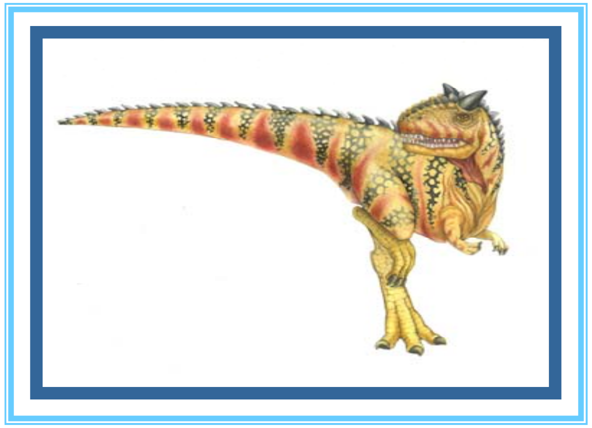
Chapter 4: Threads



Threads 개요

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

- A thread shares with its peer threads its(공유)
  - code section
  - data section
  - operating-system resources(files …)

  collectively known as a *task*.

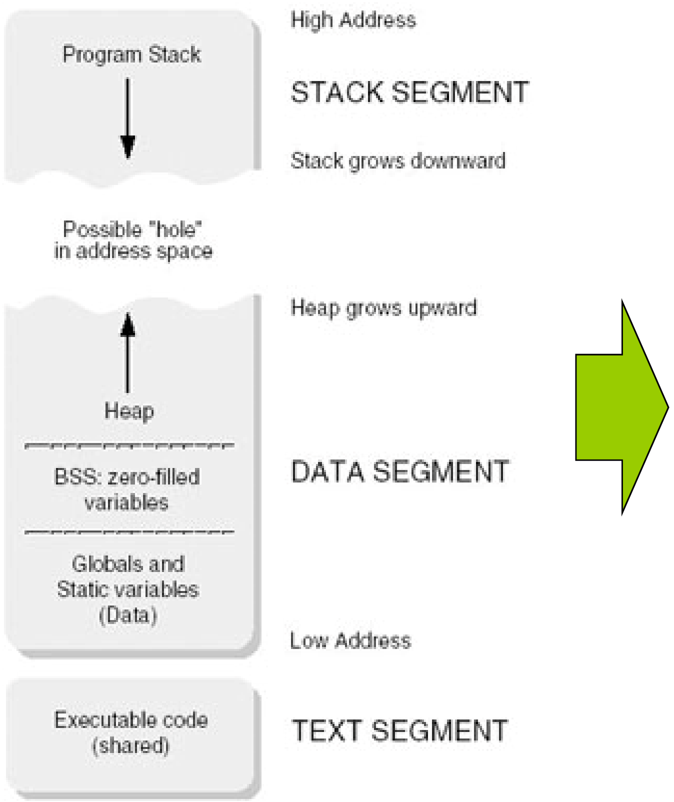
- 프로세스 : 중량 프로세스(HWP;Heavy Weight Process)
  \- 하나의 스레드를 가진 작업(task)

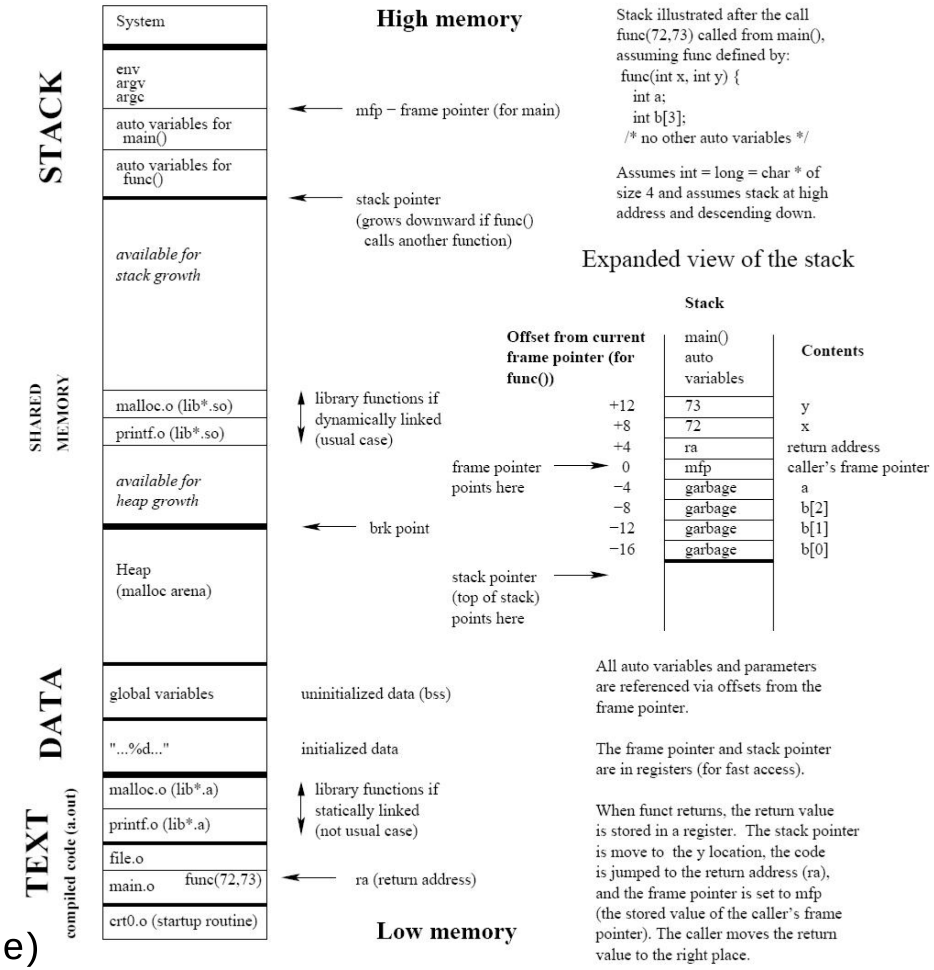
Motivation

- **Thread**들은 응용내에서 수행
- 응용내에서 분리된 **thread**들로 구현되는 작업들의 예
  - **Update display**
  - **Fetch data**
  - **Spell checking**
  - **Answer a network request**
- **Kernels are generally multithreaded**

Threads 개요

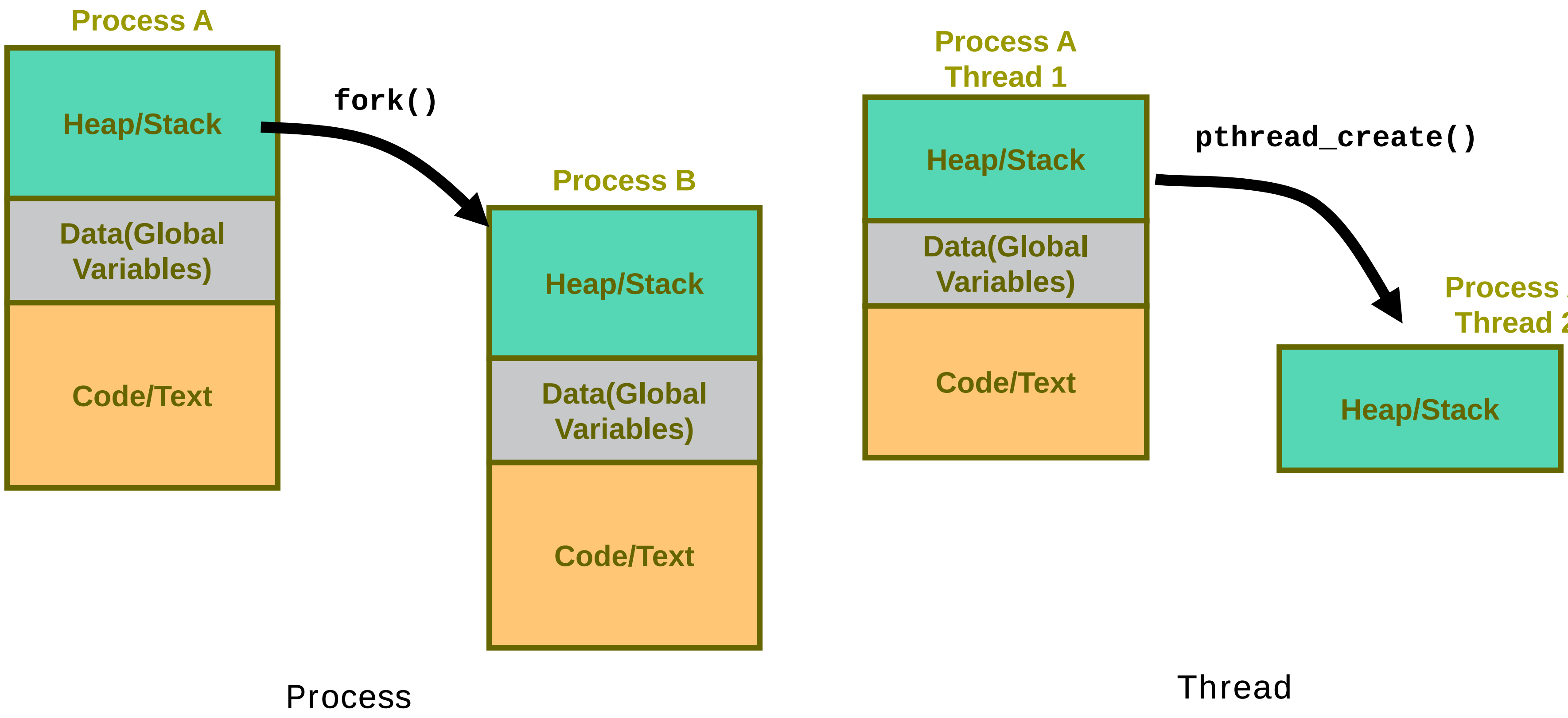
- **Process**의 메모리 구조(상세)



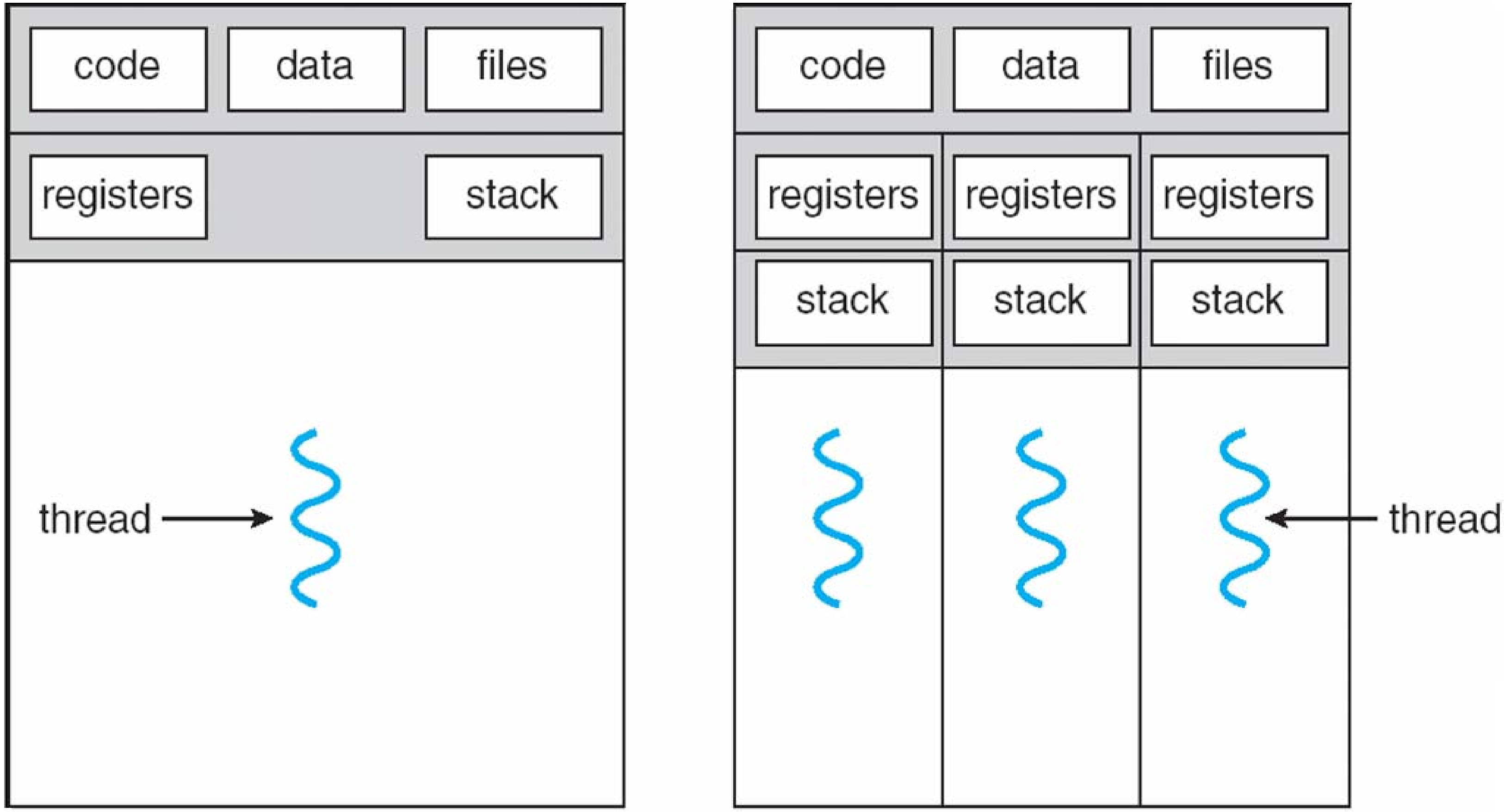


Threads 개요

- **Process**와 **Thread**의 차이



Single and Multithreaded Processes



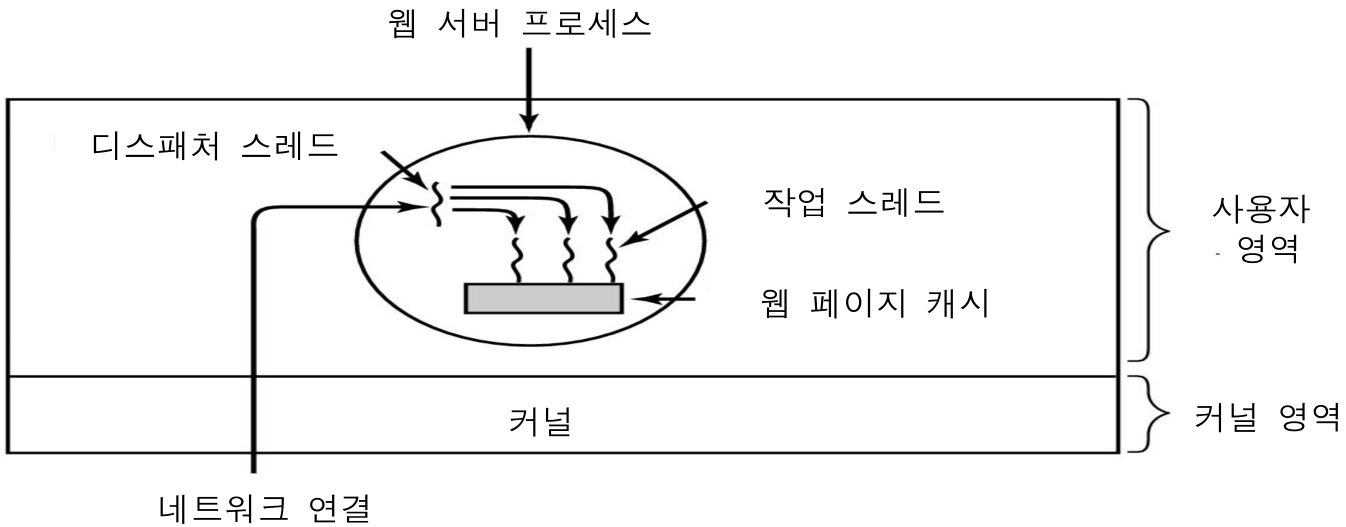
single-threaded process

multithreaded process

Thread의 장점

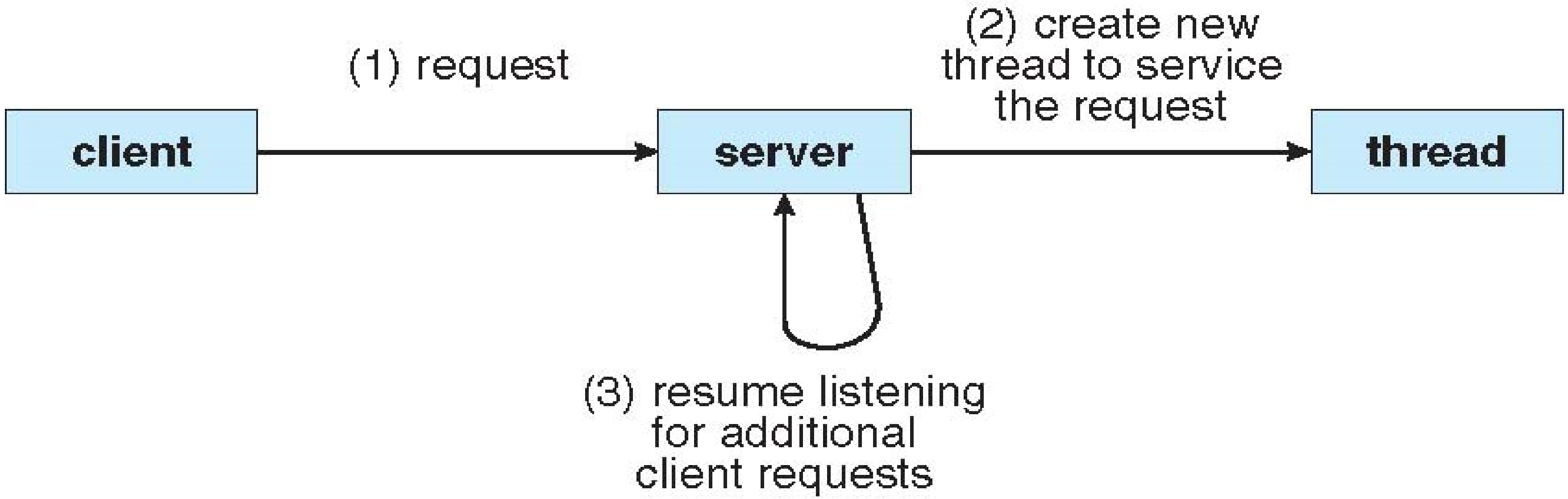
- **Responsiveness**
  - **eg) multi-threaded Web - if one thread is blocked (eg network) another thread continues (*eg display)***
- **Resource Sharing**
  - **n threads can share binary code, data, resource of the process (files, crt, …)**
- **Economy**
  - ***creating* and *context switching* *thread* (rather than a *process)***
  - **Solaris: 30배 5배**
- **Utilization of MP Architectures**
  - ***each thread* may be running in *parallel* on a *different processor***

쓰레드의 이용 예 : 웹 서버

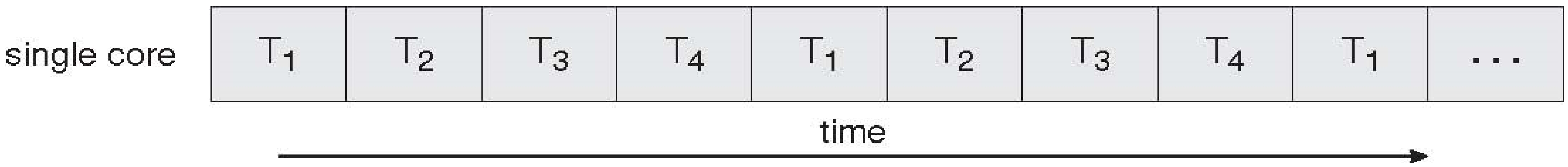


출처: 그림으로 보는 운영체제

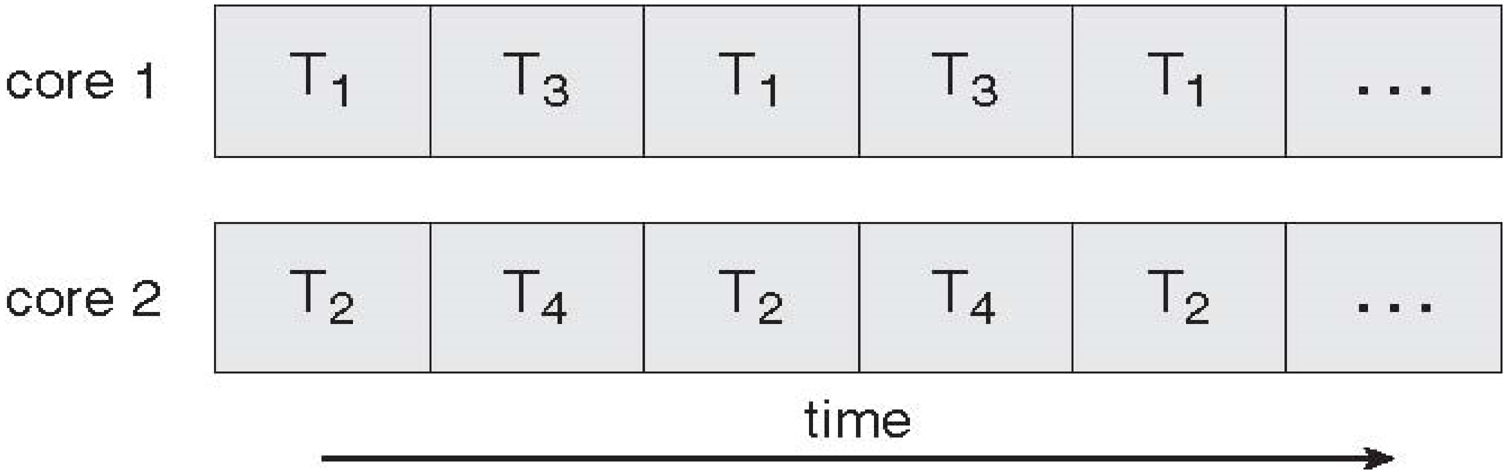
Multithreaded Server Architecture



Concurrent Execution on a Single-core System



Parallel Execution on a Multicore System



User Threads

- Thread management done by user-level threads library
  - 라이브러리는 커널의 지원없이 쓰레드의 생성과 스케쥴링, 관리를 지원
  - 커널을 통하지 않으므로, 생성과 관리가 빠르나 봉쇄형 시스템 콜을 수행하는 사용자 수준의 쓰레드는 다른 쓰레드와 함께 스케쥴링 되지 않음
- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads

Kernel Threads

- **Supported by the Kernel**
- 커널 수준에서 관리되어 생성과 관리가 느리나 다른 쓰레드와 함께 스케쥴링 될 수 있음
- **Examples**
  - **Windows XP/2000**
  - **Solaris**
  - **Linux**
  - **Tru64 UNIX**
  - **Mac OS X**

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**

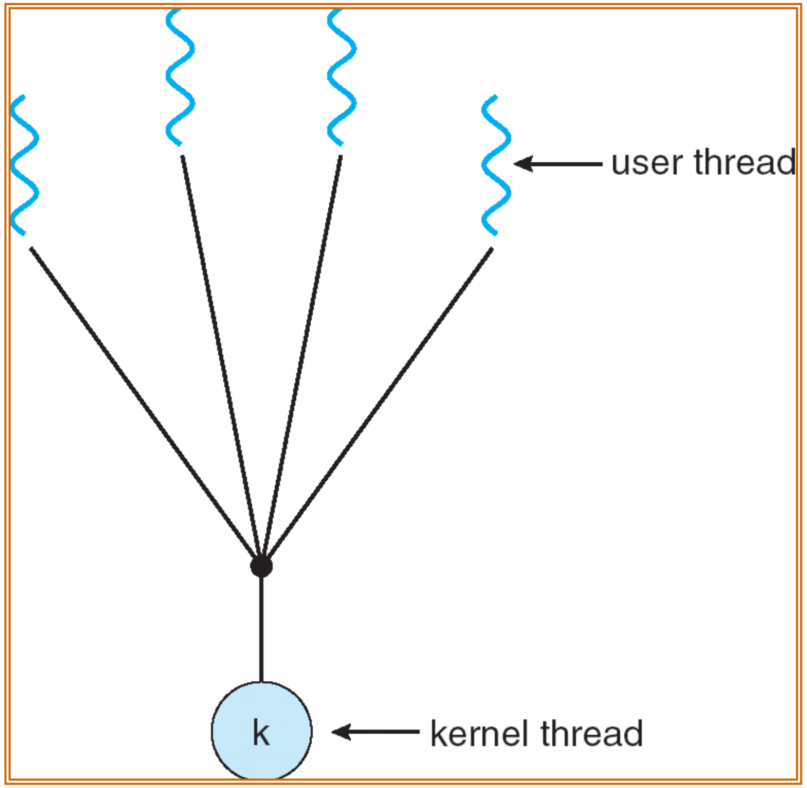
Multithreading Models

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

Many-to-One

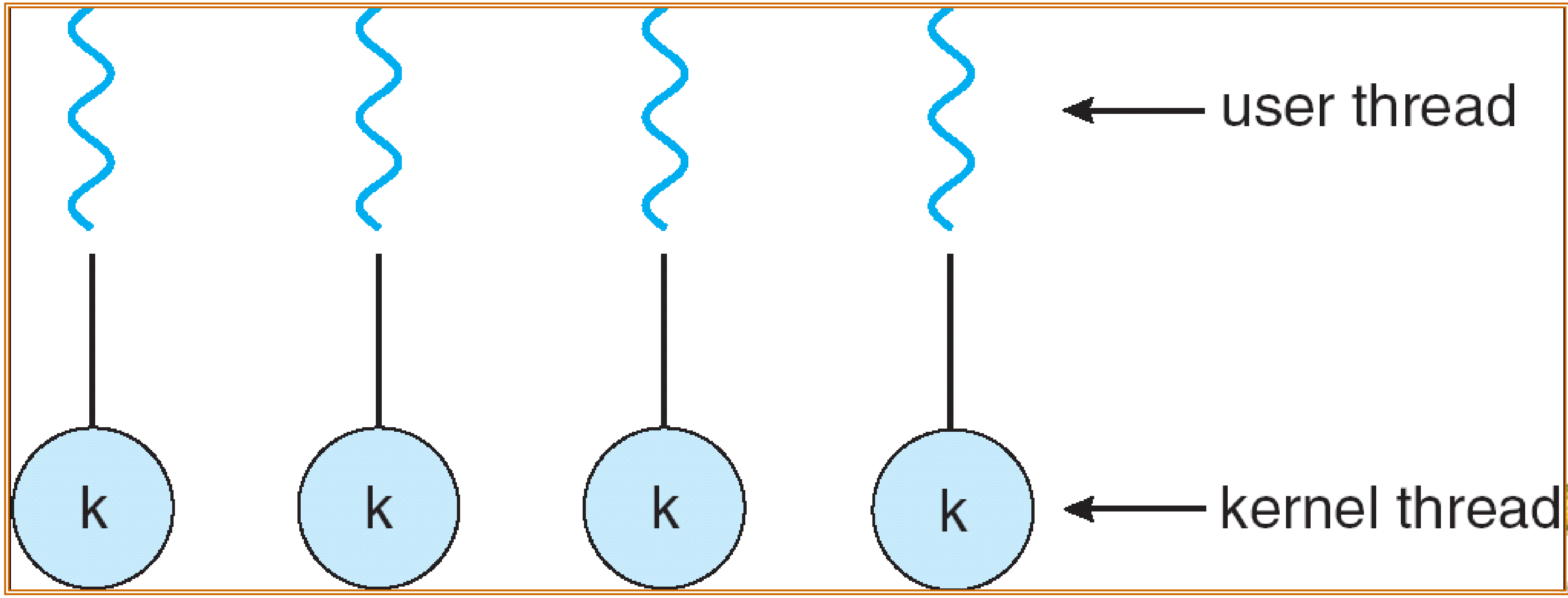
- Many user-level threads mapped to single kernel thread

- Examples:
  - Solaris Green Threads
  - GNU Portable Threads



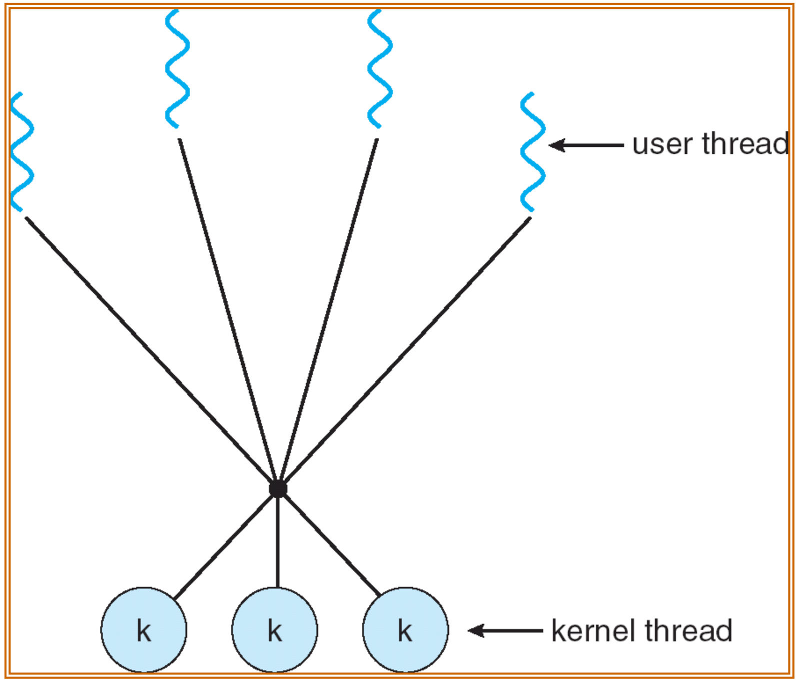
One-to-One

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



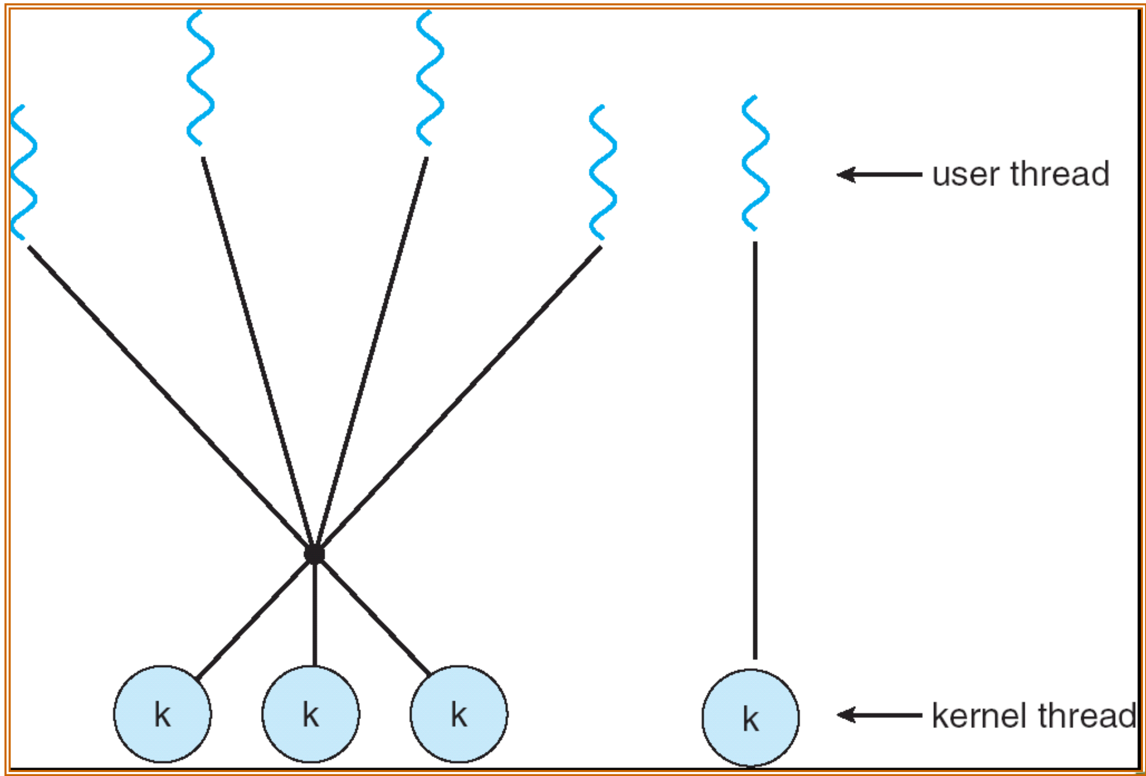
Many-to-Many Model

- 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



Thread Libraries

- Thread library provides programmer with API for creating and managing threads

- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

- POSIX Pthread
- Wind32 Thread API
- Java thread API

- Linux

Threading Issues

- Semantics of fork() and exec() system calls
- Thread cancellation of target thread
  - Asynchronous or deferred
- Signal handling
  - Synchronous and asynchronous

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 쓰레드들이 데이터베이스를 병렬로 검색하다가 그 중 한 쓰레드가 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**

Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred.
- A signal handler is used to process signals
  1. Signal is generated by particular event
  2. Signal is delivered to a process
  3. Signal is handled
- Options:
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the process

Signal Handling

- **Signal**
  - **Unix**에서 특정 **Event**가 일어났음을 알리기 위해 사용되는 단위**(**예**: Windows Message)**
- **signal handler**의 처리 순서
  1. **Signal**이 특정 **event**에 의해 생성됨
  2. **Signal**이 특정 프로세스에 전달됨
  3. **Signal**이 처리됨

Signal의 예
  - **Synchronous**
    - **Devide-by-zero,**
    - **illegal-memory-access**

- **Process**에서의 **Signal** 처리 선택사항
  - **Signal**이 적용될 특정 **Thread**에 전송
  - **Process**안에 있는 모든 **Thread**에 전송됨
  - **Process**안의 다수 **Thread**에게 전송됨
  - 그 **Process**에 전달되는 모든 **Signal**을 처리할 특정 **Thread**를 지정

Thread Pools

- 작업을 대기하는 다수의 **Thread**를 미리 생성해 놓는 **Pool**
- **Advantages:**
  - 속도 **:** 보통 새로운 **Thread**를 생성하는 것보다 존재하는 **Thread**를 사용하므로 다소 빠름
  - 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)`

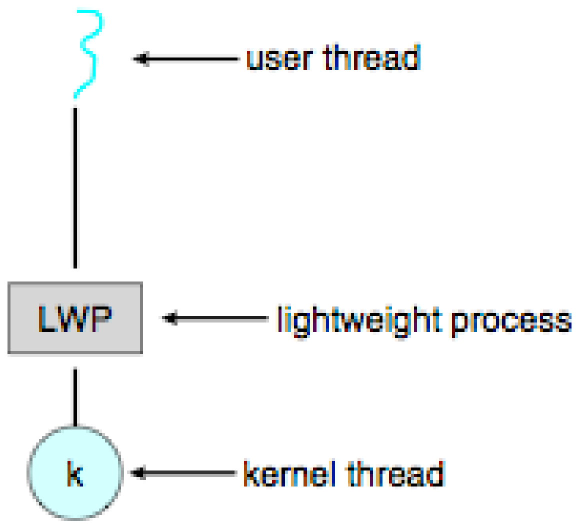
  3. Cached thread pool - pool of unbounded size

  - `static ExecutorService newCachedThreadPool()`

Scheduler Activations

- Scheduler Activation
  - Thread library와 Kernel Thread의 통신방법
  - This communication allows an application to maintain the correct number kernel threads

- LWP 자료구조
  - M:M and Two-level model들은 다수의 Kernel



운영체제 사례

- **Solaris**에서 **Thread**와 **Process**의 관계

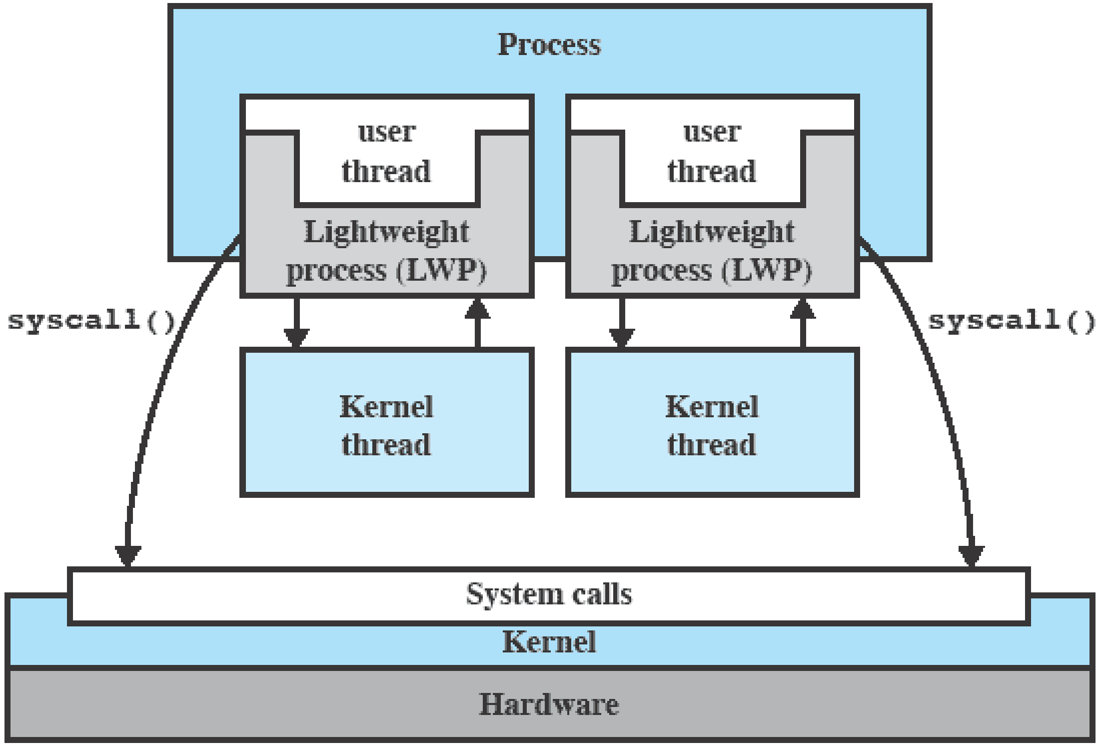


Figure 4.15 Processes and Threads in Solaris [MCDO07]

운영체제 사례

- **Unix**와 **Solaris**의 **Thread** 지원 **Process**의 비교

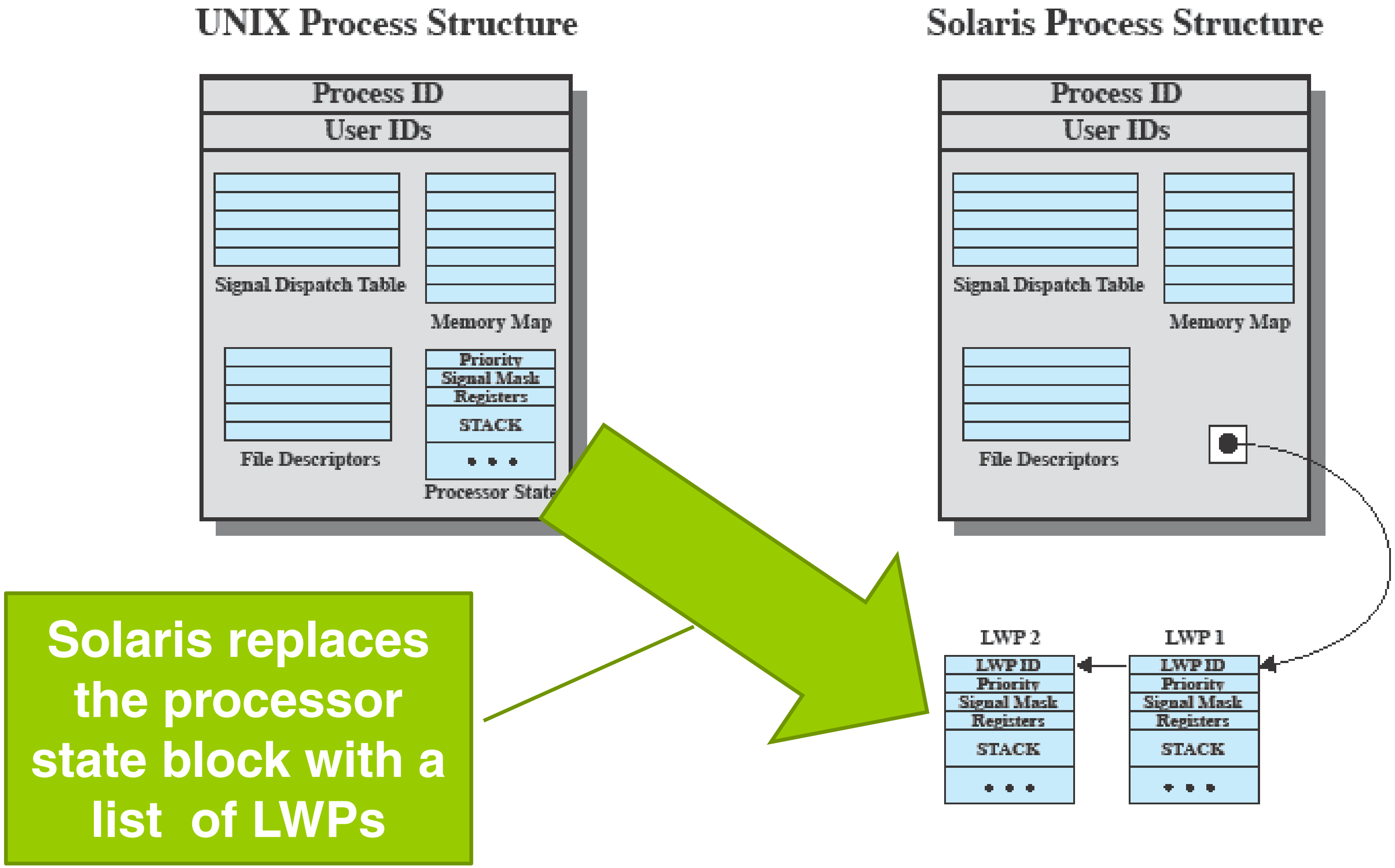


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]

운영체제 사례

- **Solaris**에서의 **Thread** 모델

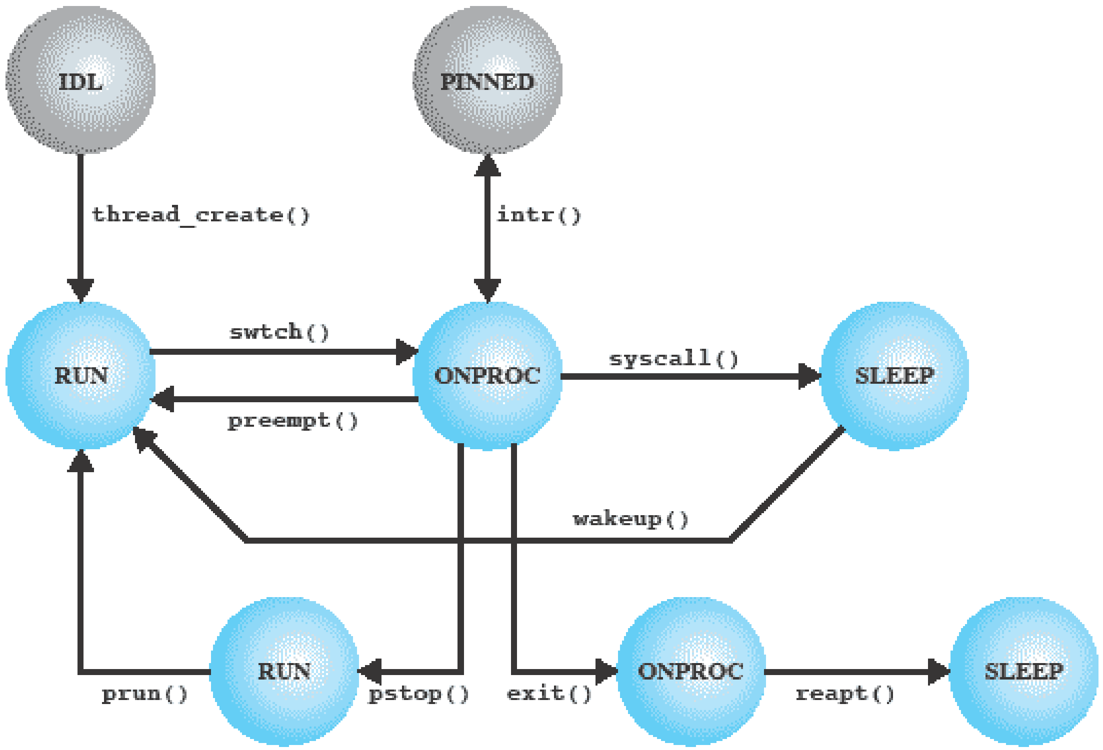


Figure 4.17 Solaris Thread States [MCDO07]

운영체제 사례

- **Linux**에서의 **Process/Thread** 모델

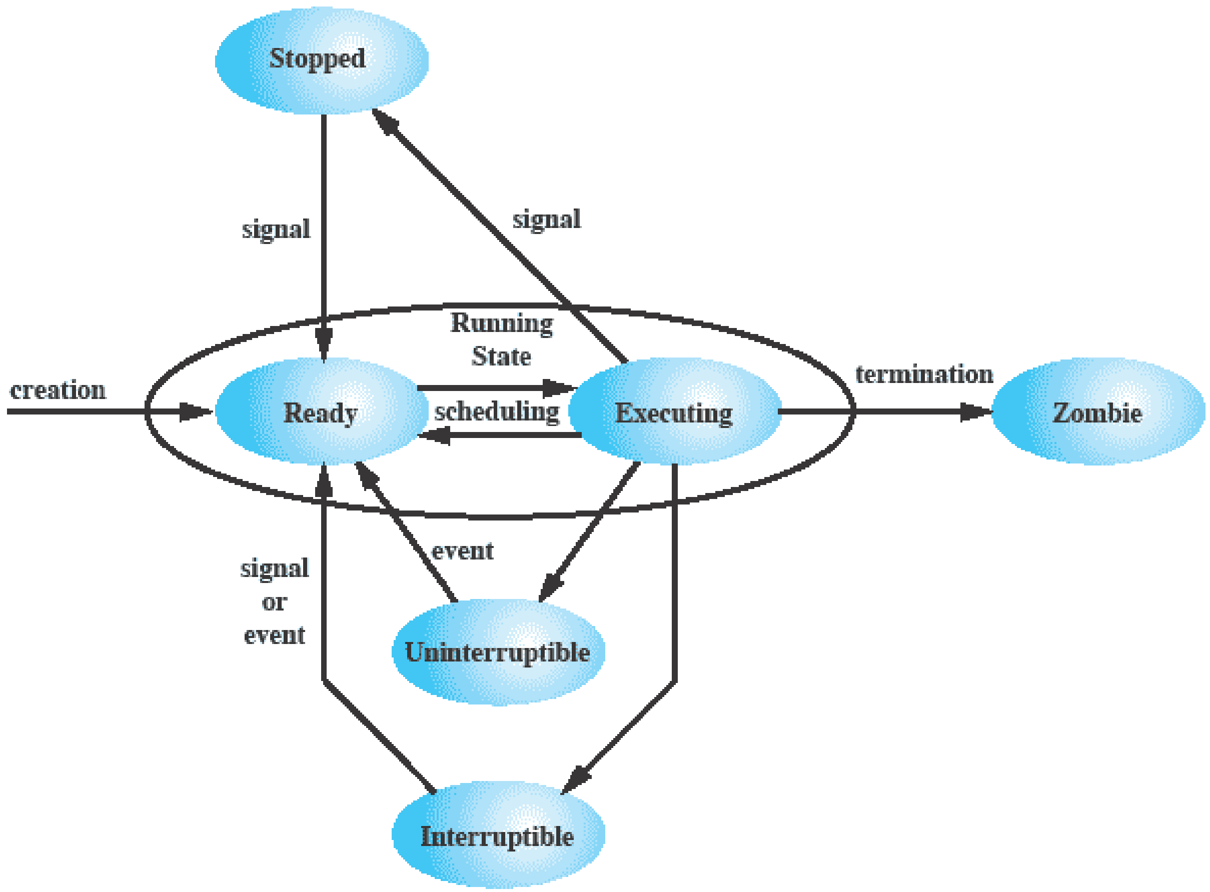
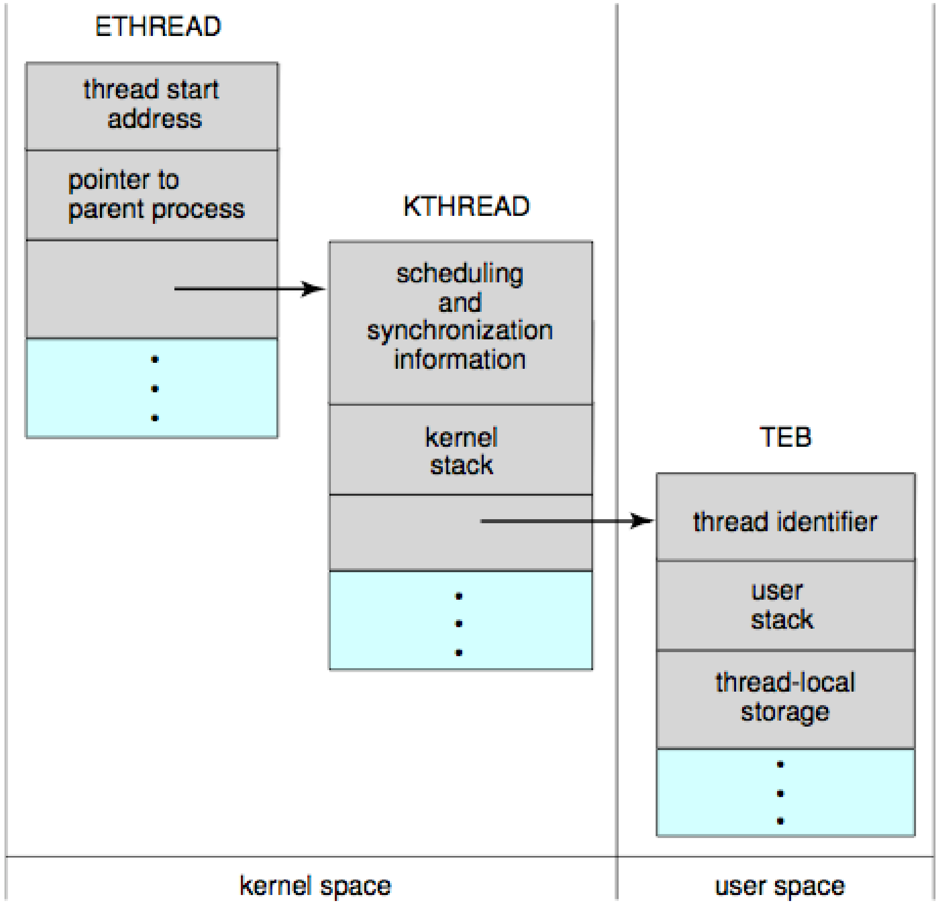


Figure 4.18 Linux Process/Thread Model

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.

Thread Programming : Windows(1)

```
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <process.h>

#define rowA 3
#define colA 4
#define rowB 4
#define colB 5

typedef struct Matrix
{
 int matrixA[rowA][colA];
 int matrixB[rowB][colB];
 int matrixC[rowA][colB];
}Matrix;
unsigned long  thread0, thread1, thread2;
unsigned __stdcall Thread0(void *pParam)//스레드 함수
{
 int nTemp=0, j, k;
 Matrix *mx = (Matrix*)pParam;

 for ( j = 0; j < colB; j++ )
  {
   for ( k = 0; k < colA; k++ )
    {
     nTemp += (mx->matrixA[0][k] * mx->matrixB[k][j]);
    }
   mx->matrixC[0][j] = nTemp;
   nTemp = 0;
  }
 thread0=1;
 return 0;
}
```

Thread Programming : Windows(2)

```
unsigned __stdcall Thread1(void *pParam)//스레드 함수
{
 int nTemp=0, j, k;
 Matrix *mx = (Matrix*)pParam;

 for ( j = 0; j < colB; j++ )
 {
  for ( k = 0; k < colA; k++ )
  {
   nTemp += (mx->matrixA[1][k] * mx->matrixB[k][j]);
  }
  mx->matrixC[1][j] = nTemp;
  nTemp = 0;
 }
 thread1=1;
 return 0;
}

unsigned __stdcall Thread2(void *pParam)//스레드 함수
{
 int nTemp=0, j, k;
 Matrix *mx = (Matrix*)pParam;

 for ( j = 0; j < colB; j++ )
 {
  for ( k = 0; k < colA; k++ )
  {
   nTemp += (mx->matrixA[2][k] * mx->matrixB[k][j]);
  }
  mx->matrixC[2][j] = nTemp;
  nTemp = 0;
 }
 thread2=1;
 return 0;
}
```

행렬곱셈
[3 * 4] * [4 * 5] -> [3*5]에서
[1*5] [1 * 5] [1 * 5]쓰레드를 통해
[3*5] 행렬 계산

Thread Programming : Windows(3)

```
void main()
{
   Matrix mx;

 int i, j;
 for(i = 0; i < rowA; i++)
 {
  for(j = 0; j < colA; j++)
   mx.matrixA[i][j] = 1;
 }

 for(i = 0; i < rowB; i++)
 {
  for(j = 0; j < colB; j++)
   mx.matrixB[i][j] = 2;
 }
 _beginthreadex(NULL, 0, Thread0, &mx, 0, NULL);//스레드 시작
 _beginthreadex(NULL, 0, Thread1, &mx, 0, NULL);//스레드 시작
 _beginthreadex(NULL, 0, Thread2, &mx, 0, NULL);//스레드 시작
 while(1)
 {
  if(thread0 && thread1 && thread2)
  {
   for(i = 0; i < rowA; i++)
   {
    for(j = 0; j < colB; j++)
     printf("%d ", mx.matrixC[i][j]);
    printf("\n");
   }
   break;
  }
 }
}
```

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;
}
```

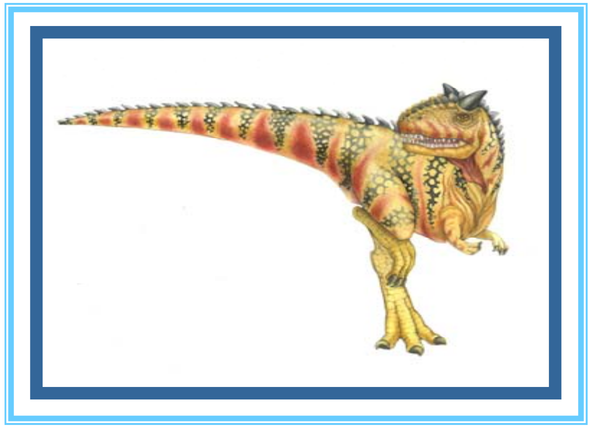
예제 : Thread Echo Server

```
/*------------------------------------------------------------------*/
/*--- 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;
}
```

End of Chapter 4



Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Pthreads Example

```
#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;
   }
```

Pthreads Example (Cont.)

```
    /* 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);
}
```

Figure 4.9 Multithreaded C program using the Pthreads API.

Win32 API Multithreaded C Program

```
#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;
   }
```

Win32 API Multithreaded C Program (Cont.)

```
    // create the thread
    ThreadHandle = CreateThread(
      NULL, // default security attributes
      0, // default stack size
      Summation, // thread function
      &Param, // parameter to thread function
      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);
    }
}
```

Figure 4.10 Multithreaded C program using the Win32 API.

Java Threads

- Java threads are managed by the JVM
- Typically implemented using the threads model provided by underlying OS
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface

Java Multithreaded Program

```
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);
   }
}
```

Java Multithreaded Program (Cont.)

```java
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>"); }
}
```

Figure 4.11 Java program for the summation of a non-negative integer.