

Chapter 12. NIO 개요

Originally made by Prof. Hanku Lee

Modified by Mingyu Lim

**Collaborative Computing Systems Lab.
School of Internet & Multimedia Engineering
Konkuk University, Seoul, Korea**

1. NIO 패키지 소개

□ NIO 패키지

- New I/O는 JDK1.4에서 새로 추가된 패키지이다.
- New I/O는 java.nio 패키지로 제공되는 기능
 - 버퍼 관리 클래스
 - 확장된 네트워크 그리고 파일
 - I/O, 문자 집합 지원
 - 정규식 문자 표현에 새로운 특징들과 개선된 성능을 제공
- java.nio 패키지는 다음과 같은 클래스들로 나뉘어진다.
 - java.nio
 - java.nio.channels
 - java.nio.channels.spi
 - java.nio.charset
 - java.nio.charset.spi
- spi가 붙은 것을 볼 수 있는데 이는 SPI(Service Provider Interface)로 프로그래머가 제공하는 클래스로 대체할 수 있는 기능을 제공해준다.

1. NIO 패키지 소개(계속)

□ 특징

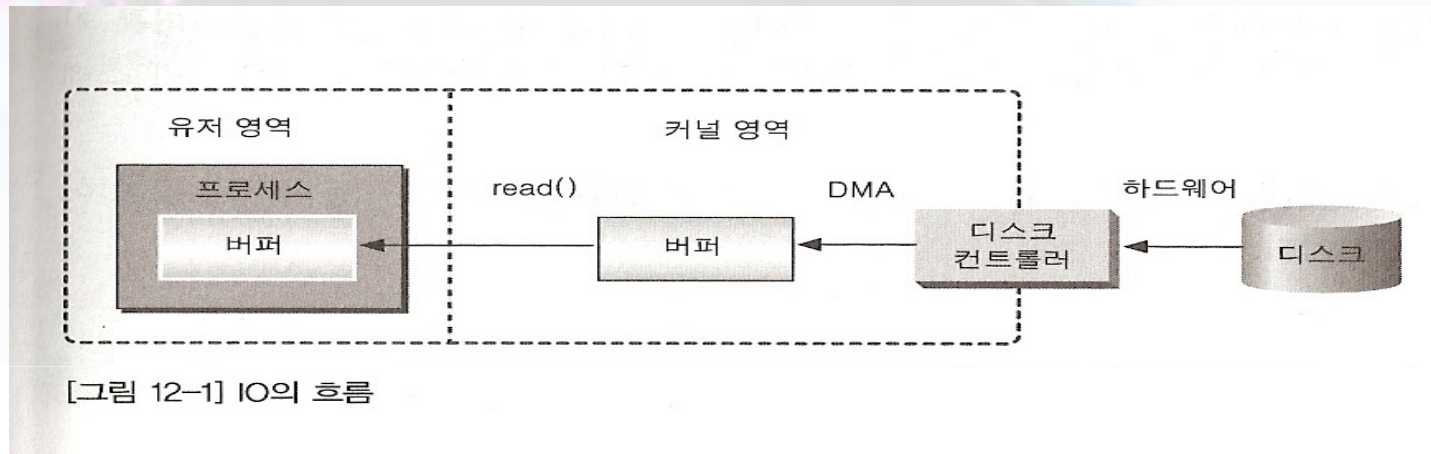
- 기본 데이터형 버퍼를 클래스로 제공
- Character-set 인코더들과 디코더
- 패턴과 어울린 기능은 Perl-style 정규식 들로 설정
- 채널, 새로운 I/O 추상화
- 메모리 매핑과 파일의 lock(잠금장치)를 지원해주는 인터페이스
- non-blocking 입출력이 가능

□ 패키지 소개

- java.nio.package : 자바 기본형 유형에 맞는 버퍼 클래스들
- java.nio.channels.package : 채널과 셀렉터
- java.nio.charset.package : 문자 암호화들
- java.nio.channels.spi.package : Service 프로바이더는 채널들을 위해 분류
- java.nio.charset.spi.package : Service 프로바이더는 문자를 위해 분류
- java.util.regex.package : 정규식 들에 의해서 지정된 패턴들에 대해서 문자 표현에 대한 클래스들
- java.lang.CharSequence.인터페이스 : 다양한 종류의 문자 순서에의 통일된 read 전용 액세스를 제공

2. 블럭킹 자바 IO

□ 블럭킹 자바 IO 영역



- 유저 영역 : 일반적인 프로세스들이 존재하는 제한된 권한을 갖는 영역
(유저 영역의 프로세스들은 h/w나 다른 프로세스에 직접적으로 접근 못함)
- 커널 영역 : 운영체제에 존재하는 영역으로 h/w 장치에 직접적으로 접근하고 다른 프로세스를 제어할 수 있는 권한을 가짐

2. 블럭킹 자바 IO(계속)

□ 블럭킹 자바 IO 영역

- 모든 IO 는 반드시 커널 영역을 직간접적으로 거침
- 프로세스를 JVM 가정 (파일 읽기 시도)
 1. 커널에 명령 전달
 2. 커널은 시스템 콜(read()) 을 사용 읽어온 파일 데이터를 커널 버퍼에 저장
 3. 모든 파일 데이터가 커널안의 버퍼로 복사되면 JVM(프로세스) 안의 버퍼로 복사를 시작한다.

CHROMATIC COLOR

* 시스템 콜

프로그래밍 언어에서 지원하지 않는 기능을 운영체제의 루틴을 호출해서 이용하는 것을 말한다. (ex 디렉토리를 만드는 루틴, 특정 디렉토리에 있는 파일 목록 읽는 루틴) 만약 응용프로그램에서 운영체제에 있는 루틴을 실행시켜 결과를 얻기 원한다면 “시스템 콜”을 이용해야 한다.

2. 블로킹 자바 IO(계속)

□ 자바 파일 읽기의 비효율성

- 커널 영역 버퍼에서 프로세스 영역 안의 버퍼로 데이터를 복사하는 점
 - 물리적 디스크에서 커널영역 버퍼로 데이터 저장하는 것은 DMA 기술로 CPU 도움없이 처리 가능
 - 커널 영역에서 프로세스 영역 버퍼로의 데이터 전달은 CPU 관여 (CPU 자원 사용 효율성 저하)
 - 커널 영역의 버퍼에 저장된 데이터를 직접 사용 하면 복사 시간 단축, 복사 대상의 데이터 가비지 컬렉션 불필요, CPU 자원의 효율성 향상
- 디스크 컨트롤러에서 커널 영역의 버퍼로 데이터를 복사하는 동안 프로세스 영역은 블로킹 되는점
 - 디바이스의 파일 데이터를 커널 영역안의 버퍼로 모두 복사할때까지 자바 프로세스는 블로킹

시스템 콜을 사용해서 입출력 작업을 수행하는 c 나 c++ 등의 저수준 언어에 비해 자바의 io 가 느린것이다.

3. 10 향상을 위한 운영체제 수준의 기술

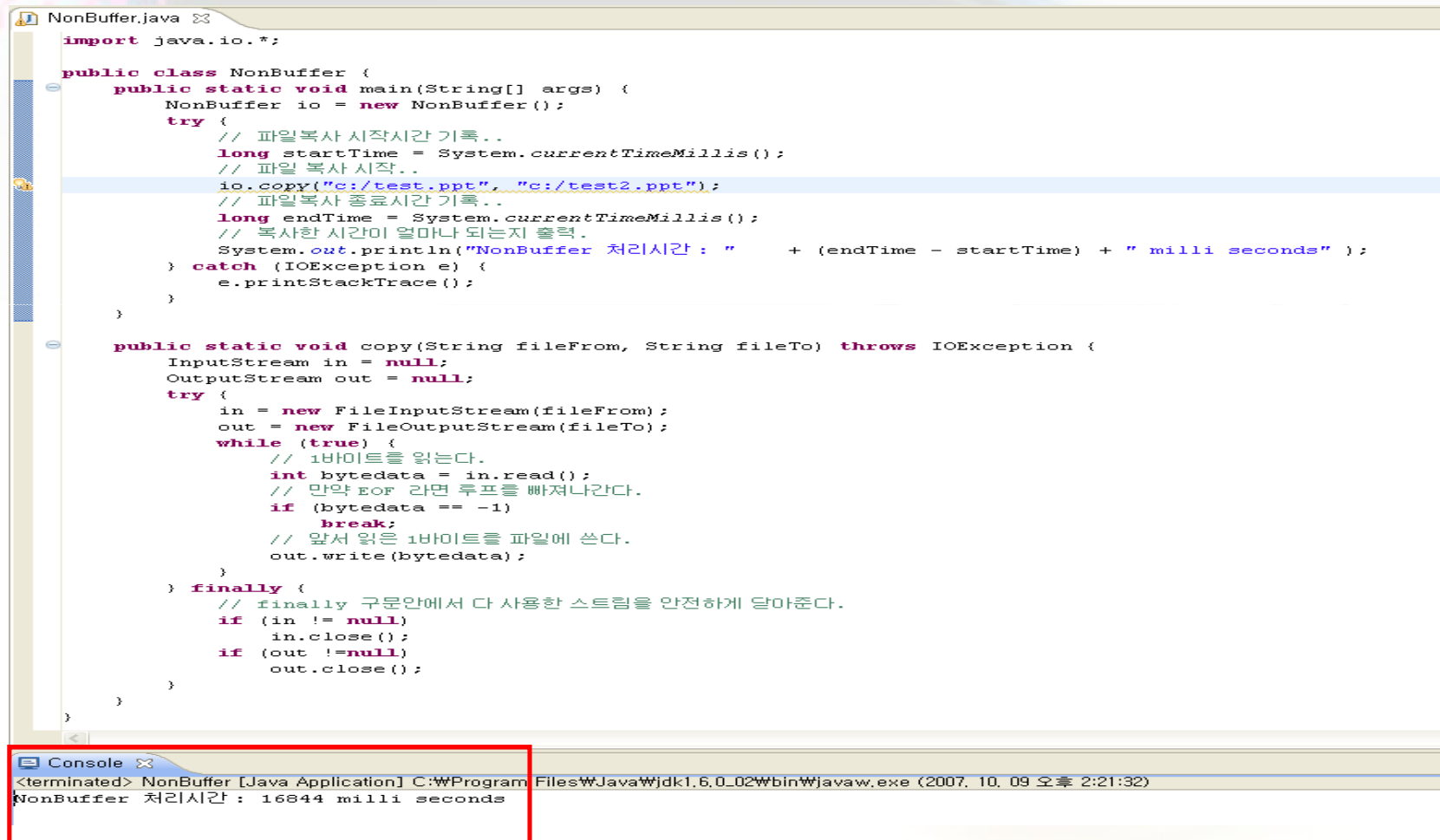
□ 버퍼

- 버퍼 : 효율적으로 데이터를 전달하기 위한 객체, 데이터를 한개씩 여러 번 보내는 것보다는 버퍼를 두고 데이터를 모아 한번에 보내는것이 효율적
- 데이터를 전송하는 곳에서는 대부분 기본적으로 버퍼를 사용
(운영체제 포함)
- 세가지 테스트 코드로 버퍼 사용에 따른 성능차이 확인(파일 복사 프로그램)
 - 원본 파일 크기 (test.ppt : 3984KB)

3. IO 향상을 위한 운영체제 수준의 기술(계속)

□ 버퍼1

- 버퍼 사용하지 않고 1바이트씩 파일을 읽어서 복사하는 예제



```
NonBuffer.java
import java.io.*;

public class NonBuffer {
    public static void main(String[] args) {
        NonBuffer io = new NonBuffer();
        try {
            // 파일복사 시작시간 기록..
            long startTime = System.currentTimeMillis();
            // 파일 복사 시작..
            io.copy("c:/test.ppt", "c:/test2.ppt");
            // 파일복사 종료시간 기록..
            long endTime = System.currentTimeMillis();
            // 복사한 시간이 얼마나 되는지 출력.
            System.out.println("NonBuffer 처리시간 : " + (endTime - startTime) + " milli seconds");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void copy(String fileFrom, String fileTo) throws IOException {
        InputStream in = null;
        OutputStream out = null;
        try {
            in = new FileInputStream(fileFrom);
            out = new FileOutputStream(fileTo);
            while (true) {
                // 1바이트를 읽는다.
                int bytedata = in.read();
                // 만약 EOF 라면 루프를 빠져나간다.
                if (bytedata == -1)
                    break;
                // 앞서 읽은 1바이트를 파일에 쓴다.
                out.write(bytedata);
            }
        } finally {
            // finally 구문안에서 다 사용한 스트림을 안전하게 닫아준다.
            if (in != null)
                in.close();
            if (out != null)
                out.close();
        }
    }
}
```

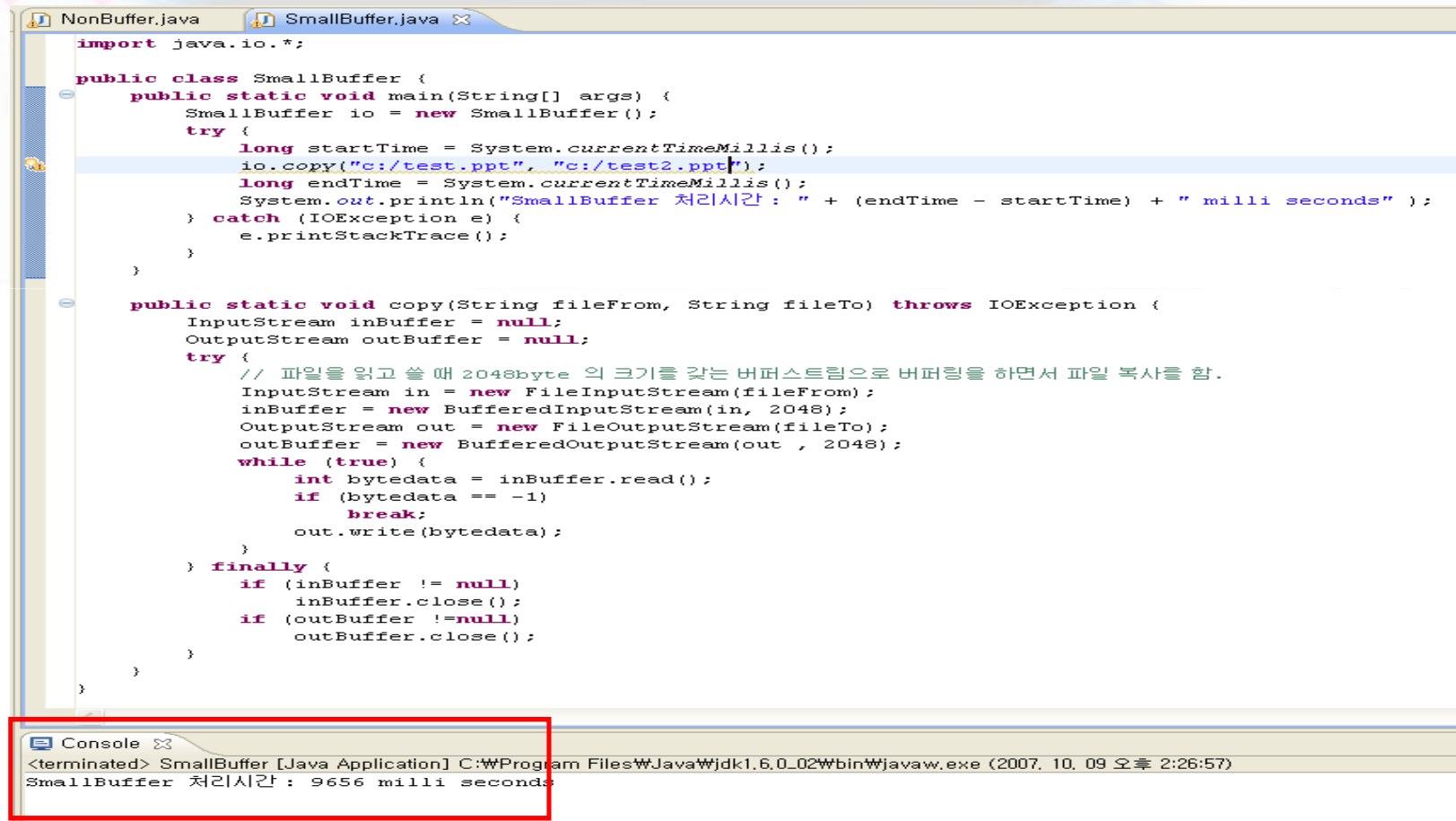
Console

<terminated> NonBuffer [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 10. 09 오후 2:21:32)
NonBuffer 처리시간 : 16844 milli seconds

3. IO 향상을 위한 운영체제 수준의 기술(계속)

□ 버퍼2

- 2048Byte의 버퍼크기를 갖는 Buffered 스트림 이용하는 예제



```
import java.io.*;

public class SmallBuffer {
    public static void main(String[] args) {
        SmallBuffer io = new SmallBuffer();
        try {
            long startTime = System.currentTimeMillis();
            io.copy("c:/test.ppt", "c:/test2.ppt");
            long endTime = System.currentTimeMillis();
            System.out.println("SmallBuffer 처리시간 : " + (endTime - startTime) + " milli seconds");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void copy(String fileFrom, String fileTo) throws IOException {
        InputStream inBuffer = null;
        OutputStream outBuffer = null;
        try {
            // 파일을 읽고 쓸 때 2048byte 의 크기를 갖는 버퍼스트림으로 버퍼링을 하면서 파일 복사를 함.
            InputStream in = new FileInputStream(fileFrom);
            inBuffer = new BufferedInputStream(in, 2048);
            OutputStream out = new FileOutputStream(fileTo);
            outBuffer = new BufferedOutputStream(out, 2048);
            while (true) {
                int bytedata = inBuffer.read();
                if (bytedata == -1)
                    break;
                out.write(bytedata);
            }
        } finally {
            if (inBuffer != null)
                inBuffer.close();
            if (outBuffer != null)
                outBuffer.close();
        }
    }
}
```

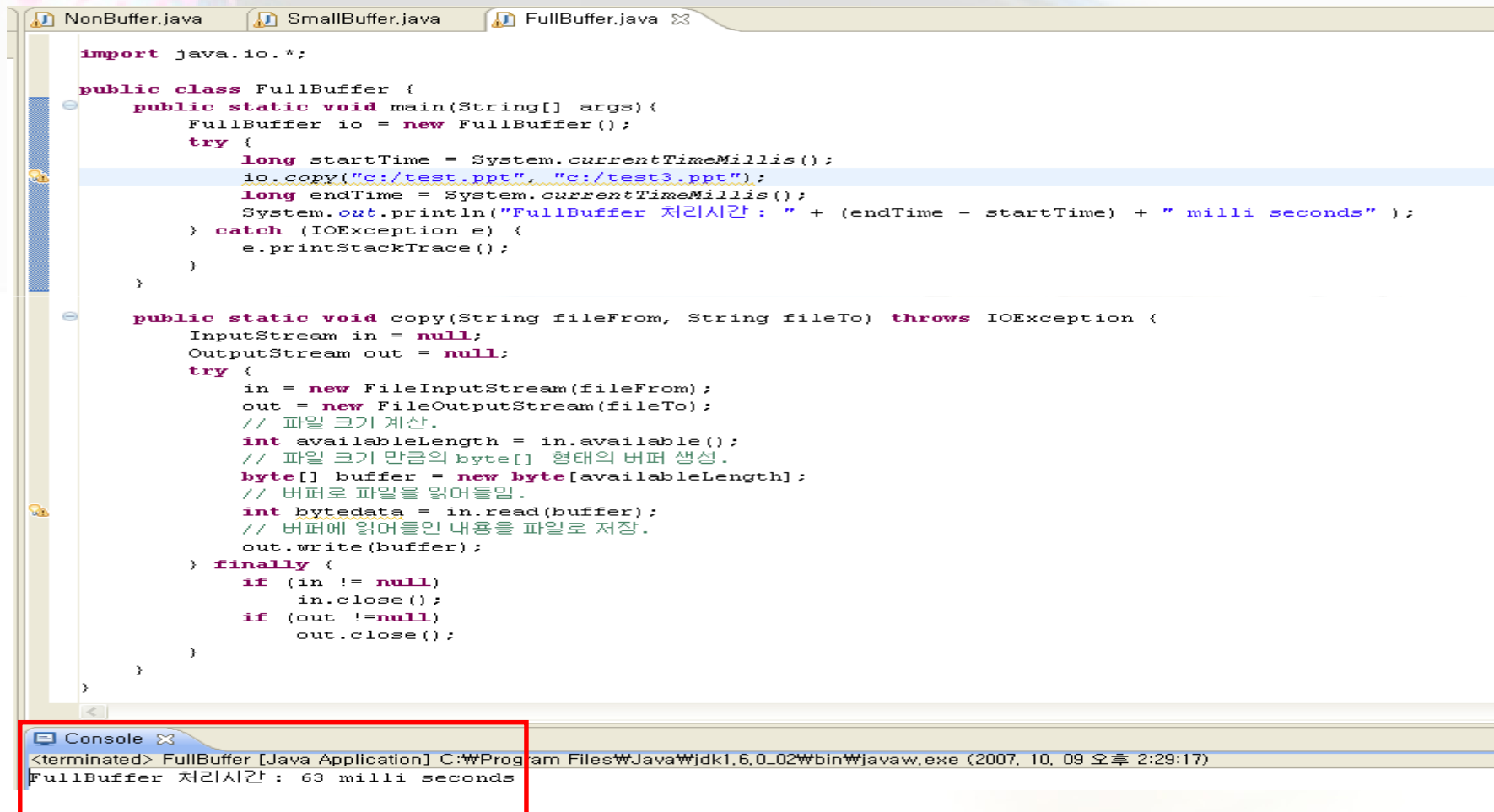
Console

<terminated> SmallBuffer [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 10. 09 오후 2:26:57)
SmallBuffer 처리시간 : 9656 milli seconds

3. IO 향상을 위한 운영체제 수준의 기술(계속)

□ 버퍼3

- 파일 크기의 버퍼를 이용하는 예제



The screenshot shows a Java IDE with three tabs: NonBuffer.java, SmallBuffer.java, and FullBuffer.java. The FullBuffer.java tab is active, displaying the following code:

```
import java.io.*;

public class FullBuffer {
    public static void main(String[] args){
        FullBuffer io = new FullBuffer();
        try {
            long startTime = System.currentTimeMillis();
            io.copy("c:/test.ppt", "c:/test3.ppt");
            long endTime = System.currentTimeMillis();
            System.out.println("FullBuffer 처리시간 : " + (endTime - startTime) + " milli seconds");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void copy(String fileFrom, String fileTo) throws IOException {
        InputStream in = null;
        OutputStream out = null;
        try {
            in = new FileInputStream(fileFrom);
            out = new FileOutputStream(fileTo);
            // 파일 크기 계산.
            int availableLength = in.available();
            // 파일 크기 만큼의 byte[] 형태의 버퍼 생성.
            byte[] buffer = new byte[availableLength];
            // 버퍼로 파일을 읽어들이м.
            int bytedata = in.read(buffer);
            // 버퍼에 읽어들이는 내용을 파일로 저장.
            out.write(buffer);
        } finally {
            if (in != null)
                in.close();
            if (out != null)
                out.close();
        }
    }
}
```

The console window at the bottom shows the following output:

```
<terminated> FullBuffer [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 10. 09 오후 2:29:17)
FullBuffer 처리시간 : 63 milli seconds
```

3. 10 향상을 위한 운영체제 수준의 기술(계속)

□ 성능 비교

클래스	버퍼크기	프로세스 시간(ms)
NonBuffer	None	16844
smallBuffer	2048byte	9656
FullBuffer	3394byte	63

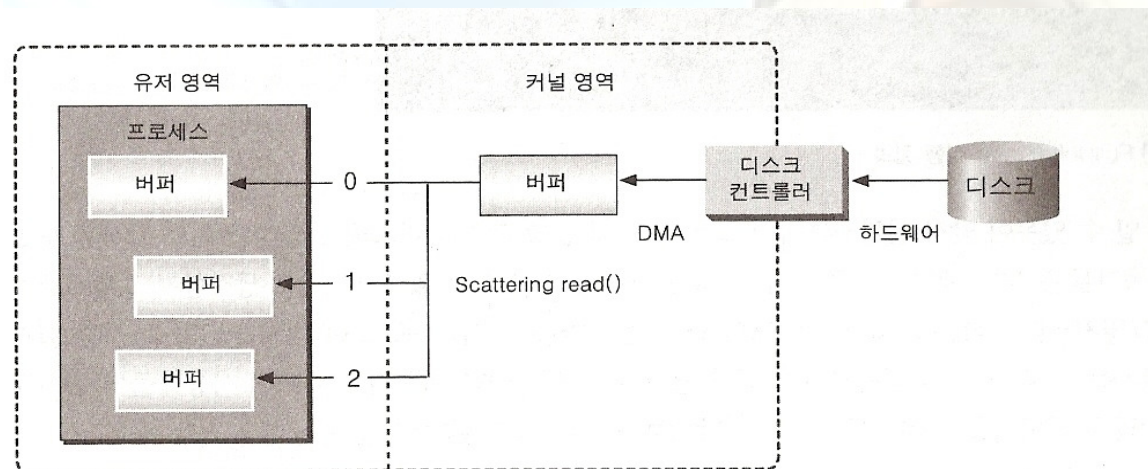
CHROMATIC COLOR
.....

ACHROMATIC COLOR
.....

3. IO 향상을 위한 운영체제 수준의 기술(계속)

□ Scatter / Gather

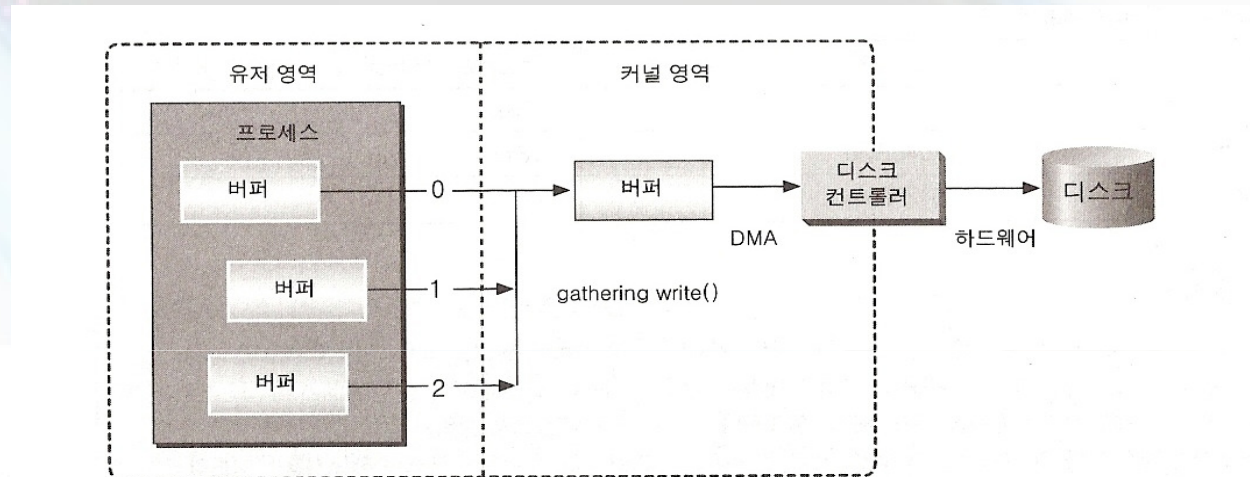
- 자바 프로그램안에 버퍼 세개를 만들어 사용 하는데 만약 동시에 각각의 버퍼에 데이터를 읽거나 쓰면 시스템 콜이 세번 발생 (상당히 비효율적)
- 위의 단점을 보완하기 위해서 운영체제 수준에서 지원하는 기술이 Scatter, Gather(시스템 콜 한번만 호출)



[그림 12-5] Scattering Read

3. IO 향상을 위한 운영체제 수준의 기술(계속)

□ Scatter / Gather



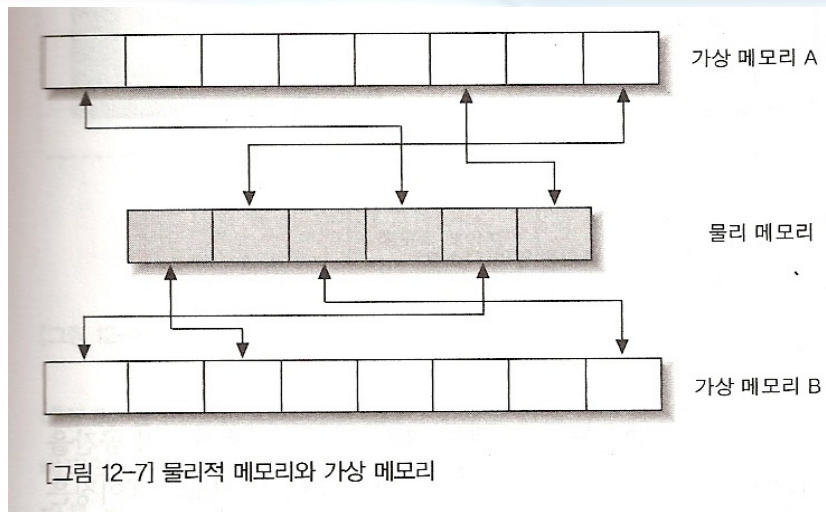
[그림 12-6] Gathering Write

- Scatter / Gather 기능을 이용하기 위해 `ScatteringByteChannel` / `GatheringByteChannel` 인터페이스를 사용

3. 10 향상을 위한 운영체제 수준의 기술(계속)

□ 가상 메모리

- 프로그램이 사용할수 있는 주소 공간을 늘리기 위해 운영체제에서 지원하는 기술
- 운영체제는 가상 메모리를 페이지(page) 라는 고정된 크기로 나누고 각 페이지는 메모리가 아닌 디스크에 먼저 저장
- 프로그램 실행되어지는 페이지의 가상 주소만 물리적 메모리 주소로 맵핑

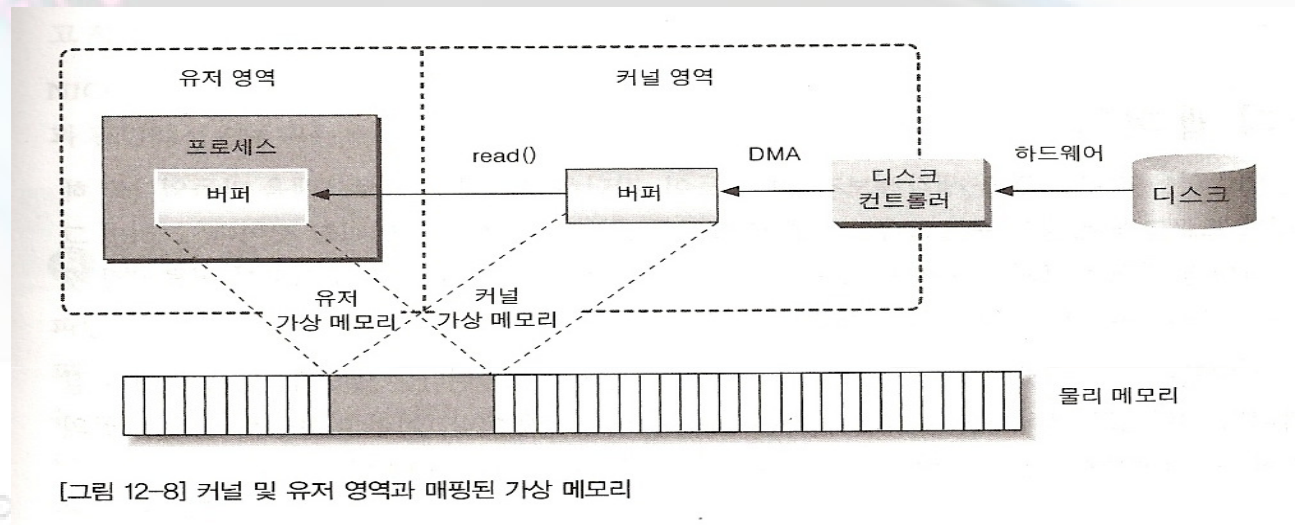


가상 메모리 사용시 이점

1. 실제 물리적 메모리 크기보다 큰 가상 메모리 공간 사용가능
2. 여러 개의 가상 주소가 하나의 물리적 메모리 주소를 참조함으로써 메모리를 효율적으로 사용

3. 10 향상을 위한 운영체제 수준의 기술(계속)

□ 가상 메모리

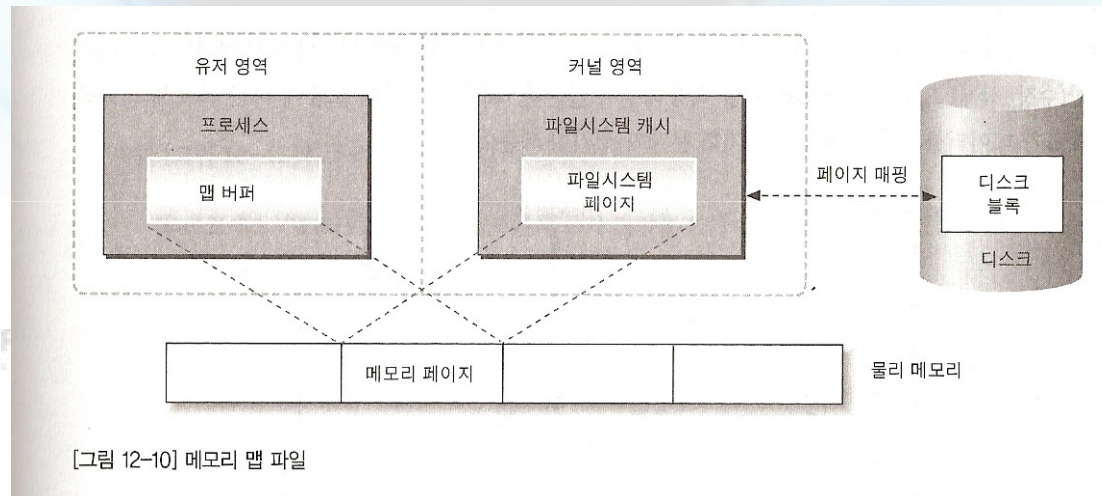


- 프로그램에서 가상메모리를 사용하게 되면 유저 영역 버퍼와 커널 영역 버퍼를 매핑시킴으로써 커널영역에서 유저 영역으로 데이터를 복사하지 않아도 된다.
- 디스크 컨트롤러부터 읽어온 데이터를 커널 영역에 저장하는것은 곧 동시에 유저 영역 버퍼에 저장하는 것이다.

3. IO 향상을 위한 운영체제 수준의 기술(계속)

□ 메모리 맵 파일

- 빈번한 시스템 콜과 불필요한 커널영역과 유저 영역간의 복사가 이루어지면 많은 가비지 발생(가비지 제거하는것은 상당히 느린 작업)
- 위의 문제점을 해결하기 위해 Memory-Mapped IO 이용



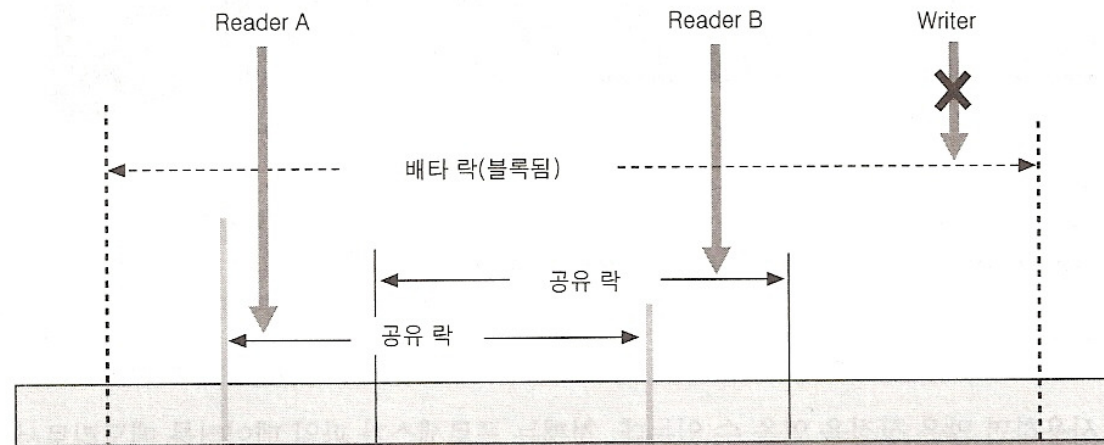
- 사용시 장점

1. read(), write() 등의 시스템 콜 불필요
2. 매우 큰 파일을 복사하기 위해 많은 양의 메모리 소비 안해도 된다.

3. 10 향상을 위한 운영체제 수준의 기술(계속)

□ 파일락

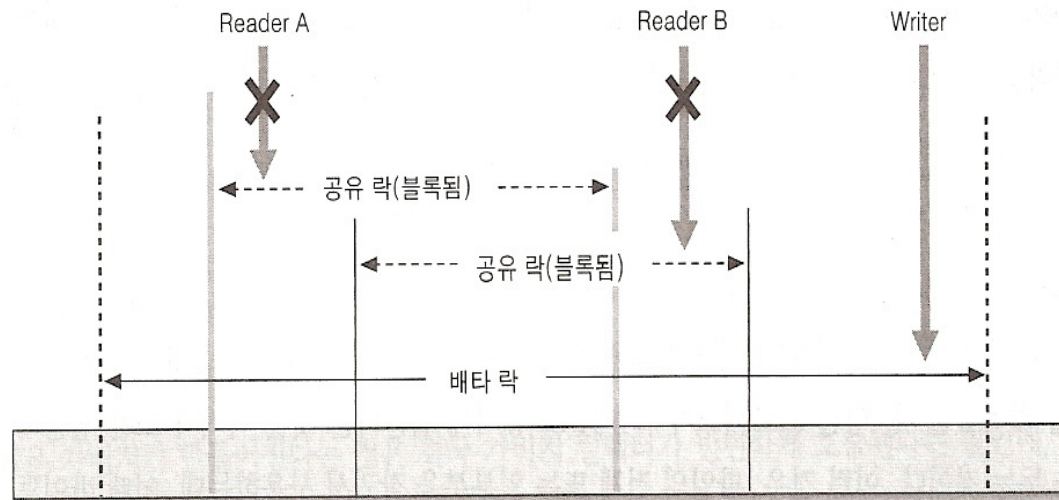
- 동기화 비슷한 개념
- 한 프로세스가 어떤 파일에 락(lock)획득시 다른 프로세스가 동시에 파일로 동시에 접근하는것을 막거나 또는 접근하는 방식에 제한을 두는것
- 파일 락은 크게 공유(shared)락과 배타(exclusive)락이 있다.
- 공유락은 읽기작업에 사용 / 배타락은 쓰기작업에 사용



[그림 12-11] 공유 락(Shared Lock)

3. 10 항상을 위한 운영체제 수준의 기술(계속)

□ 파일락



[그림 12-12] 배타 락(Exclusive Lock)

4. 자바의 새로운 변화

□ 자바의 포인터 버퍼 도입

- NIO 에서 Buffer 클래스를 도입
- J2SDK 1.4에서 새롭게 도입된 NIO에서는 커널에 의해 관리되는 시스템 메모리를 직접 사용할수 있는 Buffer 클래스 도입
(DirectByteBuffer 에 한정) : 하부구현은 c 로 제작
- 결론적으로 C / C++ 프로그래머가 사용하는 포인터가 자바에서도 생성 되었다는 의미

□ 네이티브 IO서비스를 제공하는 채널 도입

- NIO 에서는 채널을 도입 (스트림처럼 단방향, 읽고 쓰는 양방향 통신이 가능한 3가지 형식 존재)
- 채널을 이용해서 시스템 메모리인 버퍼에 직접적으로 데이터를 읽거나 쓸수 있다.
- Scatter / Gather 를 구현해서 효율적으로 IO를 처리

4. 자바의 새로운 변화(계속)

□ 셀렉터 도입

- 셀렉터는 네트워크 프로그래밍의 효율을 높이기 위한것
(POSA2 에서 소개된 Reactor 패턴의 구현체)
- NIO 에서 네트워크 프로그래밍의 확장성과 유연성, 효율성을 높이기 위해
셀렉터를 이용함으로써 단 한 개의 스레드만으로 수천에서 수만명의
동시사용자를 처리할수 있는 서버를 만들수 있게 되었다.

CHROMATIC COLOR
.....

ACHROMATIC COLOR
.....