임베디드 시스템 인터페이스용 미들웨어 설계 및 성능분석

(Design and Performance Analysis of the Interface Middleware for Embedded Systems)

김 명 선 † 이 수 원 † 이 철 훈 †† 최 훈 ††† 조 길 석 ††††
(Myoungsun Kim) (Su Won Lee) (Cheol-Hoon Lee) (Hoon Choi) (Kilseok Cho)

요 약 임베디드 응용프로그램 개발의 적시성(time-to-market)을 위해, 기존 임베디드 응용프로그램의 재사용 기법과 각기 다른 플랫폼에 인터페이스 시켜주는 인터페이스용 미들웨어에 대한 연구가 활발히 진행되고 있다. 인터페이스용 미들웨어 연구로서 기존의 MT 프로젝트, Xenomai, Legacy2linux 기술들은 미들웨어에서 제공되는 API가 특정 운영체제에 종속되거나 API의 확장성을 제공하지 못한다는 한계점이 있다. 본 논문에서 제안하는 임베디드 시스템 인터페이스용 미들웨어는 운영체제에 대한 종속성을 제거하여 다양한 운영체제를 지원한다. 또한, 미들웨어의 확장 및 동적 재구성이 가능한 컴포넌트 기반 구조로 설계되어, 자원 제한적인 임베디드 시스템에서 응용프로그램을 효율적으로 실행시킬 수 있다. 본 논문에서 제안하는 미들웨어의 추가에 따른 응용프로그램의 실행 지연은 미들웨어 API의 실행 시 최소 0.3μsec에서 최대 5μsec 정도이며, 이는 응용프로그램의 실행 성능에 큰 영향을 주지 않을 것으로 판단된다.

키워드 : 인터페이스용 미들웨어, 임베디드 시스템, 실시간 시스템, OS Changer

Abstract As various types of embedded devices are widely used, a technology that supports reuse of applications on multiple platforms is needed in order for time-to-market development of the applications. The interface middleware is one of such technology and it hides platform dependency from application programmers. Existing interface middleware such as the MT project, Xenomai and Legacy2linux have limitation in that the APIs provided by each of these middleware are fixed to a specific operating system, and the middleware does not provide dynamic expansion of its API set. In this paper, we propose a middleware which hides operating system dependencies and enables porting of applications on various operating systems. In addition, the middleware has scalable structure so that it is suitable for resource-limited embedded systems. The overhead of the middleware, i.e., the time delay occurred by the middleware is between 0.3*u*sec and 5*u*sec in most cases. We believe that the amount of overhead is reasonable and does not hurt the performance of applications.

Key words : Interface middleware, Embedded System, Real-Time system, OS Changer

† 비 회 원 : 충남대학교 컴퓨터공학과
mskim05@cnu.ac.kr
swlee@cnu.ac.kr
†† 비 회 원 : 충남대학교 컴퓨터공학과 교수
clee@cnu.ac.kr
††† 종신회원 : 충남대학교 컴퓨터공학과 교수
hc@cnu.ac.kr
†††† 정 회 원 : 국방과학연구소 3-1-8 책임연구원
kscho@add.re.kr
논문접수 : 2007년 5월 16일
심사완료 : 2007년 12월 4일

정보과학회논문지 : 컴퓨팅의 실제 및 레터 제14권 제1호(2008.2)

1. 서 론

산업용 장비나 무기체계 장비가 전자화되면서 많은 임베디드 기기가 이용되었고 그에 따른 고객의 요구를 충족시키기 위한 다양한 응용프로그램이 개발되고 있다[1,2]. 그러나 대부분의 임베디드 응용프로그램은 동일한 기능을 수행하더라도 탑재되는 시스템에 따라 상호 호환되지 못하며, 이에 따라 같은 기능의 응용프로그램을 타겟(target) 운영체제에 맞춰 다시 개발함으로써 불필요한 비용 및 시간이 소요된다[3,4]. 이런 불필요한 소모를 줄이고, 임베디드 응용프로그램의 적시성을 위해 타겟 운영체제가 변경되더라도 응용프로그램이 변경없이

동작할 수 있도록 지원할 수 있는 미들웨어가 필요하다[5,6].

이런 미들웨어에 대한 연구는 지금까지 MT(MapuSoft Technology) 프로젝트[7], Xenomai[8], Legacy2Linux[9] 등 다양하게 진행되고 있다. 그러나, 기존에 개발된 미들웨어는 제공되는 API가 특정 운영체제에 종속적이라는 문제점과 타겟 운영체제로 특정 운영체제만 지원하기 때문에 타겟 운영체제가 바뀌면 미들웨어를 대폭적으로 수정해야 한다는 문제점을 가지고 있다. 따라서 운영체제에 독립적이기 위해서는 특정 운영체제에 종속적이지 않은 API 인터페이스를 가져야 하며, 다양한 운영체제에 대한 이식성을 제공해야 한다.

본 논문에서는 임베디드 시스템에 적합한 임베디드 시스템 인터페이스용 미들웨어(IMES: Interface Middleware for Embedded System)를 소개한다. IMES는 시스템을 추상화하여 하나의 응용프로그램이 다양한 임베디드 운영체제에서 실행할 수 있게 지원하고, 자원 제약적인 임베디드 시스템 환경에서 자원을 최적화하여 사용할 수 있도록 서비스별 API 컴포넌트를 네트워크를 이용하여 다운로드한 후, 시스템을 동적으로 재구성할 수 있는 API 관리자를 지원한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 관해 기술하고, 3장에서는 본 연구에서 개발한 미들웨어의 내부 구조 및 동작 과정을 기술한다. 4장에서는 미들웨어의 기능 테스트 및 성능 평가 결과에 대해 기술하고, 5장에서는 결론 및 향후 연구 방향에 대해 제시한다.

2. 관련연구

MT 프로젝트는 한번 개발된 응용프로그램을 새로운 운영체제에 쉽게 이식할 수 있게 한다[10]. MT 프로젝트에서는 응용프로그램의 이식성을 제공하는 OS Changer와 독립 API(Independent API)를 가지는 OS Abstractor 인터페이스를 제공한다(그림 1). OS Changer는 각기 다른 운영체제에서 사용되는 API를 OS Abstractor에서 제공하는 API로 변경해주는 계층(layer)이다. OS Changer는 지원하는 상위 운영체제마다 하나씩 존재하며, 현재 VxWorks OS Changer, pSOS OS Changer, Nucleus PLUS OS Changer를 제공하고 있다. OS Abstractor는 MT 프로젝트만의 독립 API를 지원하고 하위에 존재하는 타겟 시스템을 추상화하는 레이어이다. 즉 특정 OS에 무관한 새로운 API를 정의했으며 응용프로그램에서 이 독립 API를 실행하면, 해당 독립 API와 대응되는 타겟 운영체제의 API를 호출하거나 같은 기능을 수행하여 응용프로그램과 운영체제를 연결해주는 역할을 수행한다[11]. MT 프로젝트에서 지원하는 타겟 운영체제로는 리눅스(Linux), VxWorks, pSOS, Nucleus PLUS 등이 있다. 그림 1은 MT의 구조를 나타낸다.

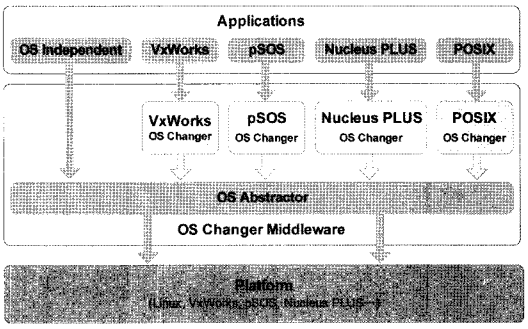


그림 1 MT 구조

Xenomai는 VxWorks, pSOS, VRTX와 같은 실시간 운영체제의 API를 리눅스상에서 에뮬레이트(emulate)한다. 응용프로그램의 중복 개발을 줄이고, 기존 실시간 운영체제 환경에 익숙한 개발자들에게 유사한 개발환경을 제공함으로써 GNU/리눅스 개발환경에 쉽게 적응하는 것을 목적으로 한다. 각각의 실시간 운영체제에서 실행되던 응용프로그램을 리눅스에서 실행시키기 위해, 각 실시간 운영체제의 API를 제공한다. 또한 리눅스에서 제공하지 않는 실시간성 및 각 실시간 운영체제와의 시스템 차이점을 보완하기 위해 Real-time Nucleus 모듈 및 HAL(Hardware Abstraction Layer), Adeos 모듈을 제공한다[12]. 그림 2는 Xenomai의 구조를 보여준다.

Legacy2linux는 임베디드 시스템에서 비용, 확장성, 신뢰성 등에서 강점을 보이는 리눅스를 기존 실시간 운영체제 대신 사용할 수 있게 하는데 초점을 두고 있다. 기존 실시간 운영체제 시스템을 지원하던 회사들이 리눅스 기반으로 시스템을 변경할 시에 기존 실시간 운영체제에서 개발된 응용프로그램을 새로운 리눅스 기반

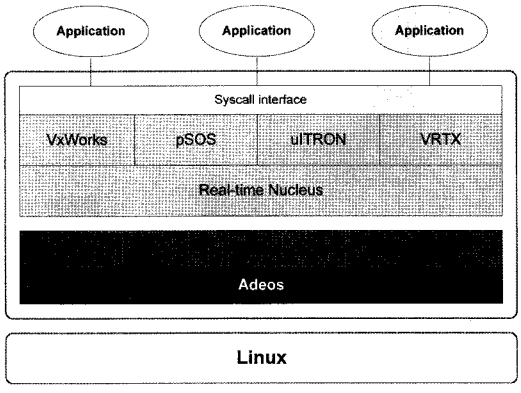


그림 2 Xenomai 구조

시스템에서도 실행될 수 있도록 하는 것이다. Montavista Software사의 후원으로 진행되고 있으며, 오픈 소스 프로젝트로 진행되고 있다[9]. VxWorks, pSOS 등의 실시간 운영체제 커널을 에뮬레이트하여 기존 실시간 운영체제에서 개발된 응용프로그램을 리눅스 환경에서 재사용할 수 있다.

Xenomai와 Legacy2linux는 여러 가지 운영체제의 응용프로그램을 리눅스에 포팅하기 위해 만들어진 인터페이스용 미들웨어로서 리눅스 외의 운영체제에서는 실행되지 않는다. 그러나 MT 프로젝트와 IMES는 시스템 추상화 계층을 두어 다양한 타겟 운영체제를 지원할 수 있다.

3. IMES 설계 및 구현

인터페이스용 미들웨어는 응용프로그램의 운영체제 의존성을 줄여, 플랫폼(platform)이 변경되더라도 정상적인 동작을 지원해야 한다. 이를 위해 운영체제와 응용프로그램 사이의 이질적인(heterogeneous) 실행 환경을 추상화시키고 서로 다른 기능을 정합(adaptation)시켜주어야 한다. 즉, 사용자에게 시스템 투명성을 제공하는 역할을 제공해야 한다. 그림 3은 본 논문에서 제안하는 IMES의 구조이다.

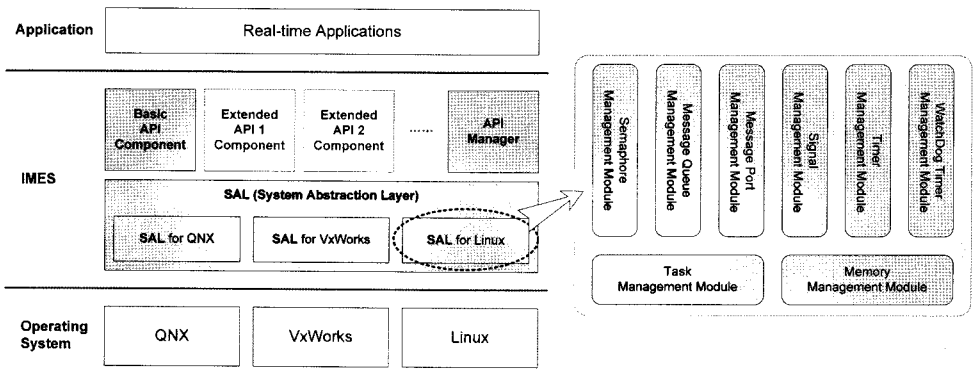
임베디드 시스템은 특정한 목적을 위해 개발되었으며, 임베디드 시스템에 탑재되는 응용프로그램 또한 특정 기능의 수행을 목적으로 개발되었다. 이러한 임베디드 시스템의 특성에 따라 본 논문에서 제안하는 IMES에서는 API를 기본 API(Basic API)와 확장 API(Extended API)로 분류하여 임베디드 시스템의 특성을 고려하였다.

- 기본 API : 다양한 임베디드 시스템에서 공통적으로 제공하고 있는 기본적인 API로서, 임베디드 응용프로그램이 동작하기 위한 필수불가결한 API들로 구성된다. 또한, 기본 API를 보편적으로 많이 사용되는 VxWorks와 유사하게 정의함으로써 새로운 API에 대한 진입장벽을 완화하였다.
- 확장 API : 기본 API 이외에 추가적인 기능을 제공하는 API로서, 서비스 별로 정의된 확장 API 컴포넌트를 이용하여 임베디드 시스템을 추가 및 확장할 수 있다.

제안하는 IMES는 기본 API, 확장 API 외에 API 관리자(API Manager)와 시스템 추상화 계층(SAL: System Abstraction Layer)으로 구성되어 있다[13,14].

- API 관리자 : IMES를 구성하는 API들의 관리를 담당하며, 임베디드 응용프로그램이 필요로 하는 API가 임베디드 시스템 내에 존재하지 않을 경우 해당 API를 원격 API 서버로부터 다운로드/설치하여 시스템을 재구성할 수 있다. 따라서, 기본 API 이외에도 다양한 확장 API를 관리하고, 시스템을 동적으로 재구성하는 것이 가능하다.
- SAL : 미들웨어의 이식성과 재사용성을 높이기 위해 정의된 계층으로서 미들웨어가 각 임베디드 시스템과 상호작용할 수 있도록 각 실시간 운영체제에 의존적인 부분을 집약한 미들웨어의 최하위 계층이다. 운영체제가 바뀌더라도 SAL 계층만 수정하면 응용프로그램을 재개발할 필요없이 수행할 수 있다. 본 논문에서는 임베디드 시스템에서 많이 사용되는 VxWorks, QNX 및 리눅스용 SAL을 개발하였다.

IMES는 MT 프로젝트, Xenomai, Legacy2linux등에서는 지원하지 않는 API 관리자를 지원하여 응용프로그램이 필요로 하는 API는 시스템에 로드(load)시키고, 사용하지 않는 API는 시스템에서 자동 언로드(unload)함으로써 메모리 사용 효율을 높일 수 있다. 또한 IMES는 기존의 인터페이스용 미들웨어와는 다르게 표준 API



A. IMES 구조 B. SAL 내부 모듈

그림 3 임베디드 시스템 인터페이스용 미들웨어 구조

를 제공하며, 표준 API를 다시 기본 API와 확장 API들로 구분함으로써 구분된 확장 API를 이용하여 각 임베디드 운영체제들의 특성에 맞게 미들웨어를 구성할 수 있다.

이러한 특성은 IMES가 다양한 운영체제들을 지원하는데 가장 큰 역할을 한다. 표 1은 IMES와 타 미들웨어들과의 차이점을 표로 나타낸 것이다.

3.1 기본 API 및 확장 API

제안하는 IMES에서 기본 API는 보편적으로 이용되는 임베디드 운영체제인 VxWorks, 리눅스, QNX에서 공통적으로 존재하고, 응용프로그램이 실행되는데 꼭 필요한 API들로 선정하였다. 선정 방법은 그림 4와 같다.

기본 API 이외에 다른 확장적인 기능이나, 각 임베디드 시스템마다 특화된 기능 등을 확장 API로 정의하며 각 임베디드 시스템의 용도에 따라 확장될 수 있다. 확장 API들은 API 관리자에 의해 응용프로그램 및 시스템 용도에 따라 추가 및 삭제되어 시스템을 동적으로 구성될 수 있다. 본 연구에서는 우선적으로 기본 API만을 구현하였다.

본 논문에서는 기본 API를 태스크, 메모리, 세마포, 타이머, 메시지 큐, 메시지 포트, 시그널, 와치독 타이머(watchdog timer) 등 총 8개의 그룹으로 분류하였으며, 각각을 컴포넌트 형태로 구현하였다. 기본 API로 선정된 API는 유사 미들웨어인 Montavista project, Xenomai project 및 Sceptre Kernel Services 에서도 지정한 표준의 API 그룹으로서 응용프로그램의 실행에 기본이 되는 API 그룹이다[15,16].

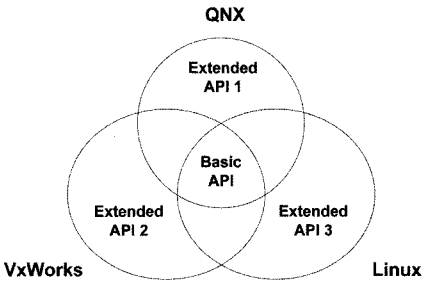


그림 4 기본 API 및 확장 API 선정 방법

IMES와 유사 미들웨어에서 제공하는 API 그룹과 그룹별 API 개수는 표 2와 같다.

3.2 API 관리자

API 관리자는 임베디드 시스템의 메모리 자원을 효율적으로 사용할 수 있도록, 응용프로그램의 실행 시 필요한 API를 동적으로 추가, 삭제, 갱신 및 저장하는 기능을 제공한다. 기본 API 및 확장 API들을 컴포넌트화하여, 시스템의 상황에 따라 필요한 컴포넌트만 적재함으로써 자원 최적화된 환경을 제공한다[17]. API 관리자의 내부 구조는 그림 5와 같다.

- 응용프로그램 관리자(Application Manager) : 응용프로그램의 설치, 삭제, 실행 및 정지 기능을 제공하는 모듈이다.
- 동적 재구성기(Dynamic Reconfigurator) : 응용프로그램에서 요구하는 API 컴포넌트의 설치 여부를 검사하여, 설치되어 있고 메모리에 로드되어 있지 않으면 해당 API 컴포넌트를 메모리에 로드하고, 더 이상 사

표 1 다른 미들웨어와 IMES간의 차이점

	IMES (제안하는 미들웨어)	MT	Xenomai	Legacy2linux
표준 API 제공	O	O	X	X
인터페이스 추상화	O	O	O	O
시스템 추상화 (멀티 운영체제 지원)	O	O	X	X
API 확장성	O	X	X	X
API 동적 재구성	O	X	X	X

표 2 다른 미들웨어의 API와 IMES의 기본 API 비교

미들웨어 / API 그룹	IMES (제안하는 미들웨어)	Montavista	Xenomai	Sceptre
태스크	23	24	15	8
메모리	12	-	4	-
세마포	6	8	8	2
메시지 큐	4	5	5	4
메시지 포트	4	-	-	-
시그널	5	-	-	4
타이머	6	-	8	-
와치독 타이머	4	5	-	-

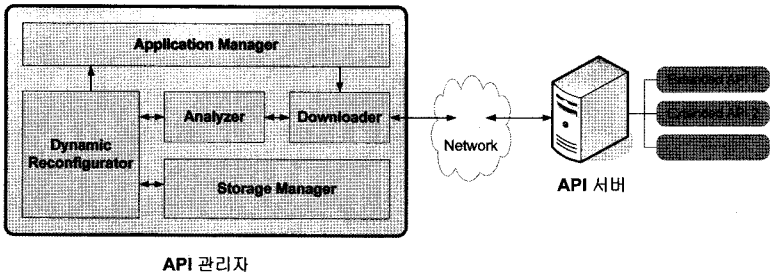


그림 5 API 관리자 구조

용하지 않는 API 컴포넌트는 메모리에서 해제 또는 삭제한다.

- 분석기(Analyzer) : 응용프로그램 및 API 컴포넌트의 명세를 분석하여 응용프로그램 및 해당 API 컴포넌트와 다른 API 컴포넌트의 대한 참조 여부를 검사하거나 API 컴포넌트의 버전을 관리한다.
- 다운로더(Downloader) : 시스템에 설치되어 있지 않은 응용프로그램이나 API 컴포넌트를 네트워크를 이용하여 다운로드하는 모듈이다. 디렉토리 서비스(directory service)를 이용하여 인터넷 상에서 해당하는 API 컴포넌트를 찾아 다운로드한다.
- 저장 관리자(Storage Manager) : 응용프로그램 및 API 컴포넌트들을 갱신할 때, 기존에 설치되어 있던 응용프로그램 및 컴포넌트에 대한 백업(backup) 기능을 제공하여 시스템 재구성으로 인해 발생할 수 있는 오류로부터 복구될 수 있도록 한다.

API 관리자는 API 컴포넌트 설치 및 삭제 시에 그 컴포넌트의 CDF(Component Description File)를 참조하는데, CDF는 API 컴포넌트의 정보 및 API 컴포넌트 간의 의존성에 대해 명시하고 있다. 또한 API 관리자는 API 컴포넌트들을 메모리에 로딩 및 언로딩 할 때, 각 컴포넌트 안의 API 주소들을 이용하여 API 참조 테이블(API Reference Table)을 구성한다.

그림 6의 A는 API 컴포넌트 설치에 대한 흐름도이

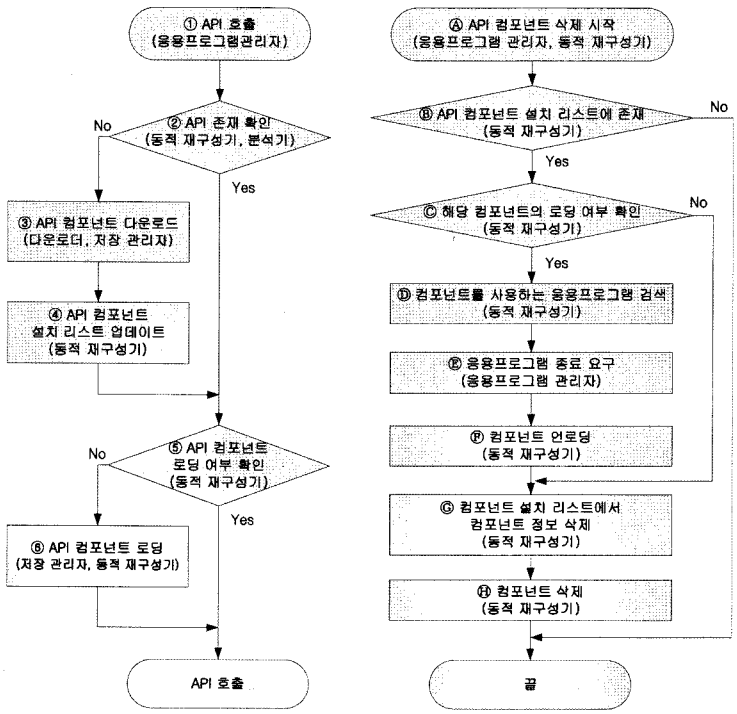


그림 6 API 관리자를 이용한 API 설치 및 삭제 흐름도

다. 응용프로그램에서 API를 호출하면(①), 동적 재구성기는 컴포넌트 설치 리스트를 확인하여 호출된 API가 속한 컴포넌트의 설치 여부 및 API의 로딩 여부를 확인한다(②). 호출된 API가 API 컴포넌트 설치 리스트에 존재하지 않으면, 해당 API가 시스템 내에 존재하지 않는 것으로 판단하여 원격 서버로부터 API 컴포넌트를 다운로드 받는다. 다운로드한 API 컴포넌트의 CDF 정보를 확인하여 의존관계가 있는 다른 컴포넌트가 존재할 경우 함께 다운로드 받는다(③). API 컴포넌트 설치 리스트를 업데이트하고(④), API 컴포넌트 설치 리스트를 확인하여 해당 API 컴포넌트가 로딩됐는지 확인한다(⑤). API 컴포넌트가 로딩되어 있지 않으면, 저장소에 저장되어 있는 컴포넌트를 로딩한다(⑥). API 관리자가 API 컴포넌트를 로딩할 때 API 참조 테이블에 컴포넌트 API에 대한 참조 정보를 반영하고, 반영된 참조 정보를 바탕으로 API를 호출 함으로써 API 설치 및 로딩 과정은 종료한다.

그림 6의 B는 API 컴포넌트 삭제에 대한 흐름도이다. 동적 재구성기는 삭제하고자 하는 컴포넌트가 존재하는지 API 컴포넌트 설치 리스트를 확인한다(A,B). 해당 컴포넌트가 존재하는 경우, 컴포넌트가 로딩된 상태인지 체크한다(C). 로딩 상태이면, 해당 컴포넌트를 사용하는 응용프로그램을 찾아 중지시키고(D,E), 컴포넌트를 언로딩한 후에(F) API 컴포넌트 설치 리스트에서 컴포넌트 정보를 삭제하고(G), 컴포넌트를 삭제한다(H).

IMES에서는 API 관리자를 사용함으로써 메모리를 효율적으로 관리 할 수 있고, 시스템의 필요에 따라 API를 동적으로 재구성하여 각기 다른 임베디드 시스템의 특성을 살릴 수 있다.

3.3 SAL

SAL은 기본 API 및 확장 API에서 정의된 API에 대한 시스템 의존적인 부분을 해당 시스템에 맞게 정합 및 추상화시켜 주는 계층으로서, 기본 API에서 제공하는 API의 기능을 구현하였다. 표 3은 QNX와 VxWorks에서 제공하는 세마포 API이다.

표 3에서 두 운영체제는 sem_post와 semGive API를 통하여 같은 기능을 제공하지만, API 이름이나, 각 API의 return 값 및 매개변수의 형태가 다르다. SAL은 이와 같이 운영체제에 따른 API 차이를 변환하는 역할을 제공한다. 또한, SAL은 직접 매핑(direct mapping)으로서의 역할뿐만 아니라, 직접 매핑이 안되는 경우 POSIX를 이용하여 API들의 기능을 구현하였다. 본 논문에서 제안하는 SAL의 내부 흐름은 그림 7의 A와 같으며, 이를 도식화하면 그림 7의 B와 같다.

SAL의 내부는 각 기능별 모듈로 구성되며, 각 모듈들은 API들의 기능을 제공하거나 모듈 내부를 관리한다. SAL은 태스크 관리 모듈, 메모리 관리 모듈, 세마포 관리 모듈, 메시지큐 관리 모듈, 메시지포트 관리 모듈, 타이머 관리 모듈, 와치독 타이머 관리 모듈, 시그널 관리 모듈로 구성된다.

- 태스크 관리 모듈(Task Management Module) : 태스크를 생성, 삭제 및 제어한다. 또한 태스크 생성 후, 즉시 스케줄링되는 경우나 태스크 생성과 스케줄링의 시작이 분리되어 있는 경우, 두 가지 방법 모두를 지원한다.
- 메모리 관리 모듈(Memory Management Module) : 기본적인 메모리 할당 방법뿐만 아니라 메모리 분할(memory partition) 할당 방법인 고정크기 분할과 가변크기 할당 방법을 지원한다.
- 세마포 관리 모듈(Semaphore Management Module) : 태스크 간에 동기화를 지원하기 위한 부분으로서 카운팅 세마포(counting semaphore)를 지원한다. 카운팅 세마포는 동기화와 상호 배제를 제공한다.
- 메시지 큐 관리 모듈(Message Queue Management Module) : 멀티태스킹(multitasking) 환경에서 태스크들 간의 정보를 주고받기 위한 통신 기능을 제공한다. 메시지 길이가 가변적인 가변길이 큐를 제공한다.
- 메시지 포트 관리 모듈(Message Port Management Module) : 메시지 큐와 유사하며, 여러 개의 메시지를 다른 태스크로 전달하는 기능을 제공한다. 메시지 큐와 다른 점은 메시지 큐는 메시지를 복사하여 전달하지만, 메시지 포트는 메시지의 포인터를 전달한다.
- 타이머 관리 모듈(Timer Management Module) : 타이머 함수들을 지원한다.
- 와치독 타이머 관리 모듈(Watchdog Timer Management Module) : 카운터의 일종으로 정기적으로 리셋(reset)되는 카운터가 리셋되지 않고 증가하면 시스템을 재부팅시키는 기능을 한다.
- 시그널 관리 모듈(Signal Management Module) : 하나 이상의 태스크에게 비동기적인 이벤트를 알리는 기능을 한다.

표 3 QNX 와 VxWorks의 API 함수 비교

QNX 함수	VxWorks 함수

Int sem_post (sem_t * sem) {
.....
}
	STATUS semGive (SEM_I semId) {
	.....
	}

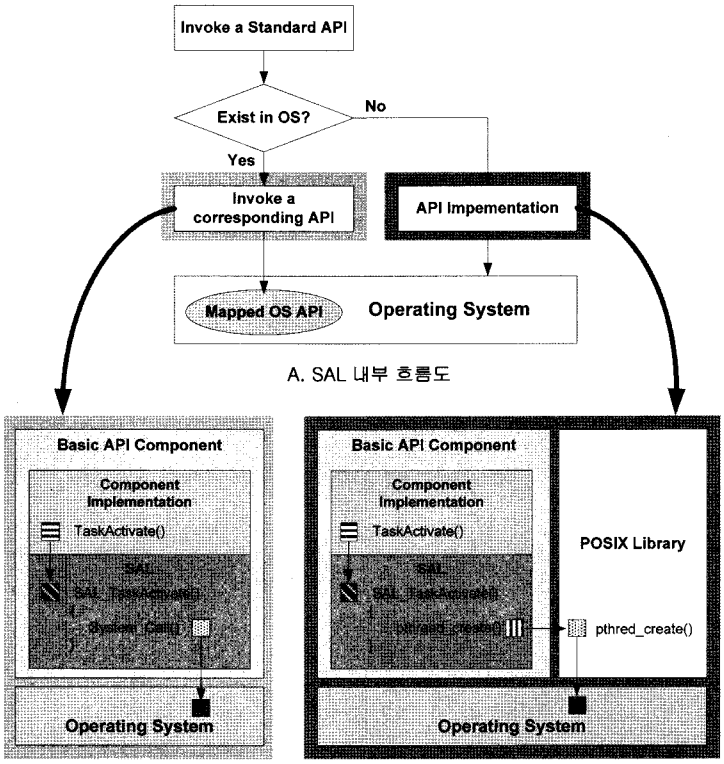


그림 7 SAL의 내부 데이타 흐름 및 구조

4. 테스트 및 성능 분석

제안한 IMES을 구현한 후, QNX, VxWorks 및 리눅스 운영체제가 설치된 컴퓨터를 사용하여 기능 및 성능 측정을 수행하였다. 본 연구는 산업용이나 무기체계 같은 중대형 임베디드 시스템 즉, PC급 기능을 수행하는 시스템을 대상으로 하였으며, 실험 환경도 이런 시스템에서 많이 사용되고 있는 Intel 프로세서를 대상으로 구성하였다. 실험에 사용된 컴퓨터들의 하드웨어 사양은 표 4와 같다. 실험에서 VxWorks가 설치된 컴퓨터의 사양과 QNX 및 리눅스가 설치된 컴퓨터의 사양이 다른 이유는 QNX와 리눅스는 컴퓨터에 각 운영체제가 직접 설치된 반면, VxWorks는 Windows 상에서 VxWorks에서 제공하는 Tornado 툴을 설치하여 Tornado에서 제공하는 가상 커널에서 테스트를 진행하였기 때문이다.

4.1 기능 테스트

각 API의 기능 및 API 관리자의 기능을 테스트하기 위해 각각의 운영체제에 IMES 및 API 기능 테스트 프로그램을 설치하고, 그림 8과 같이 테스트를 실시하였다. API 관리자의 기능을 테스트하기 위해 시스템 내에 기본 API 컴포넌트를 설치하지 않은 상태에서 테스트용 응용프로그램의 요청에 의해 기본 API 컴포넌트를 서버에서 다운로드하여 설치하도록 하였다. 그림 8의 ⓐ에서 시스템에 설치된 기본 API 컴포넌트가 메모리에 로드되는 것을 참조 테이블을 통해 확인하였다. 또한, 테스트용 응용프로그램의 실행 중에 기본 API 컴포넌트를 메모리에서 삭제한 후 API를 호출하면 해당 컴포넌트가 다시 메모리에 로드되어 계속적으로 서비스를 제공함을 확인하였다.

표 4 기능 및 성능 측정에 사용된 하드웨어 사양

	VxWorks	QNX	리눅스
CPU	Intel-PentiumⅣ 2.4GHz	Intel-PentiumⅢ 800MHz	Intel-PentiumⅣ 1.7GHz
Memory	1GB RAM	256MB RAM	1GB RAM
OS	Tornado 2.0 (VxWorks 5.5)	QNX 6.2.1	리눅스 Redhat 9.0 (Kernel 2.4.)

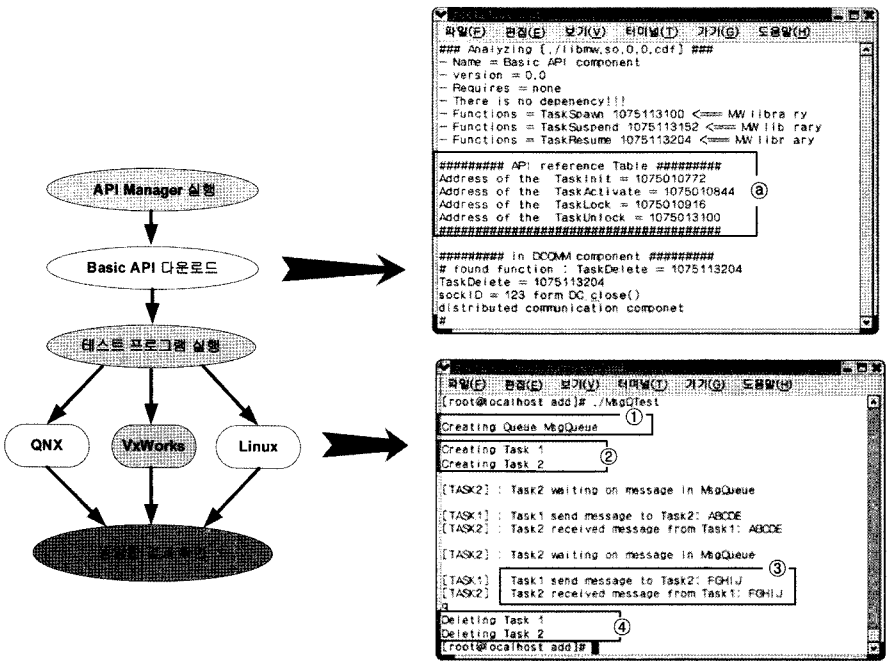


그림 8 기능 테스트 방법 및 테스트 결과

그림 8 ①~④는 태스크 및 메시지 큐를 서비스하는 API 컴포넌트에 대한 기능 테스트의 출력 결과로서, 메시지 전송을 위한 큐 생성(①), 메시지 수신을 위한 태스크 생성(②), 송수신 결과 확인(③) 및 태스크 삭제 기능(④) 등을 보여준다. 실험 결과, 시스템에 설치된 각 API 컴포넌트들이 정상적으로 동작하는 것을 확인하였다.

이와 같은 방법으로 모든 API 각각에 대해 정상 동작 상황, 비정상 동작 상황을 시뮬레이트(simulate)하는 테스트 프로그램을 통해 IMES 기능 테스트를 수행하였으며 IMES가 올바르게 동작함을 확인하였다.

4.2 성능 테스트

4.2.1 기본 API 성능 측정

응용프로그램이 운영체제의 API를 직접 호출하는 때와 응용프로그램이 IMES를 통해 API를 호출할 때의 시간 지연에 따른 성능 차이를 측정하기 위해, 미들웨어를 거치지 않고 직접 시스템 API를 호출할 때의 실행 시간과 IMES를 통해 시스템 API를 호출할 때의 실행 시간을 측정하였다.

API에는 실행 속성상 일회성 API와 수시성 API로 분류할 수 있다. 일회성 API는 태스크 생성 및 삭제, 메모리 할당 및 해제와 같이 한번 실행되면 상당 기간 그 효과가 지속되고 상대적으로 덜 빈번하게 호출되는 API이다. 따라서 응용프로그램 실행 성능에 상대적으로 적은 영향을 미친다. 그러나 수시성 API, 즉 메시지를 주고받거나 세마포를 주고받는 API들은 응용프로그램 실행 시 빈번히 사용되는 API로서 응용프로그램 성능에 큰 영향을 줄 수 있다. 따라서 본 실험에서는 응용프로그램 실행 시간에 많은 영향을 주는 SemaphoreSend, SemaphoreReceive, MessageQueueSend, MessageQueueReceive, MessagePortSend, MessagePortReceive와 같은 수시성 API를 대상으로 성능 측정을 실행하였다.

실험 1-1. 실험 1-1은 세마포 API의 IMES에 의한 시간 지연을 측정한 것이며, 표 5는 세마포 API를 각 운영체제별로 3,000회씩 반복 실험한 결과의 평균을 표로 나타낸 것이다.

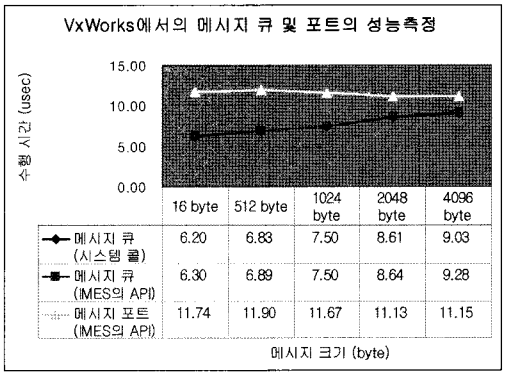
표 5 운영체제별 세마포 API의 측정결과 <단위 : μsec>

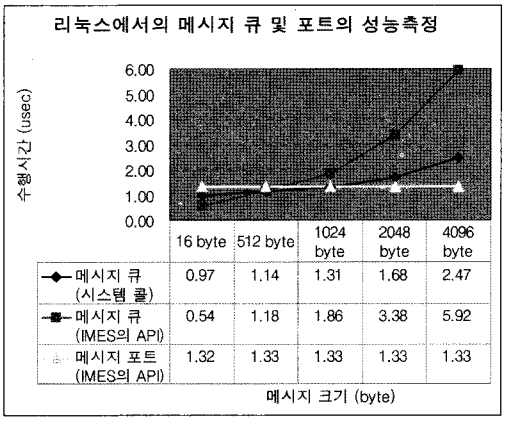
	QNX		VxWorks		리눅스	
	미들웨어 유 (IMES의 API)	미들웨어 무 (시스템 콜)	미들웨어 유 (IMES의 API)	미들웨어 무 (시스템 콜)	미들웨어 유 (IMES의 API)	미들웨어 무 (시스템 콜)
SemaphoreSend	4.04	1.82	2.28	1.19	1.12	0.27
SemaphoreReceive	4.64	1.71	2.14	2.14	0.45	0.34

세마포의 성능 측정 결과 운영체제를 직접 호출하는 시스템 콜과 미들웨어 API를 호출한 것 간의 성능 차이가 대부분 3μsec이내로 크지 않음을 알 수 있다.

실험 1-2. 실험 1-2는 메시지 큐와 메시지 포트의 IMES에 의한 시간 지연을 측정한 것이다. IMES는 메시지 큐와 메시지 포트 모두 기본 API로 제공하지만, VxWorks, QNX, 리눅스에서는 메시지 큐만 지원한다. 메시지 큐와 메시지 포트의 차이는 메시지를 넘겨주는 방식에 의해 차이가 나는데, 메시지 큐는 메시지를 복사하여 전송하고 메시지 포트는 메시지의 포인터를 전송해주어 복사가 일어나지 않는다. 본 연구에서 개발한 IMES는 두 API 컴포넌트를 모두 제공하여, 응용프로그램 개발자가 선택적으로 사용할 수 있게 하였다. 메시지 큐와 메시지 포트 성능 측정에서는 IMES에 의한 성능 지연 시간을 측정하기 위해 각 시스템에서 제공하는 API, 즉, 메시지 큐에 대한 시스템 콜과 IMES의 메시지 큐의 수행 시간에 대해 측정하였다. 또한 같은 크기의 메시지를 전송할 때 메시지 큐와 포트 간의 성능차이를 알기 위해 메시지 포트에 대해서도 측정하였다. API의 측정 결과는 메시지 큐와 포트의 전송 및 수신 함수를 함께 측정하여 1/2한 값이며, 그림 9는 메시지 크기를 16byte, 512byte, 1Mbyte, 2Mbyte, 4Mbyte로 하여 각 운영체제 별로 1,000회씩 반복 실험한 결과의 평균치를 그래프로 나타낸 것이다.

성능 측정 결과, VxWorks의 미들웨어 API는 거의 매핑 수준으로 구현되어 메시지 큐에서 데이타 크기가 증가해도 성능 차이가 0.5μsec이내로 아주 적어서 메시지 전달 성능이 매우 우수함을 확인하였다. 리눅스와 QNX의 메시지 큐 같은 경우 메시지 크기가 작을 때는 IMES의 실행 시간이 QNX 시스템 콜(system call)보다 좋다가 메시지 크기가 커지면서 시스템 콜보다 실행 시간이 커지는 것을 볼 수 있다. 이것은 리눅스와 QNX의 메시지 큐가 SAL에 구현되어서 커널 모드(kernel mode)에 진입하지 않고 메시지를 전송함으로써 메시지 크기가 작을 때에는 시스템 콜로 메시지를 전송할 때보다 성능이 좋다. 그러나 메시지 크기가 커지면서 복사할 메시지를 위한 메모리 할당 및 복사에 의한 커널 접근 시간이 길어지기 때문에, 시스템 콜에 의해 커널 내부에서 처리할 때보다 실행 시간이 더 걸리게 된다. 메시지 포트는 각 시스템 마다 메시지의 크기에 상관없이 일정한 성능을 보이며, 메시지의 크기가 큰 경우 메시지 포트를 사용하는 것이 성능상 유리한 것을 알 수 있다. VxWorks같은 경우 그림 9에 나타난 그래프의 내용만으로는 메시지 큐의 성능이 더 좋지만, 메시지 크기가 더 커질수록 메시지 큐보다 메시지 포트의 성능이 더 좋아질 것을 유추 할 수 있다.





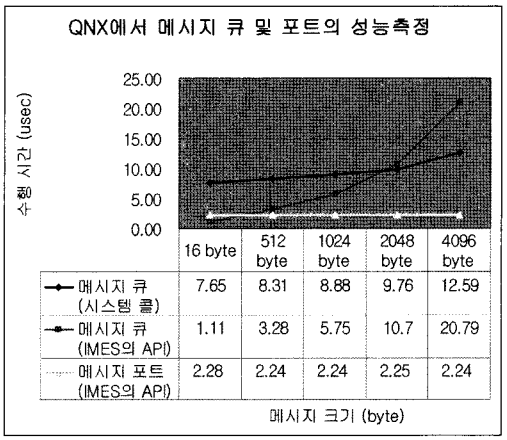


그림 9 운영체제별 메시지 큐 및 메시지 포트 성능 측정 결과

4.2.2 API 관리자 성능 측정

API 관리자의 동적 재구성에 따른 소요 시간은 네트워크에서의 소요 시간과 시스템에서의 소요 시간으로 나뉜다. 네트워크에서의 소요 시간은 설치할 API 컴포넌트를 제공하는 컴포넌트 서버에 대한 검색 시간과 컴포넌트 서버를 통한 API 컴포넌트의 다운로드 시간으로 구성된다. 시스템에서의 소요 시간은 다운로드된 API

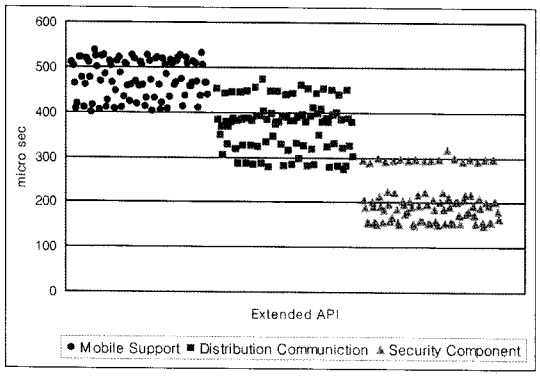
컴포넌트를 플랫폼에 설치 및 삭제하는데 소요되는 시간으로서 API 컴포넌트의 메모리 적재 및 해제 시간과 이에 따른 API 참조 테이블 갱신 시간으로 구성된다.

본 실험에서는 네트워크의 상황, 즉 서버의 위치나 네트워크 전송 방식 및 트래픽 상황에 따라 변하는 네트워크에서의 소요 시간을 배제하고, 시스템에서의 플랫폼 동적 재구성을 위한 소요 시간만을 측정하기 위하여 실험에서 사용할 API 컴포넌트를 시스템 저장 장치에 저장한 후 실험하였다.

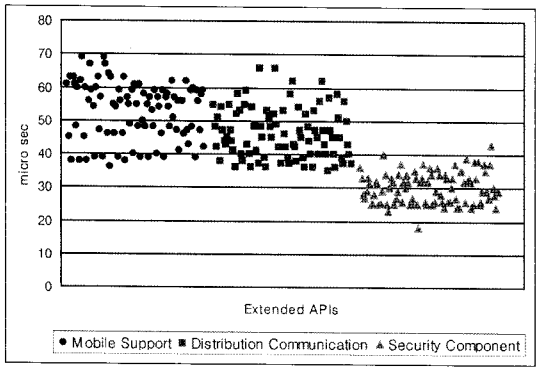
실험을 위해 세 가지 확장 API 컴포넌트를 사용하는 채팅프로그램을 실행하여 API 관리자의 성능을 측정하였다. 채팅프로그램에서 사용되는 확장 API 컴포넌트는 이동성 지원 API 컴포넌트, 분산통신 API 컴포넌트 및 보안 API 컴포넌트이며, 분산통신 API 컴포넌트는 미들웨어의 데이타 통신 기능을 제공하고, 이동성 지원 API 컴포넌트는 모바일 IP 기능을 제공한다. 보안 API 컴포넌트는 미들웨어의 보안 기능을 제공한다. 표 6은 본 실험에서 사용한 확장 API의 개수 및 크기에 대한 정보를 표로 나타낸 것이다.

표 6 확장 API 컴포넌트 정보

	컴포넌트 사이즈	API 개수
이동성지원	271 Kbyte	11 개
분산통신	11 Kbyte	27 개
보안	15 Kbyte	7 개



(a) Extended API installation times



(b) Extended API uninstallation times

그림 10 API 관리자 성능 측정 결과

그림 10은 각 확장 API 컴포넌트의 설치 및 삭제에 소요되는 시간의 분포를 보여주며, 소요 시간은 각 확장 API 컴포넌트 별로 일정한 시간 범위 안에 있음을 확인할 수 있다. 실험 결과, 본 연구에서 개발한 플랫폼 동적 재구성 기능은 컴포넌트의 크기와 컴포넌트에서 제공하는 API의 개수에 영향을 받으며, 컴포넌트를 설치하거나 삭제할 때 예측 가능한 시간 안에 플랫폼을 재구성할 수 있는 일정한 성능을 보장한다. 또한 비교적 큰 소요 시간을 보인 분산통신 API 컴포넌트의 설치와 이동성지원 API 컴포넌트의 삭제 경우, 응용프로그램 실행 시 컴포넌트 설치 및 삭제가 단 한번만 실행되기 때문에 전체 시스템의 성능에 미치는 영향은 적을 것으로 판단된다.

5. 결 론

임베디드 응용프로그램의 개발 기간 및 비용을 줄이기 위해 임베디드 응용프로그램의 재사용 필요성이 높아지고 있다. 이에 따라 재개발 없이 기존 응용프로그램을 다른 다양한 운영체제에서 실행할 수 있게 해주는 인터페이스용 미들웨어가 필요하게 되었지만, 현재 나와 있는 인터페이스용 미들웨어들은 특정 운영체제에 종속적이고, API의 확장성을 지원하지 않는다는 문제점을 갖고 있다. 본 연구에서는 이러한 문제점을 극복하기 위하여 IMES를 설계 및 구현하였으며, IMES는 각 임베디드 운영체제를 추상화시킴으로써 응용프로그램의 재사용성을 보장한다. 또한 제안하는 IMES의 API 관리자는 메모리 자원이 한정적인 임베디드 시스템에서 API의 동적 재구성을 가능하게 하여 메모리 자원을 효율적으로 사용할 수 있다. 이런 API 관리자 기능은 타 미들웨어에서도 제공하지 않는 IMES만의 특징이다. IMES는 MT 프로젝트와는 다른 표준 API 방식, 즉 표준 API를 기본 API와 확장 API들로 분류하여 설계함으로써 API의 확장성을 제공하고 각기 다른 임베디드 운영체제의 특성 또한 잘 반영되게 하였다. 또한 실험 결과에서 볼 수 있듯이 미들웨어의 존재로 인한 시스템 오버헤드를 최소화함으로써 응용프로그램의 실행 시간의 오버헤드를 최소화하였다.

참 고 문 헌

[1] 차정은, 유미선, "임베디드 S/W 재사용을 위한 공통 개발 지침 및 시범 사례 구축", 정보과학회지, 2006. 11.

[2] 한국정보산업연합회, "해외 기업의 임베디드 소프트웨어 개발 역량 강화 사례", 한국정보산업연합회, 임베디드소프트웨어산업협의회, 2006.
[3] William B. Frankes, Kyo Kang, "Software reuse research: Status and future," IEEE Trans. on Software Engineering, Vol.31, July 2005.
[4] Kao WC, Kao CC, Lin CK, Sun TH, Lin SY, "Reusable embedded software platform for versatile camera systems," IEEE Transactions on Consumer Electronics, Vol.51, No.4, 2005.
[5] Loyall JP, "Emerging trends in adaptive middleware and its application to distributed real-time embedded systems," Lecture Notes in Computer Science, Vol.2855, pp. 20-34, 2003.
[6] Peng J, Liu JD, Yang T, "Research and implementation of the real-time middleware in open system," Lecture Notes in Computer Science, Vol. 3032, pp. 803-808, 2004.
[7] Mapusoft, MT project, http://www.mapusoft.com/
[8] Xenomai, Xenomai, http://www.xenomai.org/
[9] MontaVista software, Legacy2Linux, http://legacy2linux. sourceforge.net/
[10] Business Wire, Cranes Software Brings OS Changer & OS Abstractor Software to India, http://www.findarticles.com/p/articles/mi_m0EIN/is_2005_Oct_3/ai_n15657103
[11] MapuSoft, OS Changer & OS Abstractor, http://mapusoft.com/products/
[12] Gna, Xenomai-Implementing a RTOS emulation framework on GNU/Linux, http://download.gna.org/rtai/documentation/vesuvio/pdf/xenomai.pdf
[13] 김명선, 유인선, 최 훈, "실시간을 지원하는 리눅스 인터페이스용 미들웨어 설계 및 구현", 한국정보과학회 KCC2006 추계학술발표대회, 논문집 제33권 제2호(A), pp.313-317, 2006. 10.
[14] 이승열, 이철훈, "실시간 운영체제 QNX 인터페이스용 미들웨어 설계 및 구현", 한국정보과학회 KCC2006 추계학술발표대회, 논문집 제33권 제2호(A), pp.454-458, 2006. 10.
[15] Xenomai, Native Xenomai API, http://www.xenomai.org/documentation/branches/v2.0.x/html/api/group_native.html
[16] Browaeys F. et al., "Sceptre: Proposed Standard for a real-time executive kernel," Technology and Science of Information, Vol.3, pp. 37-43, 1984.
[17] Su-Won Lee, Yong-Duck You, Hoon Choi, "Design and Implementation C Based Lightweight Component Framework," 2006 IEEE TENCON, pp.91, 2006. 11.

김 명 선
2003년 한남대학교 멀티미디어학과(학사). 2007년 충남대학교 컴퓨터공학과(석사). 관심분야는 모바일 컴퓨팅/분산 시스템 미들웨어, 임베디드 시스템, 실시간 OS

이 수 원
2005년 충남대학교 컴퓨터공학과(학사) 2007년 충남대학교 컴퓨터공학과(석사) 관심분야는 분산 시스템 미들웨어, 웨어러블 컴퓨터, 실시간 OS

이 철 훈
1983년 서울대학교 전자공학과(학사). 1988년 KAIST 전기및전자공학과(석사). 1992년 KAIST 전기및전자공학과(박사). 1983년~1986년 삼성전자 컴퓨터사업부 연구원. 1992년~1994년 삼성전자 컴퓨터사업부 선임연구원. 1994년~1995년 미국 미시건대학교 객원연구원. 2004년~2005년 미국 미시건대학교 초빙연구원. 1995년~현재 충남대학교 컴퓨터공학과 교수. 관심분야는 실시간시스템, 운영체제, 고장허용시스템

최 훈
1983년 서울대학교 컴퓨터공학과(학사) 1990년 Duke University 전산학과(석사) 1993년 Duke University 전산학과(박사). 1983년~1996년 한국전자통신연구원 광대역통신망연구부 근무. 1996년~현재 충남대학교 컴퓨터공학과 교수. 2000년 미국 NIST(National Institute of Standards and Technologies) 객원연구원. 관심분야는 모바일 컴퓨팅/분산 시스템 미들웨어, 운영체제

조 길 석
1984년 경북대학교 전자공학과(학사). 1986년 경북대학교 전자공학과(석사). 2004년 University of Florida 전기 및 컴퓨터공학과(박사). 1986년~현재 국방과학연구소 책임연구원. 관심분야는 Distributed/Dependable Computing, 임베디드 시스템