VxWorks Base Java Virtual Machine 개발

박 상 현, 고 재 진, 민 수 영
전자부품연구원
전화 : 031-610-4064 / 핸드폰 : 011-9981-0217

Vxworks Base Java Virtual Machine Development

Sang-Hyun Park, Jae Jin Ko, Soo Young Min
Korea Electronics Technology Institute
E-mail : shpark@keti.re.kr

Abstract

Nowadays, many users use internet and many set-top box needs browser for Internet. so we need to develop a Java Virtual Machine to improve browser's performance.

This paper has been studied a Java Virtual Machine based on Real-Time OS VxWroks. Java Virtual Machine handles Java byte-code quickly in Browser.

So, this paper has designed module and interfaces for Embedded Browser and Implemented them.

I. 서론

최근에 핫 이슈가 되고 있는 무선 인터넷 단말들의 브라우저들은 대부분 외국의 브라우저를 탑재하여 사용하므로 막대한 로열티를 지불하고 있는 실정이다. 인터넷 강국을 지향하는 우리나라에서는 앞으로 수많은 서비스 및 단말장치가 개발 될 것으로 믿어 의심치 않는다. 이를 위해선 적은 비용과 개발기간의 단축을 지원하기 위해 플랫폼 독립적인 범용 브라우저를 개발이 필요하게 되었고, 이러한 브라우저개발의 일환으로 자바를 빠르게 처리할 수 있는 Java Virtual Machine의 개발이 필요하다.

본 논문은 Java 기반 범용 임베디드 브라우저 개발에 관한 연구 중 브라우저에서 자바를 처리할 수 있는 자바 가상 머신 개발에 관한 연구이다. 가상머신의 구현을 위해 리얼타임 OS인 VxWorks를 사용하고, Open VM Source를 VxWorks로 포팅하고, Thread 기반의 VM을 VxWorks의 Task 기반으로 개발하고 이와 관련된 Lock 메카니즘의 적용, 메모리관리 등에 관한 모듈을 중점적으로 개발하였으며, 브라우저와의 연동을 위한 인터페이스 개발을 디자인하였다.

II. Java Virtual Machine

2.1 Java Virtaul Machine의 기본 구조

자바 가상머신은 브라우저에서 Java applet, Java script 및 Java binary를 빠르게 처리해주는 역할을 한다. 본 연구에서 설계한 가상 머신은 Open VM인 Kaffe를 참고하여 JDK 1.2를 지원하도록 설계하였다. 그림 1.은 자바 가상 머신의 구조도 이다.

2.2 클래스 로더 서브 시스템의 구조

클래스로더 서브시스템은 그림 2.와 같이 로딩(Loading)부분,링킹(Lingking)부분,초기화(Initialization)부분으로 구성된다. 로딩부분은 Method를 위한 바이너

리 데이터를 찾아서 읽어오며, 링킹부분은 읽어온 데이터에 대한 정확도를 검증(verification), 클래스 데이터를 위한 메모리 할당, 초기화 등을 수행하는 예비(preparation), 심볼릭 참조를 직접 참조로 변환시키는 확정(resolution)기능을 수행한다. 초기화 부분에서는 클래스 변수를 초기화하는 자바 코드를 호출하는 기능을 수행한다.

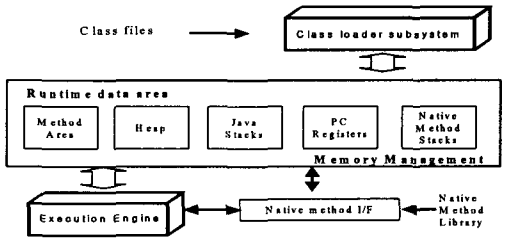

그림 1. 자바 가상 머신의 구조

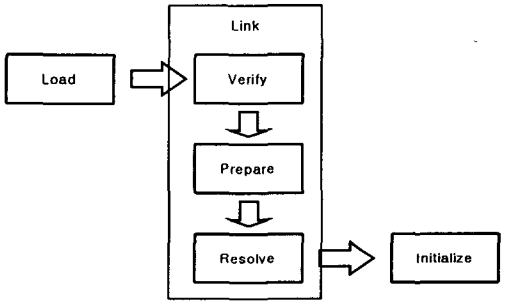

그림 2. 클래스 로더 서브시스템

2.3 Runtime Data 영역

자바 가상 머신의 Runtime Data 영역은 다음과 같이 구성하였다. 임베디드 시스템의 제한된 메모리를 효율적으로 사용하기 위하여 전체 메모리를 Heap영역으로 삼고, 각각의 목적에 따른 메모리를 할당받아 사용할 수 있게끔 설계를 하였다. 그림 2.은 Heap 영역에서의 Runtime Data Area를 나타낸 그림이다.

- Method Area : 읽어들여진 클래스에 대한 정보가 저장되는 곳으로 모든 참조되는 내용들이 저장된다.
- Heap :새로 생성되는 Object들을 위한 메모리 공간.
- PC Register : Program Counter가 저장되는 곳이다.
- Java Stack : 새로운 쓰레드가 실행될 때마다 생성되는 공간으로 하나의 쓰레드에는 하나의 Java Stack이 할당된다.
- Stack Frame : Java stack을 관리하기 위한 데이터 구조이다. 스택 프레임에는 local variables, operand stack, frame data들이 포함된다. 새로운 stack 프레임은 메쏘드가 호출될 때 생성된다.
- Native Method Stack : 자바 Method는 Native Method를 호출할 수 있으며, Native Method 역시 자바 Method를 호출 할 수 있다. 자바 Method가 Native Method를 호출할 때는 자바 Method가 JVM의 현재 상태에 대한 정보를 전달하며, Native Method가 자바 Method를 호출할 때는 Native Method Stack이 이전 Native Method를 반환할 필요가 있다.

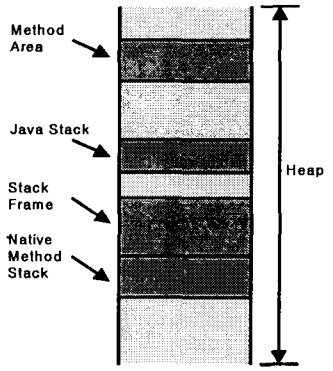

그림 3 Runtime Data Area

III. VxWorks 용 Java Virtual Macine

3.1 System Library Module 개발

System Library는 Open Java Virtual Machine의 Thread Module을 VxWorks의 Task 기반으로 동작할 수 있도록 개발된 Thread 관련 인터페이스 및 라이브러리이다. 개발된 System Libarary 모듈은 다음과 같고, 그림 4.는 System Library의 구성을 나타낸 것이다.

- Java Thread 관련 모듈 - Java library를 위한 Thread의 생성, 초기화, 삭제, 메모리 할당한다. Native thread관련 모듈에 의해 초기화된 java thread는 Task 형태로 동작한다.
- Native Thread 관련 모듈 - Native thread의 생성, 초기화, 삭제 및 스케줄링을 위한 모듈로서 Java Thread를 포함하고 있다.
- Lock Mechanism - Task의 synchronization을 위한 VxWorks Task Semaphore가 적용된다.

Memory Management - Growing Down 방식의 stack allocation에 따른 Memory 할당 방식이 적용된다.

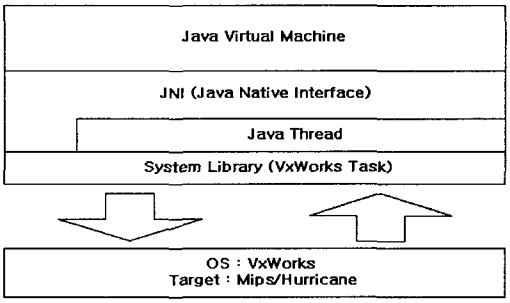

그림 4. System Library의 구성도

(1) System Library - Java Thread 관련 모듈

Java thread는 Java library의 함수의 동작을 위한 thread이다. VxWorks에서는 thread를 제공하지 않기 때문에. thread를 Task로 맵핑하여야한다. 또한 thread의 scheduling 또한 task 스케줄링으로 매핑 되어진다. 다음은 Java Thread 모듈에 필요한 인터페이스 및 그에 대한 동작을 기술한 것이다.

- jthread_init : java thread의 초기화
- jthread_createfirst :thread의 stack 영역을 할당한다.
- jthread_create : 새로운 Java thread를 생성한다. 시스템 호출 taskSpawn()을 이용 새로운 task를 생성.
- jthread_destroy : task를 위해 할당된 memory를 해제한다. 시스템 호출 SemBCreate에 의해 할당된 Semaphore를 해제한다.

(2) System Library - Native Thread 관련 모듈

Native thread는 JNI (Java Native Interface) 의 호출에 의해 동작하는 Thread이다. Native thread는 다시 Java Thread를 호출하여 Java Code에 대한 동작을 task형태로 동작하도록 인터페이스 역할을 한다. 다음은 Native Thread 모듈에 필요한 인터페이스 및 그에 대한 동작을 기술한 것이다.

- initThreads(void) : thread를 초기화.
- initNativeThreads(int nativestacksize) : main thread의 stack size를 초기화하고 Java thread 모듈 초기화를 호출한다.
- createInitialThread(const char* nm) : 초기 thread를 생성한다.
- startThread : Java library의 start 호출에 의해 생성된 native thread를 동작시킨다.
- finalizeThread : job이 끝난 thread 삭제하고 메모리를 해제한다.
- stopThread : Java library의 stop 호출에 의해 Java thread의 stop을 호출.

(3) System Library - Task의 관련 모듈

Task의 생성

Task의 생성을 초기화하기 위해서는 다음의 함수들이 이용된다. 그림 5.는 태스크 생성에 관한 코드이며 그림 6.은 태스크의 삭제를 나타낸 코드이다.

- *taskInit()* : *Initialize a new task.*
- taskVarInit() : task variable Initialize.
- taskPrioritySet(task_id, priority) : task의 우선순위 부여.

초기화후 다음 함수들에 의해 Task가 생성된다.

- taskVarAdd() : task variable을 task에 삽입.
- taskSpawn() : 새로운 task의 생성. 플래그에 정의된 함수를 호출, 및 argument를 passing 한다.

```
/* KETI Task Spawn */
jtid->VxThread, TaskId = taskSpawn(
                NULL,   /* task name */
                100-20 + pri, /* vxworks priority conversion */
                VX_FP_TASK | VX_PRIVATE_ENV | VX_NO_STACK_FILL | VX_UNBREAKABLE,
                (int) threadStackSize,
                (FUNCPTR)jtid-> func,
                0,      /* arg for firstStartThread : VxWorks first int arg */
                0,0,0,0,0,0,0,0,0);
```

그림 5. Task의 생성

Task의 삭제

- taskVarDelete() : task로부터 task variable 삭제.
- taskDelete() : task의 삭제.

```
/* free a thread context and its stack */
jthread_destroy(jthread *jtid)
{
        ...
        semDelete(TaskCreateSync);              /* KETI */
        free(jtid);
        free(currentJThread);                   /* KETI */
        taskVarDelete(0,(int *)&currentJThread); /* KETI */
        ...
}
```

그림 6. Task의 생성

3.2 Lock Mechanism

VxWorks Task의 Sync.를 위한 lock mechanism 적용. Java thread 모듈인 jthread_create에 의해 생성된 VxWorks Task는 SemGive() 호출에 의해 Block 되고, block된 Task는 firstStartThread(void* arg)에서

SemTake() 호출에 의해 Wake된다. 그림 7.는 semaphore를 이용한 Task간 동기화 설정을 나타낸다.

· Semaphore 생성 및 초기화
- SemBCreate() : binary semaphore의 생성과 초기화 한다.
· Lock : SemBCreate() 에 의해 초기화된 semaphore를 이용하여 task lock을 설정한다.
- jmutex_lock : jthread lock 인터페이스
- SemGive() : task operation 의 lock, block a task.
· Unlock
- jmutex_unlock : jthread unlock
- SemTake() : task operation 의 unlock, wakeup blocked task.
· Semaphore의 삭제
- semDelete() : SemBCreate에 의해 생성되었던 Binary semaphore를 삭제

```
extern SEM_ID TaskCreateSync;                    /* by KETI */

jthread_create(...)                              firstStartThread(void* arg)
{                                                {
    ...                                              ...
    semGive(TaskCreateSync);                         semTake(TaskCreateSync,WAIT_FOREVER);
    ...                                              ...
}                                                }
```

그림 7. Semaphore를 이용한 Task 동기화

3.3 Memory 관련 모듈

Task의 메모리를 할당을 위한 VxWorks의 stack은 Growing down 방식이 적용되며 필요한 인터페이스는 다음과 같다.

· newThreadCtx(int stackSize) : java thread의 생성시 호출.
· thread_malloc(size_t s) : stack 영역을 할당한다.
· thread_free(void *p) : 할당된 stack 영역을 해제한다.

Ⅳ. Browser와의 연동

본 장에서는 앞서 설계 및 구현한 모듈 및 인터페이스를 통한 브라우저와의 연동 관계는 그림 8.과 같이 브라우저를 통해 전달된 자바코드는 자바가상머신을 통해 처리하게되며 RTOS인 VxWorks를 통해 실시간적으로 작업을 처리하게 된다.

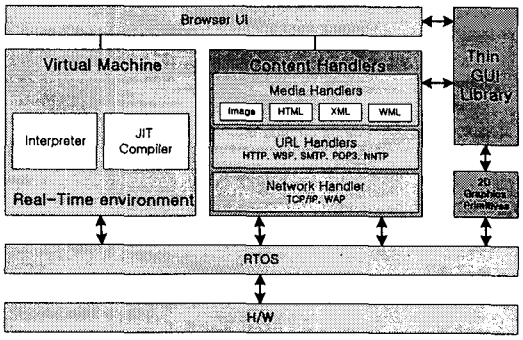

그림 8. 브라우저와 가상머신의 연동관계

Ⅴ. 결론

인터넷 사용자의 급속한 증가와 이를 활용하는 제품 영역이 PC에서 Non-PC, 즉 정보 가전제품으로 확대되는 추세에 따라 정보 가전제품의 수요 폭발이 기대되는 만큼 제품의 인터넷 구현에 필수적인 임베디드 웹 브라우저 수요 또한 상상할 수 없는 잠재력을 지니고 있다.

본 논문은 이러한 임베디드 웹 브라우저의 성능 향상을 위해 자바를 빠르게 처리할 수 있는 가상머신을 리얼타임 OS인 VxWorks기반에서 개발하였다.

본 논문에서 기술한 VxWorks기반 자바가상머신의 개발은 임베디드용 브라우저 개발에 있어 그 활용도가 높을 것으로 생각된다. 또한 아직 초기 시장형성 단계에 있는 임베디드용 웹 브라우저 시장에서 시장 선도 주자로서 주도적 역할을 할 수 있는 기반 기술의 확보라는 성과가 있을 것으로 사료된다.

참고문헌(또는 Reference)

[1] Jon Meyer, Troy Downing, "JAVA Virtual Machine," O'REILLY, March 1997.
[2] Bill Venners, "INSIDE the JAVA Virtual Machine," McGraw-Hill, 1998.
[3] Tim Lindholm, Frank Yellin, "The Java Virtual Machine Specification," Java Software, Sun Microsystems, Inc.
[4] Wind River Systems, Inc. "VxWorks Programmer's Guide 5.4," Mar 1999.
[5] Wind River Systems, Inc. "Tornado Training Labs Book", Tornado Training Workshop, Dec. 2001