림 5] SAL의 내부 API 호출 구조

4. 기능 테스트

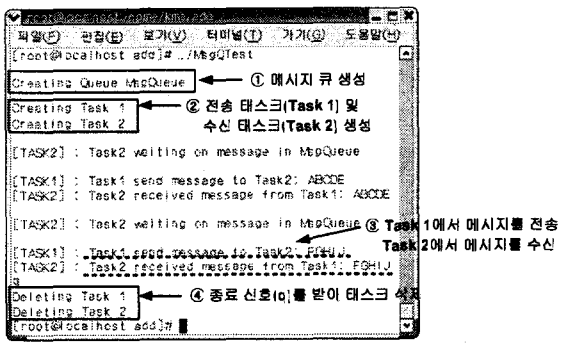
리눅스 인터페이스용 미들웨어의 개발환경 및 기능 테스트를 위한 테스트 환경은 [표 3]과 같다.

[표 3] 개발 및 테스트 환경

구분	내 용
운영체제	linux-2.4.20-18
개발 언어	C 언어
CPU	AMD Athlon XP 1500+1.30GHz
RAM	1.00GB

4.1 기본 API 기능 테스트

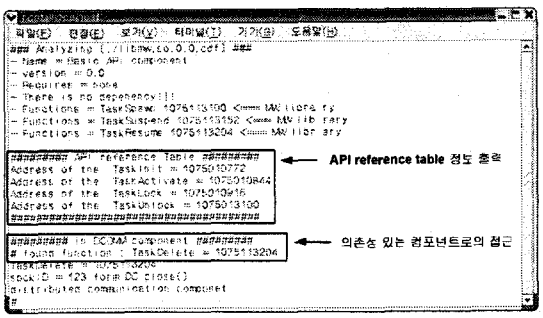
실시간성을 지원하는 리눅스 인터페이스용 미들웨어의 기능테스트는 기본 API에 대한 모든 테스트를 시행하였으며, [그림 6]은 태스크 및 메시지 큐에 대한 기능 테스트이다. 테스트 프로그램은 메시지 전송을 위한 큐를 생성한다(①). 메시지 전송을 위한 태스크(Task1)와 수신을 위한 태스크(Task2)를 생성한다(②). Task2는 생성 후에 생성된 메시지 큐에서 메시지가 전송되길 기다린다. Task1에서 메시지를 전송하면, Task2는 메시지 큐를 통해 메시지를 수신한다(③). 테스트 프로그램은 프로그램 종료 신호(q)를 받고 생성되었던 태스크들을 삭제한다(④).



[그림 6] Task 및 Message Queue 테스트

4.2 API 컴포넌트 관리자 기능 테스트

기본 API 컴포넌트를 메모리에 로드하여 테스트 프로그램이 정상적으로 동작하였고, 테스트 프로그램 실행 중에 기본 API 컴포넌트를 메모리에서 삭제한 후 API를 호출 하려고 하면 컴포넌트가 다시 메모리에 로드되어 계속적으로 서비스를 제공하였다.



[그림 7] 컴포넌트간의 참조 여부 확인

또한 API참조 테이블(API Reference Table)을 이용한 컴포넌트 간의 참조 여부 확인 및 의존관계에 있는 컴포넌트의 동적인 추가는 [그림 7]과 같이 메모리에 로드된 컴포넌트들의 API 참조 정보와 의존성 정보를 검사함으로써 확인할 수 있었다.

5. 결론

본 논문에서는 실시간 응용프로그램의 재사용성을 위해 실시간 응용프로그램을 지원하는 리눅스 인터페이스용 미들웨어를 설계 및 구현하였다. 구현된 리눅스 인터페이스용 미들웨어는 표준 API를 제공하고, API의 확장성을 제공하기 위해 확장 API을 제공한다. 또한 API 컴포넌트의 관리 및 동적 재구성을 위해 API 컴포넌트 관리자를 제공하여 기본 API 및 확장 API에 대한 설치 및 삭제를 기능을 제공하여 플랫폼의 확장 및 축소를 자유롭게 제공한다. SAL 계층을 두어 비실시간 운영체제인 리눅스에 실시간 결정성을 제공하며, 하부 운영체제에 대한 인터페이스 미들웨의 의존성 부분을 최소화하였다.

인터페이스 미들웨어가 추가됨으로써 응용프로그램에 더해지는 시간적 오버헤드는 필연적이나 대략적인 오버헤드 크기 분석을 수행한 결과 오버헤드로 인한 시간지연이 실시간성 요구사항을 만족할 만한 정도로 파악되었다[5].

API 별로 상세한 실시간 성능 분석은 향후 연구로 수행할 예정이다. 아울러 현재 구현된 리눅스 인터페이스용 미들웨어에 관한 성능 개선 및 더 강력한 실시간성 지원을 위한 알고리즘에 대한 연구를 진행할 예정이다. 또한 구현된 미들웨어에 관한 기존 미들웨어와의 성능 차이 및 미들웨어가 존재하지 않을 때와 본 미들웨어가 존재할 때와의 성능 차이를 분석할 예정이다.

6. 참고문헌

[1] 강현미, 박만곤, 장화식, " 소프트웨어 재사용에 따른 생산성 향상의 분석" , 한국정보시스템학회 97년도 추계 학술발표회, Vol.0, No.0, pp.379-388, 1997.

[2] http://www.mapusoft.com/

[3] http://snail.fsffrance.org/www.xenomai.org/

[4] http://legacy2linux.sourceforge.net/

[5] 유인선, 최훈, 이철훈, 신진화, 김종룡, "실시간 운영체제 인터페이스용 미들웨어 구조 설계", 제13차 유도무기학술대회 논문집, pp. 360-363, 2005. 10.

[6] 반석호, " 리눅스를 이용한 실시간 임베디드 시스템 설계에 관한 연구" , 서울시립대 산업대학원 석사학위논문, 2003.

[7] 김정국, " 실시간 시스템을 위한 미들웨어" , 정보처리학회지 제 3권, 제5호, pp. 30-37, 2001.

[8] 이수원, 유인선, 유용덕, 최 훈, " 컴포넌트 기반 소프트웨어 플랫폼을 위한 컴포넌트 관리자 설계 및 구현", 한국정보과학회 KCC2006, 논문집(D), 제33권 제1호, pp. 199-201, 2006. 6.

[9] 박충범, 유용덕, 최훈, "임베디드시스템용 범용플랫폼을 위한 동적 재구성 관리자 설계", 한국정보과학회 KCC2005, 논문집(A), 제32권 제1호, pp. 667-669, 2005. 7.