Chapter 4. Threads(쓰레드)

4장의 학습 목표

- 프로세스와 쓰레드의 차이를 이해할 수 있다.
- 쓰레드와 관련된 기본 설계 이슈를 설명할 수 있다.
- 사용자 수준의 쓰레드와 커널 수준의 쓰레드의 차이를 설명할 수 있다.
- Windows 8의 쓰레드 관리 기능을 설명할 수 있다.
- Solaris의 쓰레드 관리 기능을 설명할 수 있다.
- Linux의 쓰레드 관리 기능을 설명할 수 있다.
- Android의 쓰레드 관리 기능을 설명할 수 있다.

목 차

4.1 프로세스와 쓰레드(thread)

- 프로세스
  - ✓ 실행 중인 프로그램(A program in execution)
  - – **메모리 (코드, 데이터, 스택), 파일, signal, IPC, accounting, …**
  - – **CPU 레지스터 정보, 스케줄링 정보**

    ➔ **자원 소유권 (resource ownership)**

    ➔ **수행/스케줄링(execution/scheduling) 개체**

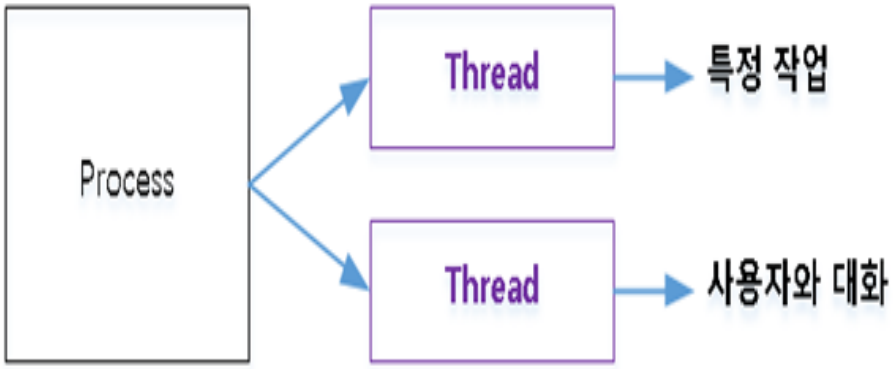
  - **현대 OS에서 프로세스는 태스크(task) 및 쓰레드(thread)라는 두 객체(특성)로 분리**
  - – **태스크: Resource Container (사용자 문맥, 시스템 문맥)**
  - – **쓰레드: 제어 흐름 (실행 정보, 레지스터 문맥)**

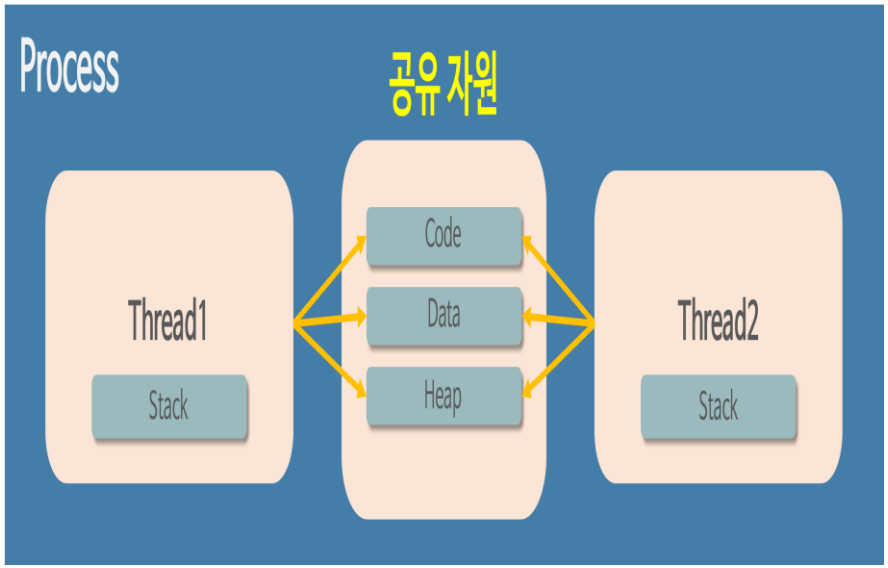
☞ **쓰레드를 경량 프로세스(lightweight process)라고 부르기도 함**

☞**쓰레드는 프로세스내에서 실행되는 흐름의 단위**

☞ **프로세스 및 태스크를 서로 혼용하여 사용하기도 함**

4.1 프로세스와 쓰레드(thread)





구분	Thread	Process
상호통신	• Library Call • 요청한 Thread만 Blocking	• System Call • Call 종료까지 전체 Blocking
구분처리	• CPU를 사용할 기본단위	• 자원을 할당할 기본단위
실행방식	• 다중 처리	• 한 개의 Process

Processes and Threads

- Examples

```
#include   <stdio.h>
#include   <sys/types.h>
#include   <stdlib.h>
#include   <unistd.h>

int a[4] = {1,2,3,4}; int b[4];

void *func1() {
   b[0] = a[0] + 1; b[1] = a[1] + 1;
   printf("In func1: %d□n", b[0]);
}
void *func2() {
   b[2] = a[2] + 1; b[3] = a[3] + 1;
   printf("In func2: %d□n", b[2]);
}

int main(void) {
   int pid;
   // child 프로세스를 생성하여 func1()을 수행
   if ((pid = fork()) < 0)
      exit(1);
   else if (pid == 0) {
      func1();
      exit(0);
   }
   wait();
   func2();
   printf("sum=%d□n", b[0]+b[1]+b[2]+b[3]);
   exit(0);
}
```

- **Results**

In func 1: 2
In func 2: 4
Sum=9 (0+0+4+5)

- **Results**

– exit(0)삭제시

In func 1: 2
In func 2: 4
Sum=14
In func 2: 4
Sum=9

Processes and Threads - Pthread

POSIX thread

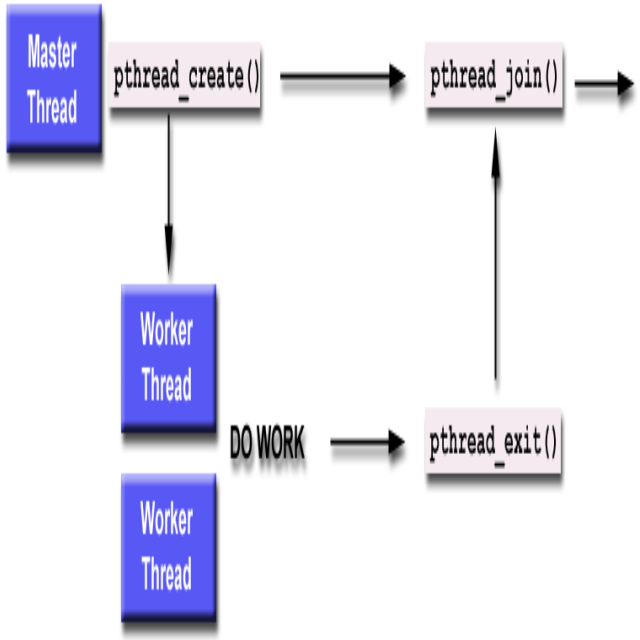
흔히 Pthread 라고 불리우며, POSIX 에서 표준으로 제안한 thread 함수모음으로 thread 를 지원하기위한 C 표준 라이브러리 셋을 제공한다. 이후 모든 예제는 Pthread 를 통해서 구현하고 설명하게 될것이다.

쓰레드의 생성과 종료

멀티 쓰레드 프로그램이 처음 시작되었을때 그것은 main()함수를 실행하는 단일 프로세스 상태로 작동하게 될것이다. 이것은 그 자체로 하나의 완전한 쓰레드이다. 이 상태에서 우리는 pthread_create(3) 함수를 부름으로써 새로운 쓰레드를 생성할수 있다.

쓰레드를 이용한 프로그램은 기본적으로 아래와 같은 순서로 작동하게 된다.

```
Master Thread
      |
      |            pthread_create() 에 의해서 worker 생성
      |
+---+---+---+      worker 시작
|   |   |   |
|   |   |   |      각각의 worker는 그들의 작업을 수행한다.
|   |   |   |
+---+---+---+      worker 를 종료한다.
      |
      |            pthread_join()에 의해서 worker 를 join 한다.
      |
Master Thread
```



worker 은 쓰레드로 바꾸어 생각할수도 있다.

Processes and Threads - Pthread

예제 thread.c

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

void* do_loop(void *data)
{
    int i;

    int me = *((int *)data);
    for (i = 0; i < 10; i++)
    {
        printf("%d - Got %d₩n", me, i);
        sleep(1);
    }
}

int main()
{
    int        thr_id;
    pthread_t p_thread[3];
    int status;
    int a = 1;
    int b = 2;
    int c = 3;

    thr_id = pthread_create(&p_thread[0], NULL, do_loop, (void *)&a);
    thr_id = pthread_create(&p_thread[1], NULL, do_loop, (void *)&b);
    thr_id = pthread_create(&p_thread[2], NULL, do_loop, (void *)&c);

    pthread_join(p_thread[0], (void **) &status);
    pthread_join(p_thread[1], (void **) &status);
    pthread_join(p_thread[2], (void **) &status);

    printf("programing is end₩n");
    return 0;
}
```

위의 프로그램을 컴파일 시키기 위해선 pthread 라이브러리를 링크시켜줘야 한다.

```
[yundream@localhost test]# gcc -o thread thread.c -lpthread
```

Processes and Threads

- pthread_create
  - ✓ 새로운 쓰레드 생성 => fork()
  - ✓ 1st arg => pthread 에 대한 포인터로서, 쓰레드가 생성되면 이 포인터가 가리키는 변수에 id 를 저장
- pthread_join
  - ✓ wait()
  - ✓ 1st arg -> 기다릴 쓰레드로서, pthread_create가 return 한 id
  - ✓ 2nd arg -> 쓰레드가 return한 값을 가리키는 ptr 에 대한 ptr
- pthread_exit
  - ✓ 쓰레드 종료 => exit()
- Compile option
  - ✓ $ gcc –o test test1.c -lpthread

Processes and Threads

- Examples

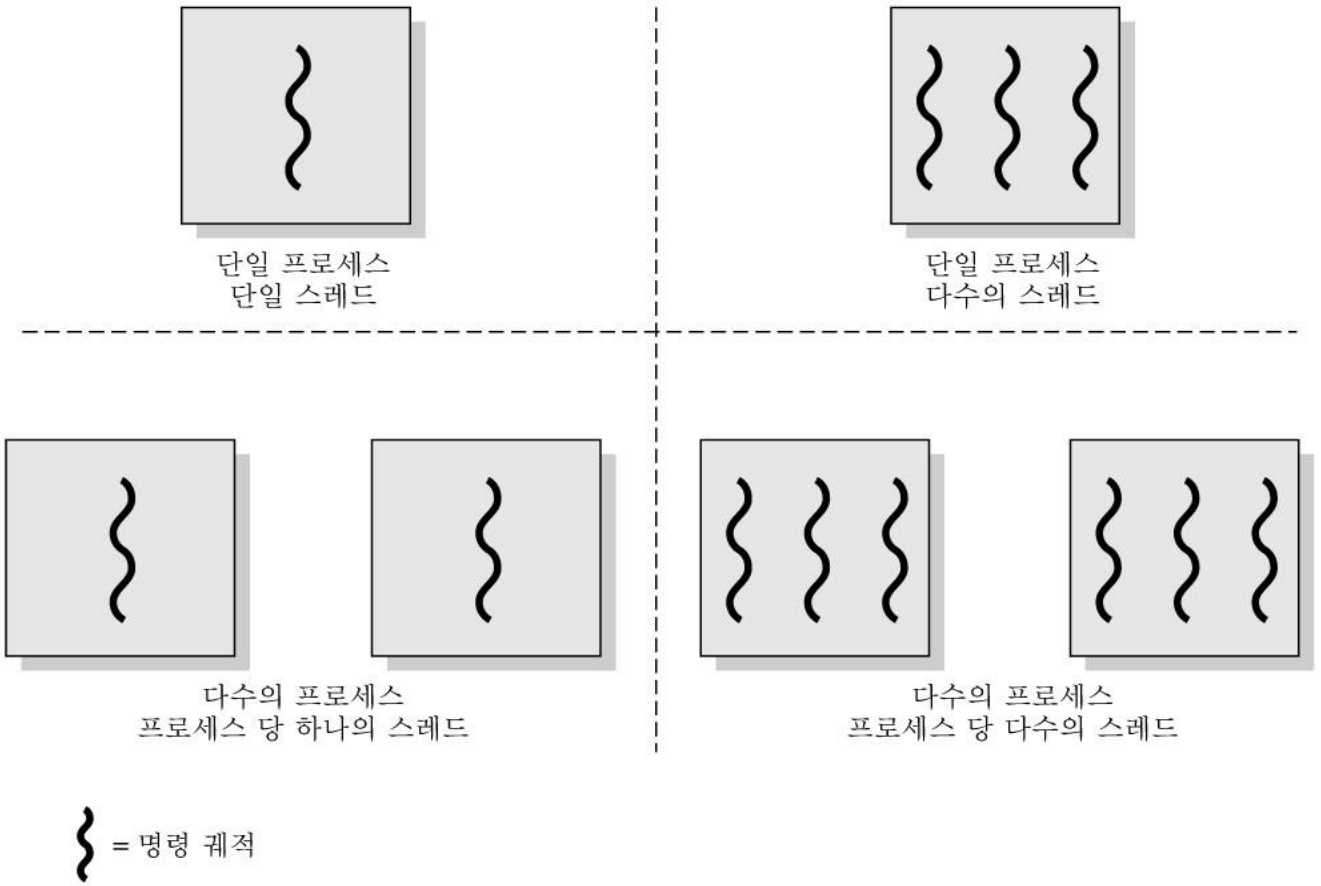
```
#include <stdlib.h>

int a[4] = {1,2,3,4}; int b[4];

void *func1() {
   b[0] = a[0] + 1; b[1] = a[1] + 1;
   printf("In func1: %d□n", b[0]);

}
void *func2() {
   b[2] = a[2] + 1; b[3] = a[3] + 1;
   printf("In func2: %d□n", b[2]);
}

int main() {
   pthread_t p_thread;

   if ((pthread_create(&p_thread, NULL, func1, (void*)NULL))
< 0)
  {
      exit(1);
   }
   pthread_join(p_thread, (void **)NULL);
   func2();
   printf("sum=%d□n", b[0]+b[1]+b[2]+b[3]);
}
```

- **Results**

In func 1: 2
In func 2: 4
Sum=14

프로세스와 쓰레드

- 단일 쓰레딩(threading) 대 다중쓰레딩(Multithreading)
  - ✓ 단일 프로세스 내에 다중 쓰레드 실행을 지원 가능



프로세스와 쓰레드

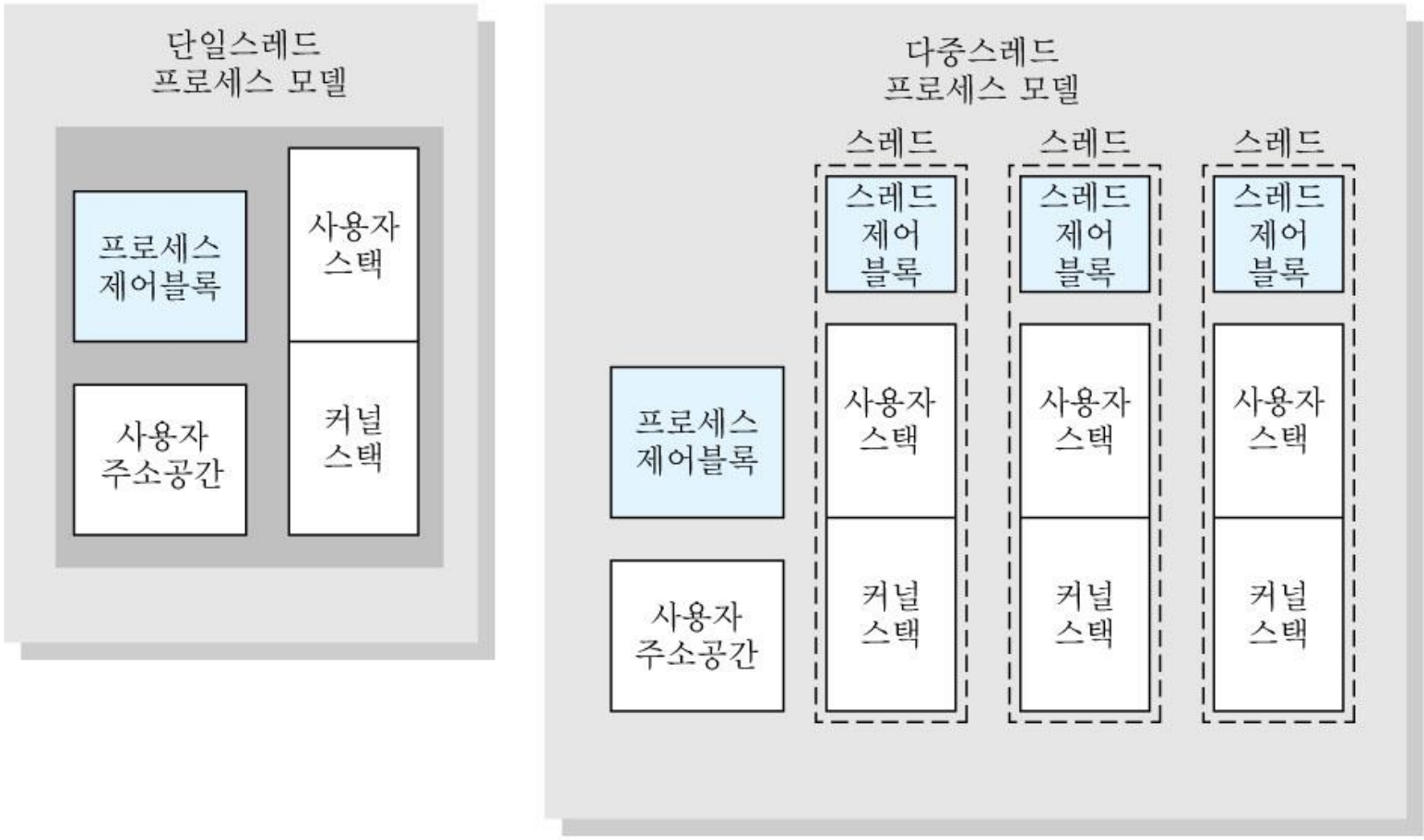
- 사례
  - ✓ MS-DOS는 단일 사용자 프로세스와 단일 쓰레드를 지원한다.
  - ✓ UNIX 계열의 여러 운영체제는 다중 사용자 프로세스를 지원하지만, 프로세스 당 하나의 쓰레드를 지원한다.
  - ✓ Java 수행시간환경(run-time environment)은 하나의 프로세스가 다중 쓰레드를 지원한다.
  - ✓ 최신 버전의 UNIX, Windows, Solaris는 다중 쓰레드를 지원하는 다중 프로세스를 사용한다.

프로세스와 쓰레드

- 다중쓰레딩 환경
  - ✓ 태스크 (또는 프로세스) 관련 사항
    - 프로세스 이미지를 유지하는 가상 주소 공간
    - 처리기, (IPC를 위한) 다른 프로세스, 파일, I/O 자원들에 대한 접근 제어
  - ✓ 쓰레드 관련 사항
    - 실행 상태 (수행, 준비, 블록, ...)
    - 수행 중이 아닐 때 저장되는 쓰레드 문맥
    - 실행 스택
    - 지역 변수 저장을 위해 각 쓰레드가 사용하는 어떤 정적 저장소 (storage)
    - 자신 프로세스의 메모리 및 자원들에 대한 접근 공유
  - ✓ 한 프로세스 내의 모든 쓰레드들은 그 프로세스의 자원들을 공유

프로세스와 쓰레드

- 쓰레드 모델 (그림 4.2)



프로세스와 쓰레드

- 쓰레드의 장점(benefits)

1. 프로세스에 비해 새로운 쓰레드 생성 시간/비용이 절약
   - Mach OS에서 쓰레드 생성시간이 UNIX 보다 10배 빠름
2. 프로세스 종료 시간보다 쓰레드 종료 시간이 짧다.
3. 한 프로세스 내의 두 쓰레드들 사이의 교환/교체 시간이 짧다.
4. 동일 프로세스 내의 쓰레드들은 메모리 및 파일을 공유하기 때문에, 이들 쓰레드들은 커널의 개입 없이 서로 통신 가능

=➔ 멀티코어 환경에서 CPU 활용을 극대화하여 병렬 실행 가능

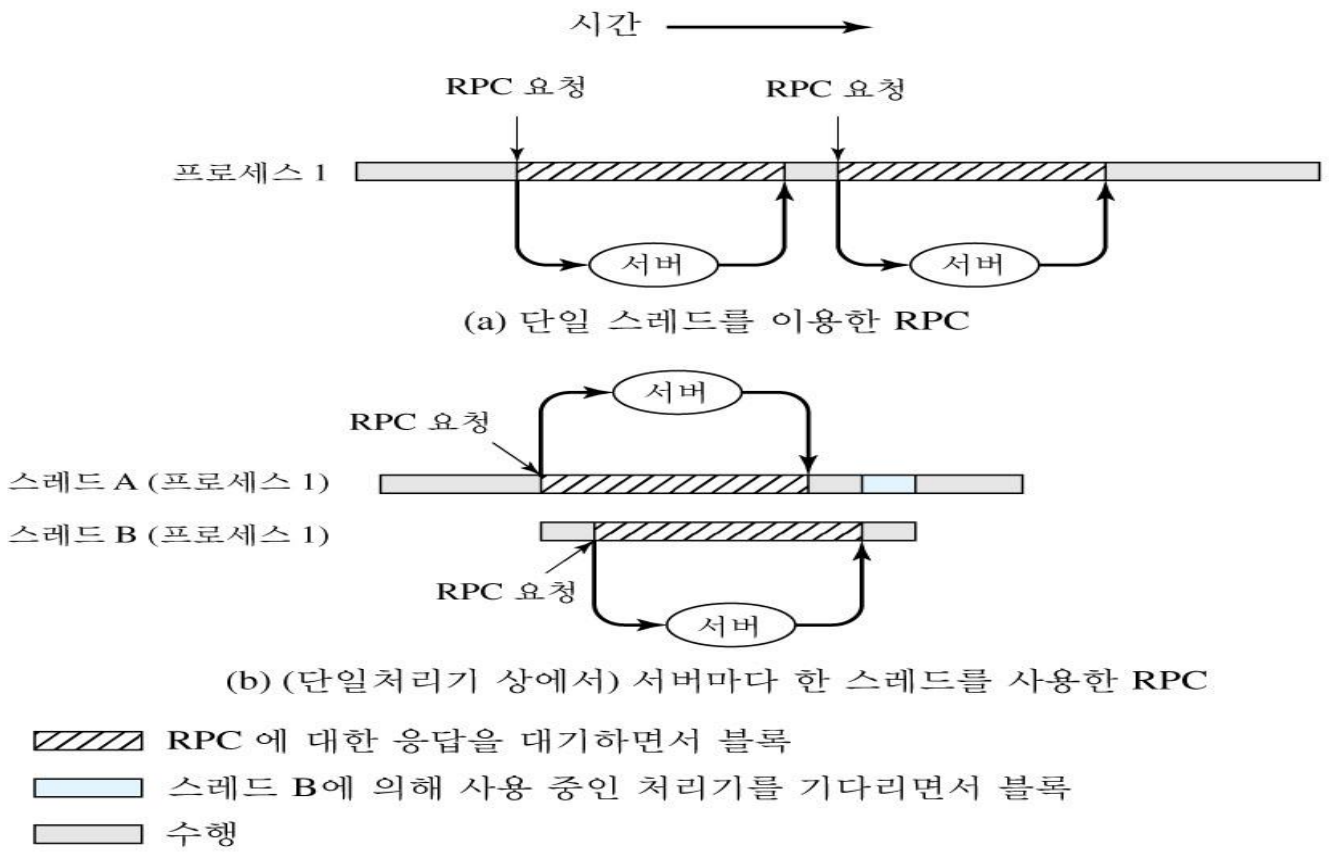
프로세스와 쓰레드

- 단일사용자 멀티프로세싱 시스템에서 쓰레드 사용 예

  - 전면(foreground)과 후면(background) 작업: 응용의 속도 향상
    예를 들어 스프레드시트 프로그램에서 하나의 쓰레드가 메뉴를 나타내고 사용자 입력을 읽는 중에, 다른 쓰레드는 사용자 명령을 수행하고 스프레드시트를 갱신할 수 있다.
  - 비동기(asynchronous) 처리:
    프로그램의 비동기적 요소들을 쓰레드를 통해 구현할 수 있다. 예를 들어 정전으로부터 보호하기 위해 1분마다 메모리(RAM) 버퍼의 내용을 디스크로 기록하는 워드프로세서를 설계할 수 있다. 이를 위한 쓰레드가 생성될 수 있는데, 유일한 업무는 주기적인 백업이며 운영체제를 통해 직접 자신을 스케줄한다. 이때 시간을 검사하거나 또는 입력 및 출력을 조정하기 위해 주 프로그램 내에 복잡한 코드를 작성할 필요는 없다.
  - 빠른 수행:
    멀티쓰레드 프로세스는 어떤 데이터 묶음(batch)을 계산하면서 동시에 어떤 장치로부터 다음 데이터 묶음을 읽어 들일 수 있다. 멀티프로세서 시스템에서 한 프로세스내의 여러 쓰레드들은 실제적으로 동시에 수행될 수 있다. 따라서 한 쓰레드가 특정 데이터 묶음을 읽기 위해 입출력 작업 완료를 기다리면서 블록(block)될지라도, 또 다른 쓰레드가 수행될 수 있다.
  - 모듈 프로그램 구조:
    다양한 활동 혹은 입출력 연산에 대한 다양한 출발・목적지를 포함하고 있는 프로그램의 경우, 쓰레드들을 사용하여 설계하고 구현하는 것이 편리하다.

프로세스와 쓰레드

- 쓰레드의 장점: 예 (그림 4.3: RPC (Remote Procedure Call))
  - ✓ 각각의 RPC에 대해 독립된 쓰레드를 사용하도록 프로그램을 재작성하면, 처리 속도 크게 향상



프로세스와 쓰레드

- 쓰레드의 기능(thread functionality)
  - ✓ 쓰레드 상태
    - 수행, 준비, 블록
    - 기본적인 쓰레드 연산:
      - 생성(Spawn), 블록(Block), 비블록(Unblock), 종료(Finish), 디스패치(Dispatch)
    - 그림 4.4: 단일 처리기 상에서 다중 쓰레딩의 예

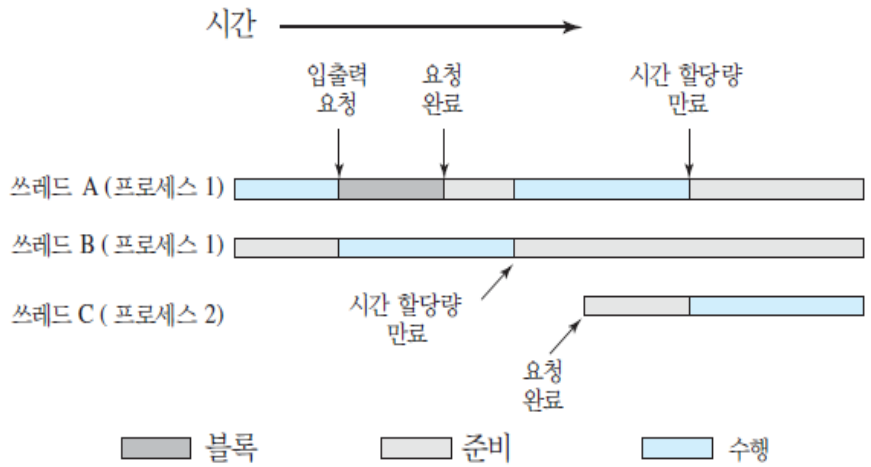
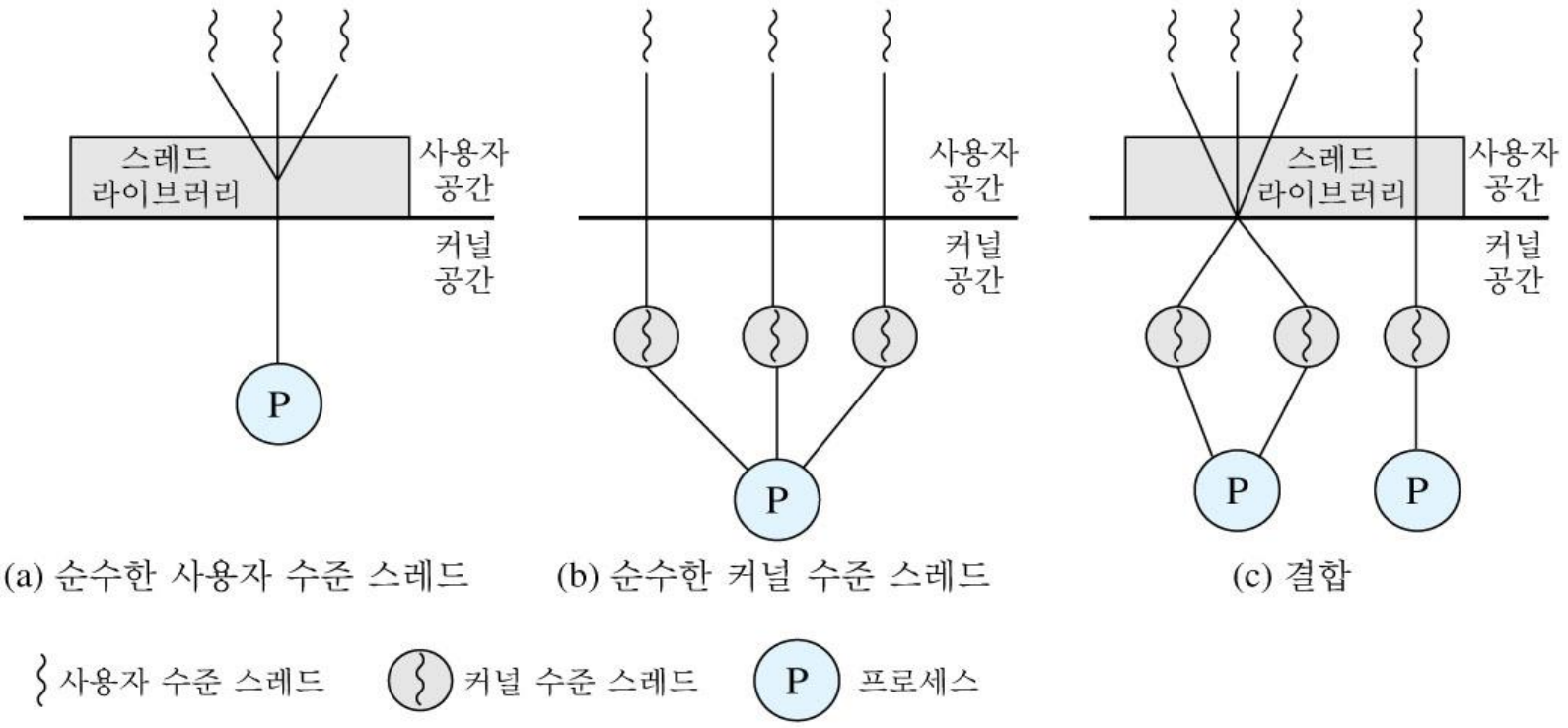


그림 4.4 단일 처리기 상에서 멀티쓰레딩 예

- 쓰레드 기능
  - ✓ 쓰레드 동기화(synchronization)
    - 한 프로세스 내의 모든 쓰레드들은 동일 주소 공간 및 자원들을 공유하므로 다른 쓰레드 데이터를 손상하지 않도록 동기화 필요
    - 공유 자원: 변수, 파일, 이중 연결 리스트(double linked lists)
    - 공유 자원에 대해 동시 접근 시(특히, 갱신 시), 일관성 유지 기법 필요
      - 예로, 한 공유 변수에 대해 읽기 연산은 동시 접근이 가능하나, 쓰기(write) 연산은 한 순간에 하나만 가능
      - 동기화 관련 내용은 제 5장 및 제 6장에서 다룸

4.2 쓰레드 유형

- 사용자 수준 쓰레드(User-level thread, ULT)
- 커널 수준 쓰레드(Kernel-level thread, KLT)=경량프로세스



프로세스와 쓰레드

- 사용자 수준 쓰레드 및 커널 수준 쓰레드
  - ✓ 사용자 수준 쓰레드
    - 응용이 모든 쓰레드 관리를 책임짐
      - **쓰레드 라이브러리가** 쓰레드 생성, 제거, 데이터 전송, 동기화, 스케줄, 문맥교환 코드 제공
    - 커널은 쓰레드의 존재를 모름
    - 이러한 접근 방법의 예로 ***cthread*** 및 ***pthread*** 등이 있음.
  - ✓ 커널 수준 쓰레드
    - 커널이 프로세스 및 쓰레드에 대한 문맥 정보를 관리
    - 쓰레드에 대한 스케줄링이 커널 수준에서 수행됨
    - 이러한 접근 방법의 예로 Windows가 있음
  - ✓ 결합된 접근 방법 (Combined approach)
    - 대표적인 예로 Solaris(UNIX)가 있음

프로세스와 쓰레드

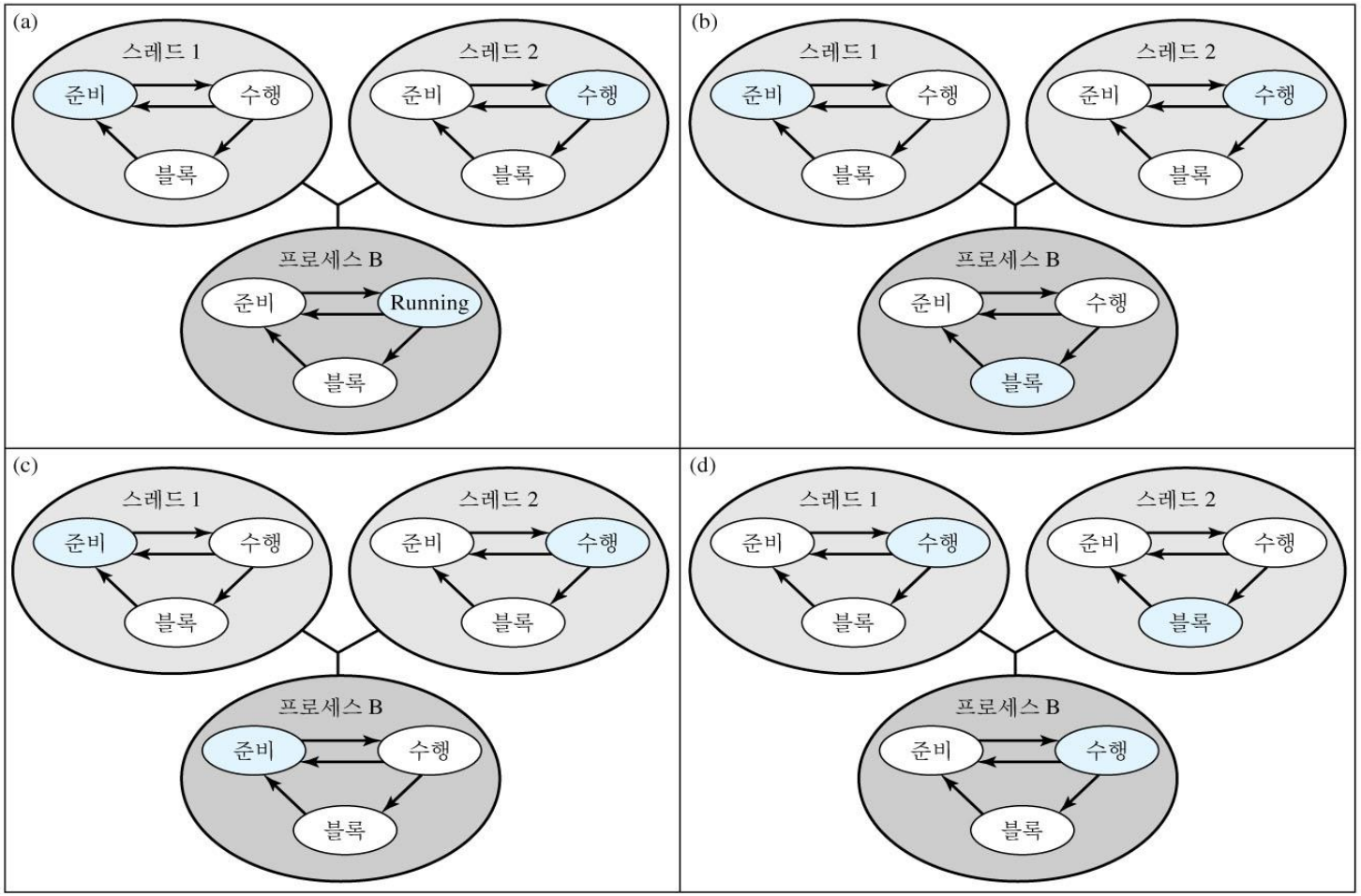
- 사용자 수준 쓰레드 및 커널 수준 쓰레드
  - ✓ 사용자 수준 쓰레드의 장점
    - 쓰레드 관리를 위한 모든 자료구조가 프로세스의 사용자 주소공간에 있게 때문에 쓰레드 교환/교체 시에 커널 모드 권한이 불필요
      - 두 번의 모드 전이 오버헤드를 절약 가능
    - 특정 응용에 적합한 스케줄링 적용 가능
    - 모든 OS에서 수행 가능하며, 기본 커널 변경할 필요 없음
  - ✓ 사용자 수준 쓰레드의 단점
    - 한 쓰레드가 블록 상태를 유발하는 시스템 호출을 수행할 경우, 자신뿐만 아니라 그 프로세스 내의 모든 다른 쓰레드들도 블록됨
    - 다중처리기의 장점(merits)을 살리지 못함.
      - 커널은 한번에 하나의 처리기에 하나의 프로세스를 할당
    - 커널 루틴 자체는 다중쓰레딩 될 수 없다.

표 4.1 쓰레드와 프로세스 연산지연(μs)

연산	사용자 수준 쓰레드	커널 수준 쓰레드	프로세스
Null Fork	34	948	11,300
Signal–Wait	37	441	1,840

프로세스와 쓰레드

- 사용자 수준 쓰레드 상태들의 예

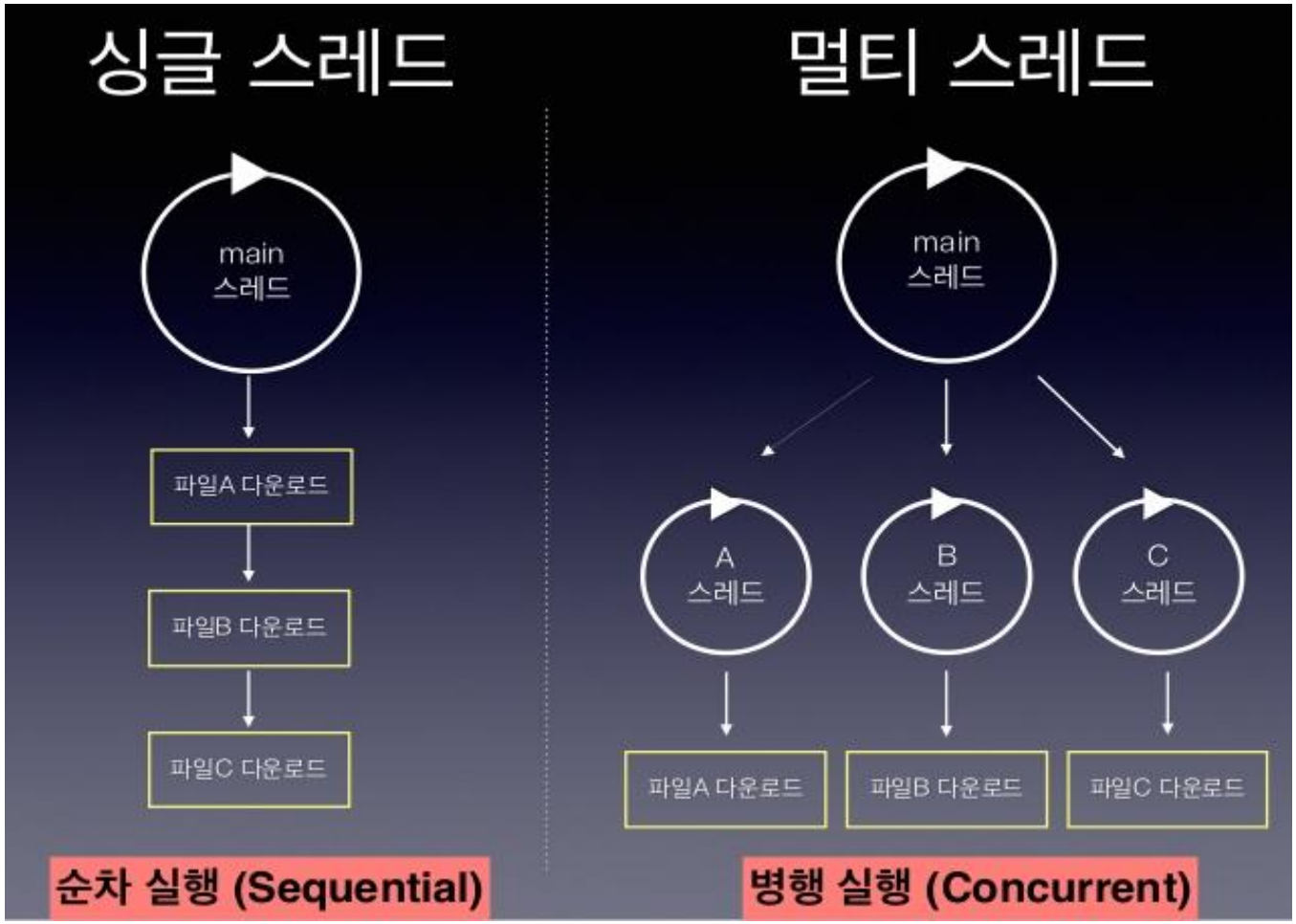


쓰레드와 프로세스 간의 관계

표 4.2 | 쓰레드와 프로세스 간의 관계

쓰레드:프로세스	설명	시스템 예
1:1	수행 중인 각 쓰레드는 자신의 주소 공간과 자원을 갖는 유일한 프로세스이다.	대부분의 UNIX 시스템
M:1	프로세스는 주소 공간과 동적인 자원 소유권을 정의하며, 여러 쓰레드가 이 프로세스 내에서 생성 및 수행될 수 있다.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	쓰레드는 한 프로세스 환경에서 다른 프로세스 환경으로 이동할 수 있다. 이것은 쓰레드가 다른 시스템 간을 쉽게 이동할 수 있도록 해준다.	Ra(Clouds), Emerald
M:N	1:M과 M:1의 특성을 혼합한 것이다.	TRIX

4.3 멀티코어와 멀티쓰레딩



4.3 멀티코어와 멀티쓰레딩

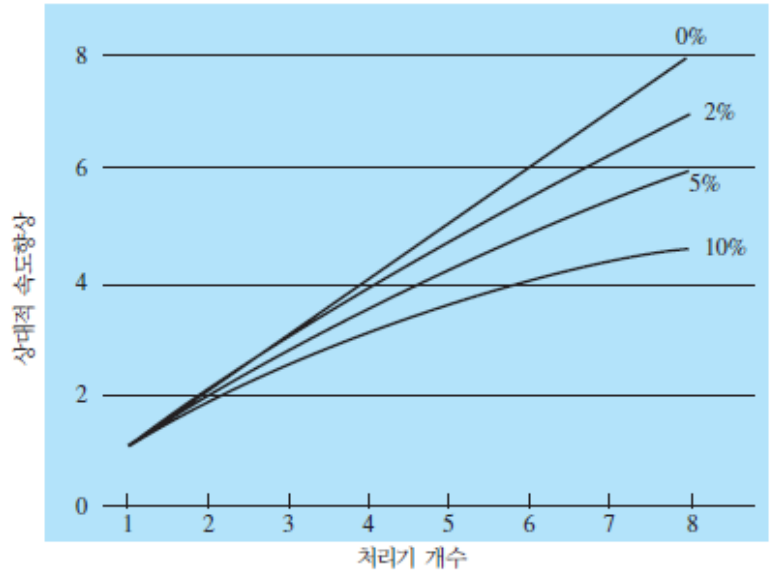
- 여러 개의 쓰레드로 구성된 하나의 응용 프로그램을 지원하기위해 멀티코어 시스템 사용
- Amdahl의 법칙

$$\text{속도향상} = \frac{\text{단일 처리기 상에서 프로그램을 실행한 시간}}{N\text{개의 병렬 처리기 상에서 프로그램을 실행한 시간}} = \frac{1}{(1-f) + \frac{f}{N}}$$

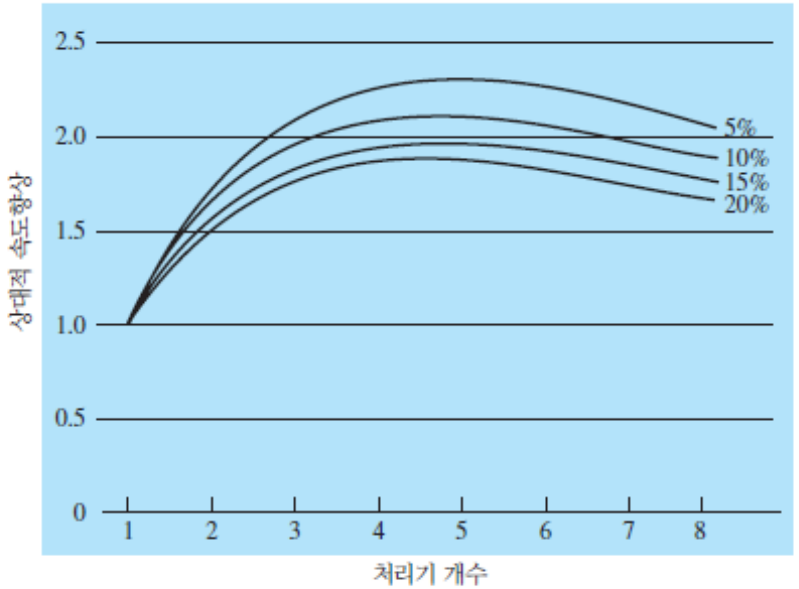
이 법칙은 프로그램의 실행 시간이 본래부터 순차적으로 동작하는(병렬화할 수 없는) 코드를 포함한 $(1-f)$의 실행 시간과 스케줄링 부하 없이 병렬로 처리 가능한 코드를 포함한 f의 실행 시간으로 구성되어 있다고 가정한다.

멀티코어와 멀티쓰레딩

- 멀티코어가 성능에 미치는 영향



(a) 순차 수행 비율이 0%, 2%, 5%, 10% 일 때 속도 향상



(b) 오버헤드를 고려한 속도향상

✓ 단지 10%의 코드가 본래부터 순차적으로 동작하는 코드라면(f=0.9), 8개의 처리기로 구성된 멀티코어시스템에서 수행되는 프로그램은 4.7만큼의 속도 향상

- 여기에 멀티프로세서 들간의 통신에 오버헤드 발생

멀티코어와 멀티쓰레딩

- 멀티코어가 성능에 미치는 영향
  - ✓ 데이터관리 시스템과 응용은 멀티코어 시스템을 효과적으로 사용할 수 있는 분야이며, 코어수 증가에 따라 처리량 증가

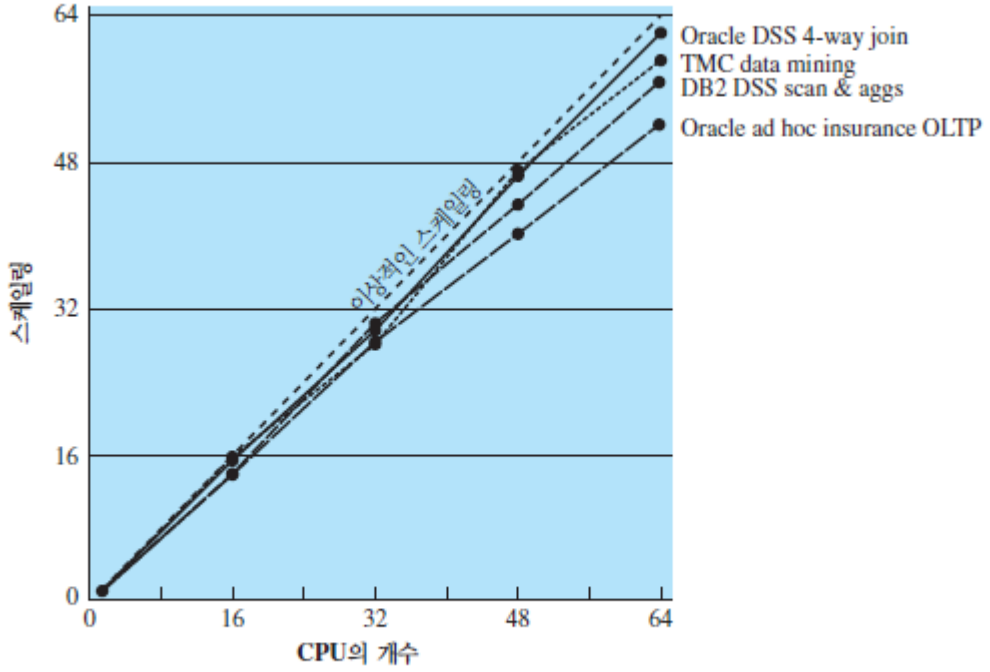


그림 4.8 멀티프로세서 하드웨어 상에서 데이터베이스 작업량의 스케일링

멀티코어와 멀티쓰레딩

- 멀티코어 성능 향상이 많은 응용
  - ✓ 멀티쓰레드화된 네이티브 응용: 멀티쓰레드화된 응용은 소수의 고도로 쓰레드화된 프로세스를 갖는 특징을 갖는다. 쓰레드화된 응용의 예는 IBM사의 Lotus Domino와 Oracle사의 Siebel 고객 관계 관리(Customer Relationship Manager, CRM)가 있다.
  - ✓ 멀티프로세스 응용: 멀티프로세스 응용은 다수의 단일 쓰레드화된 프로세스들이 존재하는 특징을 갖는다. 멀티프로세스 응용의 예는 Oracle 데이터베이스와 SAP, PeapleSoft가 있다.
  - ✓ 자바 응용: 자바 응용은 근본적인 방식으로 쓰레딩을 포함하고 있다. 자바 언어가 멀티쓰레드화된 응용을 개발하는데 매우 용이할 뿐만 아니라 자바 가상 머신도 자바 응용에 대한 스케줄링과 메모리 관리를 지원하는 멀티쓰레드화된 프로세스이다. 멀티코어 자원으로부터 직접적인 혜택을 받는 자바 응용은 Sun사의 Java Application Server, BEA사의 Weblogic, IBM사의 Websphere, 오픈 소스인 Tomcapt 응용 서버와 같은 응용 서버들을 포함한다.
  - ✓ 멀티인스턴스 응용: 개별적인 응용이 많은 수의 쓰레드를 사용해서 속도를 향상시키지 못할 지라도 다수의 응용 인스턴스를 멀티코어 구조상에서 병렬적으로 실행함으로써 속도를 향상 시킬 수 있다. 만약 다수의 응용 인스턴스가 어느 정도의 격리(isolation)를 요구한다면 가상 기술(운영체제의 하드웨어에 대한)을 사용하여 각각의 인스턴스마다 독립되고 안전한 환경을 제공할 수 있다.

응용 예제: Valve 게임 소프트웨어

- Valve는 하이브리드 쓰레딩이 8개/16개 멀티코어 시스템에서 최적 성능 발휘
  - ✓ 그림은 렌더링 모듈에 대한 쓰레드 구조, 계층적구조에서 상위수준의 쓰레드는 필요에따라 하위 수준의 쓰레드 생성

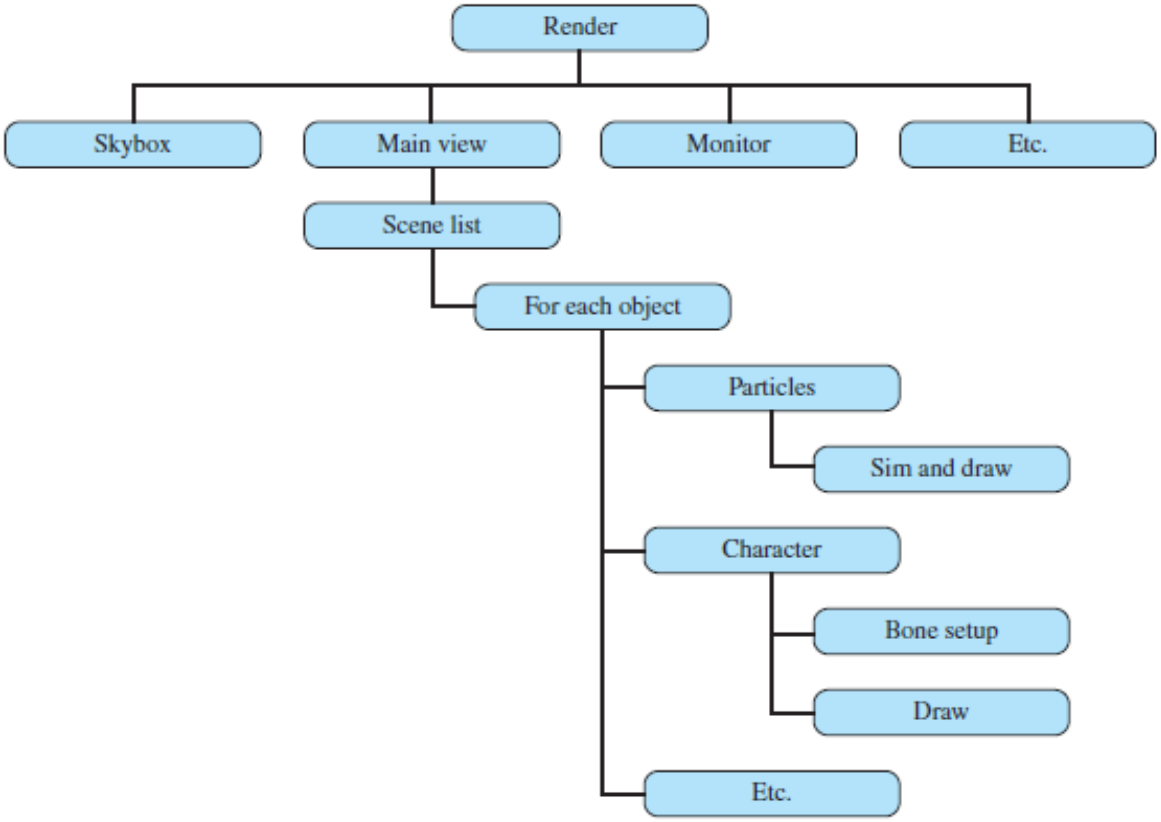


그림 4.9 렌더링 모듈에 대한 하이브리드 쓰레딩

사례 연구(Case Studies)

- Windows, Solaris, Linux

- 서로 다른 운영체제 환경에 의해 지원되는 프로세스 구조와 기능 표현 방법
  - ✓ 프로세스의 이름 부여 방법
  - ✓ 프로세스 내에서 쓰레드의 지원 여부
  - ✓ 프로세스의 표현 방법
  - ✓ 프로세스 자원의 보호 방법
  - ✓ 프로세스간 통신과 동기화를 위해서 사용되는 기법
  - ✓ 프로세스들이 상호 연관되는 방법

4.4 Windows 8의 프로세스와 쓰레드 관리

- Windows Process and Thread
  - ✓ Implemented as objects
  - ✓ Multithreading
    - process may contain one or more threads
  - ✓ 6 thread states
    - Ready, Standby, Running, Waiting, Transition, Terminated
  - ✓ Support for a variety of OS environments
    - Environmental subsystems emulate a particular process and thread structures and functionalities based on windows general objects
      - Environmental OS subsystem for Win32 and OS2: visible both process and thread information
      - Environmental OS subsystem for POSIX and 16-bits Windows: visible only process information

Windows 8의 프로세스와 쓰레드 관리

- 응용은 하나 또는 그 이상의 프로세스로 구성
- 각 프로세스는 프로그램 실행을 위한 자원을 제공
- 쓰레드는 실행을 위해 스케줄링 될 수 있는 프로세스 내의 개체
- 작업 객체는 프로세스들의 그룹을 하나의 단위로 써 관리할 수 있게 함

- 쓰레드 풀(pool)은 응용을 대신하여 비동기적인 콜백을 효과적으로 실행하는 작업 쓰레드의 집합
- 파이버(Fiber)는 응용 프로그램에 의해 수동으로 스케줄링되는 실행 단위
  - ✓ 각 쓰레드는 여러 개의 파이버를 스케쥴 가능
- 사용자모드 스케줄링 (UMS) 은 응용이 자신의 쓰레드를 통해 스케줄링하는 경량화 기법

Windows 8의 프로세스와 쓰레드 관리

- 후면 작업과 응용의 생명주기 측면에서 전통적인 Windows 접근 방법과 차이가 있음
- 개발자들은 자신이 개발한 개별 응용프로그램의 상태를 관리해야 할 책임이 있음

- 새로운 Metro 인터페이스에서는 Windows 8이 응용 프로그램의 프로세스 생명주기를 관리
  - 제한된 숫자의 응용 프로그램이 SnapView 기능을 사용하는 Metro 사용자 인터페이스에서 메인 앱(main app)과 동시에 수행
  - 하나의 Store 응용 프로그램만이 차례대로 수행
- Live Tiles은 시스템 상에서 지속적으로 수행되는 응용 프로그램의 외관을 보여줌
  - 실제적으로 푸시 알림을 받을 뿐이지 제공된 동적 콘텐트를 보여주기 위해 시스템 자원을 사용하지 않음

Metro 인터페이스

- Metro 인터페이스 에서 전면(foreground) 응용은 사용자가 사용할 수 있는 모든 처리기, 네트워크, 디스크 자원을 접근함
  - 다른 응용들은 보류(suspend) 상태로 이러한 자원을 접근할 수 없음
- 응용이 보류 상태로 전환될 때 사용자 정보의 상태를 저장하기 위한 이벤트가 발생됨
  - 응용 개발자가 이를 처리해야 함
- Windows 8은 후면 응용을 종료 시킴
  - 앱의 실행이 보류되거나 Windows에 의해 종료되었을 때 이전의 상태를 복원하기 위해서 앱의 상태를 저장할 필요가 있음
  - 앱이 전면으로 전환되었을 때는 메모리로부터 사용자의 상태를 가져오는 이벤트가 발생

윈도우 프로세스

Windows 커널에 의해 제공되는 프로세스와 서비스는 상대적으로 단순하고 범용성을 가짐

- 객체로 구현되어 있음
- 새로운 프로세스로 생성되거나 기존의 프로세스를 복사
- 수행 가능한 프로세스는 하나 이상의 쓰레드를 가지고 있음
- 프로세스와 쓰레드 객체는 동기화 능력을 내장하고 있음

Windows 8 프로세스

- Windows Process
  - ✓ 프로세스가 알고 있는 다른 객체에 대한 핸들을 포함하는 객체 테이블
  - ✓ 하나의 핸들은 이 객체에 포함된 각 쓰레드를 위해 존재

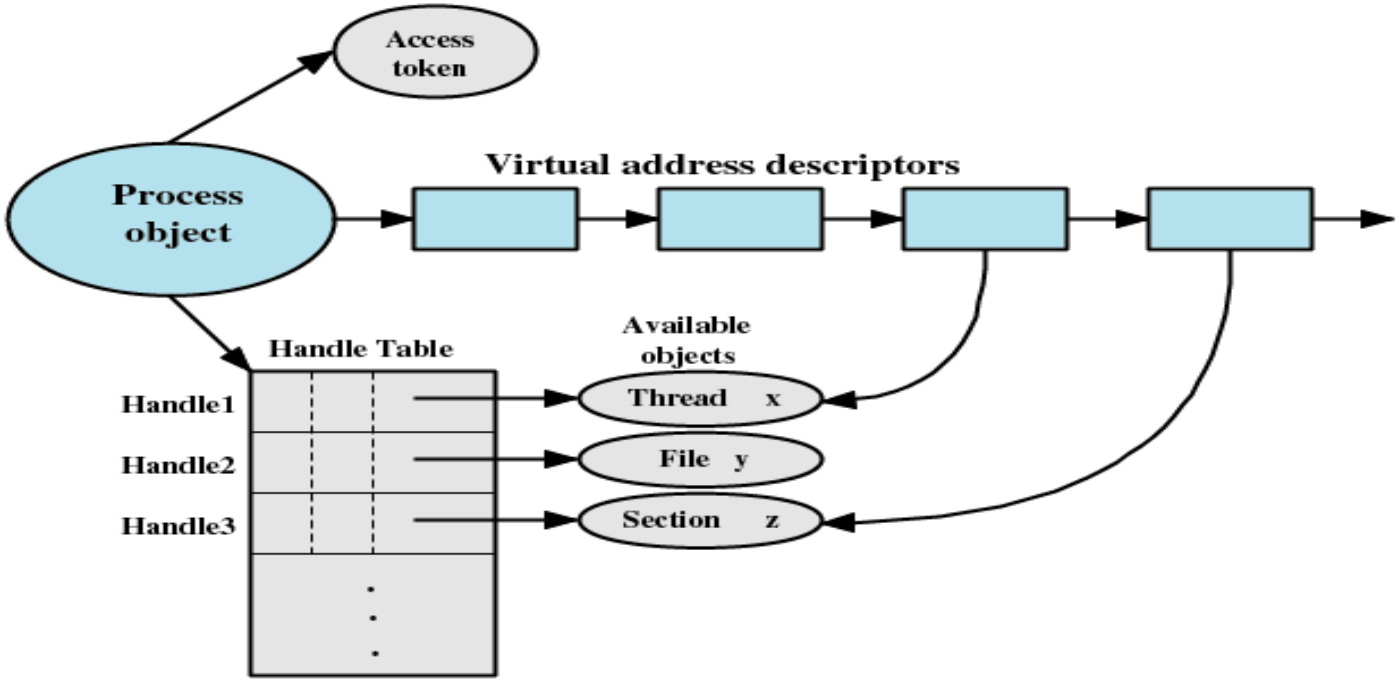
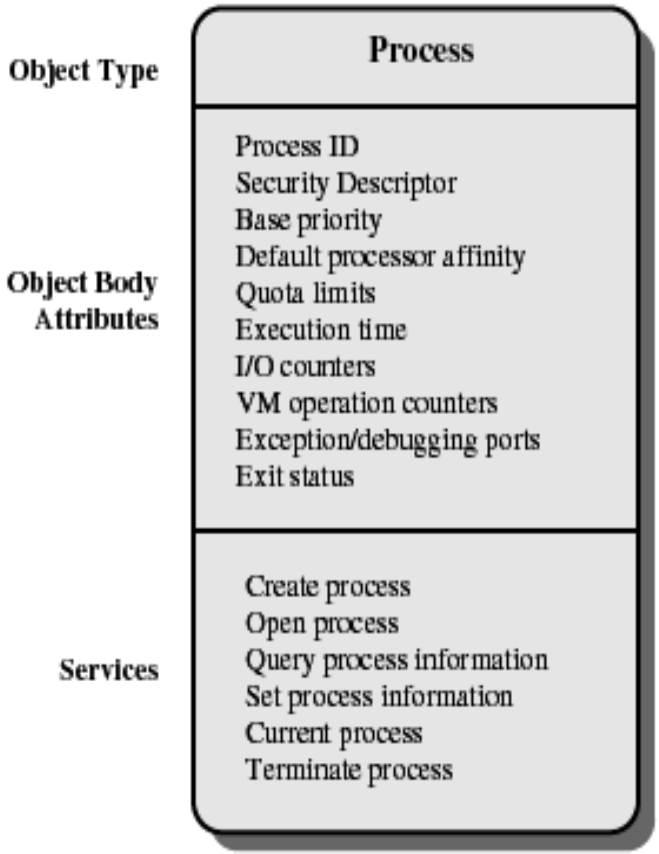


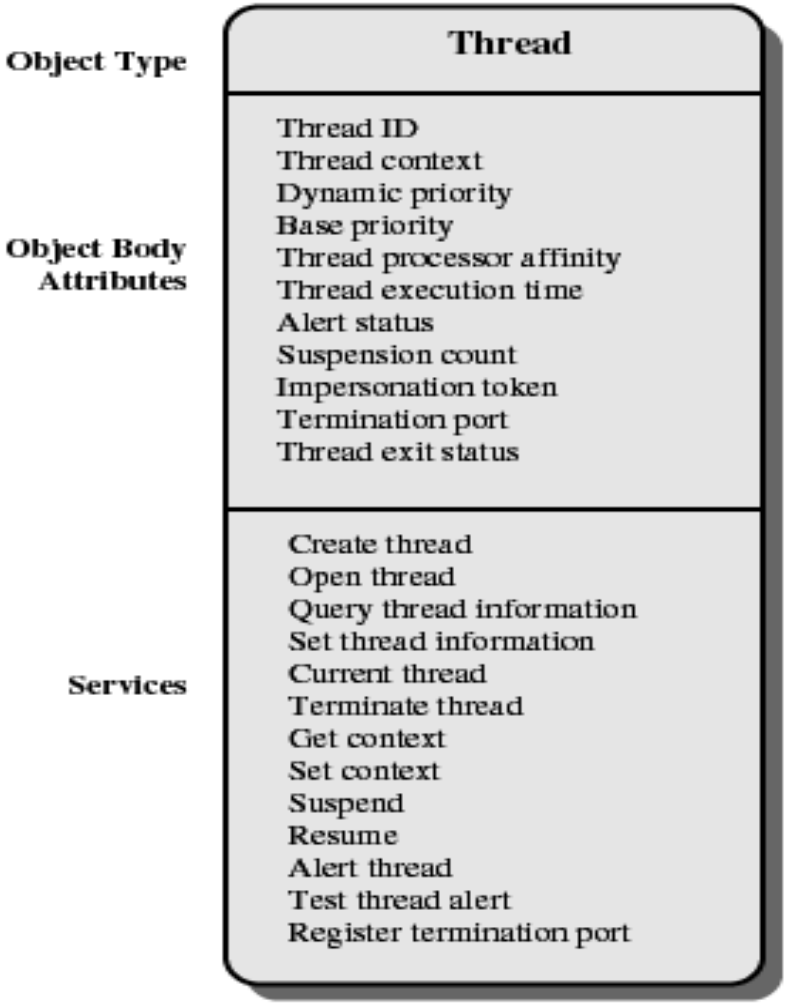
Figure 4.12 A Windows Process and Its Resources

Windows Thread and SMP Management

- Process and Thread Object



(a) Process object



(b) Thread object

윈도우 프로세스 객체 속성

표 4.3 Windows 프로세스 객체 속성

프로세스 ID	운영체제에서 프로세스를 식별할 수 있는 유일한 값.
보안 디스크립터	누가 객체를 생성했으며, 누가 객체를(로) 사용(접근)할 수 있는가, 누가 객체로의 접근이 거부되는지 등을 기술.
기준(base) 우선순위	프로세스 내의 쓰레드에 대해 기준이 되는 수행 우선순위.
기본(default) 처리기 친화성	처리기들의 기본 집합으로, 프로세스 내의 쓰레드들이 그 처리기들 상에서 수행됨
쿼터(quota) 한도	페이지로 분할된(paged) 또는 그렇지 않은 시스템 메모리, 페이징 파일 공간, 사용자 프로세스가 사용할 수 있는 처리기 시간 등에 대한 최대량
수행 시간	프로세스 내의 쓰레드가 수행된 총 시간
I/O 카운터(계수기)	프로세스 내의 쓰레드가 수행한 입출력 연산의 유형과 회수를 기록한 변수들
VM 연산 카운터	프로세스 내의 쓰레드가 수행한 가상메모리 연산의 유형과 회수를 기록한 변수들
예외상황/디버깅 포트	프로세스 내의 쓰레드 중 하나가 예외상황을 발생시켰을 때, 프로세스 관리자가 메시지를 보낼 프로세스 간 통신 채널
종료 상태	프로세스가 종료된 이유

윈도우 쓰레드 객체 속성

표 4.4 Windows 쓰레드 객체 속성

쓰레드 ID	쓰레드가 어떤 서비를 호출했을 때 쓰레드를 식별할 수 있는 유일한 값
쓰레드 문맥	쓰레드의 수행 상태를 정의하는 레지스터 값과 기타 휘발성(volatile) 데이터 값들의 집합
동적 우선순위	임의 시점에서 쓰레드 수행 우선순위
기준 우선순위	쓰레드의 동적 우선순위에 대한 하한값
쓰레드 처리기 친화성	쓰레드를 수행할 수 있는 처리기들의 집합으로, 해당 쓰레드가 속한 프로세스의 처리기 친화성과 같거나 그 일부이다.
쓰레드 수행시간	쓰레드가 사용자 모드와 커널 모드에서 수행한 누적 시간
경보(alert) 상태	쓰레드가 비동기 프로시저 호출을 수행해야하는지를 나타내는 플래그
보류 계수(count)	쓰레드의 수행 재개 없이 보류된 횟수
위장 토큰	한 쓰레드가 다른 프로세스 대신 연산을 수행하게 해주는 임시 접근 토큰(서브시스템에서 사용)
종료 포트	쓰레드가 종료되었을 때 프로세스 관리자가 메시지를 보내는 프로세스 간 통신채널(서브시스템에서 사용)
쓰레드 종료 상태	쓰레드가 종료된 이유

Windows Thread and SMP Management

- Thread states

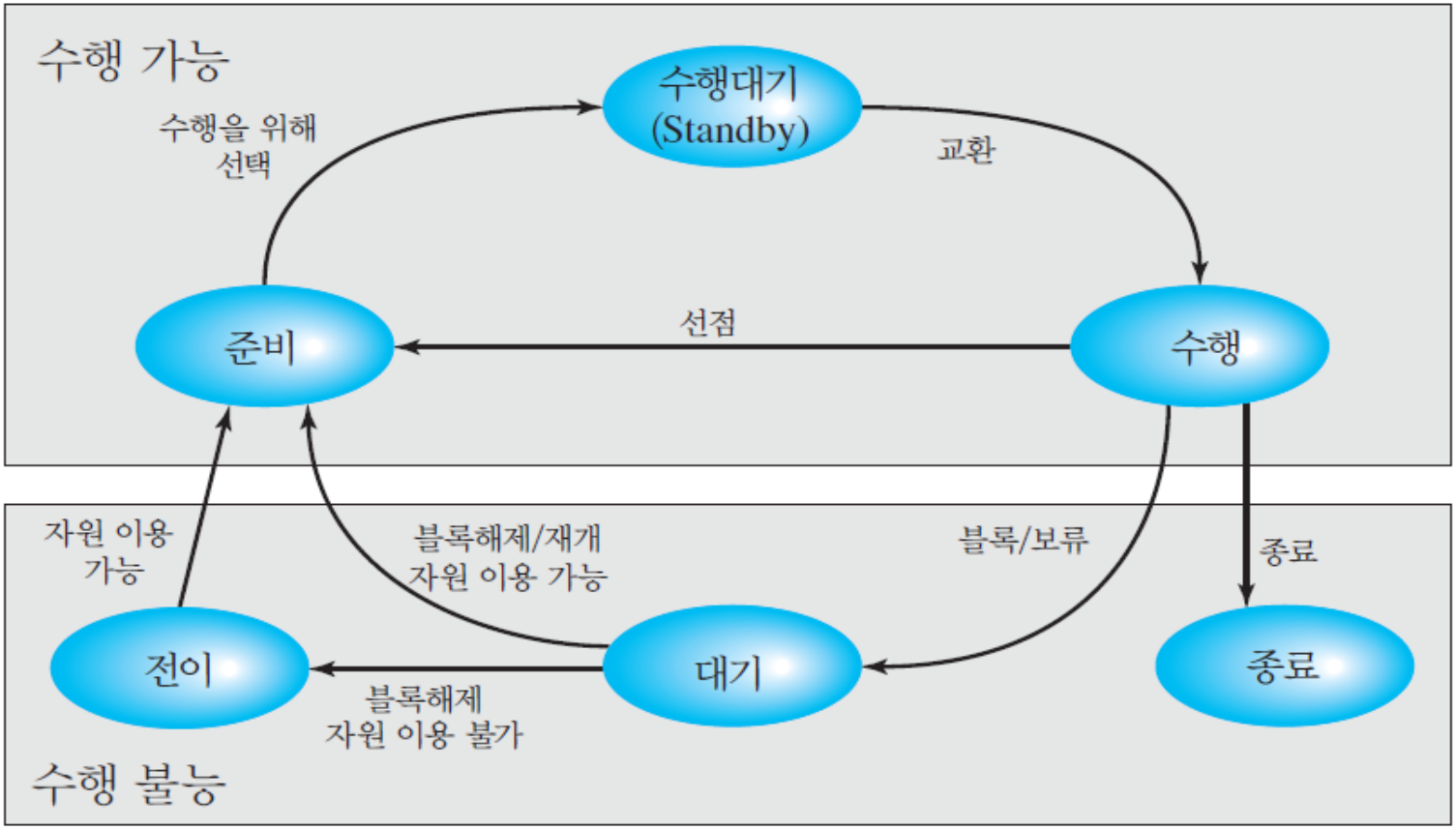


그림 4.11 Windows 쓰레드 상태

4.5 Solaris Thread and SMP Management

- Multilevel Thread Architecture
  - ✓ Process: 보통의 UNIX 프로세스
    - 사용자 주소공간, 스택, 프로세스 제어 블록(PCB) 등을 포함
  - ✓ 사용자 수준 쓰레드 (ULT)
    - 프로세스 주소공간에서 쓰레드 라이브러리를 통해 구현
  - ✓ 경량 프로세스 (Lightweight processes ≡ LWP)
    - ULT와 커널 쓰레드 사이의 사상(mapping)
  - ✓ 커널 쓰레드 ≡ 커널 수준 쓰레드 (KLT)
    - 처리기에 수행되는 기본 개체

Solaris 쓰레드 및 SMP 관리

- 프로세스와 쓰레드 관계

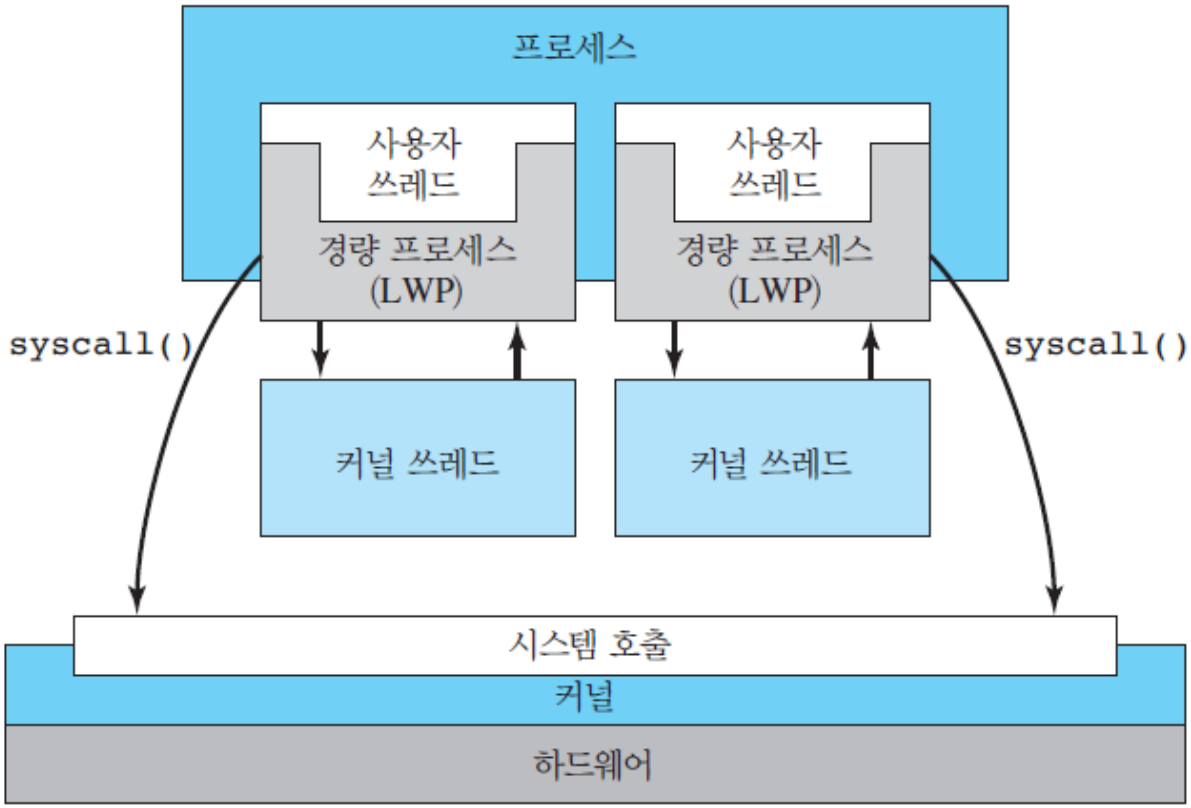
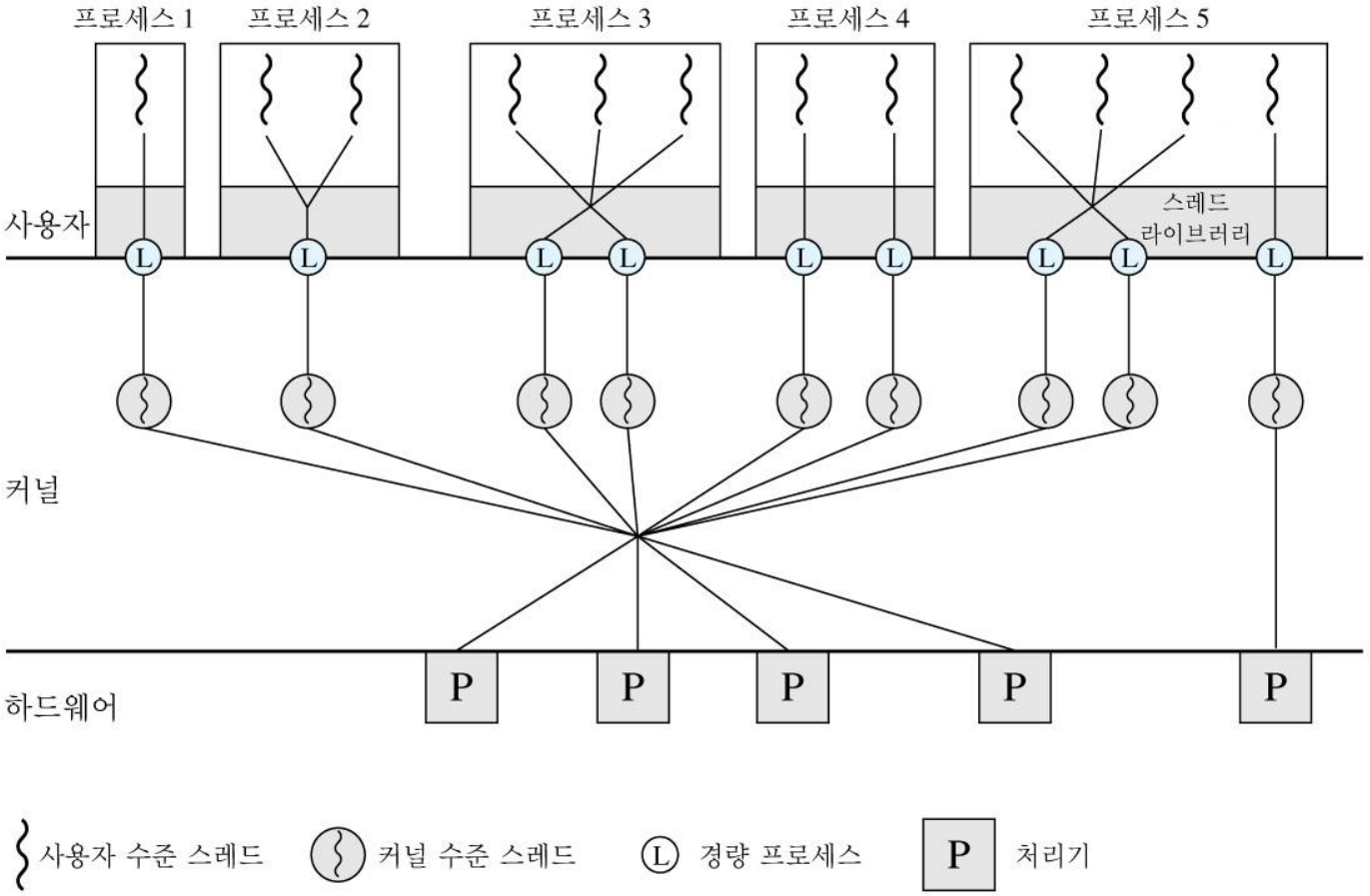


그림 4.12 Solaris에서의 프로세스와 쓰레드 [MCDO07]

Solaris 쓰레드 및 SMP 관리

- Solaris 다중쓰레드 구조의 예



Solaris 쓰레드 및 SMP 관리

- 동기 (motivations)
  - ✓ 어떤 프로그램은 논리적인 병렬성(parallelism)을 가지지만 하드웨어 병렬성을 요구하지 않는다. → 하나의 LWP에 ULT들의 집합 사상 가능
    - 예: 다중 윈도우 환경에서 한 순간에 하나의 윈도우만이 활성화
  - ✓ 어떤 응용이 (I/O 요청과 같이) 블록될 수 있는 쓰레드를 포함하는 경우에는 다수의 LWP를 이용하는 것이 효과적
  - ✓ 어떤 응용에서는 ULT를 LWP에 일대일로 사상하는 것이 효과적
    - 예: 병렬 배열 계산 (parallel array computation)
  - ✓ 혼합 형태(Hybrids): bound and unbound ULTs가 공존
    - 예: 어떤 쓰레드는 RT 스케줄링을 수행하고, 나머지 쓰레드는 후면 함수(background functions)을 수행
  - ✓ 순수 커널 쓰레드

Solaris Thread and SMP Management

- Process structure

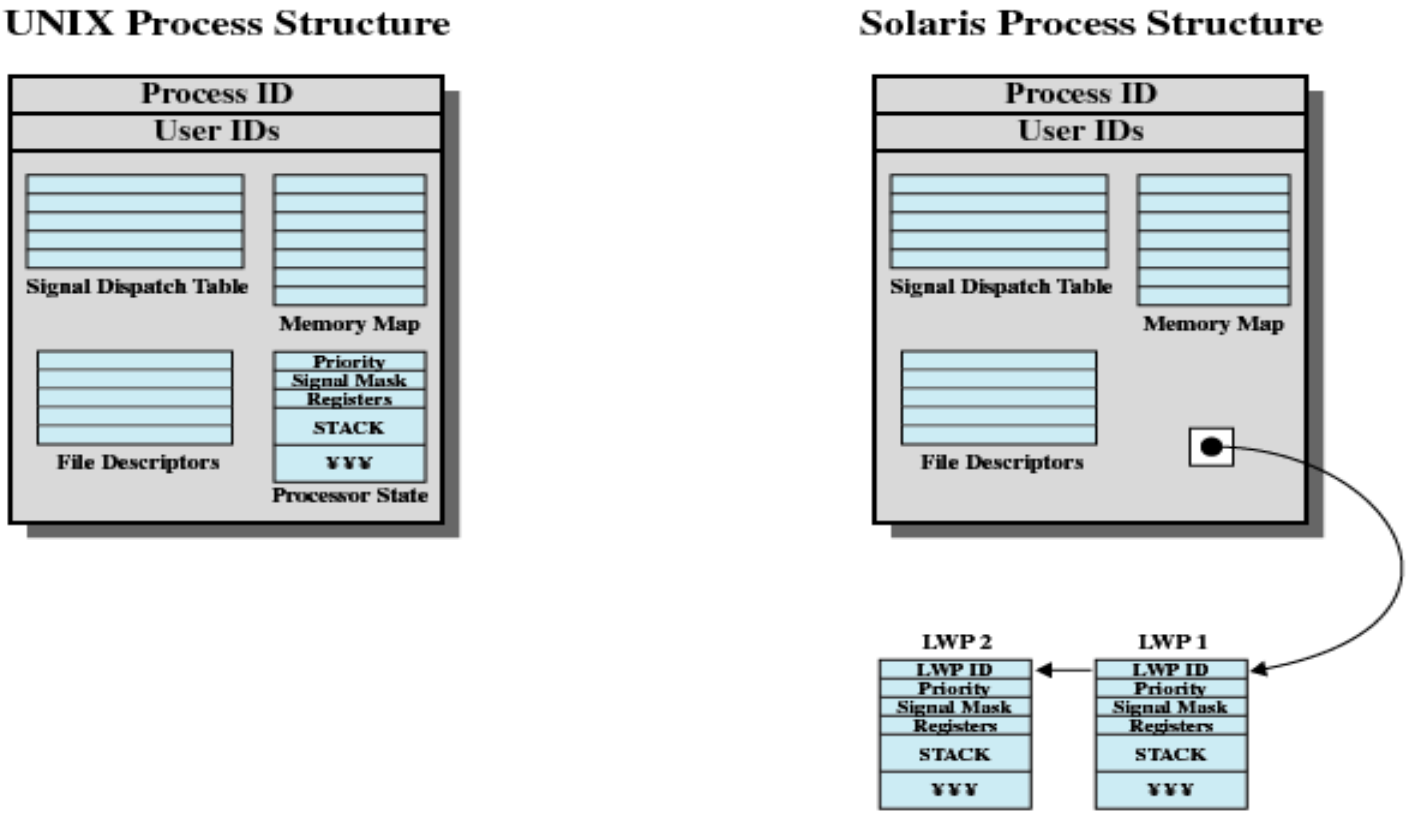


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]

Solaris Thread and SMP Management

- Thread execution

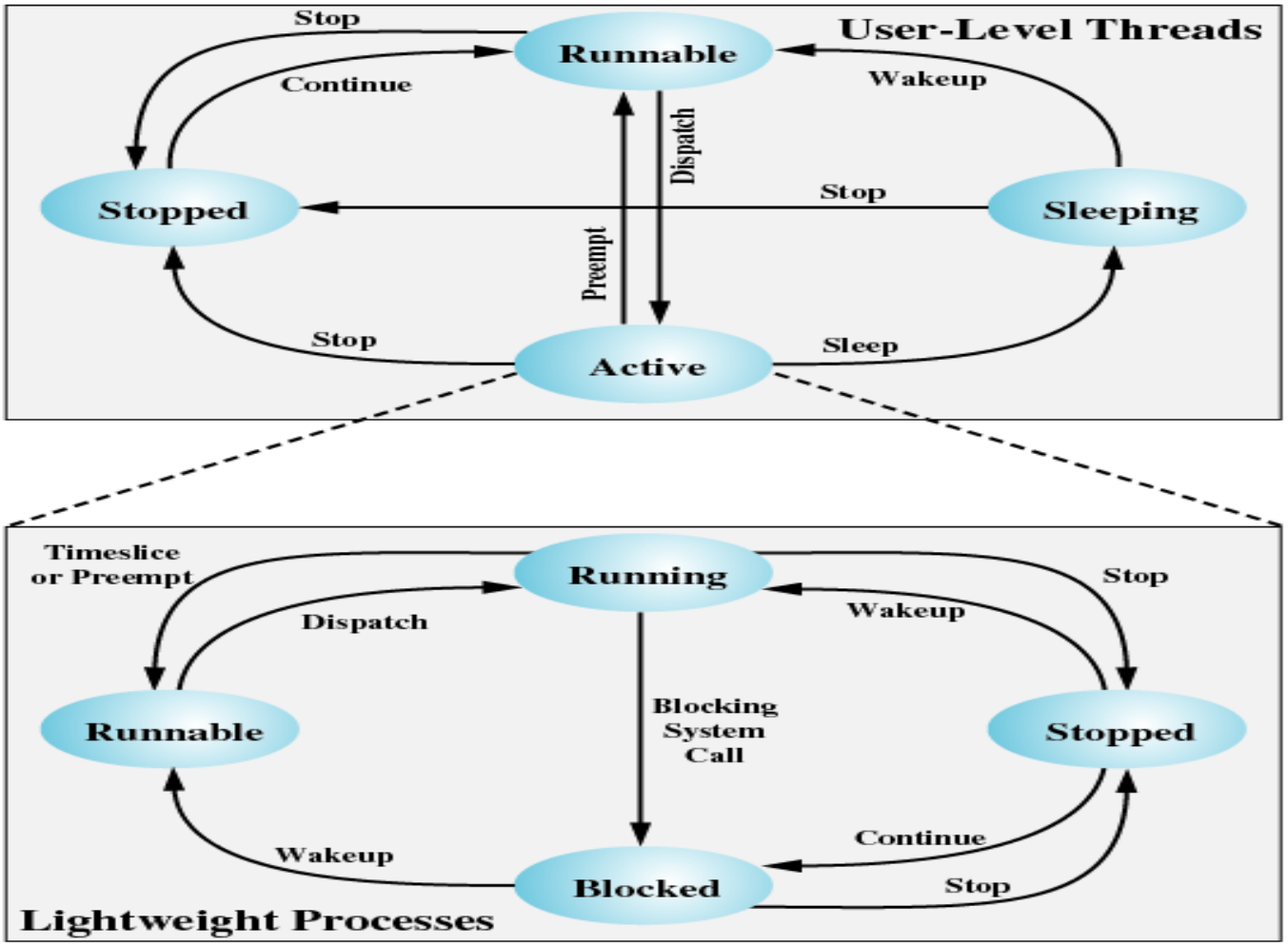


Figure 4.17 Solaris User-Level Thread and LWP States

Solaris 쓰레드 및 SMP 관리

- 활성화(active, 수행 중) 상태의 ULT 쓰레드에 대한 상태 전이의 예
  - ✓ 동기화 필요: 수면(sleeping) 상태로 전이
    - 예: 5장에서 기술될 상호배제 필요
  - ✓ 보류(suspension): 쓰레드를 보류시켜 정지(stopped) 상태로 전이
  - ✓ 선점(preemption):
    - 활성화 상태의 쓰레드가 수행가능(runnable) 상태로 전이
  - ✓ 양보(yielding):
    - thr_yield() 라이브러리 함수를 호출하면 수행가능 상태로 전이

☞ **Solaris: Interrupts as Threads**

- **인터럽트 쓰레드 도입**

4.6 Linux 프로세스 및 쓰레드 관리

- Linux 태스크 (task_struct 자료구조)
  - ✓ 상태
    - Running, Interruptable, Uninterruptable, Stopped, Zombie
  - ✓ 스케줄링 정보
    - 정책: SCHED_RR, SCHED_FIFO, SCHED_OTHER
  - ✓ 식별자(pid, uid, gid)
  - ✓ 프로세스간 통신 (IPC)
  - ✓ 링크: 부모 프로세스, 준비 상태의 프로세스들, 블록상태의 프로세스들
  - ✓ 시간과 타이머
  - ✓ 파일 시스템
  - ✓ 주소 공간
  - ✓ 처리기 의존 문맥(processor-specific context) : 레지스터 및 스택 정보

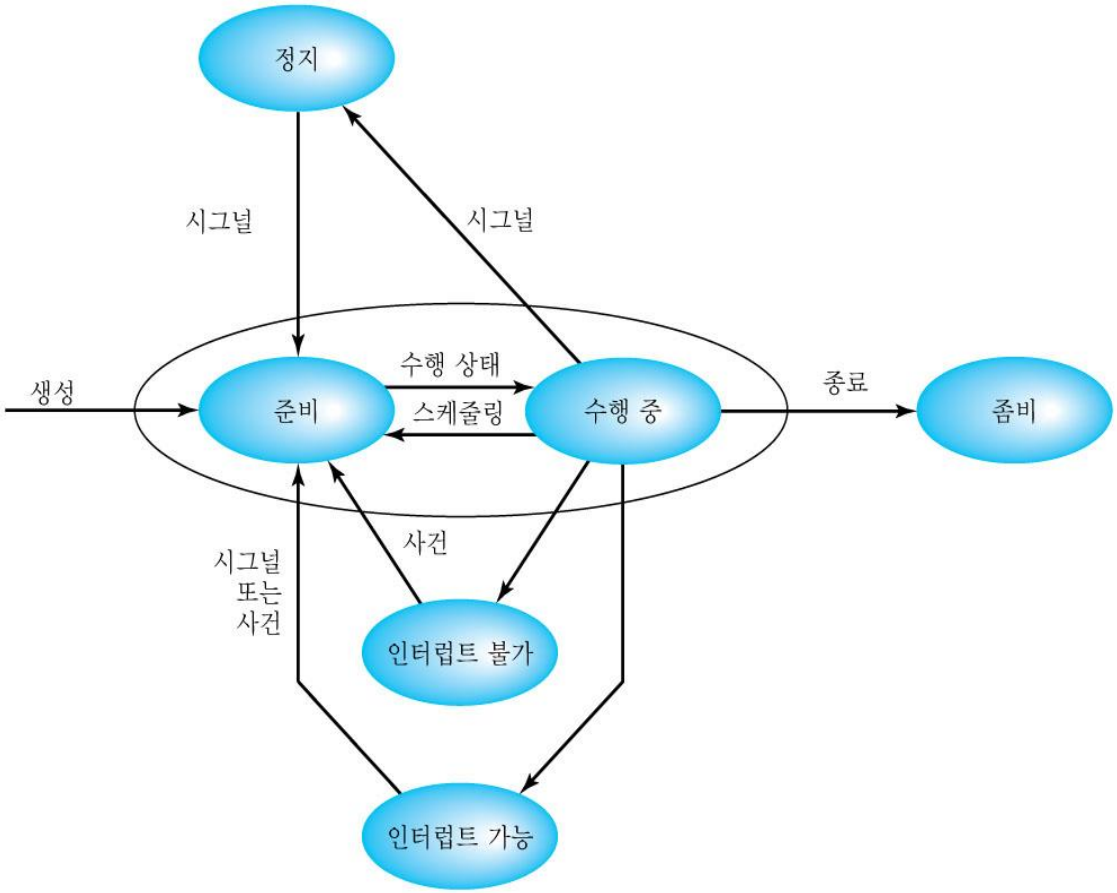
☞ **Linux: 다른 플래그(flag)를 갖는 clone() 호출은 각 프로세스에 분리된 스택 공간 생성**

☞ **쓰레드를 위해 별도의 분리된 자료구조가 없다.**

Linux 프로세스 및 쓰레드 관리

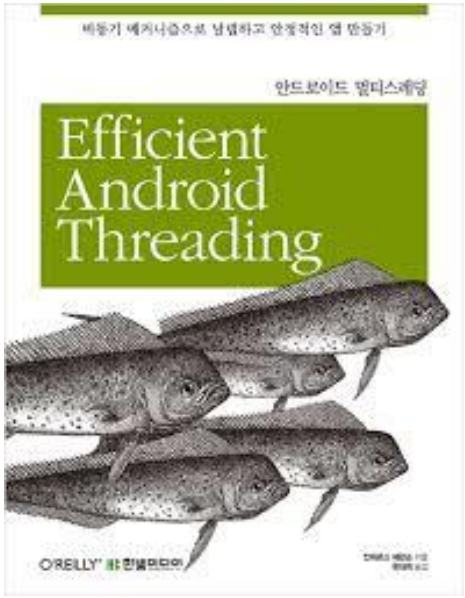
- 프로세스/쓰레드 모델
  - ✓ pthread(POSIX thread) 라이브러리 제공
  - ✓ 정지(stopped)
  - ✓ 수행 중(executing)
  - ✓ 좀비 (zombie)

  - * 시그널(signal)
  - * 사건(event)



4.7 Android 프로세스와 쓰레드 관리

- Android 응용은 앱을 구현한 소프트웨어
- Android 응용은 네 가지 유형의 응용 컴포넌트들의 여러 인스턴스로 구성
- 각 컴포넌트는 한 응용과 다른 응용 내에서 구별된 역할을 수행
- 4가지 유형의 컴포넌트:
  - 엑티비티(Activity)
  - 서비스 (Service)
  - 콘텐트 프로바이더(Content provider)
  - 브로드캐스트 리시버(Broadcast receiver)



Android 프로세스와 쓰레드 관리

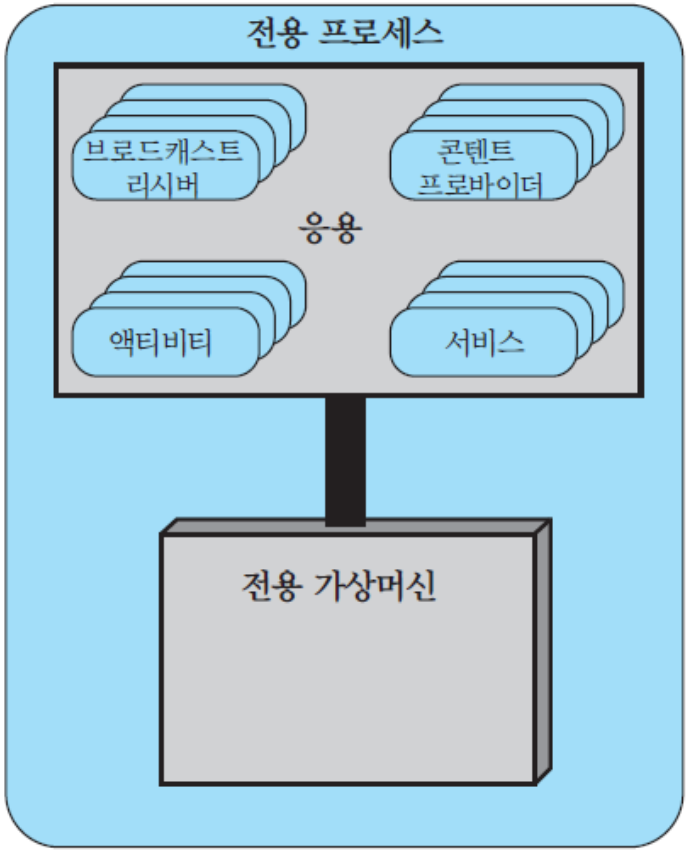
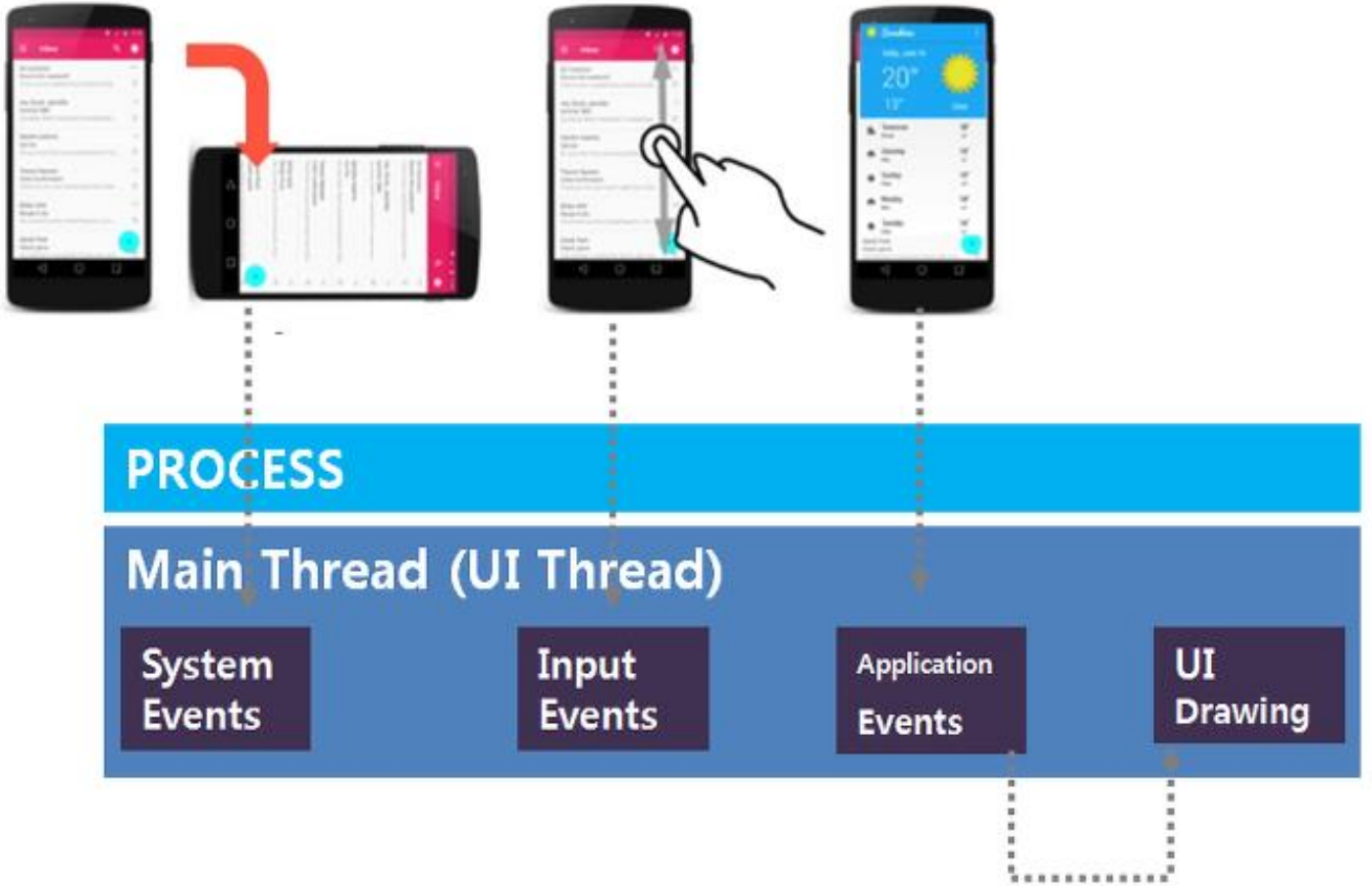


그림 4.16 Android 응용

Android 프로세스와 쓰레드 관리

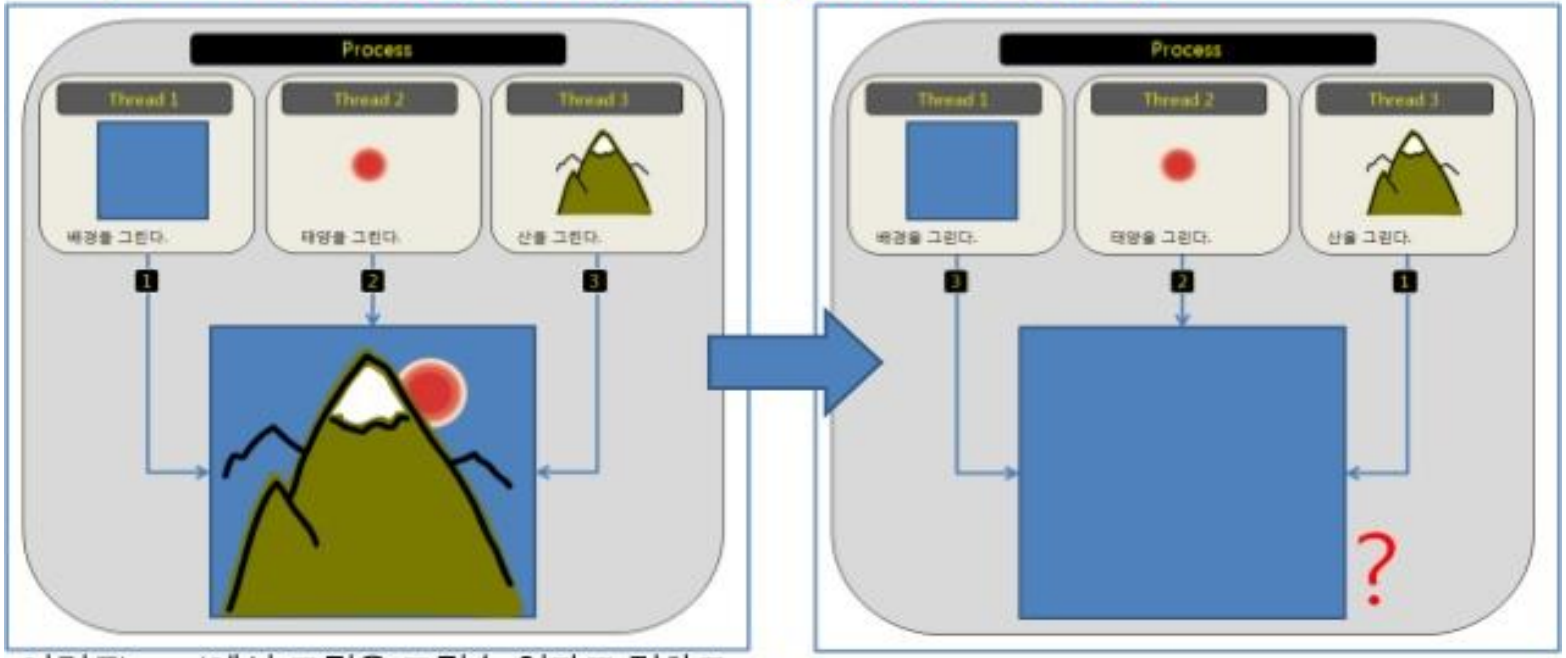


Android 프로세스와 쓰레드 관리

Thread 와 Android Main Thread

2. Android의 Main Thread

다른 Thread를 생성하여 UI 작업을 하는 것을 허용하지 않는다.



여러 Thread에서 그림을 그릴수 있다고 정하고,
Thread 1에서는 배경을 그리고,
Thread 2에서는 태양을 그리고,
Thread 3에서는 산을 그린다.
그리는 순서는 꼿 1,2,3 번 순서로 그려야 한다.

1번과 2번 과정에서 그렸던 그림이 3번의
그림으로 인해 덮어 버린다.
이 것을 UI가 꼬였다고들 한다.

Android 프로세스와 쓰레드 관리

- 응용내 여러 액티비티들은 각자가 상태 전이도상 특정상태 유지
  - ✓ 새로운 액티비티가 시작되면, 응용 SW는 Activity Manager에 의해 일련의 system call 수행

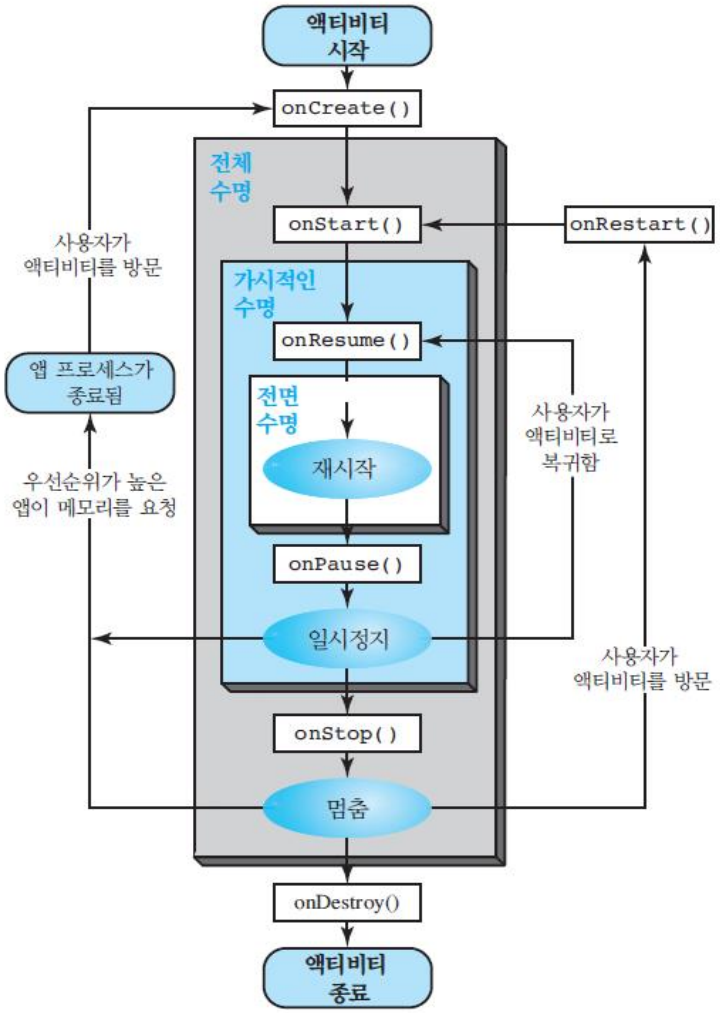
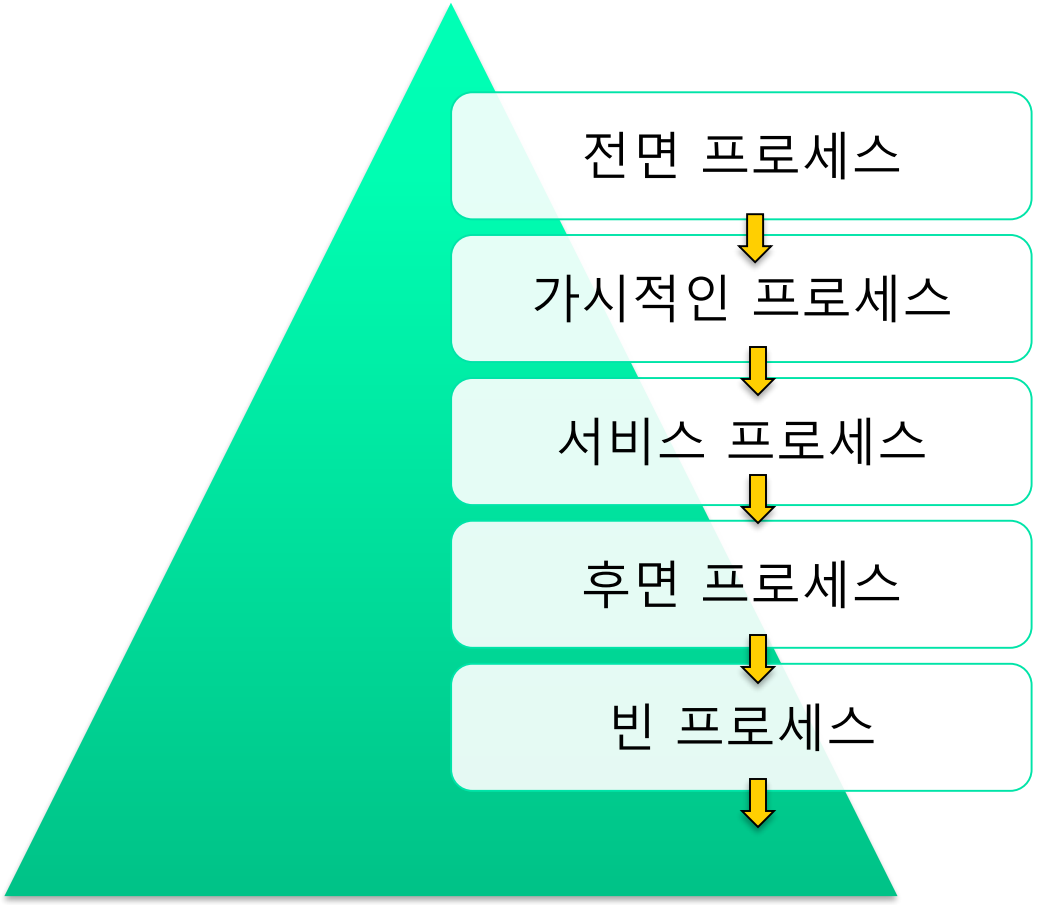


그림 4.17 액티비티 상태 전이도

프로세스와 쓰레드

- 필요한 자원을 회수하기 위해서 어떤 프로세스를 종료 시킬 것인지에 대한 결정은 우선순위 계층구조를 사용
- 가장 낮은 우선순위를 갖는 프로세스부터 종료
- 계층 구조상에서의 우선순위는 다음과 같음:



Summary

- Processes and Threads
  - ✓ resource container vs control flow
  - ✓ thread state
  - ✓ user-level thread vs kernel-level thread
- Symmetric Multiprocessing
- Microkernels
- Windows Thread and SMP Management
- Solaris Thread and SMP Management
  - ✓ user-level thread, kernel-level thread, lightweight process
- Linux Process and Thread Management