자바스레드를 이용한 운영체제 교육

김일민†

요 약

운영체제는 컴퓨터 시스템 자원의 관리 운용에 관한 과목으로서 컴퓨터 관련 전공자에게는 매우 중요하다. 운영체제의 내용 중에서 비동기 병행 프로세스 부분은 다른 부분에 비해서 매우 난이도가 높은 부분이다. 비동기 병행 프로세스 부분은 책의 설명이나 pseudo 코드만을 사용해서는 효과적인 학습효과를 거둘 수 없으므로 적절한 프로그래밍 언어로 구현된 병행 프로그램을 실행하는 것이 필요하다. 본 논문에서는 다양한 비동기 병행 코드를 자바 스레드로 구현하였고, 구현된 코드를 운영체제 교육에 활용할 것을 제안하였다.

Operating System Education Using Java Thread

Il-Min Kim†

ABSTRACT

As Operating System is a course about computer system resource management, it is very important to the computer related major students. Asynchronous concurrent processes in a Operating System class is rather difficult to understand. Because most students are not able to understands the part with the explanation and the pseudo code in a text, they need to execute the concurrent programs which are implementing the pseudo code. In this paper, we implemented executable programs using Java thread for the code and proposed those programs to apply to the Operating System education.

1. 서론

컴퓨터 공학을 전공하기 위해서는 컴퓨터 구조, 운영체제, 컴퓨터 네트워크, 데이터 베이스, 자료 구조, 프로그래밍 언어 등의 기초 과목에 대한 이해가 필수적이다. 이들 과목 중에서도 운영체제는 컴퓨터 시스템 자원의 관리 운용에 관한 기본 철학을 다루는 분야로서, 컴퓨터 관련 분야를 전공하기 위해서는 필수적으로 이수해야 하는 중요과목이다.

운영체제[1]는 운영체제의 구조, 프로세스 관리, 기억장치 관리, 보조 기억장치 관리, I/O 시스템, 분산 시스템 등으로 구성된다. 그 중에서도 프로세스 관리의 병행 프로그래밍 부분이나 비동기적 병행 프로세스 부분은 다른 부분과 비교해서 학습 난이도가 높은 부분이다. 이 부분은 상호배제 (mutual exclusion), 임계구역 (critical section), 생산자/소비자 문제, 세마포어 (semaphore), 이벤트 카운터(event counter), 교착상태(deadlock) 등을 포함하고 있으며, 병행 처리에 익숙하지 못한 학생들에게는 난해한 내용들이다. 특히 상호배제 프리미티브를 S/W적으로 구현한 코드부분이나 세마포어, 철학자들의 저녁식사 문제, 잠자는 이발사 문제 등은 교재의 설명만을 사용해서는 효과적인 학습을 기할 수 없

† 정 회 원: 한성대학교 컴퓨터 공학과 교수
논문접수: 2000년 12월 14일, 심사완료: 2001년 2월 27일
* 본 논문은 2000년 한성대학교 교내 연구비의 지원을 받아 수행되었음.

으며, 적절한 프로그래밍 언어를 사용해서 구현된 병행 프로그램을 실행해보는 것이 효과적이다.

멀티 스레드를 지원하는 프로그래밍 언어를 사용하면 운영체제의 병행 코드를 구현하는 것이 가능하다. 그러나 기존의 유닉스 시스템에서 지원되는 스레드 패키지는 사용법이 난해하고, 유닉스 운영체제에서만 실행되므로 활용이 용이하지 않다. 멀티 스레딩이 지원되며, 다양한 플랫폼에서 실행되는 무료 언어가 바로 자바이다.

본 논문에서는 현재 객체 지향 언어로서 널리 사용되고 있는 자바(java)의 스레드를 사용하여 운영체제의 병행 처리 pseudo 코드를 구현함으로서 운영체제 학습과 교육에 활용하고자 한다.

2. 운영체제의 비동기 병행 프로세스

두 개 이상의 프로세스가 공유 변수를 갱신할 때, 다른 프로세스의 접근을 금지하여야 하며 이를 상호배제라고 한다. 상호배제를 구현하는 방법 중에는 S/W적인 것과 H/W를 이용하는 것 두 가지가 있다. 일반 운영체제 교재에는 두 개의 프로세스가 임계영역에서 상호 배제를 만족하기 위해서 여러 가지 S/W적인 기법이 소개되고 있다. 그 중에 하나를 소개하면 (그림 1)과 같다[1].

```
repeat
  flag[i] := true;
  while flag[j] do no-operation;
  <임계구역>
  flag[i] := false;
  <기타 실행 부분>
until false;
```

(그림 1) 병행 프로세스 코드의 예

위의 코드는 잘못된 솔루션의 예제이며, 교착상태를 유발시킬 수 있다. 즉 두 프로세스가 while의 조건을 false가 될 때까지 무한 루프를 도는 상황이 발생할 수 있다. 간단한 코드지만 실제로 이러한 코드부분을 지도해보면 많은 학생들이 이해하지 못하는 실정이다. 그러므로 이 코드를 실제 실행시켜 보는 것이 필요하지만 현재까지의 교재에는 적절한 실행 방안을 제시하지 못하고 있으며, 또한 대부분의 전산교수들도 교재의 pseudo 코드만을 학습에 사용하고 있다.

상호배제를 위한 H/W 프리미티브로서 Test&Set, 세마포어 명령어를 사용한 병행 프로세스코드가 일반적으로 소개되며, Dijkstra가 제안한 철학자들의 저녁식사(Dinning Philosopher), 잠자는 이발사 문제(Sleeping Barber Problem), Pail이 제안한 흡연자 문제(Cigarette Smoker Problem) 등은 운영체제의 병렬 처리 부분에 꼭 등장하는 문제들이다. 특히 잠자는 이발사의 경우 매우 단순한 문제이지만 학생들이 실제 코드를 실행해보지 않고 교재의 문장만으로 문제를 파악한다는 것은 매우 어려운 일이다.

3. 자바 및 자바 스레드 소개

3.1 자바의 소개

자바[5]는 1995년 Sun사의 James Gosling에 의해서 설계된 프로그래밍 언어이다. 자바는 인터넷 환경에서 효과적으로 응용 프로그램을 개발할 수 있도록 설계된 객체지향 언어이며, 윈도우, 유닉스 등 다양한 환경에서 실행된다.

자바 소스 프로그램은 작고 단순하며 효율적인 형태인 bytecode로 번역되어 실행된다. 기존언어에서 많은 실행 오류를 유발시키고 오류 수정을 어렵게 만들었던, goto 문장이나 포인터 등은 삭제되었다. 1995년 Java 베타2 버전이 발표되었고, 1996년에는 Java 1.0버전이 발표되었다. Sun사는 자바언어를 발표한 후에 1996년부터 자바의 개발 및 관련사업을 전담하는 자바 소프트(Javasoft)를 창설하여 운영하고 있으며, 2000년 11월 현재 JDK 1.3까지 공개되어있다. 최근 수 많은 대학의 자료구조, 객체 지향 프로그래밍, 분산 병렬 프로그래밍 강좌의 구현 언어로서 채택하고 있다.

3.2 자바 스레드 소개

자바에서 멀티 스레딩[3][4] 프로그램을 작성하려면 먼저 프로그래머가 사용자 정의 스레드 클래스를 작성하여야 한다. 그 방법은 Thread 클래

스를 상속받아 새로운 클래스를 정의하는 방법과 Runnable 인터페이스를 구현하는 클래스를 정의하는 것 두 가지가 있다. Thread 클래스를 상속하는 경우는 다음 2가지의 생성자를 사용할 수 있다. 생성자의 인자 *name*은 스레드의 이름으로 설정된다.

```
Thread()
Thread(String name)
```

스레드를 실행하기 위해서는 객체 메소드 start()를 호출하여야 한다. start() 메소드는 Thread 클래스로부터 상속받은 메소드로서, 먼저 스레드의 실행준비를 한 다음 run() 메소드를 자동 호출한다. 그러므로 프로그래머는 스레드가 할 일을 run() 메소드 안에 기술하여야 한다. 다음은 사용자 정의 스레드 MyThread를 정의하는 형식이다.

```
class MyThread extends Thread {
  ....
  public void run() {
    // 스레드가 수행할 일을 기술
}}
```

앞에서 선언한 MyThread 클래스의 객체를 생성하여 실행하는 방법은 다음과 같다.

```
MyThread t1 = new MyThread();
t1.start();
```

3.3 자바 스레드 사용시 주의점

자바 스레드 프로그래밍은 단순하지 않다. 자바 스레드의 특히 많은 부분이 자바 매뉴얼에 명시되어 있지 않기 때문에 많은 실행 오류와 경험이 필요하다. 자바 스레드를 사용하여 비동기 병행 프로세스를 프로그래밍하기 위해서는 다음과 같은 주의사항이 필요하다. 첫 번째 자바에서 사용하는 스레드와 일반 운영체제 병행처리 pseudo 코드와는 많은 차이점이 있다. 자바 스레드는 객체할당 후, start() 메소드를 호출함으로서 병행처리가 시작되지만, 병행 프로세스 pseudo 코드에서는 par_begin과 par_end를 사용하므로 코드가 매우 단순하다. 그러므로 운영체제 교재의 코드와 자바로 구현된 코드는 외형상 많은 차이점이 있다. 두 번째 사용자 정의 스레드 객체들은 별도의 주소공간에 할당되므로 스레드간 공유 변수를 가지지 않는다. 그러므로 스레드간에 데이터를 공유하기 위해서는 먼저 공유 객체를 생성한 후 사용자 스레드 생성시 생성자의 인자로 넘겨주어야 한다. 인자로 넘겨받은 객체를 객체 참조변수로 가리키게 함으로서 스레드간에 데이터 객체를 공유할 수 있다. 세 번째 자바 스레드간 통신에 사용되는 wait(), notify() 메소드는 객체 메소드이므로 사용에 주의하여야 한다. 그러므로 별도의 스레드 객체간에 wait(), notify() 메소드를 사용해서는 통신이 이루어지지 않는다는 것이다. 클래스안에 공유 데이터와 이를 조작하는 메소드를 함께 명시하는 것이 필요하다. 스레드가 공유한 데이터 객체에서 사용된 notify(), wait()를 사용하여야 통신이 가능하다. 네 번째 학습자들이 자바 스레드에 대한 기초지식이 필요하다. 자바에 대한 기초 지식을 가진 학습자는 자바 스레드에 대한 학습과 병행 프로그래밍에 대한 학습을 동시에 함으로서 상호 상승효과를 거둘 수 있으리라 생각된다.

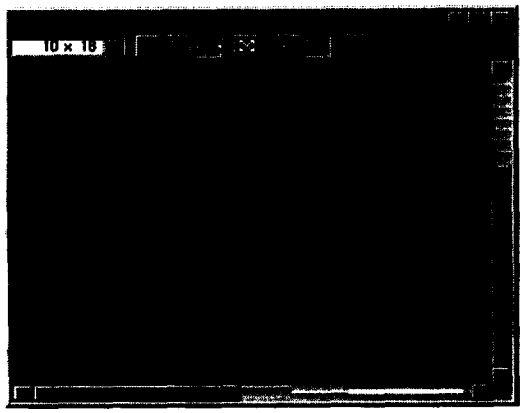
4. 상호배제 프리미티브의 구현

4.1 S/W를 사용한 구현

앞에서 예제로 사용한 (그림 1)은 상호배제 프리미티브를 소프트웨어적으로 구현한 코드이다. 두 개의 프로세스만을 고려한 코드이나 불완전하므로 교착상태를 유발시킬 수 있다. 각 프로세스는 번호를 가지며 편의상 i와 j로 표현한다. (그림 1)의 코드를 자바 스레드로 구현하면 다음 코드와 같다.

```
class Mul extends Thread {
  int id, other;
  static boolean flag[] = { false, false };
```

```
  static void work(int max) { //임의시간작업
   max *= 120;
   try {
     Thread.sleep((int)(max* Math.random()));
   } catch(Exception e) {}
  }
  static void write(int tab, String msg) {
     if(tab>0) System.out.print("\t\t");
     System.out.println(msg);
  }
  Mul(int i) { id = i; other = 1-i;}  // 생성자
  public void run() {
    int cnt = 0;
    try {
     while(++cnt<100) {
       write(id, "P"+id+" is out of CS");
       work(12);
       write(id, "P"+id+" is trying to CS");
       flag[id] = true;
       while(flag[other] == true) ;
       write(id, "P"+id+" is in CS");
       work(1);
       flag[id] = false;
      }
    } catch(Exception e) {}
  }
}
class M {
  public static void main(String[] a) {
    Mul t0 = new Mul(0);
    Mul t1 = new Mul(1);
    t0.start(); t1.start();
  }
}
```



(그림 2) 실행 결과

사용자 정의 스레드 클래스 Mul의 일반 변수들은 스레드 객체마다 할당되지만, 정적 변수들은 Mul 클래스에 하나의 인스턴스를 가지므로 모든 스레드들이 공유할 수 있다. 정적 메소드 work()는 임의시간 작업을 흉내내는 메소드이다. 임계영역에서의 작업시간을 기타 영역에서의 작업시간보다 훨씬 적게 하였으며, 두 프로세스의 번호는 0과 1로 설정하였다. 이 프로그램의 실행결과는 (그림 2)와 같다.

(그림 2)는 프로세스 P0과 P1이 서로 자신의 flag를 true로 설정함으로서 두 프로세스 모두 임계구역으로 들어가지 못하는 교착상태가 발생한 결과를 나타내고 있다.

4.2 세마포어(Semaphore)

Dijkstra가 제안한 세마포어는 상호배제를 구현하기 위한 프리미티브이다. 세마포어는 단지 P(), V()와 세마포어 초기화 메소드만을 사용해서 접근할 수 있는 변수이다. 세마포어의 값은 0이상의 양의 정수 값을 가질 수 있다. 또한 P(), V()는 atomic 연산자이다. 세마포어 s에 대한 P()연산을 기술하면 다음과 같다[2].

```
P(s):if s>0
        then s --;
        else wait(s);
```

세마포어 s에 대한 V()연산을 기술하면 다음과 같다.

```
V(s):   if (any process waits s)
        then wake up a process
        else s++;
```

세마포어를 자바 스레드로 구현하기 위해서는 wait()와 notify()메소드를 사용해야 한다. wait() 메소드를 실행한 스레드는 실행이 중단되며, 다른 스레드의 notify()에 의해서 실행을 계속하게 된다. 자바언어로 세마포어를 구현하면 다음과 같다.

```
class Semaphore {
   private int value;
   public Semaphore(int value)
```

```
    { this.value = value; }
    synchronized void p() throws Exception {
      value --;
      if(value < 0) wait();
    }
    synchronized void v() throws Exception {
      value ++;
      if( value <= 0) notify();
    }
}
```

메소드 p()와 v()앞에 키워드 synchronized를 붙임으로서 p()와 v() 메소드가 atomic 특성을 갖게 된다. 세마포어를 사용하여 상호배제를 구현 병행 프로세스의 pseudo 코드의 예는 다음과 같다[2].

```
program semaphore_example;
var active: semaphore;
procedure process1;
begin
  while true do
  begin
        // 기타 코드
        P(active);
        // Critical Section
        V(active);
        // 기타 코드
      end
end;
procedure process2;
begin
  while true do
   begin
        // 기타 코드
        P(active);
        // Critical Section
        V(active);
        // 기타 코드
       end
end;
begin { main }
  semaphore_initialize(active, 1);
  par_begin
    process1;
      process2;
  par_end;
end.
```

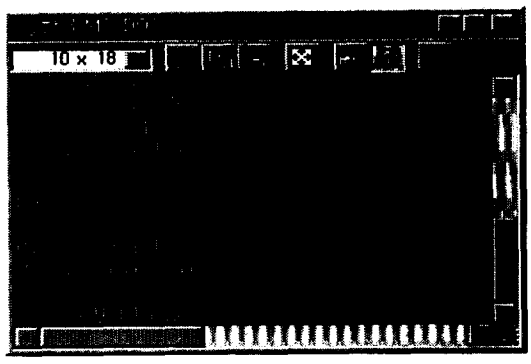
앞의 pseudo 코드를 자바 스레드를 사용하여 프로그래밍하면 다음과 같다. 사용자 정의 스레드인 MyProcess 객체가 두 개 생성되어 실행된다. 세마포어의 초기값은 1로 설정하였으며, 생성한 세마포어 s를 스레드 생성자의 인자로 넘겨주어 두 개의 스레드가 하나의 세마포어를 공유하도록 하였다. 이를 실행한 결과는 (그림 3)과 같다.

```
class MyProcess extends Thread {
    private Semaphore s;
    private int id;
    private static int MAX = 30;
    public MyProcess(Semaphore s, int id)
    {   this.s = s;
        this.id= id;
    }
    public void run() {
      for(int x=0; x<MAX; x++) {
      try {
       s.p();  // P() 연산
       // Here is critical section
       System.out.println("P"+id+" is in CS");
       sleep((int)(700 * Math.random()));
       s.v();  // V() 연산
       System.out.println("P"+id+"    out    of
CS");
       sleep((int)(700 * Math.random()));
      } catch(Exception e) {}
 }}}
 public class MutualEx {
    public static void main(String arg[]) {
      Semaphore s = new Semaphore(1);
      MyProcess p1 = new MyProcess(s, 1);
      MyProcess p2 = new MyProcess(s, 2);
      p1.start();      p2.start();
    }
 }
```

5. 병행처리를 위해 제안된 문제의 구현

5.1 철학자들의 저녁식사

Dinning Philosopher 문제는 Dijkstra가 제안하였으며 운영체제의 교착상태를 설명할 때 반드시 인용되는 문제이다. 이 문제를 구현하기 위해서는 먼저 Fork클래스를 정의하여야 한다. Fork 클



(그림 3) 실행 결과

래스는 사용가능여부 및 포크의 id를 나타내는 객체 변수가 필요하며, 포크를 집거나 놓는 메소드를 정의하여야 한다. 다음 코드는 Fork 클래스를 자바로 정의한 것이다.

```
class Fork {
  private boolean taken=false;
  private int id;
  Fork(int id)
    { this.id=id; }
  synchronized void put() {
      taken=false;
      notify();
  }
  synchronized void get() throws Exception {
      if(taken) wait();
      taken=true;
}}
```

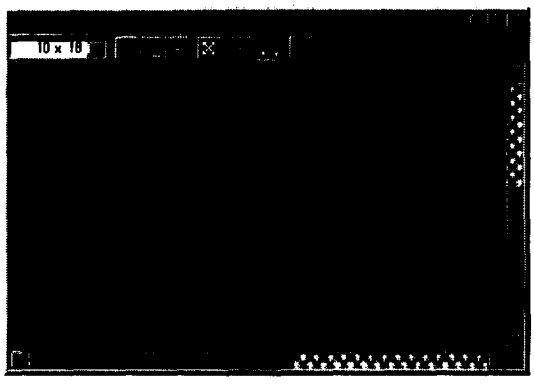
Fork 클래스의 변수 taken은 포크객체가 철학자에게 이미 할당되었는지를 나타내는 boolean 변수이며, 정수 변수 id는 포크의 식별자이다. 4명의 철학자가 식사와 사색을 반복한다고 하면 4개의 포크를 생성하여야 한다. 다음 자바 프로그램에서 포크객체는 철학자간의 공유객체로, 철학자는 자바 스레드로 구현하였다.

```
class Philosopher extends Thread {
  private int id;
  private Fork left, right;
  Philosopher(int id, Fork l, Fork r) {
    this.id = id;
    left=l;   right=r;
  }
  void display(String m) {
    for(int i=0; i<id; i++)
      System.out.print("\t\t");
    System.out.println(id+": "+ m);
  }
  public void run() {
    int cnt=0;
    try {
      while(++cnt<=100) {
        display("thinking");
        sleep((int)(Math.random()*1000));
        display("hungry");
        left.get();
        display("got left fork");
        sleep(300);    right.get();
        display("eating");        sleep(300);
        left.put();   right.put();
      }
    } catch(Exception e) {}
}}
class DiningP {
   public static void main(String args[]) {
     Fork f0 = new Fork(0);
     Fork f1 = new Fork(1);
     Fork f2 = new Fork(2);
     Fork f3 = new Fork(3);
     Philosopher p0, p1, p2, p3;
     p0 = new Philosopher(0, f0, f1);
     p1 = new Philosopher(1, f1, f2);
     p2 = new Philosopher(2, f2, f3);
     p3 = new Philosopher(3, f3, f0);
     p0.start();     p1.start();
     p2.start();     p3.start();
   }
}
```

Philosopher 스레드 객체를 생성할 때, 식별자, 사용할 포크를 인자로 사용하였다. 특히 교착상태를 유도하기 위해서 왼쪽 포크를 잡은 후에 300msec의 sleep후에 오른쪽 포크를 잡도록 하였다. 실행결과는 (그림 4)와 같으며 4명의 철학자가 모두 왼쪽 포크만을 들고 있는 교착상태를 나타내고 있다.

5.2 잠자는 이발사 문제

Dijkstra가 제안한 잠자는 이발사문제도 병행 프로그래밍의 예제로서 많이 인용된다. 이 문제



(그림 4) 철학자의 저녁식사 실행결과

를 간단히 기술하면 다음과 같다[2]. 「한 이발소에 이발용 의자 하나와 n개의 대기용 의자가 있다. 손님은 대기용 의자 중에 하나가 비었으면 들어와 기다리고, 없으면 다른 이발소로 간다. 이발사는 대기 중인 손님이 없으면 잠을 자고, 손님이 있으면 이발용의자에서 한 명씩 이발을 한다(상호배제). 도착한 손님은 대기중인 손님이 없을 때, 이발사를 깨워 이발을 시작한다. 이발사와 손님이 상호 연동하는 병행 프로그래밍을 작성하시오」

이 문제는 상당히 난해하다고 생각될 수 있지만, 자세히 고찰해보면 생산자와 소비자문제로 단순화된다. 즉 손님을 발생시키는 스레드(생산자)와 발생된 손님(생산된 데이터)을 처리하는 이발사 스레드(소비자)를 작성하면 된다. n개의 대기용의자는 생산자/소비자문제에서 queue의 크기와 같은 의미이다. 이 이발사 문제를 자바 스레드를 사용하여 구현하면 다음과 같다.

```
class Chairs {
    private int value=0;
    private int max;
    private boolean sleep = false;
    public Chairs(int max)
    { this.max = max; }
  synchronized void service() throws
Exception
  { while( value == 0) {
        sleep = true;
        System.out.println("Barber is
sleeping");
        wait();
    }
    if( value-- == max) notify();
  }
  synchronized void enter() throws Exception
  { while(value == max) {
        System.out.println("\t\tNo more chair");
        wait();
    }
    value++;
    if( sleep == true) {
        sleep = false;
        System.out.println("\t\t\tWakeup
barber");
        notify();
    }
  }
  synchronized int val() { return(value); }
}
class Barber extends Thread {
  private Chairs cs;
  private int id;
  private int MAX = 60;
  public Barber(Chairs c)
  { cs = c; }
  public void run() {
  for(int x=0; x<MAX; x++) {
  try {
    cs.service();
    System.out.println("A customer is
served");
    sleep((int)(1400*Math.random()));
  } catch(Exception e){}
  }
}}
class Customer extends Thread {
    private Chairs cs;
    private int id;
    private int MAX = 60;
    public Customer(Chairs s)
    { cs = s; }
    public void run() {
    for(int x=0; x<MAX; x++) {
      try {
        cs.enter();
        System.out.println("New customer "+
        "entered. Total # of customer
"+cs.val());
        sleep((int)(1400*Math.random()));
      } catch(Exception e){}
}}}
public class SleepBarber {
```

```
    public static void main(String arg[]) {
      Chairs cs = new Chairs(4); // 4 chairs
      Barber b = new Barber(cs);
      Customer c = new Customer(cs);
      b.start();      c.start();
}}
```

6. 결론

운영체제는 전산관련 전공자에게는 필수적인 과목이며 병행 처리는 운영체제 학습에 있어서 가장 어려운 부분이다. 많은 전공자들이 비동기 병행 처리 부분을 이해하는 데 어려움을 겪는 이유는 적절한 병행 프로그래밍 언어를 사용하지 못하고 교재의 pseudo 코드만을 사용하기 때문이다.

최근에는 운영체제의 병렬 분산 pseudo 코드대신에 자바 코드를 사용한 교재[6]가 출간되었다. 그러나 [6]에서는 모든 코드를 자바로 작성함으로 오히려 교재를 더 이해하기 어렵게 만들었다. 간결한 pseudo 코드를 사용하여 기본 개념을 이해하고 자바 언어로 구현해보는 것이 학습효과를 높이는 올바른 교수법이라고 사료된다.

본 논문에서는 분산 환경의 종합 솔루션으로서 널리 사용되고 있는 자바언어를 사용하여 운영체제의 비동기 병행 처리 부분을 구현하여 운영체제 교육에 활용할 수 있도록 하였다. 본 논문에 소개한 자바 코드를 운영체제 교육에 실제 활용해 본 결과 자바언어를 이해하고 있는 많은 학생들에게 많은 학습 효과가 있었으며, 자바 스레드 프로그래밍의 학습효과도 적지 않았다. 또한 본 논문에서 소개한 모든 소스 코드는 http://myhome.naver.com/4java/의 즐겨 찾기에 올려놓았다. 앞으로 제시된 자바 프로그램을 사용하여 애니메이션 기능을 추가한다면 보다 높은 학습 효과를 기대할 수 있을 것이다.

참고 문헌

[1] Silberschatz, Galvin (1998). Operating System Concepts (5th Edition). Addison Wesely.

[2] H. Deitel (1990). Operating Systems (2nd Edition). Addison Wesley.

[3] 김일민(2000). 알기쉬운 JAVA. 홍릉과학출판사.

[4] B. Lewis, D. Berg (2000). multithreaded programming with JAVA. Sun Micro Systems Press.

[5] The Source for Java(TM) Technology. http://java.sun.com/

[6] A. Silberschatz, P. Galvin, G. Gagne (2000). Applied Operating System Concept (1st Edition). John Wiley & Sons, Inc.

김 일 민

1984 경북대학교 전자과 학사
1989 뉴저지공대 전산학 석사
1995 아리조나 주립대 전산학 박사
1984-87 ETRI 연구원
1996 ~ 1997 삼성 SDS 교육 개발센터
1997 ~ 현재 한성대학교 컴퓨터 공학과 조교수