Extend your world

Multi-Programming with WIPI Java

October 20, 2006

㈜XCE

Document History

Date	Author	Description
2005-03-29	Park J.H.	Ver. 1.0, Created.
2006-10-18	Hoon Chung	Modified terms and sentences. Refining.

Purpose of this Document

이 문서는 WIPI Java 플랫폼상에서 실행되는 Java Contents 를 개발하는 Contents 개발자들에게 WIPI Java 플랫폼이 지원하는 멀티프로그래밍에 대한 기술적인 내용을 소개한다.

NOTE

Contents

1 개요

WIPI JAVA 에서는 Multi-Programming 을 지원한다. 하지만, WINDOW 와 같이 멀티테스킹이 가능한 것이 아니라 Parent & Child 방식으로 Stack 구조로 동작한다.

멀티프로그램을 실행하기 위해서는 다음 클래스들의 사용방법을 명확히 할 필요가 있다.

- Kernel 클래스 : 멀티프로그래밍 지원하는 클래스
- Shared 클래스 : 프로그램간에 공유 가능한 메모리 제어
- DataBase 클래스 : 프로그램간에 공유 가능한 저장공간의 제공
- File, FileSystem 클래스 : 공유 가능한 파일 시스템 접근방법 제공

2 Kernel Class

WIPI JAVA 에서 Multi-Programming 을 지원하는데 핵심이 되는 클래스로, 현재 WIPI 1.2 에서 모든 기능을 지원하지는 않는다.

```
org.kwis.msf.core
Class Kernel
java.lang.Object
  |
  +--org.kwis.msf.core.Kernel
```

```
public class Kernel
extends java.lang.Object
```

2.1 Field Summary

Field Summary	
static int	**DIR_SHARED_READ_REQ_MASK** shared directory read가능
static int	**DIR_SHARED_WRITE_REQ_MASK** shared directory write가능
static int	**DIR_SYS_READ_REQ_MASK** system directory read가능
static int	**DIR_SYS_WRITE_REQ_MASK** system directory write가능
static int	**NETWORK_ACCESS_REQ_MASK** network API사용 가능
static int	**PRGTYPE_CAPP** C 애플리케이션
static int	**PRGTYPE_CDLL** C 동적로딩 라이브러리
static int	**PRGTYPE_JAVAAPP** Java 애플리케이션
static int	**PRGTYPE_JAVADLL** Java 동적로딩 라이브러리
static int	**PRGTYPE_JAVASYSDLL** Java 시스템라이브러리
static int	**SERIAL_ACCESS_REQ_MASK** serial API사용 가능
static int	**SYSTEM1_ACCESS_REQ_MASK** system group1에 속한 API사용가능
static int	**SYSTEM2_ACCESS_REQ_MASK**

Field Summary	
	system group2에 속한 API사용가능

2.2 Method Summary

Method Summary	
static int	**execute**(java.lang.String execName, java.lang.String[] args) 플랫폼에 설치된 프로그램을 실행시킨다.
static int	**getAccessLevel**() 프로그램의 접근 수준을 구한다.
static int	**getAMID**() 응용프로그램 관리자의 프로그램의 ID를 구한다.
static java.lang.String[]	**getExecNames**(java.lang.String prgName, java.lang.String version, java.lang.String vendor) 플랫폼에 설치된 애플리케이션 중 prgName(프로그램이름), version, vendor와 일치하는 애플리케이션 식별 이름을 반환한다.
static int	**getParentPrgID**() 부모 프로그램의 ID를 구한다
static int	**getPrgID**() 현재 프로그램의 ID를 구한다.
static int[]	**getPrgInfo**() 현재 동작중인 프로그램에 대한 정보를 얻는다.
static java.lang.String	**getPrgName**() 자기 자신의 프로그램 이름을 구한다.
static int	**letThrowExceptionWhenProgramExit**() 프로그램이 종료하면 이 함수를 호출한 thread에 ProgramExitException exception이 발생하게 만든다.
static int	**load**(java.lang.String execName, java.lang.String[] args) 플랫폼에 설치된 동적 로딩 라이브러리를 로딩한다.
static int	**mExecute**(java.lang.String symName, java.lang.String[] args) 프로그램에 설치된 프로그램을 실행시킨다.
static int	**mLoad**(java.lang.String symName, java.lang.String[] args) 프로그램에 설치된 동적 로딩 라이브러리를 로딩한다.
static void	**stop**(int prgID) 프로그램을 강제로 종료시킨다.

2.3 프로그램 실행방법

SKT에서 WIPI 상에서 실행되는 모든 프로그램과 DLL은 5 자리의 CID와 5 자리의 PID로 구성된 10 자리의 이름을 가진다. 프로그램의 리스트를 얻으려면 **Kernel.getExecNames()** 메소드를 이용한다.

하지만, 프로그램 리스트만으로 프로그램에 대한 정보를 알 수 없으므로, 멀티프로그램에 앞서 실행시킬 프로그램이 있는지 없는지 확인할 경우에만 사용한다.

프로그램을 실행 예를 들면 다음과 같다.

```
int ret = Kernel.execute("CID00SID00", args);
```

명령을 실행하려면 단말기에 "**CID00SID00**"이란 프로그램이 먼저 존재해야 한다. 그리고 `args`는 String 배열 객체를 생성해서 넣어야 하는데, 여기에 들어간 `args` 들은 `stratClet`, `startJlet()`에 파라미터로 전달된다

3 Shared Class

프로그램 간에 공유하는 메모리를 생성, 관리 해주는 클래스로, WIPI C 와 연계해서 프로그램 할 때 프로그램에 공유할수 있는 메모리를 제공해 준다. 하지만, 자바와 C 의 메모리 구조가 다르기 때문에 사용 방법에 주의를 요한다.

```
java.lang.Object
  |
  +--org.kwis.msf.core.Shared
```

```
public class Shared
extends java.lang.Object
```

3.1 Method Summary

Method Summary	
`static byte[]`	**`createBuf`**`(java.lang.String name, int size)` 공유 버퍼를 생성(create)한다.
`static void`	**`deleteBuf`**`(byte[] sharedBuf)`
`static void`	**`destroyBuf`**`(byte[] sharedBuf)` 생성된 공유 버퍼를 파괴한다.
`static byte[]`	**`getBuf`**`(java.lang.String name)` 공유 버퍼를 얻는다.
`static byte[]`	**`resizeBuf`**`(byte[] sharedBuf, int size)` 공유 버퍼의 크기를 변경한다.

공유 메모리를 생성하고 관리하는데 필요한 클래스로 WIPI JAVA 와 WIPI C 에서 동시에 프로그램간에 메모리를 공유할 수 있지만 다음과 같은 사용상의 주의 사항이 있다.

- WIPI_JAVA 는 기본적으로 메모리에 헤더 정보를 20byte 가지고 있다. WIPI C 로 공유 버퍼를 얻어서 사용할 경우에도 20 바이트 더 할당하고 앞에 20 바이트를 건드리지 않아야 한다.
- 공유버퍼로 생성한 공간에 대한 GC 나 Memory Compaction 이 이루어 지지 않는다.
- 공유버퍼는 4byte 단위로 할당이 된다.

실제 사용예

1) 자바에서 생성할 경우

WIPI JAVA

```
byte[] buf = Shared.createBuf("test_j", 100);  // 실제로 120바이트의 버퍼가 생성된다.
```

WIPI C

```
void* ptr = MC_knlGetSharedBuf("test_j");
M_Byte* mbuf = MC_GETDPTR(ptr);          // 자바에서 생성된 버퍼를 가져온다.
      //  주의사항은 처음 20바이트에 헤더 값을 바꿀경우 VM이 오류를 발생시킬 수 있다.
```

2) C 에서 생성할 경우

WIPI C

```
void* ptr = MC_knlCreateSharedBuf("test_c", 100 +20);
// C에서는 원래 사이즈보다 20바이트 크게 생성해야 한다.
// 또, 20바이트 이후부터 사용해야 한다.
```

WIPI JAVA

```
Byte[] buf = Shared.getBuf("test_c");
```

4 Database Class

MIDP 의 RMS 와 유사한 클래스한 클래스로. 데이터를 레코드 단위로 저장한다. 차이점은 레코드의 사이즈가 일정하다.

DataBase 는 생성할 때 다음의 3 가지 방법으로 공유 방법을 지정할 수 있다.

- `FileSystem.PRIVATE_ACCESS`
- `FileSystem.SHARED_ACCESS`
- `FileSystem.SYSTEM_ACCESS`

이중에 FileSystem.SHARD_ACCESS와 FileSystem.SYSTEM_ACCESS로 데이터 베이스를 열었을 경우 프로그램 간에 데이터베이스를 공유할 수 있다.

주의사항으로는 한 프로그램에서 데이터 베이스를 열었을 경우 다른 프로그램에서 데이터 베이스를 열수 없는 경우가 있다. 단말 특성에 따라 다르기 때문에 개발자의 주의를 요한다.

5 File & FileSystem Class

File과 FileSystem은 파일을 생성, 관리하는 클래스이다. 여기서 생성되는 파일은 DataBase와 마찬가지로 3 가지 접근방법을 가진다.

WIPI_JAVA에서 SHARED_ACCESS로 파일을 생성하면 파일은 "java"라는 공용 폴더에 생성된다. 따라서 WIPIC와 연계하는 프로그램은 WIPIC프로그램을 작성할 때 공유 폴더를 "java"로 명시해야 두 프로그램간에 파일을 공유 할 수 있다. 또, 모든 자바파일이 같은 폴더를 공유하기 때문에 사용상에 주의를 요한다.

6 멀티 프로그램 실행 예

6.1 JAVA 에서 다른 프로그램 실행

- "Kernel-정보"를 선택하면 현재 프로그램의 id 와 name 을 반환해 준다.
- "다른 프로그램"을 선택하면 "TEST_BUF"라는 이름으로 공유버퍼를 생성하여 버퍼에 String 값을 복사하고, "**B0001TEST0**" 프로그램이 시작된다.

```
import org.kwis.msf.core.*;
import org.kwis.msp.lcdui.*;
import org.kwis.msp.lwc.*;

public class MultiTest extends Jlet {

    byte[] buf;

    ShellComponent shell = new ShellComponent();
    ListComponent list = new ListComponent(ListComponent.SELECT_IMPLICIT);
    LabelComponent label = new LabelComponent();
    DialogComponent dg = new DialogComponent(label, "LOG",
DialogComponent.TYPE_OK);

    public static final String TITLE = "msf.core";
    public static final String[] LISTS = { "Kernel-정보", "다른프로그램" };

public void startApp(String[] args)
```

```
    {
        for (int i = 0; i < LISTS.length; i++) {
            list.append(LISTS[i], null);
        }
        list.setActionListener(new Action(), null);
        shell.setTitle(TITLE);
        shell.addComponent(list);
        shell.show();
    }

public void pauseApp()
    {
    }

    public void destroyApp(boolean unconditional)
    {
    }

class Action implements ActionListener {
        public void action(Component cmp, Object o)
        {
            int index = list.getSelectedIndex();

            switch (index) {
                case 1:
                    StringBuffer strBuf = new StringBuffer();
                    strBuf.append("prgID : ");
                    strBuf.append(Kernel.getPrgID());
                    strBuf.append('\n');
                    strBuf.append("prgName : ");
                    strBuf.append(Kernel.getPrgName());
                    label.setLabel( strBuf.toString() );
                    dg.doModal();
                    break;
                case 2:
                    buf = Shared.createBuf("TEST_BUF", 50);
                    byte[] tempBuf = "WIPI JAVA 전달".getBytes();
                    System.arraycopy(tempBuf, 0, buf, 0, tempBuf.length);
                    String args[] = {"1", "2", "3"};
                    Kernel.execute("B0001TEST0", args);
                    break;
            }
        }
    }

}
```

6.2 WIPI C 에서 WIPI JAVA 프로그램 실행

```
M_Int32 rec = MC_knlExecute(“A000100001”, 6, "WAM", “CONTENT/B0001TEST2”,
              tmpArgs[0], tmpArgs[1], tmpArgs[2], tmpArgs[3], tmpArgs[4]);
```

WIPI JAVA 는 WIPIC 에서 player 로 동작하여 Content 를 실행시키는 방법으로 진행된다. 다른 프로그램을 실행시키는 API 는 MC_knlExecute()를 이용한다. 다음은 파라미터에 대한 설명이다.

1. "A000100001" – WIPI_JAVA 의 이름이다. Player 의 이름으로 사용된다.
2. 인자로 넘기는 파라미터의 수로 넘겨야 될 사이즈보다 2 더해서 넘겨야 한다.
3. "WAM" 실제로 프로그램을 실행하는 것은 WAM(Wipi Application Menenger)이 관리한다.
4. WIPI_JAVA 의 실제 컨텐츠의 이름을 명시한다. 주의할 것은 이름 앞에 "CONTENT/"를 붙여야 프로그램이 실행된다.
5. 실제로 startJlet(String[] args)에 args 로 넘어가는 파라미터 들이다. 주의할 것은 2 번의 사이즈-2 만큼 넘어간다. 따라서 파라미터에 맞춰 2 의 인자의 수를 조절한다.