다중 플랫폼 지원을 위한 WAP 추상 커널 계층

강 영 만[†] · 한 순 희[†] · 조 국 현[††]

요 약

이동단말(mobile phone, PDA, smart phone, notebook PC 등)에서 WAP을 구현하고자 할 경우, 이동단말마다 운영체제가 상이하므로 프로그램 흐름의 제어, 인터럽트, IPC등 운영체제 특성을 반영한 별도의 구현이 필요하다. 이로 인하여 개발기간의 단축이 어려움은 물론 개발 비용 증가, 개발인원의 투여, 시장 조기 진입의 어려움 등이 존재한다. 본 논문은 WAP을 다중 플랫폼에서 구현하기 위한 기저를 제공하는 추상 커널 계층(Abstract Kernel Layer)의 설계와 구현에 관한 것이다. 이는 REX, Palm, MS-DOS, MS-Window, UNIX 및 Linux를 포함한 각종 운영체제를 지원하는 커널 계층을 설계하여, 기기 종속적인 부분을 최소화하고 일관적인 인터페이스를 지원하여 개발 기간을 단축하고 소프트웨어의 유지보수를 용이하게 하는데 그 목적이 있다. 또한 추상 커널 계층은 mobile phone과 PDA에 탑재하여 그 실용성을 입증하였다.

WAP Abstract Kernel Layer Supporting Multi-platform

Young-Man Kang[†] · Soon-Hee Han[†] · Kuk-Hyun Cho[††]

ABSTRACT

In case of implementing a complicated application like WAP (Wireless Application Protocol) in a mobile terminal with the characteristic of bare machine and versatile kernel aspects of which are control, interrupt and IPC (Inter Process Communication), a special methodology should be needed. If not, it will cause more cost and human resources, even delayed product into launching for the time-to-market. This paper suggests AKL (Abstract Kernel Layer) for the design and implementation of WAP on basis of multi-platform. AKL is running on the various kernel including REX, MS-DOS, MS-Window, UNIX and LINUX. For the purpose of it, AKL makes machine-dependant features be minimized and supports a consistent interface on API (Application Program Interface) point of views. Therefore, it makes porting times of a device be shorten and makes easy of maintenance. We validated our suggestion as a consequent of porting WAP into PalmV PDA and mobile phone with AKL.

키워드 : WAP, WML, WMLScript, Micro Browser(마이크로 브라우저), Virtual task(가상타스크)

1. 개 요

이동단말을 이용한 인터넷 서비스는 점점 증가하고 있으며, 최근에 출시되는 대부분의 이동단말에는 이를 지원하는 기능이 필수적으로 구현되어야 하는 실정이다. 이러한 흐름 중의 하나인 WAP(Wireless Application Protocol)은 많은 이동 단말에서 채택되고 있으며, 국제 표준으로 자리 매김하고 있다[1-3]. 그러나 이동단말(mobile phone, PDA, smart phone, notebook PC 등)에서 WAP을 구현하고자 할 경우, 이동단말마다 운영체제가 상이하므로 프로그램 흐름의 제어, 인터럽트, IPC등 운영체제 특성을 반영한 별도의 구현이 필요하다. 이로 인하여 개발기간의 단축이 어려움은 물론 개발 비용증가, 개발인원의 투여, 시장 조기 진입의 어려움 등이 존재한다.

이동단말마다 운영체제가 상이하다는 것은 개발 Client는 각 이동단말에 대한 별도의 소프트웨어 버전을 가져야 함을 의미한다. 본 논문에서는 이러한 개발 환경을 고려하여 개발 Client가 플랫폼과 독립적으로 동작되도록 설계하려는데 그 목적이 있다.

REX, Palm, MS-DOS, MS-Window, UNIX 및 Linux를 포함하여 모든 운영체제에 일관적인 프로그램 흐름의 제어, 인터럽트, IPC등을 고안한다면 동일 버전으로서 모든 운영체제를 지원할 수 있는 Client의 개발이 가능하다.[4,5,6,7,8] 물론 개발 Client 전체를 운영체제와 별개인 버전으로 만들 수는 없다. 입출력 제어에 해당하는 GUI 제어, Keypad 제어 등은 이동단말마다 다를 것이므로 이에 대한 처리모듈은 각 이동단말마다 달리해야 한다. 그러나 입력에 대한 정형화, 처리로직, 결과의 정형화를 기하여, 모든 이동단말에 대하여 별도의 처리부분을 최소화한다.

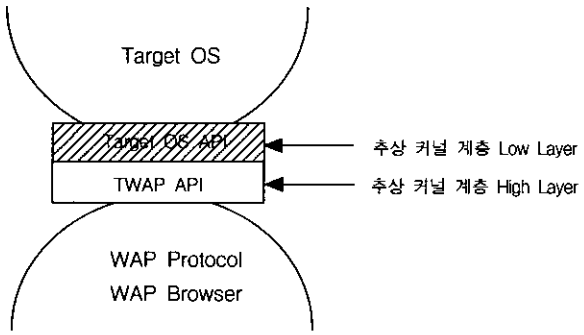
† 정 회 원 : 여수대학교 전자통신정보공학부 교수 겸 텔슨전자 수석연구원
†† 정 회 원 : 개방형 컴퓨터 통신연구회 회장
논문접수 : 2000년 12월 21일, 심사완료 : 2001년 5월 11일

2. 추상 커널 계층 개념모델 및 가상 타스크

2.1 개념 모델

추상 커널 계층은 다중 플랫폼을 지원하기 위한 목적을 달성하기 위하여 별도의 구조를 도입한다. 별도의 구조가 시스템 종속적인 부분을 흡수하여 시스템 비 종속적인 기능으로 대응시킴으로써 어느 플랫폼에 구현 WAP이 이식되든지 동일한 기능을 발휘하도록 하는 점이 근본 개념이다.

추상 커널 계층의 개념모델은 (그림 1)과 같다.



(그림 1) 추상 커널 계층 개념 모델

이는 Target 운영체제 커널하에서 동작되는 별도의 커널을 두는 개념과 동일하다. 관련 예로서 Linux 하에서 동작되는 DOS/NT/Win98 emulation을 들 수 있다. 개발 WAP 브라우저/프로토콜은 Target OS의 API가 아닌 별도로 정의된 API를 이용한다. 별도로 정의된 API는 어느 플랫폼이나 동일하므로 개발 WAP 브라우저/프로토콜의 내역은 바뀌지 않는다[3, 9-14]. 다만 해당 API를 관련커널(REX, Palm, DOS, Windows, Linux etc.)의 기능으로 대응시키는 작업이 필요하다. (그림 1)에서 제시된 추상 커널 계층 High Layer와 추상 커널 계층 Low Layer에 관한 정의는 아래와 같다.

- 추상 커널 계층 High Layer : 개발 WAP 브라우저/프로토콜의 사용 API의 알고리즘이 정의되는 부분
- 추상 커널 계층 Low Layer : 정의된 API를 상위 kernel (REX, Palm 등)에 적용하는 부분

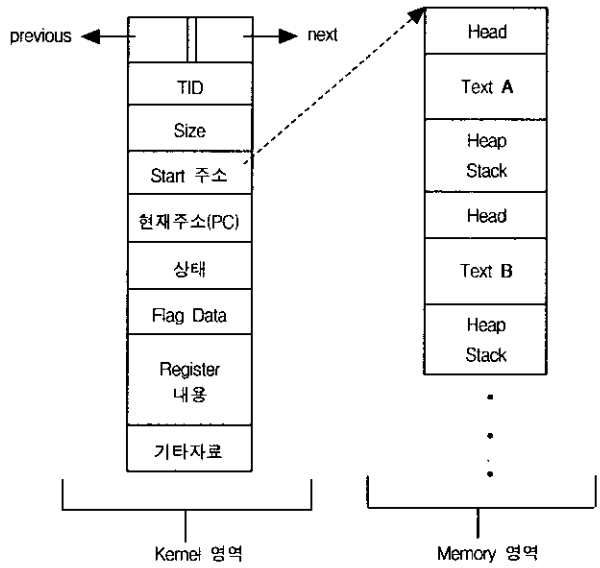
2.2 가상 타스크(Virtual Task : Vtask)

모듈은 WML(Wireless Markup Language)[9], WSP(Wireless Session Protocol)[12], WTP(Wireless Transaction Protocol)[13], WTLS(Wireless Transport Layer Security)[15] 및 WDP(Wireless Datagram Protocol)[14]등의 WAP 단위 기능을 수행하는 독립시킬 수 있는 추상적 개념의 용어이다. 각 모듈은 보다 세분화되면서 프로시저 개념에서 단위기능을 수행할 수 있는 구현측면으로 대응될 수 있다. 즉 한 개의 모듈은 그 모듈상태의 진행상 여러 개의 단위 기능으로 구성될 수 있으며, 이들 각각을 VTask로 정의한다. VTask는 모듈의 구현측면으로 본다. 구현의 편의에 따라서 한 모듈은 한 개의 VTask로 혹은 두 개 이상의 VTask로 구성될 수 있다. 이러한 배경으로 아래에서 제시되는 모듈과 VTask를 설명한다.

2.2.1 추상 커널 계층 상위 커널 Task

일반적으로 커널에서 지원하고 있는 타스크는 Head, 독립된 코드, Heap 및 Stack 영역으로 구성된다.

Head/Code/Heap/Stack으로 구성된 형상이 프로그램의 메모리 상에서 동작 이력이며 이를 관리하는 기능과 함께 할 경우 이를 Task라 정의한다. 다중 Task를 지원하는 Kernel에서 Task 관리정보는 TCB(Task Control Block)에 (그림 2)와 같은 형태로 저장되어 운영된다.



(그림 2) 다중 타스크를 지원하는 커널에서의 TCB 구조

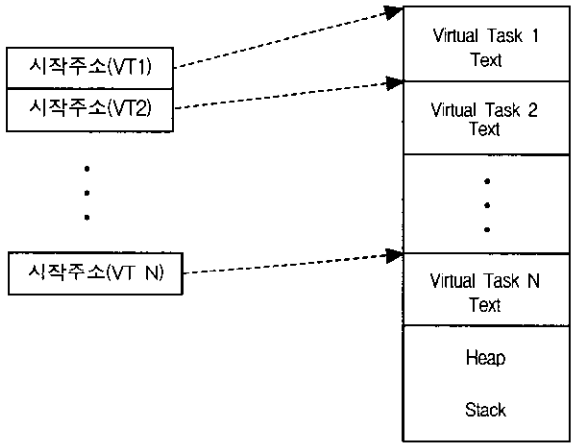
TCB에 관련 task의 수행 내역이 모두 담겨져 있으므로 task에 대한 관리는 TCB를 기준으로 이루어진다. Task 사이의 제어권 이양방식이 Task 자의적일 경우 비선점(non-preemption)이라 하고 커널에 의한 이양방식을 선점(pre-emption)이라 한다. 실시간을 요하는 응용기능이 동작되는 경우 선점 방식을 적용하는 것이 일반적이다.

2.2.2 추상 커널 계층 VTask

추상 커널 계층에서 지원하는 Task를 가상 Task라 하며 VTask로 명명한다. Target OS에서 지원되는 Task는 Kernel 영역에서 Task의 관리 목적의 TCB((그림 2) 참조)와 사용자 영역에서 독립된 Heap/Stack/Text 등을 보유하는 실제 Task로 구성된다.

이에 비하여 VTask는 소규모의 VTCB(Virtual Task Control Block)과 전체 VTask에 대한 한 개의 Heap/Stack과 각 VTask 마다 독립된 Task Image인 Text로 구성된다.

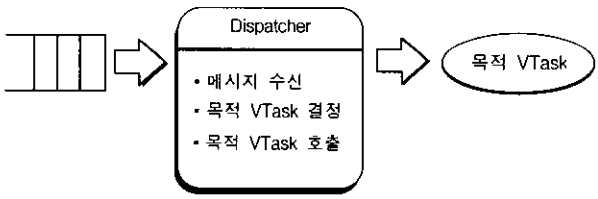
(그림 3)과 같이 실제 VTask를 관리하기 위한 관리자료가 간단하다. 관리자료는 각 VTask에 대한 시작 위치를 지시하는 시작주소를 한 엔트리로 하여 리스트로 구성된다. 리스트 내의 위치가 VTask의 ID(Identification)가 된다.



(그림 3) 추상 커널 계층 가상 Task 구조

2.2.3 추상 커널 계층 Scheduler

추상 커널 계층 Scheduler의 동작은 message-driven 방식이다. 즉 추상 커널 계층 전체의 main은 scheduler로서 처리할 메시지가 존재하는 경우에 동작된다. 처리할 메시지가 발생되는 경우 Scheduler는 관련 VTask를 호출한다. 어떤 VTask가 어떤 메시지에 대응하는지에 관한 정의는 추상 커널 계층 하에서 동작되는 응용 프로그램의 설계 시 정의된다. 또한 메시지의 구조는 제5장 추상 커널 계층 자료구조/알고리즘에서 설명한다.



(그림 4) 추상 커널 계층 Dispatcher

추상 커널 계층 Scheduler는 이후 dispatcher라고 정의하고, 메시지 큐로부터 한 메시지를 수신하여 메시지의 내용을 보고 목적 VTask를 호출하는 역할을 담당한다.

3. 추상 커널 계층 개념구조

3.1 추상 커널 계층 개념 구조도

본 연구에서 설계한 추상 커널 계층의 개념 구조도는 (그림 5)와 같다. 추상 커널 계층은 Dispatcher, 메시지 큐, 타이머, Virtual Task Control Table, Vtask 등으로 구성되고, 각각은 다음의 기능을 수행한다.

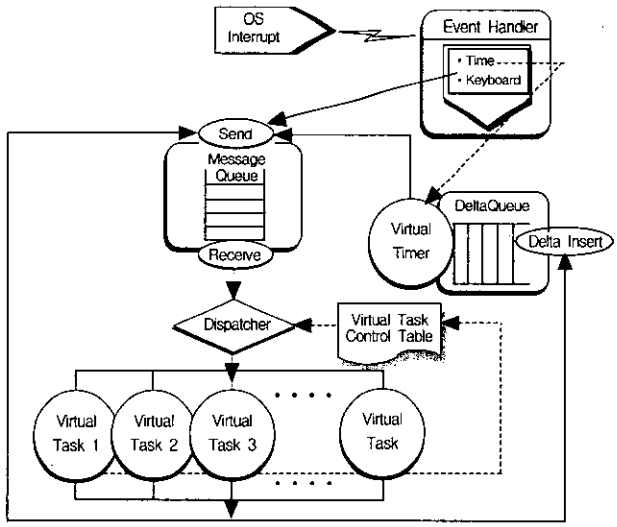
3.2 추상 커널 계층 구성 요소

① Dispatcher

메시지 큐로부터 한 메시지를 추상 커널 계층에서 정의한 API 중 하나인 "Receive"로 수신하여 대상 VTask를 결정하고 정의된 인자(parameters)로 호출한다.

② 메시지 큐

FIFO 방식의 메시지 큐로서 "Send", "Receive" API를 이용하여 접근 할 수 있다. 모든 VTask는 "Send" API를 이용하며 Dispatcher 만이 "Receive" API를 이용할 수 있다.



(그림 5) 추상 커널 계층 개념 구조도

③ 타이머

UNIX의 delta-queue 방식을 도입하였다. 각 VTask에서 타이머가 필요한 경우 "GetTime" 이라는 API를 이용한다. 타이머의 취소는 "DeleteTime" API를 이용한다. 대응하는 delta-queue의 한 엔트리가 "delete" 된다. 원하는 양만큼의 시간이 지난 후 관련 VTask에 "send" API를 이용하여 알린다.

④ Virtual Task Control Table

각 VTask의 현재 진입 점을 유지하고 있는 목록이다. 이 목록에 준하여 각 VTask는 현재의 처리 부분이 유지되며 Dispatcher에 의하여 호출된다. 다음 상태로의 전이는 "setStateTransition"이라는 API를 이용한다. 상태의 유지는 VTask의 책임이다.

⑤ VTask

VTask는 제어권에 관하여 수동적이라는 점에서 일반 subroutine의 처리개념과 동일하나 차이점으로서 상태전이를 기반으로 구현되어야 한다는 점이 다르다. 즉 다음 루틴의 호출은 관련 VTask의 호출로 대응되므로 호출은 "Send" API에

의하여 이루어진다. 자신은 다음 자료의 처리를 위하여 상태 전이가 필요하며 이러한 사실을 "setStateTransition" API를 이용하여 선언한다.

4. 추상 커널 계층 자료구조/알고리즘

4.1 Dispatcher

추상 커널 계층은 자신의 수행시점을 상위 커널인 Target OS로부터 통보 받고 다음의 두 가지 흐름을 거쳐 종료된다.

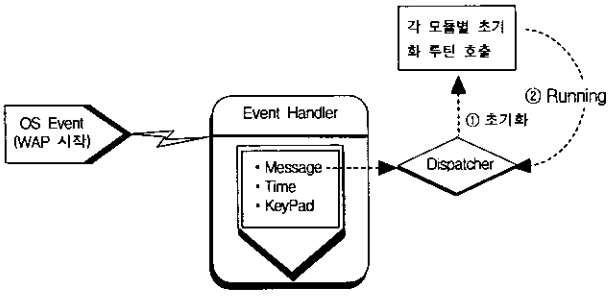
- 초기화, running, 정상종료
- 초기화, running, 비정상종료

이러한 과정에 제어권을 주도적으로 잡고 VTask에 정의된 내역을 구동시키는 임무는 Dispatcher가 담당한다.

4.1.1 초기화 과정

휴대 단말로부터 명시적인 WAP 브라우저의 시작을 알리는 이벤트에 대하여 최초로 시작되는 부분을 설명하고 있다.

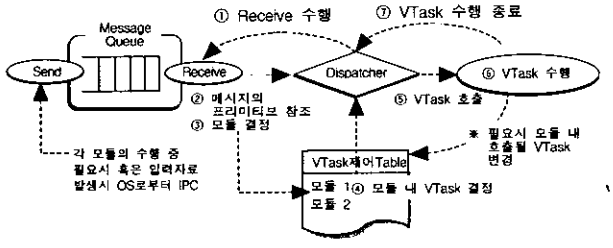
Dispatcher는 각 모듈마다 유일한 이름을 가지고 있는 초기화 프로시저를 호출하는데, 이 프로시저는 각 모듈에서 제공한다고 가정한다. 각 모듈별 프로시저에서 초기화 프로시저를 호출한 후 정상적인 "running" 상태로 전이한다. 이 외에도 메시지 큐의 초기화 인터럽트 관리를 위한 초기화 등의 과정을 수행한다.



(그림 6) Dispatcher를 통한 모듈별 초기화

4.1.2 Running 과정

초기화 과정을 완료한 Dispatcher는 Running 상태로 진입하게 된다. Running 상태는 파괴 상태에 이르기 전까지



(그림 7) Dispatcher Running 과정

WAP의 총체적 정상동작을 수행하는 과정이다. 개념적으로 (그림 8)과 같다.

(그림 7)에서 ①→⑦ 까지의 흐름을 한 주기라고 정의하며, 이들 각 요소에 대한 설명은 아래와 같이 번호별로 대응되어 제시한다.

① Receive 수행

Running 상태는 메시지 큐에 메시지가 존재하는 상태에서 진행이 되며 존재치 않는 상황에서는 "idle"상태이다. 이러한 진행은 "receive" 내부에서 일어난다. 현재 추상 커널 계층은 이와 같은 방식으로 구현되어 있다.

② 메시지 프리미티브 참조

일단 메시지를 "receive" API로 수신한 다음 수신된 메시지의 프리미티브 필드를 참조하여 목적 모듈을 결정한다.

③ 목적 모듈 결정

목적 모듈의 결정은 "프리미티브/1000 - 1"로 하는데, 이는 각 모듈 별 프리미티브의 범위가 1000이기 때문이다. 결과로 나온 정수 값의 몫이 목적 모듈이 된다.

④ 모듈 내 VTask 결정

정수의 목적 모듈 값을 인덱스로 하여 controlTable-OfVTask(VTask 제어 테이블)의 내용이 해당 모듈의 현재 VTask에 대한 function pointer이다. 다음은 controlTable-OfVTask의 구조이다.

```
#define     Mod1   1
#define     Mod2   2
#define     Mod3   3
#define     Mod4   4
#define     Mod5   5

typedef VTask (*vtask)() ;
 VTask controlTableOfVTask[numOfVTask] ;
```

⑤ VTask 호출

단계 "①"에서 수신된 메시지의 내용을 인자로 그리고 단계 ④에 결정된 function pointer가 지시하는 VTask를 호출한다. 다음은 이러한 내용을 반영하고 있는 소스 코드이다.

```
/* n은 목적 모듈 번호이며, m은 수신된 메시지의 pointer이다.
 메시지에 관한 자세한 내용은 관련 절을 참조*/

(controlTableOfVTask[n])(m->source, m->primitive, m->data) ;
```

인자로서 source, primitive 및 data등이 전달한다. 이로서 제어권은 추상 커널 계층을 떠나 관련 VTask로 이양되며 VTask는 자신의 일을 처리한 후 제어권을 다시 추상 커널 계층로 반환한다.

⑥ VTask 수행

만일 현재 주기에서 VTask가 자신의 일을 처리하는 과

정에서 자신이 종료된 후에 다음 주기에 다른 VTask를 수행을 원한다면 다음 주기의 "과정 ④ 모듈 내 VTask 결정"에서 이를 반영하도록 하기 위하여 아래와 같은 명령을 추상 커널 계층에 전달하여야 한다.

```
/* 아래의 문장은 Mod2 모듈이 현재의 VTask(에 disconnect 처리)가 종료 된 시
점에서 다음 주기에 수행되어야 할 VTask가 Mod2Idle 임을 지시하고 있는 추상
커널 계층 API 이다 */
#include      controlTableOfVTask.h

nextState(Mod2, Mod2Idle) ;
```

위의 "nextState" API는 controlTableOfVTask의 Mod2(인덱스1) function pointer를 "VTask Mod2Idle()"로 지시하게 한다. 다음 주기에서 VTask를 호출하기 위한 Dispatcher의 "(controlTableOfVTask[n])(m -> primitive, m -> data) ; "는 Mod2Idle로 제어권을 넘기게 된다.

⑦ VTask 수행종료

현재 주기의 관련 VTask가 종료된 후에 제어권이 추상 커널 계층의 Dispatcher로 전달된다. 이 과정에서 Dispatcher의 내부 정리를 통하여 현재의 주기를 완료하고 다음 주기로서 다시 "① Receive 수행 : 단계로 진행된다.

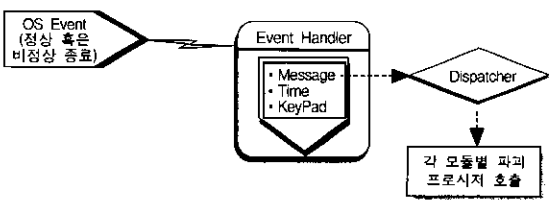
4.1.3 종료 과정

추상 커널 계층은 각 모듈의 파괴 프로시저에 대한 정보를 가지지 않으므로, 각 모듈마다 유일한 파괴 프로시저 이름으로서 Dispatcher에 의하여 실행시키도록 할 수 있다.

파괴프로시저가 호출되는 시점은 단말기 사용자에 의한 WAP 브라우저의 명시적 종료 혹은 단말기 자체의 상태(battery 탈장 혹은 완전소모, 오 동작에 의한 reset 등)에 의한 종료 두 가지를 들 수 있다. 어떤 경우가 되든지 WAP 입장에서는 다음의 두 가지 중 하나의 상태에서 종료를 하게 된다.

- 네트웍 측(WAP gateway, proxy 혹은 server)과의 상호동작이 없는 상태의 종료
- 네트웍 측과의 상호동작이 진행 중인 상태에서의 종료

위의 두 가지 경우를 모두 고려하여 각 모듈의 종료 프로시저는 적절하게 대처를 하여야 한다. WAP 프로토콜 모듈 입장에서는 세션, 트랜잭션 등의 마무리로 메모리 상의 찌꺼기가 남지 않도록 하여야 한다. 이러한 상황을 적절히 대처하지 않는 경우에 다음의 WAP 동작에 지대한 영향을 줄 수 있다. 추상 커널 계층 입장에서 종료의 처리는 (그림 8)과 같다.



(그림 8) 추상 커널 계층 종료 처리

① OS로부터의 명시적인 종료 이벤트를 추상 커널 계층 이벤트 처리기가 수신

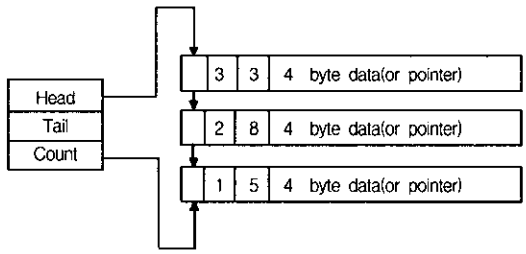
② 추상 커널 계층 이벤트 처리기가 Dispatcher의 종료 기능을 호출

③ Dispacher가 각 모듈의 종료 프로시저를 호출

4.2 메시지 큐

4.2.1 메시지 큐 자료구조

추상 커널 계층은 전체적으로 하나의 메시지 큐가 존재하며, 형태는 (그림 9)와 같다.



(그림 9) 추상 커널 계층 Message queue

메시지 큐는 FIFO로 삽입/삭제가 이루어진다. FIFO의 구성은 연결리스트(Linked List)이며 시작은 "Head" 포인터가 끝은 "Tail" 포인터가 지시하고 있다.

- "Head"가 지시하고 있는 메시지가 Dispatcher의 메시지 큐에 대한 "receive" 요청에 의하여 전송될 메시지이며 "receive"되는 순간 삭제되는 메시지이다.
- "Tail"은 여타의 VTask에 의하여 메시지 큐로 "send"될 때 메시지 큐에 묶어질 지시자 이다.
- "Count"는 현재 메시지 큐에 존재하는 메시지의 수를 나타낸다.

메시지 자체의 구성은 아래와 같이 next, source, primitive 및 data등의 네 개의 필드로 구성된다.

*next	source	primitive	data(4 byte)

- next : FIFO로 적용을 위하여 다음 메시지에 대한 포인터이다.
- source : 해당 메시지를 전송한 VTask가 속한 모듈의 이름이다. 모듈은 "controlTableOfVTask.h"에 정의된다.
- primitive : 각 모듈에서 고유하게 정의한 프리미티브이다. 이를 근거로 하여 Dispatcher는 목적 VTask를 결정한다.
- data : 송/수신 VTask들 사이에 필요하며, 그들만이

알 수 있는 부가정보를 담고 있는 필드이다. 부가정보가 4 byte 이내이면 메시지 내의 "data" 필드에 직접 표현한다. 4 byte를 상회하는 자료인 경우 별도의 변수(array, structure 등)에 자료를 담고 이 자료에 대한 지시자를 "data" 필드에 배정한다.

4.2.2 메시지 큐 API

① send API

send API는 한 VTask가 다른 VTask에 메시지를 보낼 때 사용한다. 전송하는 주체는 원하는 자신의 식별자와 수신 주체의 식별을 위한 프리미티브와 자료를 보내야 한다. send API는 제공된 자신의 식별, 프리미티브, 자료를 Dispatcher가 관장하고 있는 메시지 큐에 FIFO 규칙을 적용하여 삽입한다. send API의 형식은 아래와 같다. 모든 VTask 공히 의사 전달을 위하여 사용할 수 있는 API이다.

send(VTask source, Primitive prmtv, Data *data)	
인자(Parameter)	설 명
source	송신하는 VTask의 식별자
prmtv	목적 VTask가 처리하여야 하는 기능의 식별자
* data	prmtv와 연관된 부가적인 자료이며 송/수신 VTask 만이 알수 있는 자유 형식의 자료

② receive API

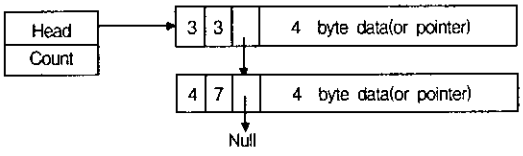
receive API는 한 VTask가 다른 VTask로부터 보내진 의사(명령) 내역을 수신하는 수단이다. 추상 커널 계층에서는 Dispatcher 만이 이 API를 사용할 수 있다. 메시지 큐에 적용되는 FIFO 방식으로 한 메시지를 삭제한다. 형식은 다음과 같다.

recv(VTask *source, Primitive *prmtv, Data *data)	
인자(Parameter)	설 명
* source	송신한 VTask의 식별자이며, recev 호출자인 Dispatcher와 메시지 큐를 처리하는 recev API 간의 자료 공유를 위하여 포인터 처리
* prmtv	목적 VTask가 처리하여야 하는 기능의 식별자, 동일 이유로 포인터 처리
* data	prmtv와 연관된 부가적인 자료이며 송/수신 VTask 만이 알수 있는 자유 형식의 자료로서 메시지의 자료 필드 내용의 복사 본

4.3 델타 큐(Delta-queue)

한 개의 실제 타이머를 마치 여러 개의 타이머가 있는 것처럼 서비스하는 기능이 구현된다.

4.3.1 델타 큐 자료구조



(그림 10) Delta queue 구조

단일 연결 리스트이며, 그 형태는 (그림 10)과 같다.

델타 큐의 시작은 "head" 포인터가 마지막의 노드는 "null"을 값을 가진다.

- "Head"는 일련의 노드들로 구성된 리스트를 지시한다. 리스트의 각 노드는 특정 VTask에서 요청한 시간 지연 값과 부가 정보이다. 일정단위 시간의 흐름마다 시간 인터럽트에 의하여 가상 타이머 VTask가 이를 인지하는 순간 Head가 지시하고 있는 처음의 node의 키 값이 하나 감소된다. "0" 값까지 감소되는 순간 가상 타이머 VTask는 "send" API를 호출한다. 그리고 처음 노드는 삭제된다.
- "Count"는 현재 델타 큐에 존재하는 노드의 수를 나타낸다.

노드 자체의 구성은 아래의 그림처럼 next, source, primitive 및 data등의 네 개의 필드로 구성된다.

key	primitive	*next	data(4 byte)

- key : FIFO로 적용을 위하여 다음 메시지에 대한 포인터이다.
- primitive : 요청시간을 대표하는 타이머의 이름이다. 일예로서 WSP_ELAPSE_T1, WML_ELAPSE_T2등을 들 수 있다. 이 역시 일반 프리미티브와 동일하게 유일한 값을 가져야 한다.
- *next : 델타 큐 적용 알고리즘에 의한 다음 노드에 대한 포인터이다.
- data : 가상 타이머를 요청하는 VTask 입장에서 요청 시간 경과 시 부가적으로 알아야 하는 정보를 담고 있다. 참고적으로 요청시간의 경과는 "send" API에 의한 메시지 수신으로 알려지게 된다. 부가정보가 4 byte 이내이면 메시지 내에 "data" 필드에 직접 표현한다. 4 byte를 상회하는 자료인 경우 별도의 변수(array, structure 등)에 자료를 담고 이 자료에 대한 지시자를 "data" 필드에 배정한다.

4.3.2 큐의 운영

큐의 운영은 기본적으로 VTask들로부터의 시간 요청/삭제와 타이머 인터럽트에 의한 최초 델타 리스트 노드의 key 값 감소 등의 두 가지이다. 타임 인터럽트에 의한 key 값 감소 후 "0" 값에 다다르면 해당 델타 노드의 프리미티브와 자료를 인자로 하여 아래와 같이 send API를 호출한다.

```
send(VTime, deltaNode->primtv, deltaNode->data) ;
```

VTime은 가상 타이머의 VTask 이름이다. "0"의 값을 조사하는 경우 유의하여야 할 점이 있다. 이 전에 동일한

지연시간을 요청하는 VTask가 여러 개 있는 경우 델타 큐의 삽입 알고리즘에 의하여 해당 수만큼의 노드가 모두 "0"이 된다는 사실이다. 따라서 그 만 큼의 send API를 호출하여야 한다. 이러한 기능을 수행하는 델타 라이브러리를 "deltaOperate"라 하며 "deltaOperate"는 타이머 인터럽트가 발생하는 경우 이벤트 처리기로부터 직접 호출된다.

4.3.3 델타 큐 API

① GetTime API

각 VTask에서 타이머가 필요한 경우 "GetTime" 이라는 API를 이용한다. "GetTime"의 호출로 인하여 가상 타이머가 관리하고 있는 delta-queue에 일정 양식의 형태로 "insert"가 실시된다.

GetTime(Time tic, Primitive prmtv, Data *data)	
tic	요청 지연 시간
prmtv	요청 지연 시간 후 자신에게 알려 주는 프리미티브, 일 예로서 WML VTask가 현재 WML_START_T1을 요구하는 시점에서 해당 타이머가 지연된후 자신에게 알려주어야 하는 프리미티브를 WML_EXPIRE_T1이라한다면 이를 프리미티브로 정의할 수 있다.
* data	prmtv와 연관된 부가적인 자료이며 송/수신 VTask만이 알수 있는 자유 형식의 자료

② DeleteTime API

타이머의 취소는 "DeleteTime" API를 이용한다. 대응하는 delta-queue의 한 엔트리가 "delete" 된다. 이러한 API가 이용되는 경우는 프로토콜의 정상적인 흐름에서 주어진 시간 내에 ACK등의 응답이 수신되는 경우에 호출된다.

DeleteTime(Primitive prmtv)	
prmtv	GetTime에서 정의한 프리미티브

4.4 이벤트 처리기

이벤트 처리기는 추상 커널 계층이 구현되는 시스템마다 상황이 다르다. 그러나 추상 커널 계층의 모듈입장에서 이벤트 처리기와 직접 상호 동작하는 상황이 아니므로 직접적인 관련은 없으며, 아래의 이벤트를 반드시 고려하여야 한다.

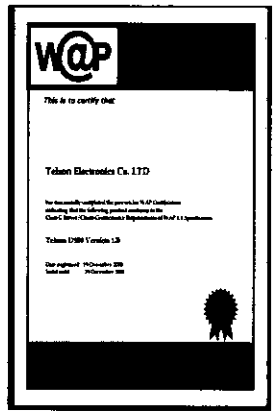
- Time 이벤트, Time 인터럽트 : 단위 틱당 타이머 H/W로부터의 인터럽트에 대응되는 처리부이며, tic에 의한 단위시간의 조정 후 Delta Timer의 "deltaOperate"를 호출한다. 특히 단위 시간의 조정은 tic의 누적치가 현재 정의된 시간에 해당하는 누적치를 초과하는지에 대한 여부의 조사이다.
- 입력 인터럽트 : 입력 자료의 인식을 위한 인터럽트이다. 이 인터럽트에 의하여 입력된 자료는 추상 커널 계층의 입력자료 처리 VTask로 전송된다. 이 VTask는 추상 커널 계층 내부 VTask이며 문자단위의 처리를 기본으로 한다. Phone인 경우 KeyPad 인터럽트로 대체된다.
- socket 인터럽트 : 궁극적인 무선 데이터서비스의 형태는 UDP/IP 방식이며, 이는 API 차원에서 socket layer로 지원된다. 이는 유선상의 socket layer의 기능과 유사하므로 유/무선의 차이는 없다. 따라서 통신 방식은 UDP/IP socket 통신 모델을 준수하여 구현된다. 특히 데이터의 수신시 socket 인터럽트를 통한 관련 VTask로의 전송이 이루어지고, 이는 추상 커널 계층 내부 VTask이며 문자열 처리를 기본으로 한다.

5. 구현 및 결과

2, 3, 4장에서 기술한 바에 따라 추상 커널 계층 모델은 다음의 각 환경에서 구현되었으며, 그 결과를 간략하게 기술한다.

embedded 시스템에 적절한 기능을 구현하려면 많은 제약이 따르며, 개발비용뿐 만 아니라 개발 시간이 절대적인 변수가 된다. 따라서 WAP을 이동단말기기에 구현하는 시간과 비용 절약을 위해서 Window 상에서 수행되는 emulator의 개발을 본 연구를 위해 먼저 진행하였다.

(그림 11) WAP Emulator 실행



(그림 12) WAP 인증서

(그림 11)은 PC상에서 수행되는 Emulator의 실행 장면 중의 하나이며, 이는 SK Telecom의 n-top 서비스 초기화면이다. 구현된 Emulator는 WML Content 뿐만이 아니라 WML Script로 작성된 모든 컨텐츠를 보여준다. 그림에서 Code Editor는 Emulator와 Gateway 사이에 전달되는 패킷

정보를 분석하여 보여주며, WML Editor는 전송되는 WML Source나 WML Script를 보여준다.

또한 Telson에서 생산한 D100 phone에 탑재하여 Opengroup에서 시행하는 WAP 인증을 받았으며, WAP 인증을 받은 client 중 CDMA 방식으로는 유일하다.

인증 시 시험 대상으로 지정 받은 gateway는 Nokia Artus Messaging Platform WAP2.2, UP.Link 4.2.1, Ericsson WAP GW/Proxy R4B 세 가지이며, 이들을 대상으로 지정된 인증 시험을 거쳐 인증서를 취득하였다.

(그림 12)는 Opengroup으로부터의 인증서이다.

인증된 mobile phone과 관련된 정보는 http : //www.opengroup.org/wap/cert/register.html에서 찾아볼 수 있으며, 등록된 정보 중 Telson과 관련된 정보가 본 구현을 통한 것이다.

6. 결 론

무선 인터넷의 도약에 따른 관련 기술들의 구현은 이동단말 제조업체나 서비스 제공자들에게는 단기간 내에 해결되어야 할 과제이다. 그러나 다양한 플랫폼과 개발 여건에 적응하려면 많은 시간이 필요한 부분이기도 하다. 본 논문에서는 이동 단말에 적합한 WAP 구현시 그 기간을 단축하고, 일관적인 인터페이스를 제공하는 추상 커널 계층을 설계하고, Target OS나 하드웨어에 종속된 부분을 최소화함으로써 다양한 플랫폼에 용이하게 이식될 수 있는 시스템을 구현하였다. 더불어 본 논문에서 제시한 추상 커널 계층은 REX 운영체제를 채택하고 있는 Mobile phone, Palm 운영체제를 채택하고 있는 PDA 및 Windows 하에서 수행되는 PC Emulator 등으로 구현하여 그 실용성을 입증하였다.

참 고 문 헌

[1] WAP Forum, "Wireless Application Protocol white paper," Wireless Internet Today, June, 2000.

[2] WAP Forum, "Wireless Application Environment Specification," April, 1998.

[3] WAP Forum, "Wireless Application Protocol Architecture Specification," April, 1998.

[4] Michael Barr, "Programming Embedded Systems," O'Reilly, 1999.

[5] QUALCOMM, "Real-Time Executive System (REX)," QUALCOMM Inc., 1999.

[6] 3Com, " Palm OS Programming Development Tools Guide," 1999.

[7] Neil Rhodes and Julie McKeehan, "Palm Programming : The Developer's Guide," O'Reilly, 1998.

[8] David Pogue, "Palm Pilot : The Ultimate Guide, 2nd Edition," O'Reilly, 1999.

[9] WAP Forum, "Wireless Markup Language Specification," April, 1998.

[10] WAP Forum, "WMLScript Language Specification," April, 1998.

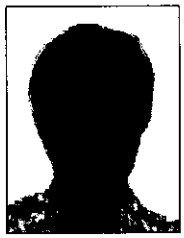
[11] WAP Forum, "WMLScript Standard Libraries Specification," April, 1998.

[12] WAP Forum, "Wireless Session Protocol Specification," April, 1998.

[13] WAP Forum, "Wireless Transaction Protocol Specification," April, 1998.

[14] WAP Forum, "Wireless Datagram Protocol Specification," April, 1998.

[15] WAP Forum, "Wireless Transport Layer Security Specification," April, 1998.



강 영 만

e-mail : ymkang@cs.yosu.ac.kr

1985년 광운대학교 전자계산학과 졸업(학사)

1987년 광운대학교 대학원 전자계산학과 (이학석사)

2001년 현재 광운대학교 대학원 전자계산학과 박사과정 수료

1985년~1991년 한국전자통신연구소 선임 연구원

1991년~현재 여수대학교 전자통신정보공학부 부교수

1999년~현재 텔슨전자 수석연구원 겸임

관심분야 : 무선 인터넷, 망관리 등

한 순 희

e-mail : shhan@yosu.ac.kr

1983년 경북대학교 전자공학과 졸업 (공학사)

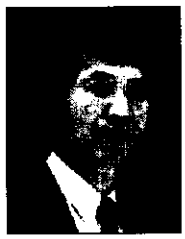
1985년 광운대학교 대학원 전자계산학과 (이학석사)

1993년 광운대학교 대학원 전자계산학과 (이학박사)

1989년~현재 여수대학교 전자통신정보공학부 교수

1999년~현재 텔슨전자 수석연구원 겸임

관심분야 : 무선 인터넷, 객체지향 프로그래밍 등



조 국 현

e-mail : khcho@cs.kwangwoon.ac.kr

1977년 한양대학교 전자공학과(공학사)

1984년 일본 동북대학교 대학원 (공학박사)

1984년~현재 광운대학교 교수, 광운대학교 이과대학 학장, 전자계산소장 역임

1999년~현재 전산대학원장, 한국정보과학회 이사, 개방형 컴퓨터통신연구회 부회장 역임, 현재 개방형 컴퓨터통신연구회 회장

관심분야 : 정보통신망 관리, 분산처리, 멀티 캐스팅 및 정보통신 분야의 표준화