4장. 쓰레드 (Thread)

순천향대학교 컴퓨터공학과
이 상 정

강의 목표 및 내용

- 목표
  - 다중 쓰레드 컴퓨터 시스템의 기초를 이루는 CPU 이용의 기본 단위인 쓰레드를 소개
  - Pthreads API 및 Win32와 Java 쓰레드 라이브러리 소개
- 내용
  - 개요
  - 다중 쓰레드 모델
  - 쓰레드 라이브러리
  - 쓰레드 관련 문제들
  - 운영체제 사례

쓰레드 개요

- 쓰레드는 CPU 이용의 기본 단위
- 쓰레드는 쓰레드 ID, 프로그램 카운터, 레지스터집합, 스택으로 구성
- 쓰레드는 같은 프로세스에 속한 다른 쓰레드와 코드, 데이터 섹션 그리고, 열린 파일이나 신호와 같은 운영체제 자원들을 공유

단일 및 다중 쓰레드(multithreaded) 프로세스

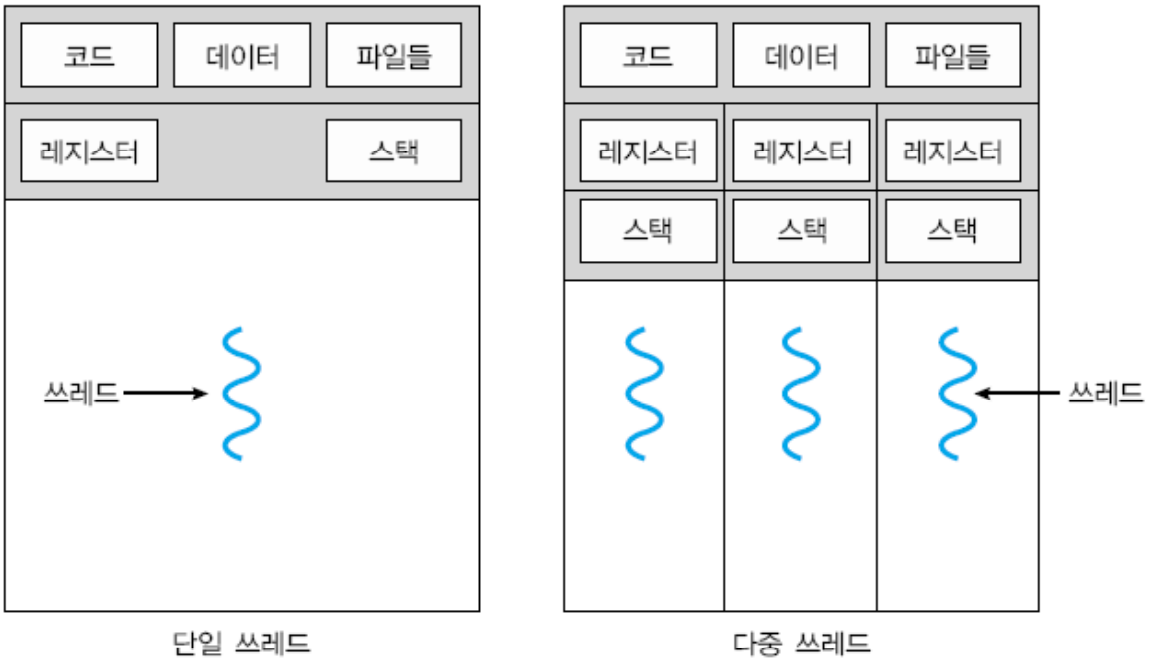
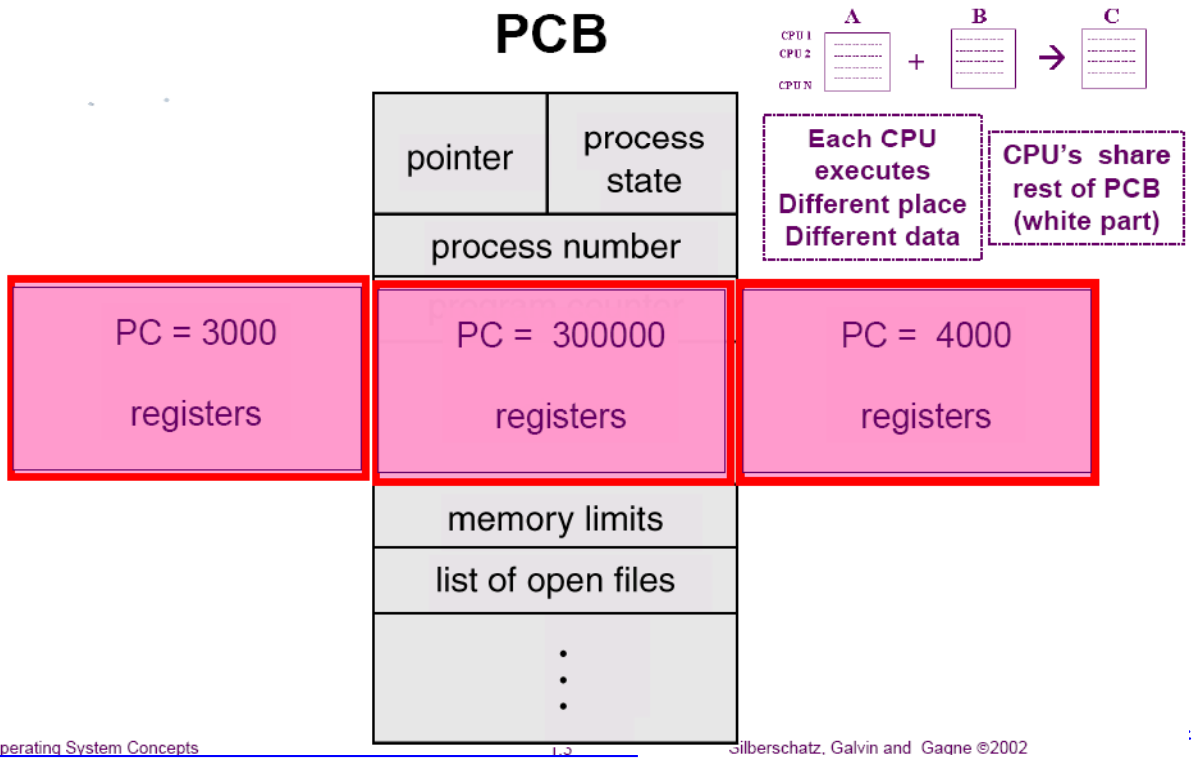


그림 4.1 단일 및 다중 쓰레드 프로세스

한 프로세스 – 3개 쓰레드 PCB



다중 쓰레드 장점

- 응답성 (responsiveness)
  - 사용자에 대한 응답성을 증가
  - 다중 쓰레드 웹 브라우저는 한 쓰레드가 이미지 파일을 적재하고 있는 동안, 다른 쓰레드에서 사용자와의 상호 작용이 가능
- 자원 공유 (resource sharing)
  - 프로세스의 자원들과 메모리를 공유
  - 한 응용 프로그램이 같은 주소 공간 내에 여러 개의 다른 작업을 하는 쓰레드 수행 가능
- 경제성 (economy)
  - 쓰레드 생성과 문맥교환 오버헤드가 프로세스 보다 적음
- 다중 처리기 구조의 활용 (utilization of multiprocessor architectures)
  - 각각의 쓰레드가 다른 처리기(CPU)에서 병렬로 수행

사용자 및 커널 쓰레드

- 사용자 쓰레드 (user thread)
  - 사용자 수준 쓰레드 라이브러리로 관리
  - 주요 쓰레드 라이브러리 예
    - POSIX Pthreads, Win32 쓰레드, Java 쓰레드
- 커널 쓰레드 (kernel thread)
  - 커널이 지원
  - 예
    - Windows XP/2000, Solaris, Linux, Tru64 UNIX, Mac OS X

다중 쓰레드 모델 (Multithreading Model)

- 사용자 쓰레드와 커널 쓰레드와의 연관관계
  - 다대일 모델 (Many-to-One Model)
  - 일대일 모델 (One-to-One Model)
  - 다대다 모델 (Many-to-Many Model)

다대일 모델 (Many-to-One Model)

- 많은 사용자 수준 쓰레드를 하나의 커널 쓰레드로 사상
  - 사용자 공간의 쓰레드 라이브러리가 쓰레드 관리
  - 한 쓰레드가 봉쇄형 시스템 호출을 할 경우, 전체 프로세스가 봉쇄
  - 다중 쓰레드가 다중 처리기에서 병렬로 작동할 수 없음
  - Solaris의 쓰레드 라이브러리 (green thread), GNU Portable 쓰레드

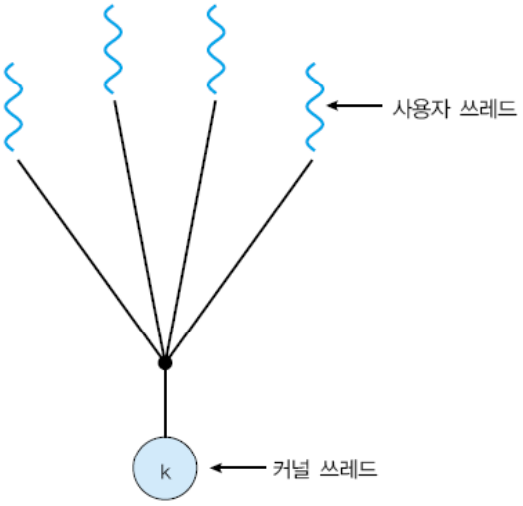


그림 4.2 다대일 모델

일대일 모델 (One-to-One Model)

- 각 사용자 쓰레드를 각각 하나의 커널 쓰레드로 사상
  - 하나의 쓰레드가 봉쇄적 시스템 호출을 하더라도 다른 쓰레드가 실행 가능
  - 다중 처리기에서 다중 쓰레드가 병렬로 수행되는 것을 허용
  - 사용자 수준 쓰레드를 생성할 때 커널 쓰레드를 하는 오버헤드로 응용 프로그램의 성능저하
  - Windows NT/XP/2000, Linux, Solaris 9

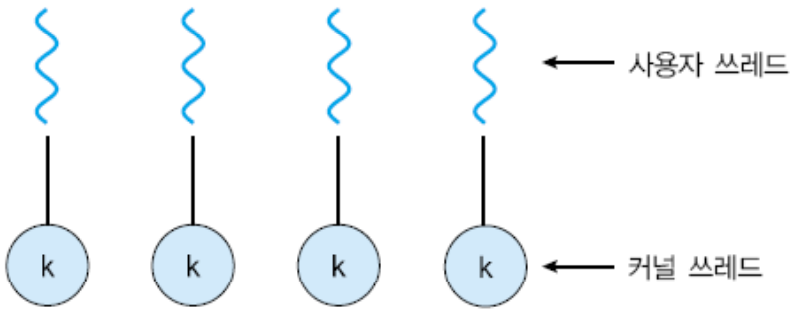


그림 4.3 일대일 모델

다대다 모델 (Many-to-Many Model)

- 여러 개의 사용자 수준 쓰레드를 그보다 작거나 같은 수의 커널 쓰레드로 다중화
  - 운영체제가 개발자는 필요한 만큼 많은 커널 쓰레드 생성을 허용
  - Solaris 9 이전 버전

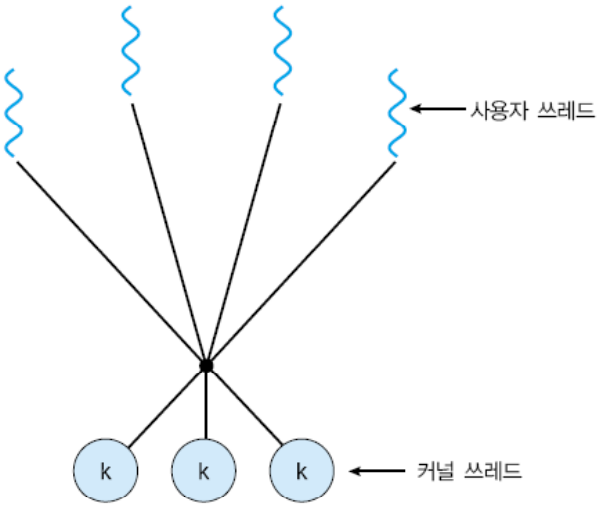


그림 4.4 다대다 모델

쓰레드 라이브러리 (Threads Library)

- 쓰레드 라이브러리는 프로그래머에게 쓰레드를 생성하고 관리하기 위한 API를 제공
- 쓰레드 라이브러리를 구현
  - 커널의 지원 없이 완전히 사용자 공간에서만 라이브러리를 제공
  - 운영체제에 의해 지원되는 커널 수준 라이브러리를 구현
- 주로 사용되는 세 종류 라이브러리
  - POSIX 표준안의 쓰레드 확장판인 Pthread
    - 사용자 또는 커널 수준 라이브러리로서 제공
  - Win32 쓰레드 라이브러리
    - Window 시스템에서 사용 가능한 커널 수준 라이브러리
  - Java 쓰레드 API
    - Java 프로그램에서 직접 쓰레드 생성과 관리

Pthread

- POSIX(IEEE 1003.1c)가 쓰레드 생성과 동기화를 위해 제정한 표준 API
  - 쓰레드의 동작에 관한 명세일 뿐이지 그것 자체를 구현한 것은 아님
  - 대부분의 UNIX 운영체제에서 적용
    - Solaris, Linux, Mac OS X
- 다중 쓰레드 프로그램 예
  - 명령어 라인에서 N 값 입력

$$sum = \sum_{i=0}^{N} i$$

Pthread API 사용 예 (1)

```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   if (argc != 2) {
      fprintf(stderr,"usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
      return -1;
   }
```

```
    /* get the default attributes */
    pthread_attr_init(&attr);
    /* create the thread */
    pthread_create(&tid,&attr,runner,argv[1]);
    /* wait for the thread to exit */
    pthread_join(tid,NULL);

    printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
    int i, upper = atoi(param);
    sum = 0;

    for (i = 1; i <= upper; i++)
        sum += i;

    pthread_exit(0);
}
```

그림 4.6 Pthread API를 사용하는 다중 쓰레드 C 프로그램

Win32 API 사용 예

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */
/* the thread runs in this separate function */

DWORD WINAPI Summation(LPVOID Param)
{
    DWORD Upper = *(DWORD *)Param;
    for (DWORD i = 0; i <= Upper; i++)
        Sum += i;
    return 0;
}

int main(int argc, char *argv[])
{
    DWORD ThreadId;
    HANDLE ThreadHandle;
    int Param;
    /* perform some basic error checking */
    if (argc != 2) {
        fprintf(stderr,"An integer parameter is required\n");
        return -1;
    }
    Param = atoi(argv[1]);
    if (Param < 0) {
        fprintf(stderr, "An integer >= 0 is required \n");
        return -1;
    }
```

```
    // create the thread
    ThreadHandle = CreateThread(
        NULL, // default security attributes
        0, // default stack size
        Summation, //thread function
        &Param, // parameter to thread fuction
        0, // default creation flags
        &ThreadId); // returns the thread identifier

    if (ThreadHandle != NULL) {
        // now wait for the thread to finish
        WaitForSingleObject(ThreadHandle, INFINITE);

        // close the thread handle
        CloseHandle(ThreadHandle);

        printf("sum = %d\n",Sum);
    }
}
```

그림 4.7 Win32 API를 사용한 다중 쓰레드 C 프로그램

Java 쓰레드(Java Thread)

- Java 쓰레드들은 JVM이 관리
- Java 프로그램에서 쓰레드를 생성하는 기법
  - Thread 클래스로부터 파생된 새로운 클래스를 생성하고, Thread 클래스의 run() 메소드를 무효화(override)하는 것
  - Runnable 인터페이스를 구현하는 클래스를 정의

```
public interface Runnable
{
    public abstract void run();
}
```

Java 쓰레드 예

```
class Sum
{
   private int sum;

   public int getSum() {
      return sum;
   }

   public void setSum(int sum) {
      this.sum = sum;
   }
}

class Summation implements Runnable
{
   private int upper;
   private Sum sumValue;

   public Summation(int upper, Sum sumValue) {
      this.upper = upper;
      this.sumValue = sumValue;
   }

   public void run() {
      int sum = 0;
      for (int i = 0; i <= upper; i++)
         sum += i;
      sumValue.setSum(sum);
   }
}
```


운영체제

```
public class Driver
{
   public static void main(String[] args) {
      if (args.length > 0) {
         if (Integer.parseInt(args[0]) < 0)
            System.err.println(args[0] + " must be >= 0.");
         else {
            // create the object to be shared
            Sum sumObject = new Sum();
            int upper = Integer.parseInt(args[0]);
            Thread thrd = new Thread(new Summation(upper, sumObject));
            thrd.start();
            try {
               thrd.join();
               System.out.Println
                  ("The sum of "+upper+" is "+sumObject.getSum());
            } catch (InterruptedException ie) { }
            }
         }
         else
            System.err.println("Usage:Summation<integer value>");}
}
```

그림 4.8 음수가 아닌 정수의 합을 계산하는 Java 프로그램

쓰레드와 관련된 문제들 (1) (Threading Issues)

Fork() 및 Exec() 시스템 호출

- 한 프로그램의 쓰레드가 fork()를 호출하면 새로운 프로세스는 모든 쓰레드를 복제 또는 호출한 쓰레드만 복제하는가?
- fork() 후 exec() 호출하는 경우 호출한 쓰레드만 복사해주는 것이 적절

취소 (cancellation)

- 쓰레드 취소(thread cancellation)는 쓰레드가 끝나기 전에 그것을 강제 종료시키는 작업
- 2가지 방식
  - 비동기식 취소(asynchronous cancellation)
    한 쓰레드가 즉시 목적 쓰레드를 강제 종료
  - 지연 취소(deferred cancellation)
    목적 쓰레드가 주기적으로 자신이 강제 종료되어야 할지를 점검

쓰레드와 관련된 문제들 (2)

신호 처리 (signal handling)

- 신호는 UNIX에서 프로세스에게 어떤 사건(event)이 일어났음을 알려주기 위해 사용
- 신호는 다음과 같은 형태로 전달
  - 신호는 특정 사건이 일어나야 생성
  - 신호가 생성되면 프로세스에게 전달
  - 신호가 전달되면 반드시 처리
- 다중 쓰레드 프로그램에서는 어느 쓰레드에게 신호를 전달?
  - 신호가 적용될 쓰레드에게 전달
  - 모든 쓰레드에게 전달
  - 몇몇 쓰레드들에게만 선택적으로 전달
  - 특정 쓰레드가 모든 신호를 전달받도록 지정
- 신호를 전달하는 표준 UNIX 함수는 kill(aid_t aid, int signal)
- POSIX Pthread는 pthread_kill(pthread_t tid, int signal) 함수도 제공

쓰레드와 관련된 문제들 (3)

- 쓰레드 풀 (thread pool)
  - 프로세스를 시작할 때 아예 일정한 수의 쓰레드들을 미리 풀로 생성
  - 장점
    - 새 쓰레드를 만들어 주기보다 기존 쓰레드로 서비스해 주는 것이 더 빠름
    - 쓰레드 풀은 동시에 존재할 쓰레드 개수에 제한
- 쓰레드별 데이터 (thread-specific data)
  - 각 쓰레드가 자기만 액세스할 수 있는 자료
  - 대부분의 쓰레드 라이브러리들은 어떤 형태로든 이와 같은 쓰레드별 데이터를 지원

실습과제 (1)

1. 다음과 같은 p.150 그림 4.6의 수정된 버전을 아래와 같이 두 개의 터미널에서 실행하고 결과 분석
   - 사용된 pthread API 설명 분석
   - 컴파일 시 다음과 같이 -pthread 옵션을 기술
     $ gcc -pthread prog.c
   - 터미널1에서 소스 작성하고 실행 시작 후 문자 입력 전 대기
   - 터미널2를 생성하고 다음을 실행하고 출력 분석
     $ ps -am -L
   - 터미널 1에서 문자 입력하고 실행 결과 분석

실습과제 (2)

```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   if (argc != 2) {
            fprintf(stderr,"usage: a.out <integer value>₩n");
            return -1;
      }
      if (atoi(argv[1]) < 0) {
            fprintf(stderr,"%d must be >= 0₩n",atoi(argv[1]));
            return -1;
      }
```



```
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d₩n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);

   sum = 0;
   for (i = 1; i <= upper; i++)
            sum += i;

   printf("Input any character to continue₩n");
   getchar();

   pthread_exit(0);
}
```

실습과제 (3)

2. 다음과 같은 p.166 그림 4.11의 수정된 버전을 다음과 같이 두 개의 터미널에서 실행하고 결과 분석
   - 터미널1에서 소스 작성하고 실행 시작 후 문자 입력 전 대기
   - 터미널2를 생성하고 다음을 실행하고 출력 분석
     $ ps –am –L
   - 터미널 1에서 문자 입력하고 실행 결과 분석


```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
  int pid;
  pthread_t tid;
  pthread_attr_t attr;

  pid = fork();

  if (pid == 0) { /* child process */
        pthread_attr_init(&attr);
        pthread_create(&tid, &attr, runner, NULL);
        pthread_join(tid, NULL);
        printf("CHILD: value = %d₩n", value); /* LINE C */
  }
  else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d₩n", value); /* LINE P */
  }
}
```

```
void *runner(void *param)
{
  value = 5;

  printf("Input any character to continue₩n");
  getchar();

  pthread_exit(0);
}
```


실습과제 (4)

3. 다음은 p.167 행렬 곱하기 프로젝트의 C와 Java프로그램이다. 프로그램을 실행하고 교과서의 내용을 참조하여 프로그램 내용과 결과를 분석하라.
   - Java 프로그램은 PC 윈도우에서 실행
     - 가상머신에서 실행할 경우 JDK 설치 후 실행

$$C_{i,j} = \sum_{n=1}^{K} A_{i,n} \times B_{n,j}$$

28	23	18	=	1	4	X	8	7	6
41	34	27		2	5		5	4	3
54	45	36		3	6				

```
// C 버전 행렬 곱하기 프로그램

#include <pthread.h>
#include <stdio.h>

#define M 3
#define K 2
#define N 3

//Matrixes
int A [M][K] = {{1,4}, {2,5}, {3,6}};
int B [K][N] = {{8,7,6}, {5,4,3}};
int C [M][N];

struct matrixValues
{
      int i; //Matrix Row
      int j; //Matrix Column
};

void *matrixThread(void *param);
```

```
int main(int argc, char *argv[])
{
      int i, j;

      for(i = 0; i < M; i++)
      {
          for(j = 0; j < N; j++)
          {
              // Create structure adding row and
              // column identifieres
              struct matrixValues *mval = (struct matrixValues*)
                                  malloc(sizeof(struct matrixValues));
              mval->i = i;
              mval->j = j;

              // Create Thread to calculate value of matrix
              // at row column
              pthread_t tid;
              pthread_attr_t attr;
              pthread_attr_init(&attr);
              pthread_create(&tid, &attr, matrixThread, mval);
              pthread_join(tid,NULL);
          }
      }
```



```
      // Dispaly results of matrix multiplication
      printf("₩nResults for Matrix Multiplication₩n");

      // Cycle through result matrix and print value
      for(i = 0; i < M; i++)
      {
          for(j = 0; j < N; j++)
          {
              printf("%d₩t", C[i][j]);
          }
          printf("₩n");
       }
      return 0;
}

// Thread method to calculate value from mutiplying A an B matricies
void *matrixThread(void * param)
{
      // Create structure and assign data from param to it.
      struct matrixValues threadVal;
      threadVal = *(struct matrixValues*)param;

      int a = threadVal.i;
      int b = threadVal.j;
      //Perform the matrix multiplication
      C[a][b] = (A[a][0] * B[0][b]) + (A[a][1] * B[1][b]);
}
```

```
// Java 버전 행렬 곱하기 프로그램

public class HW4JavaThread
{
    private static final int M = 3;
    private static final int K = 2;
    private static final int N = 3;

    public static void main(String Agrv[])
    {
      // Initialize matrices
      int[][] A = {{1,4}, {2,5}, {3,6}};
      int[][] B = {{8,7,6}, {5,4,3}};
      int[][] C = new int[M][N];

      for(int i = 0; i < M; i++)
      {
          // Create treads to calculate matrix results
          for(int j = 0; j < N; j++)
          {
             //Create and start thread
            Thread jThread = new Thread(new
      HW4Thread(i, j, A, B, C));
            jThread.start();
```

```
            try
            {
               //Wait for threads to complete

               // Wait for threads to complete
               jThread.join();
            }
            catch(InterruptedException ie)
            {
               System.err.println(ie);
            }
        }
      }

      System.out.println("Matrix Multiplication
      Results");
      //Print results
      for(int i = 0; i < M; i++)
      {
          for(int j = 0; j < N; j++)
          {
              System.out.print(C[i][j] + "₩t");
          }
          System.out.println();
      }
    }
}
```



```
//Thread class
class HW4Thread implements Runnable
{
      private int row, column;
      private int[][] A, B, C;

      public HW4Thread(int r, int col, int[][]a, int[][]b, int[][]c)
      {
          //Assign references
          row = r;
          column = col;
          A = a;
          B = b;
          C = c;
      }

      //Thread begins execution here
      public void run()
      {
           //Calculate value for matrix
          C[row][column] = (A[row][0]*B[0][column]) +
      (A[row][1]*B[1][column]);
      }
```