

Chapter 4: Threads



Chapter 4: Threads

- ❑ Overview
- ❑ Multithreading Models
- ❑ Threading Issues
- ❑ Pthreads
- ❑ Windows XP Threads
- ❑ Linux Threads
- ❑ Java Threads

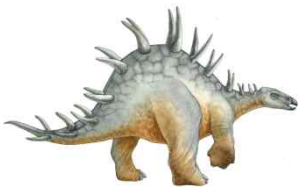


Threads 개요

- A *thread* (or *lightweight process*) is a basic unit of CPU utilization; it consists of:
 - thread ID
 - program counter
 - register set
 - stack space

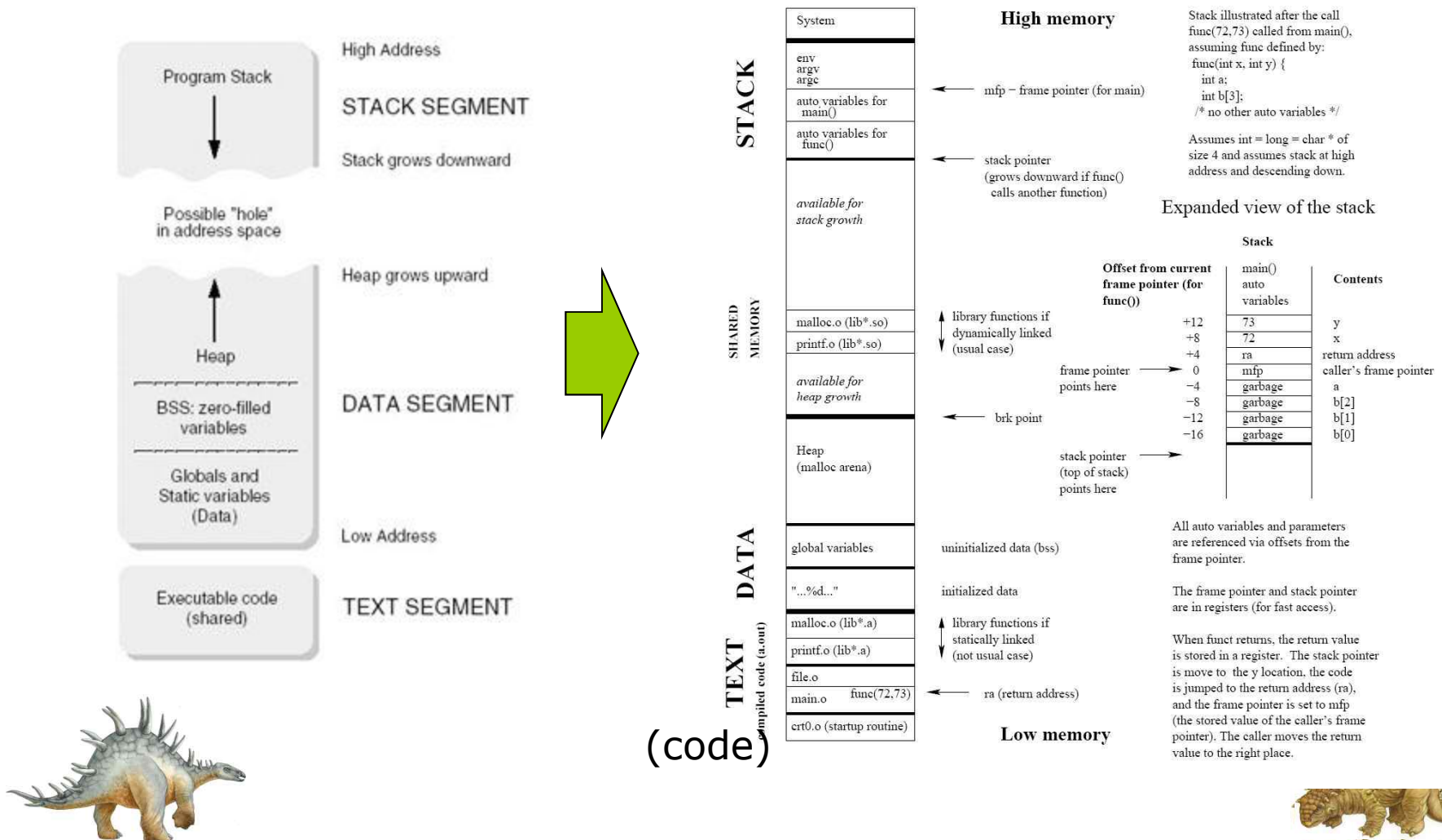
- A thread shares with its peer threads its:
 - code section
 - data section
 - operating-system resourcescollectively known as a *task*.

- A traditional or *heavyweight* process is equal to a task with one thread



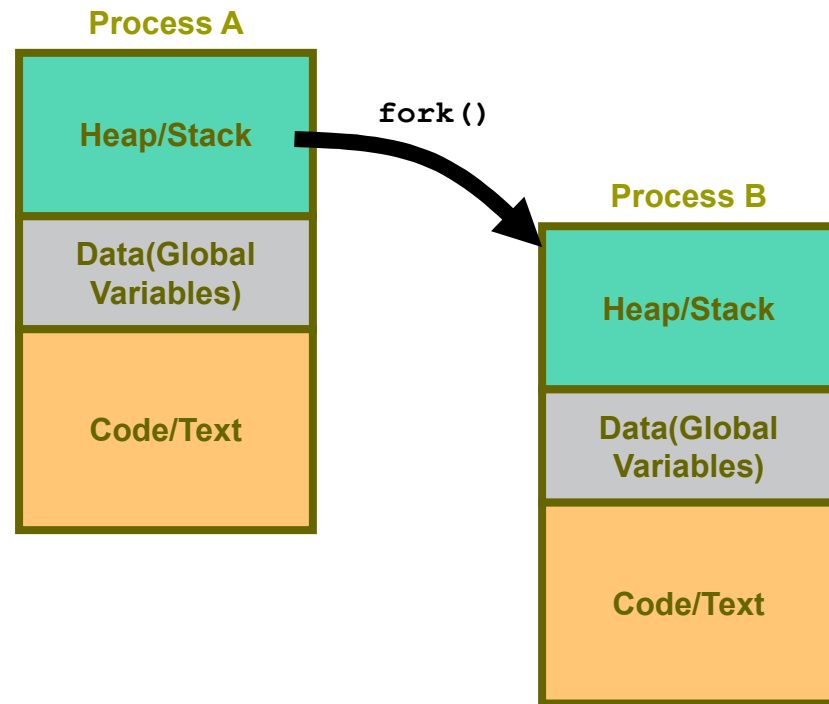
Threads 개요

□ Process의 메모리 구조(상세)

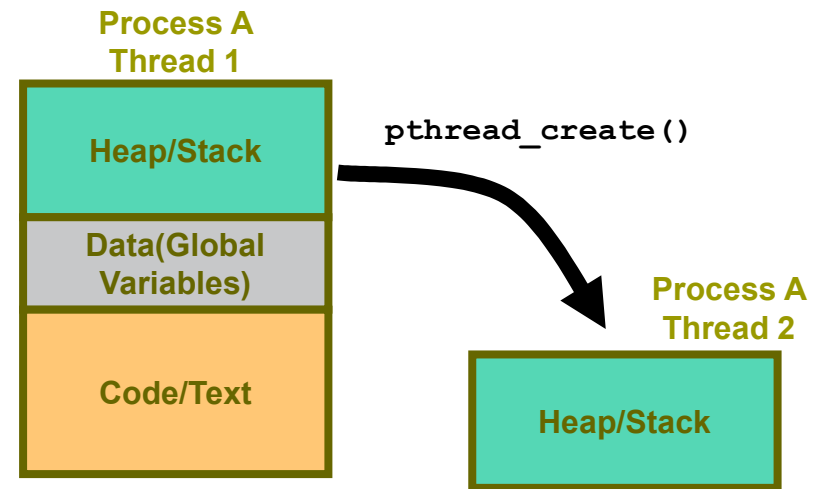
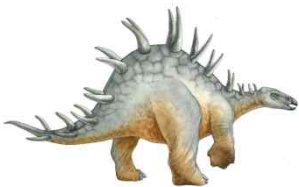


Threads 개요

□ Process와 Thread의 차이



Process

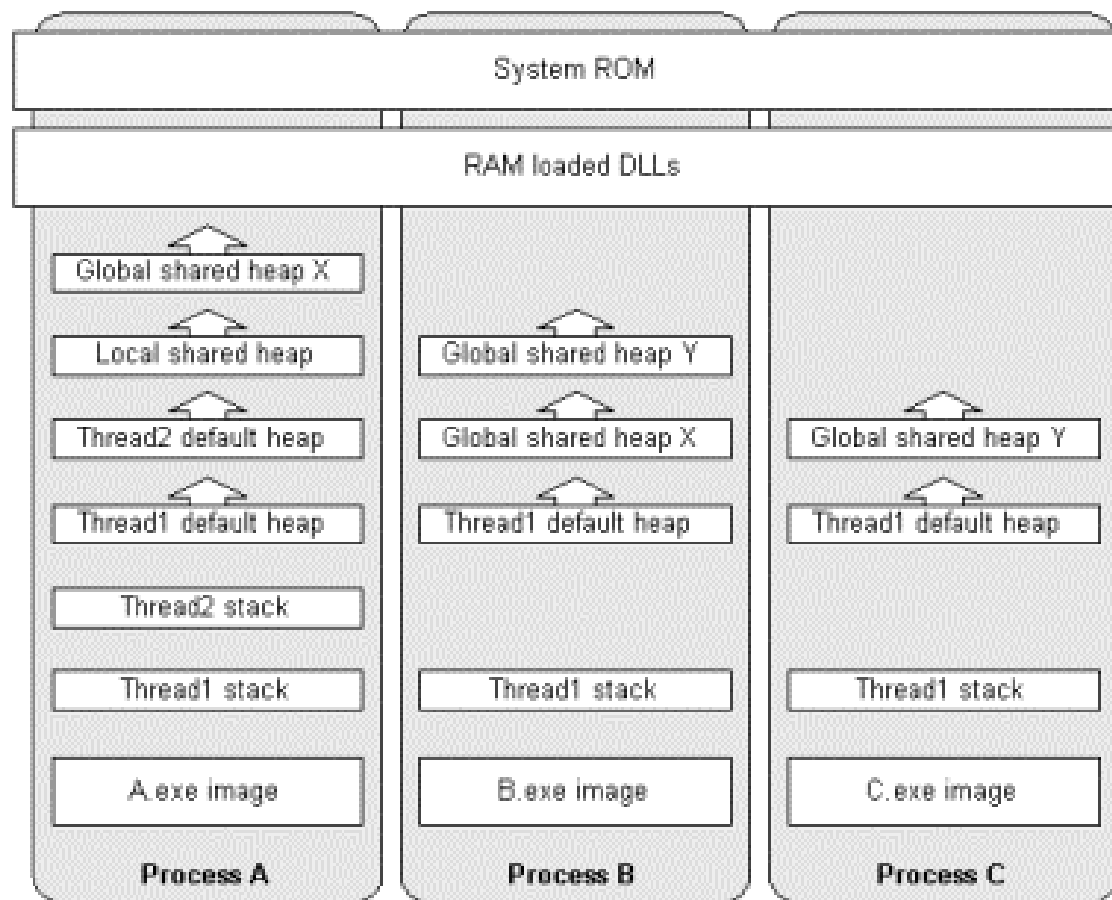


Thread

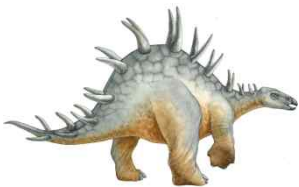
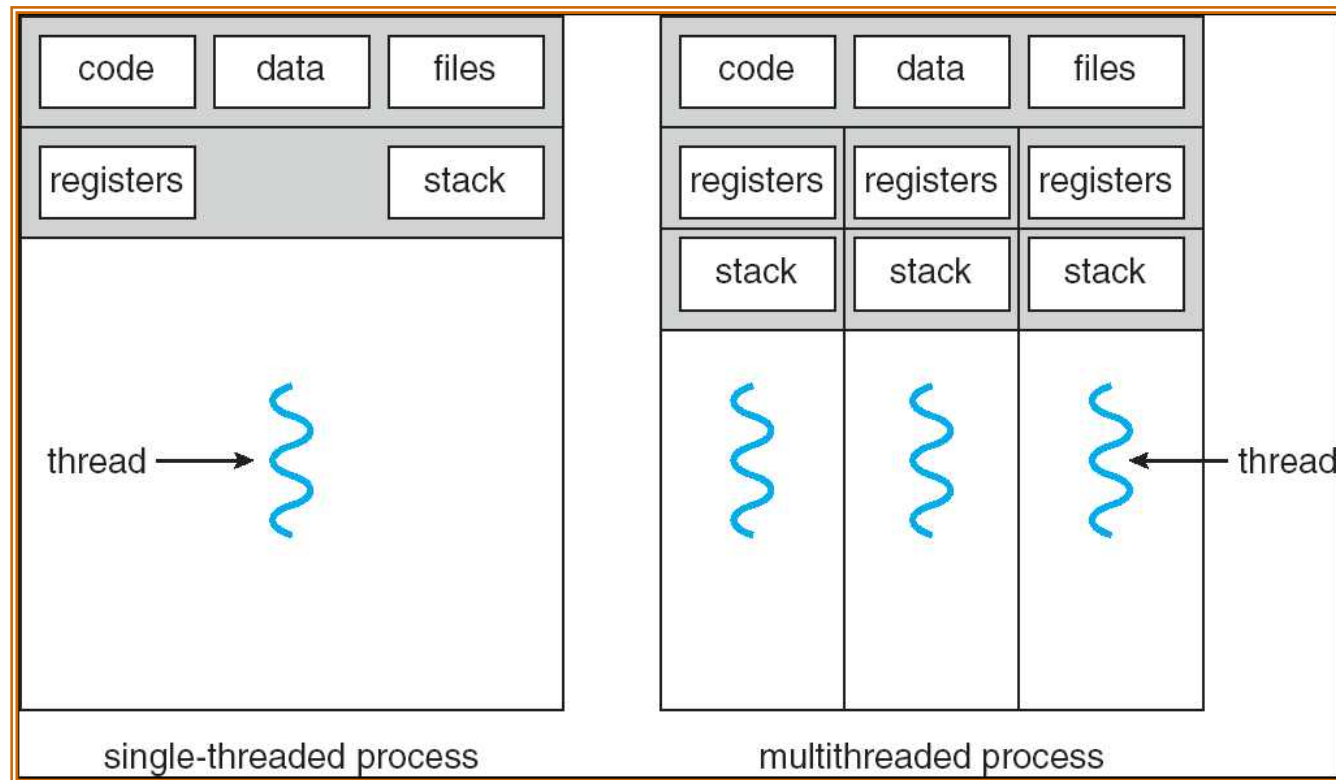


Threads 개요

□ Process와 Thread의 차이



Single and Multithreaded Processes



Benefits

□ Responsiveness

- eg) multi-threaded Web - if one thread is blocked (eg network) another thread continues (eg *display*)

□ Resource Sharing

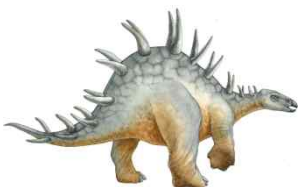
- n threads can share binary code, data, resource of the process (files, crt, ...)

□ Economy

- creating and context switching thread (rather than a process)
- Solaris: 30µH 5µH

□ Utilization of MP Architectures

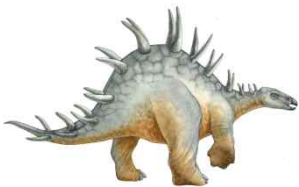
- each thread may be running in *parallel* on a different processor



User and Kernel Threads

□ User Thread

- Thread management done by user-level threads library
- 라이브러리는 커널의 지원없이 쓰레드의 생성과 스케줄링, 관리를 지원
- 커널을 통하지 않으므로, 생성과 관리가 빠르나 봉쇄형 시스템 콜을 수행하는 사용자 수준의 쓰레드는 다른 쓰레드와 함께 스케줄링 되지 않음



User and Kernel Threads

□ Kernel Thread

■ Supported by the Kernel

- 커널 수준에서 관리되어 생성과 관리가 느리나 다른 쓰레드와 함께 스케줄링 될 수 있음

■ Examples

- Windows 95/98/NT/2000

- Solaris

- Tru64 UNIX

- BeOS

- Linux

Java는 JVM에 의해 지원되므로,
커널 쓰레드와 사용자 쓰레드의 중간 형태



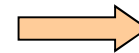
User and Kernel Threads

- Some are supported by *kernel*

eg) Windows 95/98/NT

Solaris

Digital UNIX



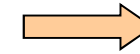
*Kernel
Threads*

- Others are supported by *library*

eg) POSIX *Pthreads*

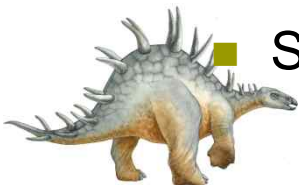
Mach *C-threads*

Solaris *threads*



*User
Threads*

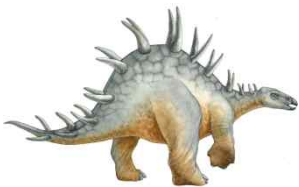
- Some are real-time threads



Multithreading Models

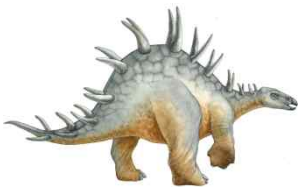
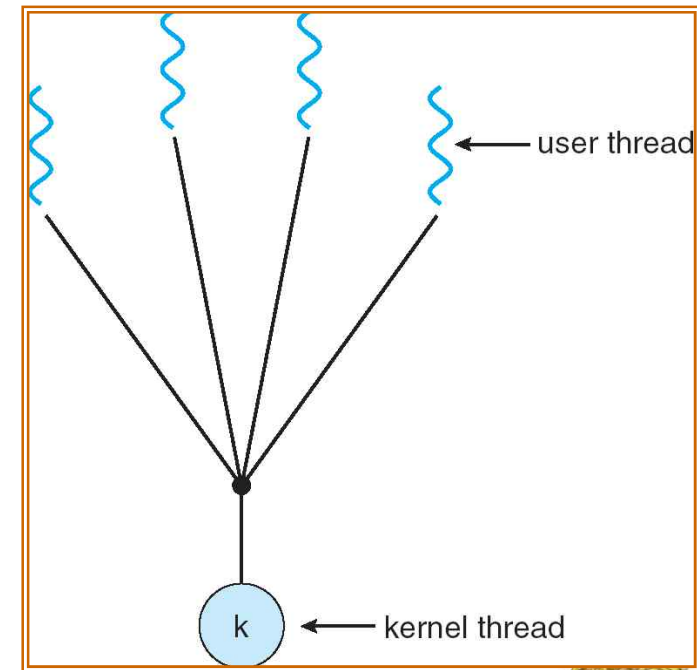
Mapping user threads to kernel threads:

- Many-to-One
- One-to-One
- Many-to-Many
- Two-level Model : Many-to-Many 모델의 변형



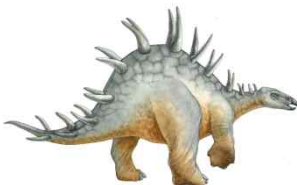
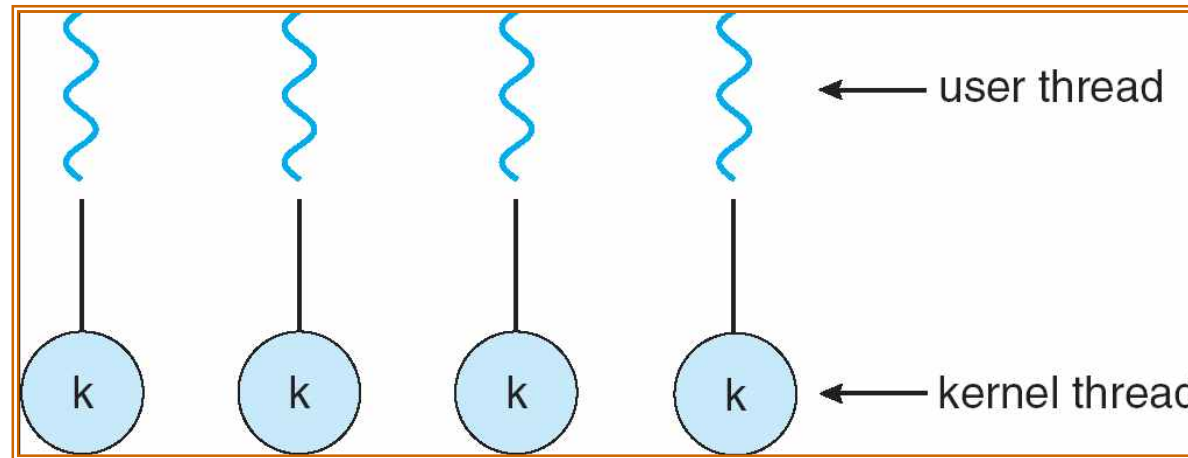
Many-to-One

- ❑ Many user-level threads mapped to single kernel thread
- ❑ Examples:
 - Solaris Green Threads
 - GNU Portable Threads



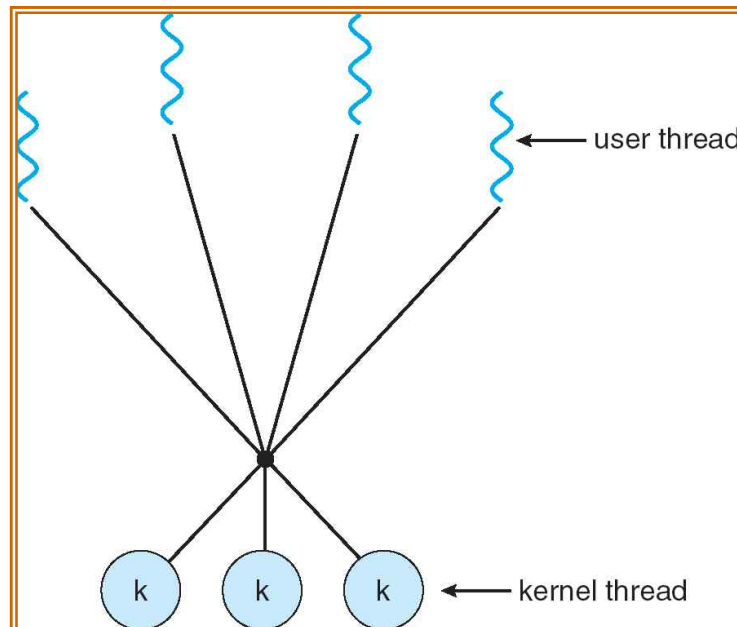
One-to-One

- Each user-level thread maps to kernel thread
- Examples
 - Windows NT/XP/2000
 - Linux
 - Solaris 9 and later



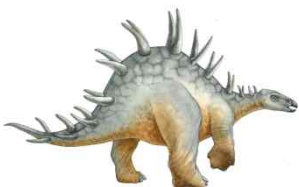
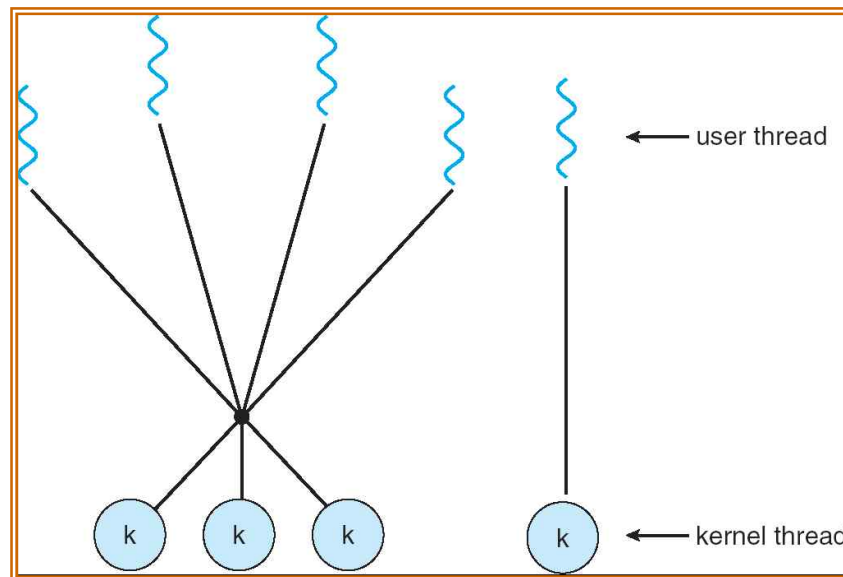
Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
 - Solaris prior to version 9
 - Windows NT/2000 with the *ThreadFiber* package



Two-level Model

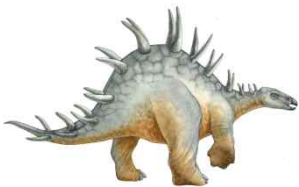
- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Examples
 - IRIX, HP-UX, Tru64 UNIX, Solaris 8 and earlier



Java Threads

- Java threads are managed by the JVM
- Java threads may be created by:
 - Implementing the Runnable interface

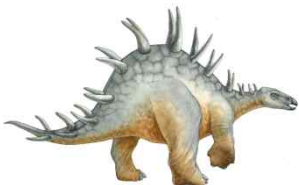
```
public interface Runnable
{
    public abstract void run();
}
```



Java Threads - Example Program

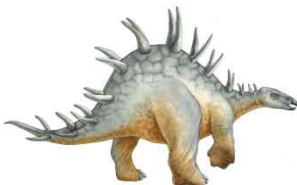
```
class MutableInteger
{
    private int value;
    public int getValue() {
        return value;
    }
    public void setValue(int value) {
        this.value = value;
    }
}

class Summation implements Runnable
{
    private int upper;
    private MutableInteger sumValue;
    public Summation(int upper, MutableInteger sumValue) {
        this.upper = upper;
        this.sumValue = sumValue;
    }
    public void run() {
        int sum = 0;
        for (int i = 0; i <= upper; i++)
            sum += i;
        sumValue.setValue(sum);
    }
}
```

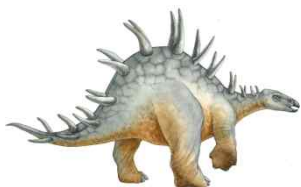
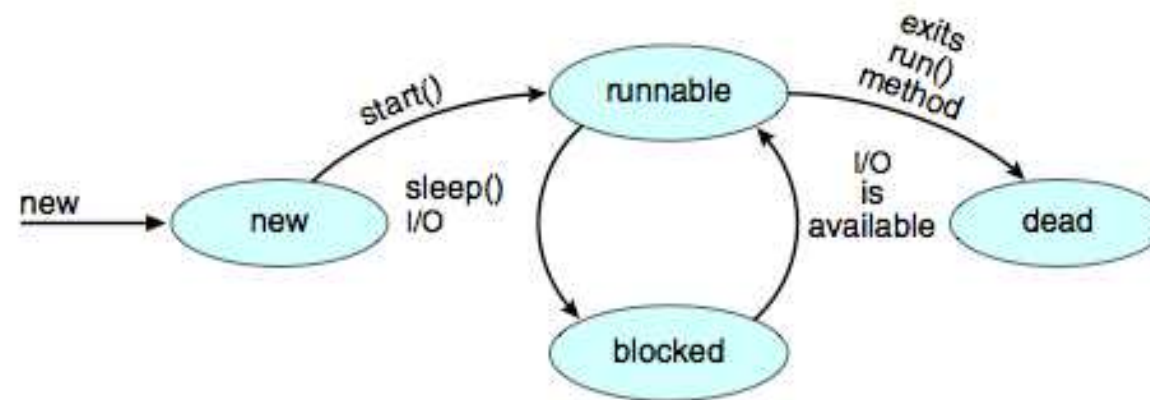


Java Threads - Example Program

```
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
                MutableInteger sum = new MutableInteger();
                int upper = Integer.parseInt(args[0]);
                Thread thrd = new Thread(new Summation(upper, sum));
                thrd.start();
                try {
                    thrd.join();
                    System.out.println
                        ("The sum of "+upper+" is "+sum.getValue());
                } catch (InterruptedException ie) { }
            }
        }
        else
            System.err.println("Usage: Summation <integer value>");
    }
}
```



Java Thread States



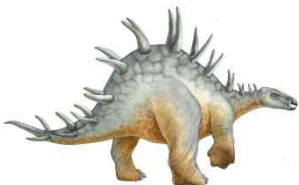
Java Threads - Producer-Consumer

```
public class Factory
{
    public Factory() {
        // First create the message buffer.
        Channel mailBox = new MessageQueue();

        // Create the producer and consumer threads and pass
        // each thread a reference to the mailBox object.
        Thread producerThread = new Thread(
            new Producer(mailBox));
        Thread consumerThread = new Thread(
            new Consumer(mailBox));

        // Start the threads.
        producerThread.start();
        consumerThread.start();
    }

    public static void main(String args[]) {
        Factory server = new Factory();
    }
}
```



Java Threads - Producer-Consumer

```
class Producer implements Runnable
{
    private Channel mbox;

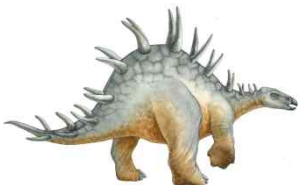
    public Producer(Channel mbox) {
        this.mbox = mbox;
    }

    public void run() {
        Date message;

        while (true) {
            // nap for awhile
            SleepUtilities.nap();

            // produce an item and enter it into the buffer
            message = new Date();

            System.out.println("Producer produced " + message);
            mbox.send(message);
        }
    }
}
```



Java Threads - Producer-Consumer

```
class Consumer implements Runnable
{
    private Channel mbox;

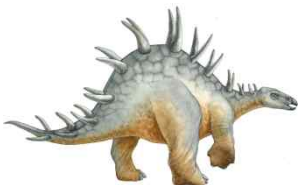
    public Consumer(Channel mbox) {
        this.mbox = mbox;
    }

    public void run() {
        Date message;

        while (true) {
            // nap for awhile
            SleepUtilities.nap();

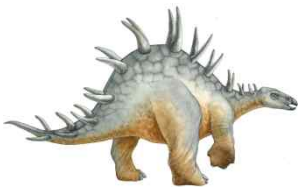
            // consume an item from the buffer
            message = (Date)mbox.receive();

            if (message != null)
                System.out.println("Consumer consumed " + message);
        }
    }
}
```



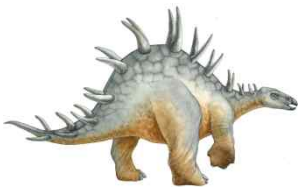
Threading Issues

- ❑ Semantics of **fork()** and **exec()** system calls
- ❑ Thread cancellation
- ❑ Signal handling
- ❑ Thread pools
- ❑ Thread specific data
- ❑ Scheduler activations



Threading Issues – Semantics of fork() and exec()

- **Multithread** 프로그램에서 **fork()**를 호출한다면, 한 개의 **thread**를 생성할 것인가? 아니면 모든 **multithread**를 모두 복사해서 생성할 것인가?
- 두 개 다 지원



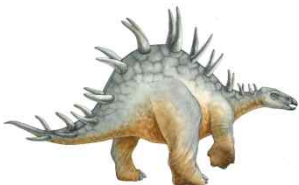
Threading Issues – Thread Cancellation

❑ Terminating a thread before it has finished

- 예를 들면, 여러 스레드들이 데이터베이스를 병렬로 검색하다가 그 중 한 스레드가 결과를 찾은 경우,
- 또는 웹 브라우저에서 사용자가 **stop**을 클릭한 경우

❑ Two general approaches:

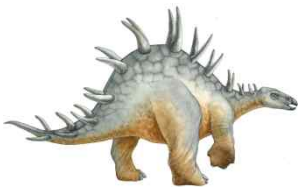
- **Asynchronous cancellation** terminates the target thread immediately
- **Deferred cancellation** allows the target thread to periodically check if it should be cancelled



Thread Cancellation

Deferred cancellation in Java
Interrupting a thread

```
Thread thrd = new Thread(new InterruptibleThread());  
thrd.start();  
.  
.  
thrd.interrupt();
```

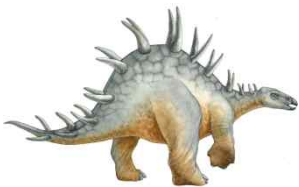


Thread Cancellation

Deferred cancellation in Java
Checking interruption status

```
class InterruptibleThread implements Runnable
{
    /**
     * This thread will continue to run as long
     * as it is not interrupted.
     */
    public void run() {
        while (true) {
            /**
             * do some work for awhile
             * . . .
             */

            if (Thread.currentThread().isInterrupted()) {
                System.out.println("I'm interrupted!");
                break;
            }
        }
        // clean up and terminate
    }
}
```



Signal Handling

- Signal

- Unix에서 특정 Event가 일어났음을 알리기 위해 사용되는 단위(예: Windows Message)

- **signal handler**의 처리 순서

1. Signal이 특정 event에 의해 생성됨
2. Signal이 특정 프로세스에 전달됨
3. Signal이 처리됨

Signal의 예
Synchronous
Divide-by-zero,
illegal-memory-access

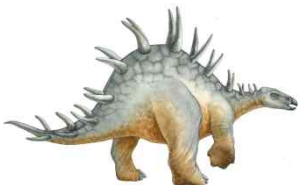
- Process에서의 Signal 처리 선택사항

- Signal이 적용될 특정 Thread에 전송
- Process안에 있는 모든 Thread에 전송됨
- Process안의 다수 Thread에게 전송됨
- 그 Process에 전달되는 모든 Signal을 처리할 특정 Thread를 지정



Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
 - 속도 : Usually slightly faster to service a request with an existing thread than create a new thread
 - 시스템 자원 할당의 한계 설정 : Allows the number of threads in the application(s) to be bound to the size of the pool



Thread Pools

□ Java provides 3 thread pool architectures:

1. **Single thread executor** - pool of size 1.

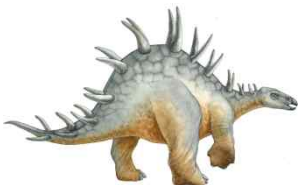
- `static ExecutorService newSingleThreadExecutor()`

2. **Fixed thread executor** - pool of fixed size.

- `static ExecutorService newFixedThreadPool(int nThreads)`

3. **Cached thread pool** - pool of unbounded size

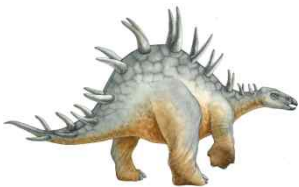
- `static ExecutorService newCachedThreadPool()`



Thread Pools

A task to be serviced in a thread pool

```
public class Task implements Runnable
{
    public void run() {
        System.out.println("I am working on a task.");
        . . .
    }
}
```



Thread Pools

Creating a thread pool in Java

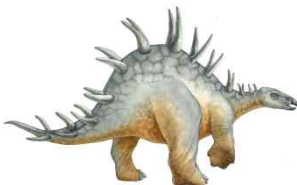
```
import java.util.concurrent.*;

public class TPEXample
{
    public static void main(String[] args) {
        int numTasks = Integer.parseInt(args[0].trim());

        // create the thread pool
        ExecutorService pool = Executors.newCachedThreadPool();

        // run each task using a thread in the pool
        for (int i = 0; i < numTasks; i++)
            pool.execute(new Task());

        // Shut down the pool. This shuts down the pool only
        // after all threads have completed.
        pool.shutdown();
    }
}
```



Thread Specific Data

- Allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)



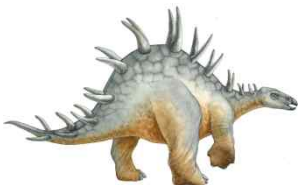
Thread Specific Data

Thread-specific data in Java.

```
class Service
{
    private static ThreadLocal errorCode =
        new ThreadLocal();

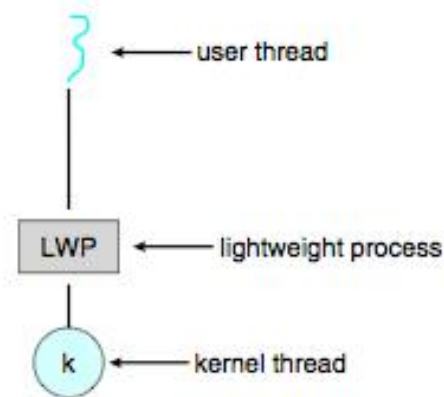
    public static void transaction() {
        try {
            /**
             * some operation where an error may occur
             * . . .
             */
        }
        catch (Exception e) {
            errorCode.set(e);
        }
    }

    /**
     * get the error code for this transaction
     */
    public static Object getErrorCode() {
        return errorCode.get();
    }
}
```



Scheduler Activations

- Scheduler Activation의 목적
 - M:M and Two-level model들은 다수의 Kernel Thread와 Application들과의 통신을 필요로 하며, 통신을 위해 사용되는 방법
 - 통신 방법 : Scheduler activations provide **upcalls**
 - a communication mechanism from the kernel to the thread library
 - This communication allows an application to maintain the correct number kernel threads



운영체제 사례

□ Solaris에서 Thread와 Process의 관계

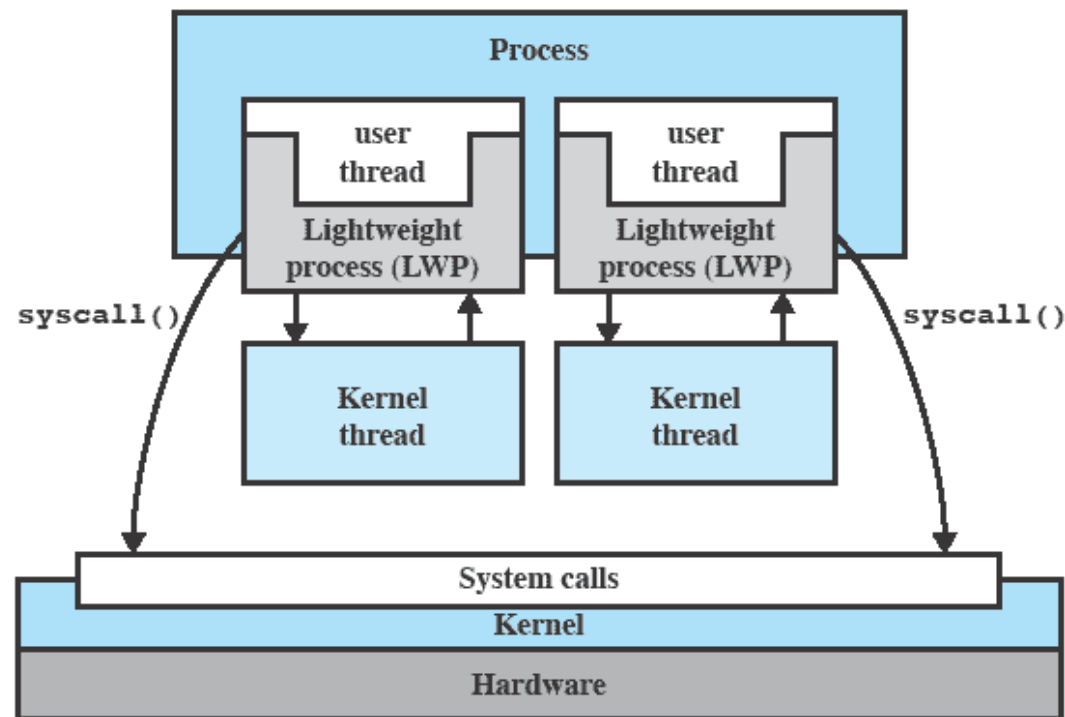


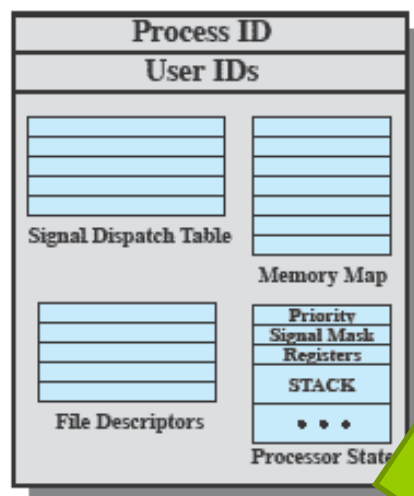
Figure 4.15 Processes and Threads in Solaris [MCDO07]



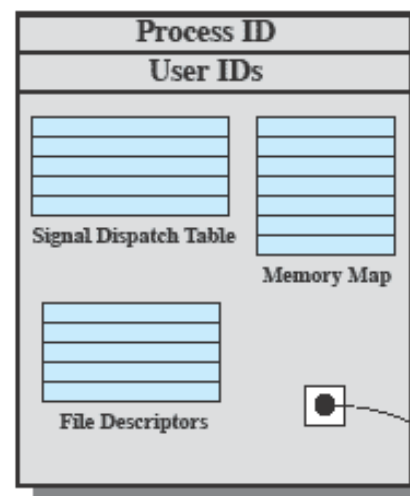
운영체제 사례

□ Unix와 Solaris의 Thread 지원 Process의 비교

UNIX Process Structure



Solaris Process Structure



Solaris replaces the processor state block with a list of LWPs

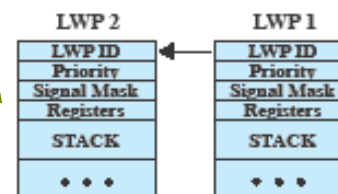


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]



운영체제 사례

□ Solaris에서의 Thread 모델

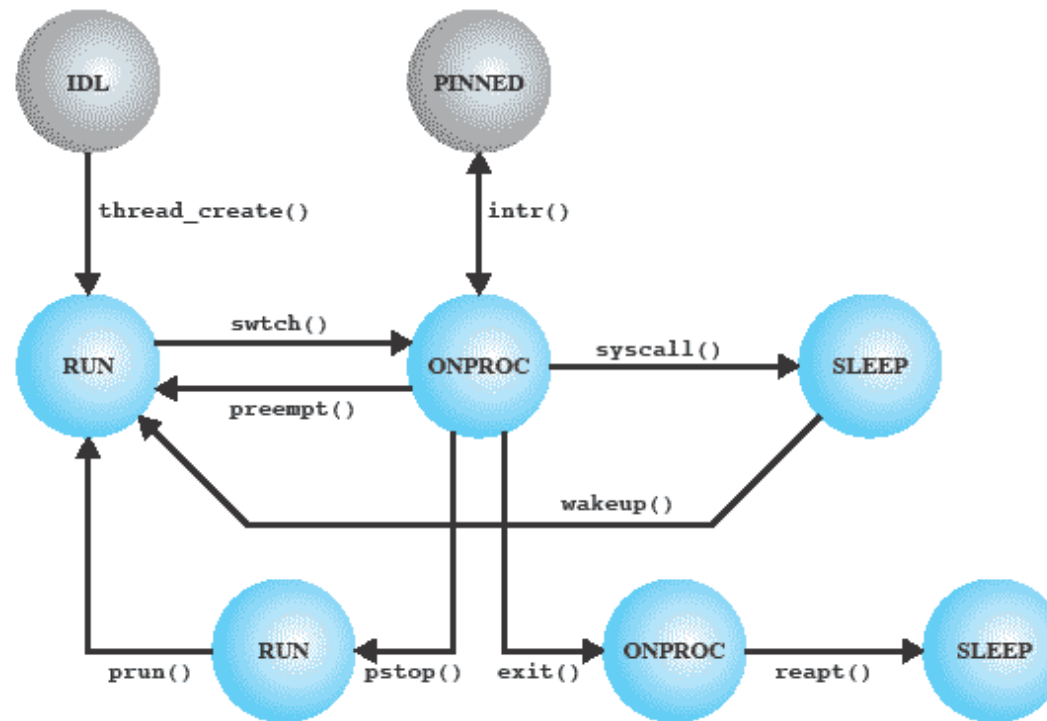
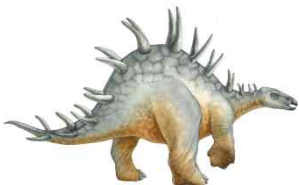


Figure 4.17 Solaris Thread States [MCDO07]



운영체제 사례

□ Linux에서의 Process/Thread 모델

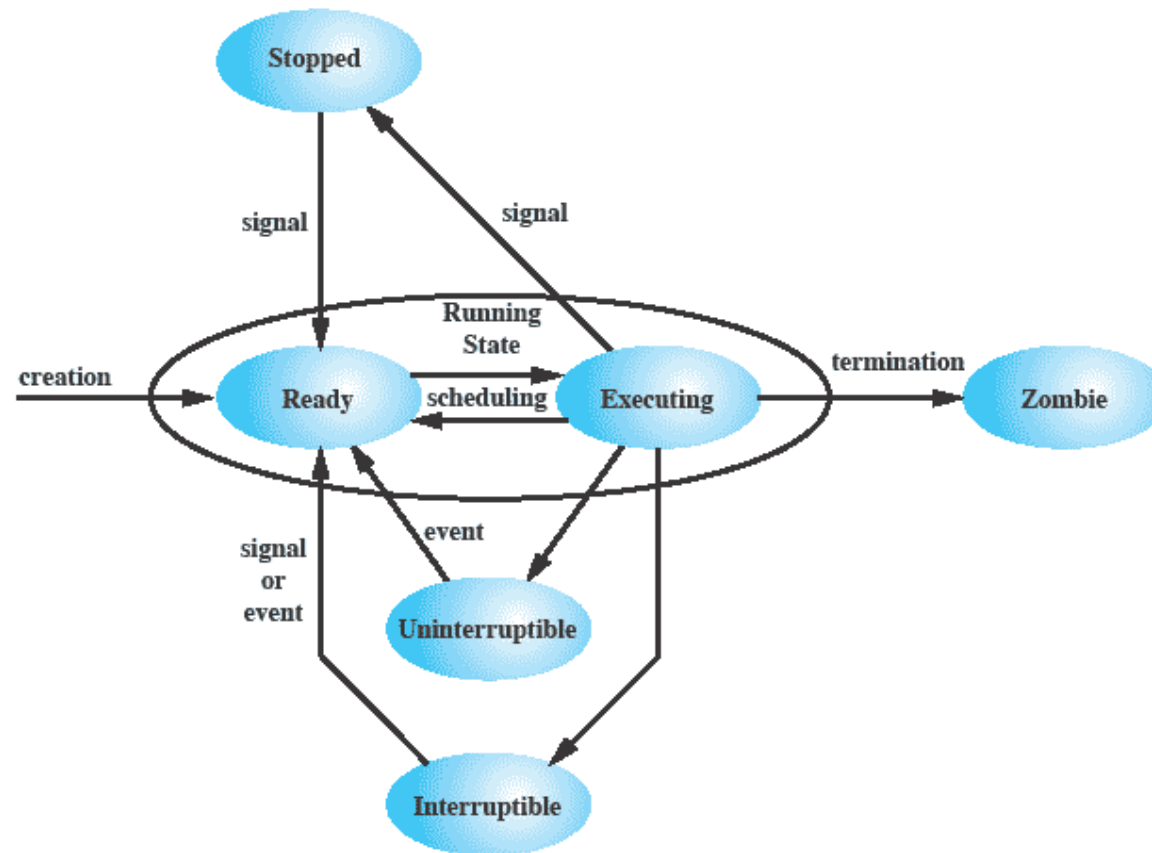
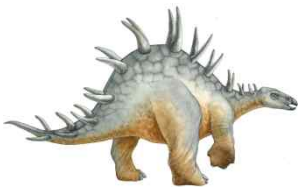


Figure 4.18 Linux Process/Thread Model

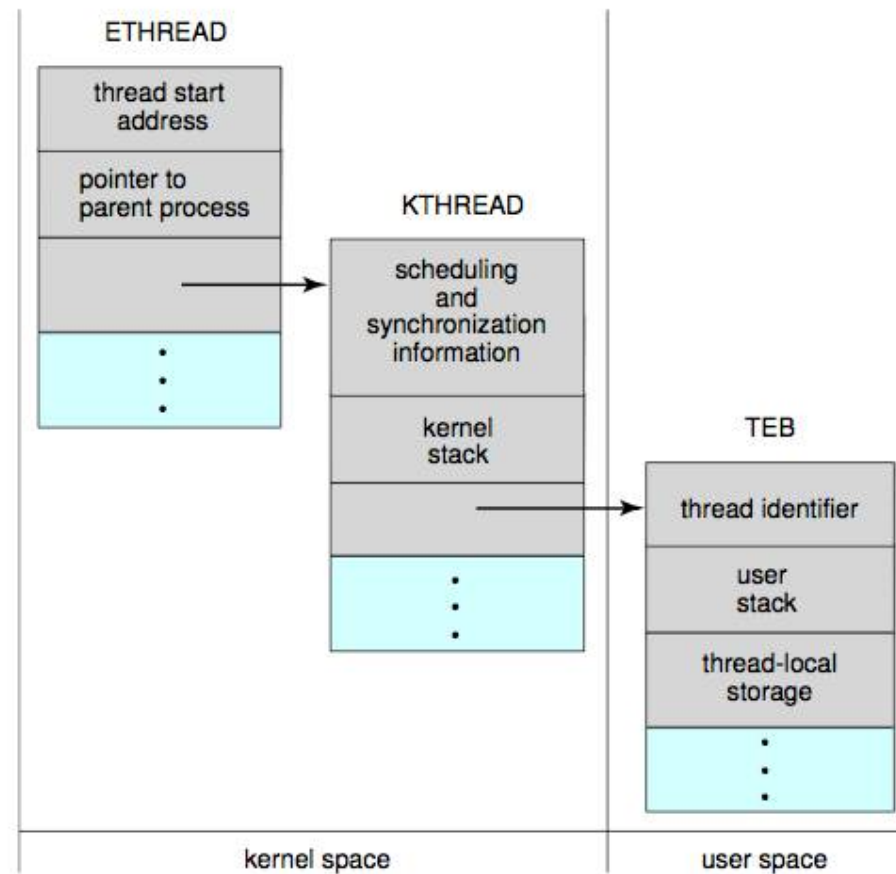


운영체제 사례: Windows XP Threads

- Implements the one-to-one mapping
- Each thread contains
 - A thread id
 - Register set
 - Separate user and kernel stacks
 - Private data storage area
- The register set, stacks, and private storage area are known as the **context** of the threads



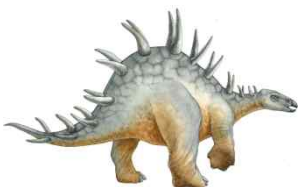
운영체제 사례: Windows XP Threads



운영체제 사례: Linux Threads

- ❑ Linux refers to them as *tasks* rather than *threads*
- ❑ Thread creation is done through **clone()** system call
- ❑ **clone()** allows a child task to share the address space of the parent task (process)

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.



예제 : Thread Echo Server

```

/*****
*** echo-thread.c
***
*** An echo server using threads.
*****/
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <resolv.h>
#include <arpa/inet.h>
#include <pthread.h>

void PANIC(char* msg);
#define PANIC(msg) { perror(msg); exit(-1); }

/*-----*/
/*--- Child - echo servlet ---*/
/*-----*/
void* Child(void* arg)
{
    char line[100];
    int bytes_read;
    int client = *(int *)arg;

    do
    {
        bytes_read = recv(client, line, sizeof(line), 0);
        send(client, line, bytes_read, 0);
    }
    while (strncmp(line, "bye\r", 4) != 0);
    close(client);
    return arg;
}

```

예제 : Thread Echo Server

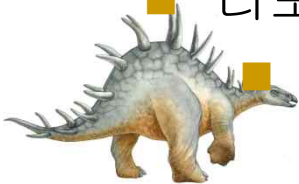
```
/*-----*/
/*--- main - setup server and await connections (no need to clean ---*/
/*--- up after terminated children.          ---*/
/*-----*/
int main(void)
{   int sd;
    struct sockaddr_in addr;

    if ( (sd = socket(PF_INET, SOCK_STREAM, 0)) < 0 )
        PANIC("Socket");
    addr.sin_family = AF_INET;
    addr.sin_port = htons(9999);
    addr.sin_addr.s_addr = INADDR_ANY;
    if ( bind(sd, (struct sockaddr*)&addr, sizeof(addr)) != 0 )
        PANIC("Bind");
    if ( listen(sd, 20) != 0 )
        PANIC("Listen");
    while (1)
    {   int client, addr_size = sizeof(addr);
        pthread_t child;

        client = accept(sd, (struct sockaddr*)&addr, &addr_size);
        printf("Connected: %s:%d\n", inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));
        if ( pthread_create(&child, NULL, Child, &client) != 0 )
            perror("Thread creation");
        else
            pthread_detach(child); /* disassociate from parent */
    }
    return 0;
}
```

Report

- 앞의 Echo Server를 확장하여 계산 서버를 작성하시오
 - 입력 : 사칙연산 숫자1 숫자2
 - 출력 : 위의 입력을 계산한 결과
 - C 또는 자바로 구현
- 기한 : 2010년 10월 29일까지
- 리포트 포맷(hwp) :
 - 1page : 이름, 학번, 리포트 제목
 - 2page : source
 - 3page : 실행 화면
- 리포트 제출 방법 : kwangwoo@gmail.com 에 선착순 도착
 - 이메일 제목 : [운영체제] Calc Server-060303



End of Chapter 4

