http://dx.doi.org/10.5392/JKCA.2015.15.04.017

범용 운영체제의 이식성 향상을 위한 인터페이스 미들웨어 설계 및 구현

Design and Implementation of Interface Middleware for Improved Portability on General Operating System

김연일, 이상길, 이승일, 이철훈
충남대학교 컴퓨터공학과

Yeon-Il Kim(kyi3426@cnu.ac.kr), Sang-Gil Lee(sk0137@cnu.ac.kr),
Seung-Il Lee(silee@cnu.ac.kr), Cheol-Hoon Lee(clee@cnu.ac.kr)

요약

운영체제 상에서 동작하는 응용프로그램은 운영체제의 환경과 지원하는 표준 라이브러리들이 다르기 때문에 운영체제에 매우 높은 의존성을 가진다. 이러한 이유로 동일한 기능을 수행하는 응용프로그램도 운영체제에 따라 새롭게 구현해야 되며 이는 응용프로그램 개발 이후의 유지 보수나 관리 측면에서도 시간적, 경제적 낭비를 초래한다. 이를 해결하기 위해 Cygwin이나 MinGW 등의 연구가 진행되고 있지만, 가상환경이나 툴을 제공하는 것일 뿐 응용프로그램 자체에 대한 이식성을 지원하는 것은 아니다. 따라서 본 논문에서는 표준 C라이브러리와 POSIX를 이용한 래퍼 형식의 범용 운영체제를 위한 인터페이스 미들웨어를 설계하여 응용프로그램이 가상환경이나 코드 수정 없이 동일한 동작을 지원한다. 미들웨어는 API를 기본과 확장으로 분류해 선택적으로 적재하여 응용프로그램의 크기를 효율적으로 관리할 수 있도록 한다. 또한, 응용프로그램을 인터페이스 미들웨어를 탑재한 Linux, Unix, Windows와 Cygwin을 비교 실험하고 기능 및 성능 평가를 수행하였다.

■ 중심어 : | 인터페이스 미들웨어 | 운영체제 | 응용프로그램 |

Abstract

The applications program that running on Operating System has high dependence. Because environment of OS and standard libraries that supports are different. For those reason, Applications that perform the same function should be implemented in accordance with the new operating system. This results in a temporal and economic waste not only in subsequent maintenance of application but also in management. Even though, to solve this problem Cygwin or MinGW has been distributed, they do not support the portability of the application but provide a virtual environment and the tool. Therefore, in this paper, we design the wrapper format interface middleware using the POSIX and standard C library to support the application performing the same function on virtual environment and without code modification. The middleware can be selectively loading the API that is classified by basic and extend. This allows to managing the application size efficiently. Also, perform the comparative experiments and performance evaluation for application, on equipped with the Interface Middleware Linux, Unix, Windows and on Cygwin

■ keyword : | Interface Middleware | Operating System | Application Program |

* 이 논문은 2014년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임
(No. 한국연구재단에서 부여한 과제번호 : 2013R1A1A2062171)
접수일자 : 2014년 12월 09일 심사완료일 : 2015년 01월 06일
수정일자 : 2015년 01월 05일 교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

I. 서 론

운영체제는 시스템 하드웨어를 관리할 뿐 아니라 응용프로그램을 실행하기 위하여 하드웨어 추상화 플랫폼과 공통 시스템 서비스를 제공하는 시스템 소프트웨어이다. 잘 알려진 현대의 PC운영체제는 마이크로소프트사의 Windows, 오픈 소스 기반의 Linux, 솔라리스사의 Unix가 있으며 이러한 운영체제들은 GNU나 IEEE와 같은 표준화 단체에 의해 동일한 구조의 표준 라이브러리를 지원한다. 하지만 운영체제별로 시스템 구성이 다르며 동일한 API(Application Program Interface)를 지원하는 것은 아니다. Windows는 Windows API와 표준 C라이브러리를 지원하지만 Linux와 Unix는 POSIX와 표준 C라이브러리를 지원한다. 동일한 라이브러리 유형이라도 운영체제나 라이브러리 버전에 따라 API 프로토콜을 다르게 가질 수 있으며, 이와 같은 내용은 응용프로그램의 이식성을 저하시키는 요인이 될 수 있다. 또한 응용프로그램 개발 시 프로그램언어 선택에 따라 차이점이 존재한다. 자바와 같은 플랫폼 독립적인 언어를 이용하여 개발할 경우 표준 자바 API를 사용하여 CPU나 운영체제의 종류와 무관하게 동일한 동작을 보장할 수 있다. 자바에서는 JNI 기술을 사용하여 표준 C 라이브러리나 POSIX를 지원하나 기존 라이브러리를 자바 바이트 코드로 변환하여 JVM 상에서 실행하기 때문에 실행 속도 저하의 단점이 존재하게 되며 C/C++ 언어를 사용하기 위해 JNI와 JVM의 구조에 대한 충분한 지식을 요구한다. 따라서 C/C++ 언어 기반에서 운영체제가 변경되더라도 응용프로그램을 변경하지 않고 그대로 실행 가능하게 하여 응용프로그램의 수명 주기(Life Cycle)를 장기간 보장할 수 있는 보완장치가 필요 시 된다. 이런 보완장치에 대한 연구는 Windows에서 POISX API 제공을 위한 Cygwin, MinGW, SFU(Windows Service For Unix) 등이 존재하며 임베디드 시스템의 미들웨어 연구로는 Xenomai, Legacy2Linux, IMES(Interface Middleware for Embedded System) 등이 진행되고 있다. 그러나 Cygwin, MinGW, SFU는 가상환경이나 툴을 제공함으로써 응용프로그램 자체에 대한 이식성을 제공하지 못하며 실행 시간이 길다는 단점이 존재하며, 임베디드 시스템의 미들웨어는 모듈 재사용 프레임워크만을 제공하거나[1] 특정 운영체제에 종속적[2]이기 때문에 타겟 운영체제가 변경되면 미들웨어를 수정해야 되며 Windows는 지원하지 않는 문제점이 존재한다. 이러한 문제를 해결하기 위해 본 논문에서는 범용 운영체제를 위한 인터페이스 미들웨어를 소개한다. 인터페이스 미들웨어는 C/C++ 언어 기반으로 운영체제에 종속적이지 않은 API 인터페이스를 가지며 POSIX와 표준 C라이브러리 기반으로 응용프로그램의 이식성 향상을 지원한다.

본 논문의 구성은 2장에서 관련연구로 Windows에서 호환성을 제공하는 미들웨어인 Cygwin과 SFU, 임베디드 운영체제용으로 개발한 인터페이스 미들웨어를 기술하며, 3장에서 응용프로그램에 이식성을 제공하기 위한 인터페이스 미들웨어 설계 내용을 다룬다. 4장에서는 실험 및 결과를 기술하여 인터페이스 미들웨어가 정상적으로 동작함으로써 설계 방안의 가능성을 확인한다. 마지막으로 결론 및 향후 연구과제에 대해 기술한다.

II. 관련 연구

1.인터페이스 미들웨어

1.1 Cygwin

Cygwin은 Windows에서 구동되는 Unix와 같은 환경의 실질적인 POSIX 시스템 콜 기능을 제공하는 에뮬레이션 레이어다. Cygwin은 cygwin1.dll이라는 DLL파일을 통해 POSIX API를 제공하며, Unix에서 사용하는 툴의 집합으로 구성되어 있다. Cygwin이 설치되면 사용자는 표준 Unix 유틸리티들을 사용하여 bash처럼 Cygwin에서 제공하는 셸이나 Windows 커맨드 프롬프트를 실행시킬 수 있다. 또한, 개발자는 표준 마이크로소프트의 Windows API와 Cygwin API를 이용하여 Win32 콘솔이나 GUI 응용프로그램을 만들 수 있다. 이러한 Cygwin을 통해 소스코드를 크게 바꾸지 않고 Unix 프로그램을 Windows 환경에 이식하는 것이 가능

하다[3].

Cygwin의 핵심은 런타임 라이브러리인 Cygwin1.dll이다. 이것은 Unix에서 빈번히 사용되는 셸이나 명령 등을 Windows에서 소스 코드로부터 컴파일 할 수 있도록 하여주며, 이를 통해 Unix 상의 코드를 Windows 상에서 수정 없이 사용 가능하도록 한다. 하지만 Cygwin은 에뮬레이터로써 Cygwin을 시작하고 프로그램을 실행하는 데 오랜 시간이 소요되며, 실제 응용프로그램의 구동은 Cygwin 상에서 mount를 통하여 동작시킬 수 있다. 또한, Cygwin은 상업적인 용도로 이용한다면 추가적인 라이센스 비용을 지불해야 한다[4].

1.2 SFU

SFU는 마이크로소프트에서 개발된 소프트웨어 패키지로 Windows NT나 이후 운영체제 상에서 Unix 하위 시스템 혹은 전체적인 Unix 환경의 일부분에 해당하는 기능을 제공하는 소프트웨어 패키지이다. 주된 기능은 SFU를 이용하여 NFS server를 구성할 수 있고 NFS client로써 작동 할 수도 있으며 Telnet을 통해 SFU에 접속 유지 관리 할 수 있도록 도와준다. 또한, CShell이 포함되어 있기 때문에 Unix 교육용으로 활용되기도 한다[5].

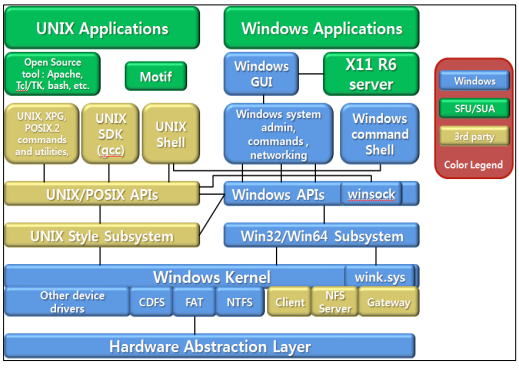


그림 1. Unix 상호 운영성 구조

위의 [그림 1]은 Unix 상호운영의 구조를 나타낸 것이다. Windows 커널에서 NFS server 및 NFS client를 지원하며 다양한 Unix 응용프로그램을 지원하는 구조는 기존의 Windows 응용프로그램을 위한 Win32/64 서브시스템과 동일한 구조로 지원이 된다. Third-Party에서 제공하는 확장기능으로 사용자나 고객의 요구에 따라 필요한 기능을 추가 할 수 있으며 이러한 확장기능을 통해 기존의 Unix 환경과 동일한 환경으로 커스터마이징할 수 있도록 지원한다[6].

기존에는 별도 유틸리티를 제공하였으나, Windows Server 2003 R2에서부터 내장된 기능으로 제공한다. 이를 통해 현재는 응용프로그램으로서가 아닌 하나의 Windows 기반의 서브시스템 중 하나가 되었다[7].

2. 임베디드 운영체제용 인터페이스 미들웨어

2.1 Xenomai

Xenomai는 VxWorks, pSOS, VRTX와 같은 실시간 운영체제의 API를 리눅스상에서 에뮬레이트한다. 응용프로그램의 중복 개발을 줄이고, 기존 실시간 운영체제 환경에 익숙한 개발자들에게 유사한 개발환경을 제공함으로써 리눅스 개발환경에 쉽게 적응하는 것을 목적으로 한다[8]. 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[9]. Xenomai는 여러 운영체제의 응용프로그램을 리눅스 환경에서 구동하기 위한 미들웨어로서 리눅스 이외의 운영체제에서 동작하지 않는 단점이 있다. 아래의 [그림 2]는 Xenomai의 구조를 보여준다.

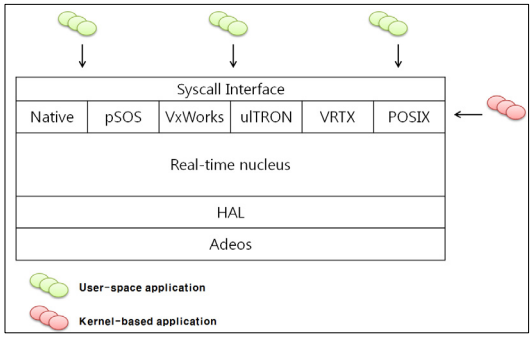


그림 2. Xenomai 구조

2.2 IMES

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[10]. 여기서 미들웨어는 분산 환경에서 네트워크 운영체제와 응용프로그램 상에 위치하여 시스템의 이질적인 환경을 추상화 시켜주고 서로 다른 기능을 정합시키는 계층이다. [그림 3]는 IMES의 구조를 나타내고 있다.

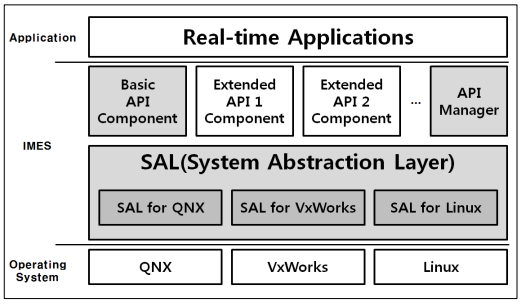


그림 3. IMES 구조

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

- 기본 API 및 확장 API : IMES에서 기본 API는 보편적으로 이용되는 임베디드 운영체제인 VxWorks, Linux, QNX에서 공통적으로 존재하고, 응용프로그램이 실행되는데 꼭 필요한 API들로 선정하였다. 선정 방법은 [그림 4]와 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.

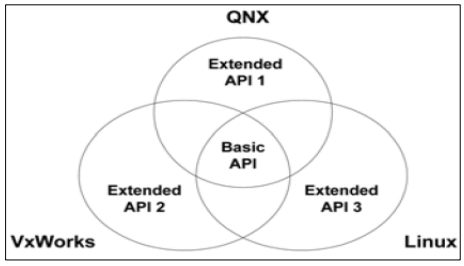


그림 4. 기본 API 및 확장 API 선정 방법

- 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.
- System Abstraction Layer : SAL은 기본 API 및 확장 API에서 정의된 API에 대한 시스템 의존적인 부분을 해당 시스템에 맞게 정합 및 추상화시켜 주는 계층으로 운영체제에 따른 API 차이를 변환하는 역할을 제공한다. 아래의 [그림 5]는 SAL 내부 데이터 흐름을 나타낸 것으로 API가 호출될 경우 OS에서 API를 지원하면 SAL은 해당 API를 직접 매핑(direct mapping)하는 역할을 수행한다. 만약 직접 매핑이 안되는 경우 POSIX를 이용하여 API들의 기능을 구현하였다.

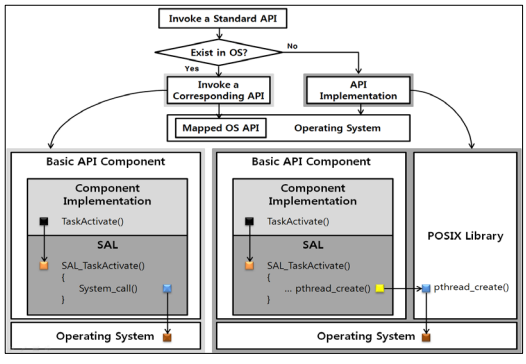


그림 5. SAL 내부 데이터 흐름 및 구조

SAL의 내부는 각 기능별 모듈로 구성되며, 각 모듈들은 API들의 기능을 제공하거나 모듈 내부를 관리한다. IMES는 임베디드 운영체제의 특성에 맞게 미들웨어를 구성하였으며 QNX, VxWorks, Linux에서 정상 동작을 지원한다. 하지만 임베디드 운영체제에 중점을 두어 개발되었기 때문에 지원하는 기본 API의 수가 적으며 Windows 환경을 지원하지 않는 문제가 있다.

Ⅲ. 인터페이스용 미들웨어 설계

1. 인터페이스 미들웨어 구조 설계

본 논문에서 제안하는 인터페이스 미들웨어는 응용프로그램과 운영체제에서 가지는 표준 인터페이스 라이브러리 사이에 위치하며 Unix와 Linux에서 가지는 시스템의 이질적인 자원과 기능들을 추상화하여 라이브러리에서 지원하는 인터페이스 기능을 통합화 시키는 계층이다. 즉, 인터페이스 미들웨어를 통해 다양한 운영체제 상에서 표준 라이브러리인 C라이브러리와 POSIX기반으로 작성된 응용프로그램을 별도의 추가 작업 없이 사용 가능하도록 하여 사용자에게 응용프로그램 구현 및 관리의 용이성과 편리성을 제공한다. 다음의 [그림 6]는 인터페이스 미들웨어의 구조를 나타낸 것으로 응용프로그램 하위 계층에 위치하여 기본 API와 확장 API를 선택적으로 응용프로그램에게 지원한다. 또한 Windows 환경에서 POSIX API를 지원하기 위한 프로세스 관련 기능들을 정의해 놓은 계층과 Unix 타입을 정의해 놓은 계층이 존재한다.

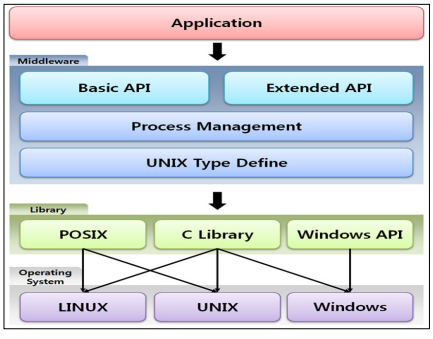


그림 6. 인터페이스 미들웨어 구조

인터페이스 미들웨어는 내부적으로 C 라이브러리와 POSIX API를 호출 하며, 존재하지 않는 API의 경우 라이브러리에서 지원하는 API들을 조합하여 그 기능을 수행하도록 한다. 이처럼 인터페이스 미들웨어는 라이브러리 인터페이스의 조합으로 제공되는 래퍼 형식 때문에 함수 호출에 대한 오버헤드가 발생한다. 이러한 오버헤드를 줄이기 위해 미들웨어 개발단계에서는 최적화가 고려되어야 한다.

1.1 기본 API 및 확장 API

본 논문에서는 응용프로그램의 메모리 자원을 효율적으로 사용하기 위한 방안으로 기본 API와 확장 API를 정의하였다. 기본 API와 확장 API는 표준 C라이브러리 API와 POSIX API로 구성되며 정적 링크(Static Link)방식으로 컴파일한다. 하지만, 응용프로그램의 목적에 따라 확장 API의 코드를 컴파일 하도록 함으로써 미들웨어의 크기를 유동적으로 가질 수 있고, 이로 인해 인터페이스 미들웨어를 사용하는 응용프로그램의 크기를 효율적으로 관리할 수 있는 이점을 얻을 수 있다. 기본 API는 자주 사용되는 API의 집합으로 항상 메모리에 적재하도록 한다. 확장 API는 사용자에 의해 선택적으로 활성화하여 메모리에 적재되도록 한다. 아래의 [표 1][표 2]은 API들을 정리한 것을 나타낸다.

표 1. 표준 C 라이브러리 API 분류

API 범주	기본 API 수	확장 API 수
산술 연산	9	13
시간 관리	3	5
프로세스 관리	1	2
문자열 관리	25	19
메모리 관리	8	0
파일 및 디렉터리 관리	16	2
입출력 관리	5	2
시스템 관리	1	1
예외 및 신호 처리	0	1

표 2. POSIX API 분류

API 범주	기본 API 수	확장 API 수
예외 및 신호처리	8	16
프로세스 관리	2	17
프로세스 간 통신	12	13
파일 및 디렉터리 관리	15	34
운영체제 객체의 보안	2	18
쓰레드 관리	5	72
시간 관리	0	3
메모리 관리	0	2
네트워크 관리	6	21
기타	3	17

기본 API와 확장 API의 선택은 인터페이스 미들웨어의 개발자가 정의하는 것으로 명확한 기준은 존재하지 않는다. 본 논문에서는 "The GNU C Library Reference Manual[11]"와 "POSIX Programmer's guide[12]"을 참고하여 API를 분류하였다. API항목은 표준 C 라이브러리에서 113개, POSIX에서 266개의 API로 총 379개의 API를 정의하였으며 API는 수행 연산 특성에 따라 표준 C라이브러리에서 9개의 범주로 POSIX에서 10개의 범주로 나누었다.

1.2 Process Management 및 Unix Type Define

POSIX API는 Unix 기반 운영체제에 정의된 표준 인터페이스 규격으로 각 운영체제가 지원하는 시스템 콜로 구현되어 있다. POSIX API는 Windows에서는 지원하지 않으나 기능적으로 유사한 함수들을 제공하고 있다. 하지만 프로세스와 관련된 API들은 각 운영체제에서 크게 차이점이 존재하며 Windows에서는 구현이 불가능하다. 이러한 문제로 인해 Windows 환경에서 POSIX API 지원을 하기 위해 인터페이스 미들웨어에 프로세스 관리를 위한 계층을 설계하였다. 이 계층은 Windows 환경에서만 참조하며 프로세스 테이블을 생성한다. 프로세스 테이블은 각 인덱스마다 프로세스 관련 정보를 가지는 구조체를 저장하고 프로세스가 생성되거나 삭제되면 관련 정보나 리스트는 수정되어 일관성을 유지하도록 한다. 다음의 [그림 7]은 프로세스 관리 구조를 보인다.

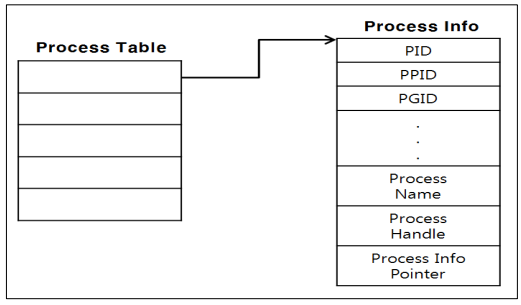


그림 7. 인터페이스 미들웨어 프로세스 관리 구조

또한 Unix와 Linux 상에서 지원하는 여러 가지 자료형이나 구조체 타입은 Windows 상에서 동일하게 지원하지 않는다. 이를 위해 인터페이스 미들웨어에서는 기본 API와 확장 API에서 사용하는 자료형과 구조체 타입을 조사한 후 API의 기능과 특성에 맞게 재정의하는 계층을 추가적으로 설계하였다.

2. 인터페이스 미들웨어 구현 방안

본 논문에서 제안하는 인터페이스 미들웨어의 개발 방안은 "실시간 운영체제 UbiFOSTM 인터페이스용 미들웨어 설계 및 구현[13]"의 내용과 마이크로소프트에서 제공하는 "UNIX Custom Application Migration Guide"를 참고하였다. 미들웨어의 구현은 다음과 같은 방안들을 통해 구현할 수 있다.

- 윈도우에서 POSIX 함수의 기능을 제공하는 오픈소스 라이브러리 또는 윈도우가 제공하는 호환성 라이브러리 사용
- 마이크로소프트가 직접적으로 지원하는 업계 표준 함수 사용
- 라이브러리가 아닌 매크로를 이용해서 한 운영체제에서 다른 운영체제의 기능 구현

본 논문에서 제안하는 인터페이스 미들웨어는 첫번째, 두번째 방식을 지향하며 이는 내부적으로 다음의 3가지 경우를 고려해야 한다.

- 동일한 기능을 하는 API를 지원하는 경우
- 함수 Protocol(함수 호출 규약)이 다른 경우
- 가변 인자를 가지는 경우

이번 절에서는 인터페이스 미들웨어 개발을 위해 위의 사항들을 고려하여 구현을 위한 가이드라인을 제시한다. 아래의 [그림 8]는 미들웨어 구현에서 위의 특성을 가지는 인터페이스들의 구현 방안을 Flow-Chart를 통해 도식화 한 그림이다.

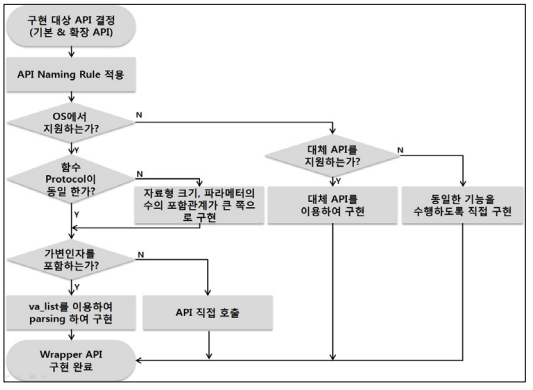


그림 8. 인터페이스 미들웨어 API 설계 방안

우선 구현대상 API가 결정되면 해당 API에 wrap이라는 접두사를 붙이는 호출명 규칙을 적용하고 API를 운영체제에서 지원하는지 확인한다. 지원하지 않는다면 동일한 기능을 수행할 수 있는 대체 API 및 API 조합을 이용하여 구현하며, 이조차 불가능하다면 직접적인 구현을 통해 해당 인터페이스의 기능을 지원하도록 한다. 만약 구현 대상 API를 운영체제에서 지원한다면 Protocol의 동일 여부를 검사한다. 만약, Protocol이 다르다면 그 의미가 더 큰 쪽으로 구현한다. 예를 들어 반환값이 int 형과 char 형을 가진다면 int 형으로 통일한다. 이와 마찬가지로 인자가 2개와 4개라면 4개로 통일하도록 한다.

마지막으로 가변 인자를 포함하는지 확인하며, 가변 인자를 포함하는 경우 이를 위해 va_list를 이용하여 추가적인 파싱 과정을 포함해야 한다. 가변 인자란 printf(), scanf()와 같이 제약 없이 원하는 인자만큼 전달하는 인자를 의미하며 만약 이러한 가변 인자를 포함하지 않는 경우, 내부적으로 인터페이스를 호출/리턴하는 방식으로 단순한 형태의 구현이 가능해 진다.

2.1 표준 C 라이브러리 API 구현

표준 C 라이브러리 API를 분석한 결과 Unix, Linux, Windows의 인터페이스는 대부분 동일한 프로토콜 지원을 통해 동일한 기능이 수행된다는 결과를 얻었다. 특별히 반환 값이 다른 경우 의미가 큰 쪽으로 통일시켰으며 가변 인자를 가지는 함수에 대해서 파싱을 통해 구현하였다. 다음의 [그림 9]은 가변 인자를 가지는 인터페이스 중 하나인 wrap_printf()의 구현 코드이다.

```
void wrap_printf(const char* format, ...){
    va_list argptr;
    char* temp_buf;

    temp_buf = (char*)malloc(256);
    memset(temp_buf, 0, 256);

    va_start(argptr, format);
    vsprintf(temp_buf, format, argptr);

    printf("%s", temp_buf);

    free(temp_buf);

    va_end(argptr);
}
```

그림 9. wrap_printf 구현 코드

2.2 POSIX API 구현

POSIX API는 Unix와 Linux에서 동일한 인터페이스를 가진다. 하지만 Windows 환경에서는 제공하지 않기 때문에 대체 API를 사용하거나 직접 API를 구현하여 작성하여야 하며, 이는 많은 부분의 노력을 필요 시한다. 다음의 [그림 10]은 Windows API인 CreateFIle()을 사용하여 구현한 유형인 wrap_open()의 동작과정이다.

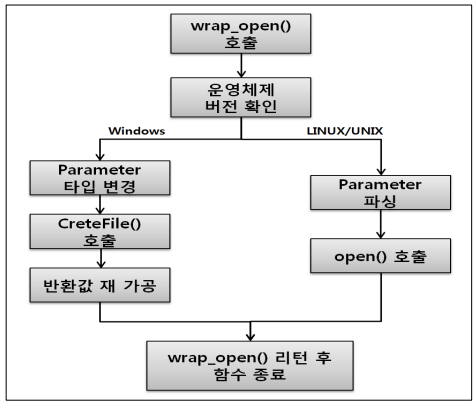


그림 10. wrap_open의 동작 과정

3. 인터페이스 호출명 재정의

미들웨어의 API를 기존의 표준 API명과 동일하게 구현할 경우 함수 재정의 에러가 발생하기 때문에 미들웨어 상에서 인터페이스명을 기존의 API명과 동일하게 구현할 수 없다. 만약 표준 API명과 다른 함수 호출명을 사용할 경우, 응용프로그램에서 표준 API를 호출하는 코드마다 인터페이스 미들웨어 API명으로 변경해야 하는 불편함이 생기며 이는 응용프로그램의 이식성을 저하시키는 요인이 될 수 있다. 본 논문에서는 이러한 문제점을 해결하기 위해 인터페이스 미들웨어에서 구현하는 API들의 함수 호출명에 일관된 규칙을 적용한 후 #define 매크로를 이용하여 기존의 함수명과 동일하게 동작하도록 구현하였다. 다음의 [그림 11]은 정의된 문법의 구조를 나타낸다.

```
#define source_string destination_string
```

그림 11. 호출명 재정의 문법 구조

#define매크로를 이용하여 응용프로그램 코드상에 존재하는 모든 source string을 destination string으로 변경하는 작업이 컴파일 과정 중에 이루어진다. 이러한 과정을 통해 인터페이스의 이름을 재정의하여 개발자에게 지원한다.

Ⅳ. 실험환경 및 결과

1. 실험환경 및 실험방법

인터페이스 미들웨어의 기능이 정상적으로 동작하는 것을 검증하기 위해 구성된 테스트 환경은 아래의 [표 3]와 같다.

표 3. 인터페이스 미들웨어 검증 테스트 환경

	Unix	Linux	Windows	Cygwin
OS	SunOS Solaris 5.10	openSUSE 12.3	Windows 7 (32bit)	Cygwin NT 6.1
Compiler	GCC 4.6.3	GCC 4.7.2	Visual C++ 6.0	GCC 4.8.2

테스트는 동일한 PC에서 진행하였으며 CPU는 Intel Pentium G860 3.0GHz 기반의 환경을 가지고 있다. 운영체제는 Unix, Linux, Windows를 포함하며 Cygwin에 대한 실험도 진행한다. 실험방법은 함수의 기능을 확인할 수 있는 동일한 응용프로그램을 작성한 후, 각각의 운영체제에서 동일한 동작의 수행여부를 확인하는 방법으로 하였다. 또한 함수들의 수행시간을 측정하여 인터페이스 미들웨어의 성능평가를 수행하였다.

2. 실험결과

2.1 기능 평가

기능평가는 크게 3가지로 대체 가능한 API, 간단한 연산으로 직접 구현한 API, 복잡한 연산으로 직접 구현한 API로 나누어 진행하였다.

아래의 [그림 12]는 현재 작업 디렉터리를 변경하는 chdir()함수의 구현 화면을 나타낸 것으로chdir()함수는 윈도우 운영체제에서 제공하지 않으며, 대체 가능한 Windows API인 SetCurrentDirectory()를 지원하여 준다.

```
int wrap_chdir(const char* path){
#ifdef LINUX
        return chdir(path);
#elif WINDOWS
        return SetCurrentDirectory(path);
#endif
}
```

그림 12. chdir 구현 화면

아래의 [표 4]에서 수행한 테스트 코드는 현재 작업 디렉터리를 변경 후 출력하는 프로그램이다. 다음 표를 통해 테스트환경 모두 정상적으로 수행하게 되며, 결과 화면을 통해 그 연산이 동일하게 수행하는 것을 확인할 수 있다.

표 4. 운영체제에서 지원하는 대체 가능한 API

	chdir()

Test Code	int main(int argc, char* argv[]){
	system("ls");
	printf("₩nchdir() Test Program₩n₩n");
	chdir("./test_dir");
	system("ls");
	printf("₩n");
	}

Unix	# ./a.out a.out chdir_test.c test_dir chdir() Test Program test1.txt.txt test2.txt.txt test3.txt.txt
Linux	sslab@ubuntu:~/work$ ./a.out a.out chdir_test.c chmod_test.c opendir_test.c test_dir chdir() Test Program text1.txt.txt text2.txt.txt text3.txt.txt
Windows	$A_Project.lib pthread.h pthreadVCE2.lib testprogram.dsw TEST_DIR pthreadVC2.dll pthreadVSE2.dll testprogram.ncb chdir() Test Program test1.txt.txt test2.txt.txt test3.txt.txt
Cygwin	$ ./a.exe a.exe main.cpp main.cpp~ TEST_DIR chdir Execute! test1.txt test2.txt test3.txt

아래의 [그림 13]는 지정된 디렉터리를 검색하기 위해 디렉터리를 열기 하는 opendir()함수의 구현 화면의 일부를 나타낸 것으로 윈도우상에서 지원하지 않고 동일한 기능을 수행하는 함수 또한 제공하지 않는다. 이러한 이유로 Windows API의 조합과 간단한 연산을 통해 동일한 동작을 수행하는 함수를 구현하였다.

```
DIR* wrap_opendir(const char *name){
#ifdef LINUX
        return opendir(name);
#elif WINDOWS
        char *path; HANDLE h; WIN32_FIND_DATAA *fd; DIR *dir;
        char namebuff[256]; TCHAR buff[256]; int namlen;
                .....
        if (namebuff[namlen -1] == 'WW' || namebuff[namlen -1] == '/') {
            path = (char*)_alloca(namlen + 2);
            strncpy(path, namebuff, namlen + 2);
            path[namlen] = '*';
            path[namlen + 1] = 'W0';
        } else {
            .....
        }

        if ((fd = (WIN32_FIND_DATAA*)malloc(sizeof(WIN32_FIND_DATA))) == NULL) {
            return NULL;
        }

        if ((h = FindFirstFileA(path, fd)) == INVALID_HANDLE_VALUE) {
            free(fd);
            return NULL;
        }
            .....
        dir->h = h;  dir->fd = fd; dir->has_next = TRUE; dir->d_position = 0;
        strcpy(dir->path,path);
  return dir;
#endif
}
```

그림 13. opendir 구현 화면

표 5. 간단한 연산으로 직접 구현한 API

	opendir()

Test Code	int main(int argc, char* argv[]){
	DIR *dir_info;
	struct dirent *dir_entry;
	dir_info = opendir(".");
	if(dir_info){
	while(dir_entry = readdir(dir_info))
	printf("%s₩n", dir_entry->d_name);
	}
	closedir(dir_info);
	}

Unix	# ./a.out . .. chdir_test.c a.out core test_dir opendir_test.c
Linux	sslab@ubuntu:~/work$ ./a.out a.out chmod_test.c chdir_test.c .. test_dir . opendir_test.c
Windows	. .. Debug include main.cpp testprogram.dsp testprogram.dsw testprogram.ncb testprogram.opt testprogram.plg TEST_DIR Press any key to continue
Cygwin	$ ./a.exe . .. .a.c.swp .main2.cpp.swp a.exe main.cpp TEST_DIR

위의 [표 5]에서 수행한 테스트 코드는 현재 위치의 디렉터리를 열기한 후 디렉터리의 전체 목록을 출력해 주는 프로그램이다. 다음 표를 통해 테스트환경 모두 정상적으로 수행하게 되며, 결과 화면을 통해 현재 디렉터리를 열기하여 디렉터리 목록을 출력하는 것을 확인할 수 있다.

아래의 [그림 14]는 대상 파일의 권한을 변경하는 chmod()함수의 구현 화면의 일부를 나타낸 것이다. chmod()는 윈도우 환경에서 지원하지 않아 직접 구현하였으며, chmod() 내부적으로는 사용자가 정의한 함수를 몇 차례 재호출 하는 방식으로 복잡한 연산을 수행한다. 이러한 유형의 함수는 다양한 테스트를 통해 정확성 검증에 주의해야 한다.

```
int wrap_chmod (const char * file_name, mode_t mode){
#ifdef      LINUX
            return chmod(file_name, mode);
#elif       WINDOWS
            HANDLE hFile;
            DWORD allowedAceMasks [] = {FILE_GENERIC_READ,
                                        FILE_GENERIC_WRITE,
                                        FILE_GENERIC_EXECUTE};
            DWORD deniedAceMasks [] =  {FILE_GENERIC_READ & ~SYNCHRONIZE,
                                        FILE_GENERIC_WRITE & ~SYNCHRONIZE,
                                        FILE_GENERIC_EXECUTE & ~SYNCHRONIZE};

            hFile = CreateFile( file_name, GENERIC_READ, FILE_SHARE_READ,
                                NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
                  ...

            if (hFile == INVALID_HANDLE_VALUE)
            {
               if(SECURITY_DEBUG) printf(" [SECURITY_DEBUG] File not found₩n");
               return -1;
            }
            else
            {
               return = ChangeFilePermissions(mode, (char*)file_name,
                                              allowedAceMasks, deniedAceMasks);
            }
#endif
}
```

그림 14. chmod 구현 화면

표 6. 복잡한 연산으로 직접 구현한 API

	chmod()

Test Code	`int main(int argc, char* argv[]){`
	`  system("ls -al test.txt");`
	`  printf("₩n");`
	`  if(!chmod("test.txt", 0111)) printf("File Mode Change₩n");`
	`  else printf("File Mode Change Fail₩n");`
	`  system("ls -al test.txt");`
	`}`

Unix	`# ./a.out`
	`---x--x--x  1 root  root  21  5월 28일  13:39 test.txt`
	
	`File Mode Change`
	`---x--x--x  1 root  root  21  5월 28일  13:39 test.txt`

Linux	`sslab@ubuntu:~/work$ ./a.out`
	`-rw-r--r-- 1 sslab sslab 17 2014-11-02 20:03 test.txt`
	
	`File Mode Change`
	`---x--x--x 1 sslab sslab 17 2014-11-02 20:03 test.txt`

Windows	`----------+ 1 *Administrators  None  0 Nov  2 11:59 test.txt`
	
	`File Mode Change`
	`---x--x--x  1 *Administrators  None  0 Nov  2 11:59 test.txt`
	`Press any key to continue`

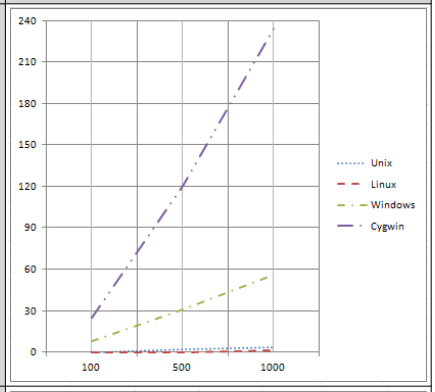
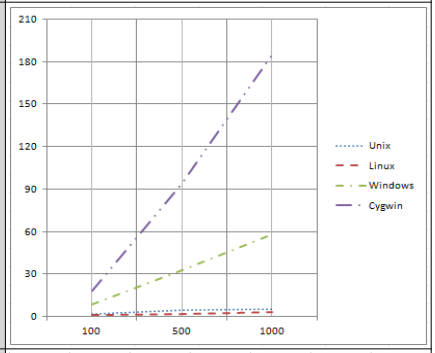
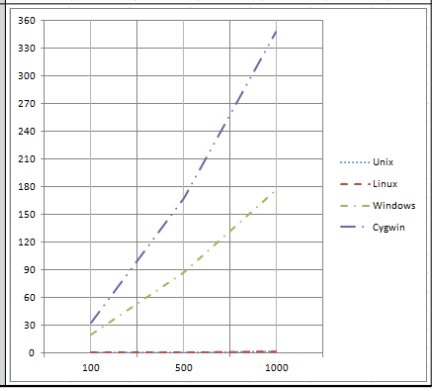
Cygwin	`$ ./a.exe`
	`-rwxr-xr-x 1 Administrators None 0 11월  2 12:00 test.txt`
	
	`File Mode Change`
	`---x--x--x 1 Administrators None 0 11월  2 12:00 test.txt`

위의 [표 6]에서 수행한 테스트 코드는 대상 파일의 권한을 변경하고 이를 확인하는 프로그램이다. 다음 표를 통해 테스트환경 모두 정상적으로 수행하게 되며, 결과 화면을 통해 대상 파일의 권한이 정상적으로 변경되는 것을 확인할 수 있다.

2.2 성능 평가

앞 절에서는 인터페이스 미들웨어의 기능을 평가하기 위해 API들의 수행 결과를 확인하였다. 이번 절에서는 API 설계 방안에 따라 수행시간이 어느 정도 소요되는지를 평가한다.

표 7. API 수행 시간 정리

	수행 시간 그래프
chdir()	
opendir()	
chmod()	

위의 [표 7]은 각 운영체제에서 성능평가를 위한 API 함수의 수행 시간을 측정한 결과를 그래프로 나타낸 것이다. 성능 평가는 Unix, Linux, Windows에서 동일하게 평가할 수 있는 방식을 사용해야 되기 때문에 sys/timeb.h의 ftime()함수를 이용하여 동일한 조건으로 성능을 검증하였다. 또한 변별력 있는 결과 값을 얻

기 위해 각 API별로 100, 500, 1000번의 반복수행을 통해 결과를 확인하였으며, x축의 값은 API의 동작 횟수를 의미하고 y축은 ms 단위의 수행시간을 의미한다.

Windows 환경에서 제공하는 API를 이용하거나 간단한 연산을 통해 인터페이스 미들웨어 함수를 구현한 경우 비교적 빠른 성능으로 수행하는 것을 chdir(), opendir() API를 통해 확인할 수 있다. 반면 Cygwin API의 결과를 통해 Cygwin API는 래퍼 함수 방식이 아닌 하드 코드로 구현된 것을 예상할 수 있으며 본 논문의 인터페이스 미들웨어 함수보다 수행 시간이 오래 걸리는 것을 확인할 수 있다. 마지막으로 다수의 Windows API로 커널 자원에 접근하여 조작하는 연산인 chmod()함수의 경우, 수행 시간이 대폭 증가하는 것을 확인할 수 있으며 이는 다양한 Windows API를 이용하였을 때 래퍼 함수방식을 통한 구현의 단점을 보이고 있다.

V. 결론 및 향후 연구과제

본 논문에서는 응용프로그램의 이식성과 재사용성 향상을 위한 인터페이스 미들웨어를 개발하였다. 인터페이스 미들웨어는 표준 라이브러리와 응용프로그램 사이에 존재하며 메모리 자원을 효율적으로 사용하기 위해 기본 API와 확장 API로 나누어 구현 및 설계하였다. 또한 기존의 표준 라이브러리 API명과 동일한 호출명을 사용하여 미들웨어 API와 표준 라이브러리 API의 혼동을 방지하였다.

인터페이스 미들웨어를 검증하기 위한 방안으로 동일한 코드를 통한 기능평가에서 정상적으로 동작하는 것을 확인할 수 있었으며, 함수의 수행시간을 비교하여 본 논문에서 제안하는 미들웨어가 Windows에서 Unix 계열이 동작하도록 고안된 에뮬레이터인 Cygwin에 비해 수행시간 측면에서 우수한 것을 확인할 수 있다. 향후 연구과제로는 본 논문에서 제안한 인터페이스 미들웨어 API가 기존의 API와 동일한 성능을 얻기 위한 래퍼 함수의 최적화 방안에 대한 연구진행이 필요하다.

참 고 문 헌

[1] 이철식, 김영상, 문석환, “임베디드 시스템에서의 자원 재사용을 위한 임베디드 미들웨어 설계 및 구현”, 한국차세대컴퓨팅학회 논문지, Vol.10, No.5, pp.6-12, 2014.

[2] 송병열, 장철수, 김성훈, 최 훈, “실시간 확장 플랫폼 RTX를 위한 개방형 미들웨어”, ICROS 학술대회, pp.27-28, 2014.

[3] *Cygwin User's Guide*, Red Hat, 2003.

[4] http://www.cygwin.com

[5] Microsoft Service For Unix : Installation and Configuration for eSlim Korea, eSlimKorea Product.

[6] http://technet.mocrosoft.com/ko-kr/library/cc700776.aspx

[7] http://technet.mocrosoft.com/en-us/library/cc771470.aspx

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

[9] Gna, Xenomai-Implementing a RTOS emulation framework on GNU/Linux, http://download.gna.org/rtai/documentation/vesuvio/pdf/xenomai.pdf

[10] 김명선, 이수원, 이철훈, 최 훈, 조길석, “임베디드 시스템 인터페이스용 미들웨어 설계 및 성능 분석”, 한국정보과학회 논문지, 제14권, 제1호, pp.52-62, 2008.

[11] http://www.gnu.org/software/libc/manual/pdf/libc.pdf

[12] D. Lewine, *POSIX Programmers Guide*, O'Reilly Media, 1991

[13] 송예진, 이철훈 “실시간 운영체제 UbiFOSTM 인터페이스용 미들웨어 설계 및 구현”, 한국콘텐츠학회논문지, 제5권, 제1호, pp.1-52, 2007.

저 자 소 개

김 연 일(Yeon-Il Kim) 준회원

- 2014년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2014년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 임베디드 시스템

이 상 길(Sang-Gil Lee) 준회원

- 2014년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2014년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 임베디드 시스템

이 승 일(Seung-Il Lee) 준회원

- 2013년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2013년 9월 ~ 현재 : 충남대학교 컴퓨터공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 임베디드 시스템

이 철 훈 (Cheol-Hoon Lee) 정회원

- 1983년 2월 : 서울대학교 전자공학과(공학사)
- 1988년 2월 : 한국과학기술원 전기 및 전자공학과(공학석사)
- 1992년 2월 : 한국과학기술원 전기 및 전자공학과(공학박사)
- 1983년 3월 ~ 1986년 2월 : 삼성전자 컴퓨터 사업부 연구원
- 1992년 3월 ~ 1994년 2월 : 삼성전자 컴퓨터 사업부 선임연구원
- 1994년 2월 ~ 1995년 2월 : Univ. of Michigan 객원연구원
- 1995년 5월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙연구원

<관심분야> : 실시간시스템, 운영체제, 고장허용 컴퓨팅, 로봇 미들웨어