운영체제: 강의노트 05

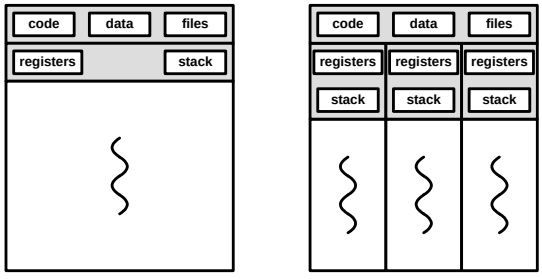
A. Silberschatz, P.B. Galvin, G. Gagne
Operating System Concepts,
Sixth Edition, John Wiley & Sons, 2003.

Part II. Process Management

5 스레드

5.1 개요

- **스레드(thread)**는 프로세스와 마찬가지로 CPU의 작업 단위이다. 그러나 스레드는 일반 프로세스(**중량 프로세스(heavyweight process)**)와 달리 **경량 프로세스(lightweight process)**라 한다.
- 프로세스는 프로그램에 수행과 그것에 필요한 자원을 하나의 개체에서 관리하기 위한 방법이지만 스레드는 자원의 관리보다는 프로그램의 하나의 실행 흐름으로서 CPU의 실행되기 위해 스케줄되어야 하는 개체이다.
- 스레드는 스레드 식별자, 프로그램 카운터, 레지스터 집합, 스택(함수 호출 처리에 필요한 메모리)만을 가지며, 텍스트 부분, 데이터 부분, 다른 운영체제 자원은 같은 프로세스에 속한 다른 스레드와 공유한다.
- 하나의 프로세스에서 병행 수행되는 스레드는 주소 공간과 같은 자원을 서로 공유하는 반면, 하나의 컴퓨터에서 병행 수행되는 프로세스는 물리적 메모리, 디스크 등을 공유한다.
- 모든 프로그램을 다중 스레드를 사용하도록 바꾼다고 하여 성능이 향상되는 것은 아니다.
  - 좋은 예) 웹 서버(요청마다 다른 스레드가 처리), 웹 브라우저(그림과 텍스트를 별도 스레드가 처리)
  - 나쁜 예) 단일 프로세서 환경에서 계산 중심 프로그램

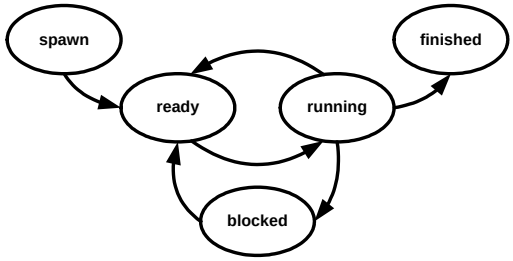


(a) 단일 스레드 (b) 다중 스레드

<그림 5.1> 단일과 다중 스레드 프로세스

5.1.1 동기

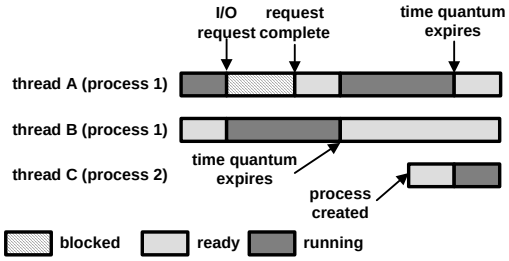
- 한 응용 프로그램에서 여러 가지 일을 동시에 수행할 수 있다. 각 일을 처리하기 위해 자식 프로세스를 생성하여 처리할 수 있지만 프로세스의 생성은 비용이 크며, 자원 공유가 쉽지 않다.
- 이 비용을 줄이고 자원 공유를 쉽게 하기 위해 도입된 개념이 스레드이다. 유닉스 구현의 경우에 스레드 생성 비용은 프로세스보다 대략 10배 정도 빠르다.



<그림 5.2> 스레드의 상태 변화

5.1.2 스레드의 특성

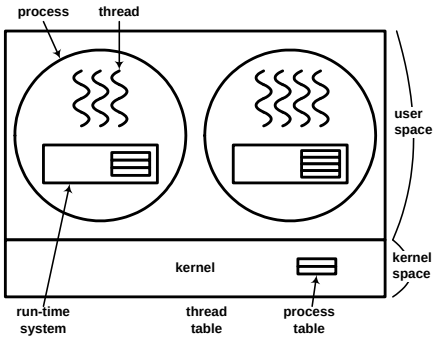
- 스레드도 프로세스와 마찬가지로 같은 상태 변화를 한다. 그림 5.2 참조.
- 프로세스는 부모와 자식 프로세스를 구분하지만 같은 프로세스 내에 스레드는 보통 이런 구분을 하지 않는다.



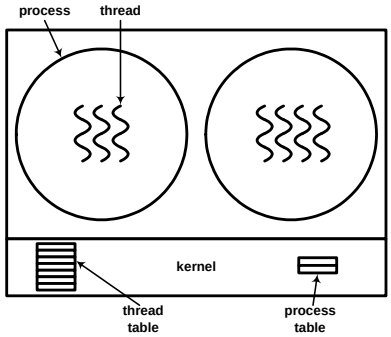
<그림 5.3> 단일 프로세서 시스템에서 다중 스레드의 수행

5.1.3 장점

- **응답성(responsiveness)**: 다중 스레드 프로그램은 그것의 일부가 블록되어도 나머지 부분은 계속 실행될 수 있다.
- **자원 공유(resource sharing)**: 기본적으로 같은 프로세스의 스레드는 메모리와 그 밖에 프로세스의 자원을 공유한다. 스레드는 같은 주소 공간을 공유하므로 스레드 간 통신이 더 용이하다.
- **경제성(economy)**: 프로세스마다 메모리와 자원을 할당하면 많은 오버헤드가 발생한다. 그러나



(a) 사용자 스레드



(b) 커널 스레드

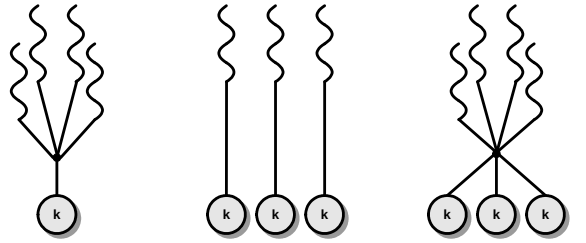
<그림 5.4> 사용자와 커널 스레드

프로세스 내의 스레드는 자원을 공유하므로 생성 비용이 저렴하며, 스레드 간에 문맥 전환이 용이하다.

- **다중 프로세서 구조의 활용**

5.1.4 사용자와 커널 스레드

- 스레드는 사용자 공간에 구현할 수 있고, 커널 공간에 구현할 수 있다. 또한 혼합하여 두 공간 모두에서 구현할 수 있다.
- **사용자 스레드(user thread)**
  - 사용자 공간에서 스레드 라이브러리를 통해 제공한다.
  - 이 라이브러리는 커널의 지원 없이 스레드의 생성, 스케줄링, 관리를 제공한다. 즉, 커널은 전혀 관여하지 않는다. 따라서 모드 변경이 필요없어 스레드의 생성과 관리를 빠르게 할 수 있다.
  - 프로세스마다 자체적 스케줄링 알고리즘을 가질 수 있다. 그러나 커널은 스레드 단위로 시분할을 하지 않는다.
  - 커널이 단일 스레드일 때 프로세스의 하나의 스레드가 블록되면 전체 프로세스가 블록된다. 또한 다중 프로세서의 장점을 이용할 수 없다.
    - 블록되지 않는 입출력(비동기식)을 사용하면 이것을 극복할 수 있다.



(a) 다대일 모델 (b) 일대일 모델 (b) 다대다 모델

<그림 5.5> 다중 스레드 모델

  - 운영체제가 스레드 기능을 지원하지 않으면 이것이 유일한 방법이다. 반대로 생각하면 운영체제 변경 없이 스레드를 지원할 수 있다.
  - 예) POSIX의 Pthread, Mach의 C-thread, Solaris의 UI-thread
- **커널 스레드(kernel thread)**
  - 운영체제가 직접 지원한다.
  - 커널이 스레드의 생성, 스케줄링, 관리를 담당한다.
  - 모드 변경이 필요하기 때문에 스레드의 생성과 관리가 사용자 스레드보다 느리다.
  - 하나의 스레드가 블록되어도 같은 프로세스에 있는 다른 스레드를 계속 실행할 수 있다.
  - 예) 윈도우즈 NT, 윈도우즈 2000, Solaris 2는 모두 커널 스레드를 지원한다.

5.2 다중 스레드 모델

- 보통 운영체제는 사용자 스레드와 커널 스레드를 모두 제공한다.
- 커널은 커널 스레드의 존재만 인식하며, 커널은 커널 스레드를 스케줄하여 실행한다.

5.2.1 다대일 모델

- 여러 사용자 스레드를 하나의 커널 스레드로 매핑한다.
- 장점: 스레드의 관리가 사용자 모드에서 이루어지므로 처리가 빠르다.
- 단점: 하나의 스레드가 블록되면 전체 프로세스가 블록된다. 또한 다중 프로세스에서 다중 스레드가 동시에 수행될 수 없다.

5.2.2 일대일 모델

- 각 사용자 스레드는 하나의 커널 스레드로 매핑된다.

- 장점: 하나의 스레드가 블록되어도 나머지 스레드는 계속 실행된다. 또한 다중 프로세서에서 동시에 수행될 수 있다.
- 단점: 비용이 비싸다. 따라서 보통 스레드의 수를 제한한다.
- 예) 윈도우즈 NT, 윈도우즈 2000, OS/2

5.2.3 다대다 모델

- 여러 사용자 스레드를 동일 수 또는 그 이하의 수의 커널 스레드로 매핑한다.
- 커널 스레드의 수는 응용 프로그램에 따라 또는 기계의 특성에 따라 제한될 수 있다.
- 장점: 비용이 일대일 모델보다 저렴하며, 다대일 모델과 달리 스레드의 동시 수행이 가능하다.
- 예) Solaris 2

5.3 스레드 쟁점

5.3.1 fork와 exec 시스템 호출

- 스레드 개념을 사용하면 기존 fork와 exec 시스템 호출을 변경해야 한다.
- 이를 위해 보통 유닉스는 두 종류의 fork 시스템 호출을 제공한다.
  - 종류 1. 프로세스의 하나의 스레드가 fork 시스템 호출을 하면 프로세스의 모든 스레드가 복사된다.
  - 종류 2. 프로세스의 하나의 스레드가 fork 시스템 호출을 하면 그 스레드만 복사된다.
- 프로세스의 하나의 스레드가 exec 시스템 호출을 하면 전체 프로세스가 새 프로그램으로 교체된다.
- 두 종류의 fork 중 어느 것을 선택하느냐는 응용 프로그램에 의해 결정된다. 만약 fork한 다음에 바로 exec를 호출할 것이면 종류 1의 fork을 사용하는 것은 낭비이다.

5.3.2 취소

- 스레드 취소(thread cancellation)는 실행 중인 스레드를 강제로 종료하는 것을 말한다. 이 때 취소할 스레드를 종종 목표 스레드(target thread)라 한다.
- 스레드 취소의 두 가지 형태
  - 비동기식 취소: 목표 스레드가 바로 종료된다.
  - 지연된 취소: 목표 스레드는 취소 명령을 받고 바로 종료하지 않고 주기적으로 자신이 종료되어야 하는지를 검사하여 적절한 시기(Pthread에서는 이 시기를 취소점(cancellation point)라 한다)에 종료한다.
- 스레드 취소의 어려움: 비동기식 취소에서는 스레드에게 할당된 자원이 모두 자동으로 해제되지 않을 수 있다.

5.3.3 신호 처리

- 유닉스에서는 신호를 이용하여 특정 사건이 일어났음을 프로세스에게 통보한다. 신호는 소프트웨어 인트럽트이다.
- 신호는 동기식 또는 비동기식으로 수신된다. 동기식 신호란 자신에 의해 생성된 신호를 말하며, 이 신호는 그것을 생성한 프로세스에 전달된다. 비동기식 신호는 외부에 의해 생성된 신호를 말하며, 그것의 생성은 보통 예측할 수 없다.
- 신호 처리 과정
  - 단계 1. 특정 사건의 발생에 의해 신호가 생성된다.
  - 단계 2. 생성된 신호는 프로세스에게 전달된다.
  - 단계 3. 프로세스는 신호를 처리한다.
- 동기식 신호의 예: 불법적 메모리 접근
  - 신호 발생의 원인을 제공한 프로세스에게 신호가 전달된다.
- 비동기식 신호의 예: CTRL+C
  - 신호가 수행중인 프로세스의 외부적 요인에 의해 발생한 경우
- 신호 처리기의 종류
  - 기본 신호 처리기: 모든 신호마다 그것을 처리하는 기본 신호 처리기가 있다. 이 처리기는 커널이 수행한다.
  - 사용자 정의 신호 처리기: 사용자가 신호 처리기를 정의하면 기본 신호 처리기 대신에 이 처리기가 신호를 처리한다.
- 스레드에서 신호 처리는 어떻게?
  - 방법 1. 신호에 해당하는 스레드에게만 전달
  - 방법 2. 프로세스의 모든 스레드에게 전달
  - 방법 3. 프로세스의 일부 스레드에게만 전달
  - 방법 4. 프로세스의 모든 신호를 수신하는 특정 스레드를 두어 처리
- 동기식 신호는 방법 1이 적합하며, CTRL+C와 같은 신호는 방법 2가 적합하다.
- 보통 UNIX에서는 스레드가 수신할 신호와 무시할 신호를 지정할 수 있도록 한다. 일반적으로 신호가 발생되면 그 신호를 무시하지 않는 첫번째로 발견된 스레드에게 전달된다.
- Solaris는 방법 4를 지원한다.

```
#include <pthread.h>
#include <stdio.h>
int sum; /* 모든 스레드가 공유하는 데이터 */
void *runner(void *param);
main(int argc, char *argv[]){
        pthread_t tid; /* 스레드 식별자 */
        pthread_attr_t attr; /* 스레드의 속성 */
        ...
        /* 기본 속성 가져오기 */
        pthread_attr_init(&attr);
        /* 스레드 생성 */
        pthread_create(&tid,&attr,runner,argv[1]);
        /* 생성한 스레드가 종료될 때까지 대기 */
        pthread_join(tid,NULL);
        ...
}
 void *runner(void *param){

        ...
        pthread_exit(0);
}
```

<그림 5.6> Pthread API 사용의 예

5.3.4 스레드 풀

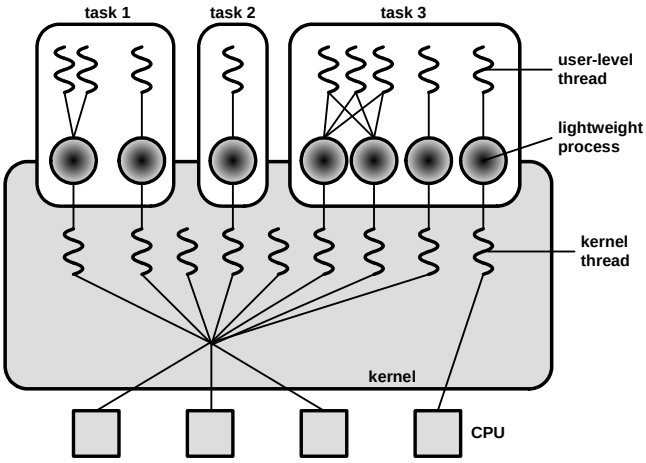
- 스레드 풀(thread pool)은 프로세스가 생성할 수 있는 스레드의 수를 적절하게 관리하기 위해 사용되는 메커니즘이다.
- 스레드를 필요할 때마다 생성하지 않고 프로세스가 시작될 때 적정한 수의 스레드를 생성하여 스레드 풀에 대기하도록 한다. 스레드가 필요하면 이 풀에 있는 하나 스레드를 활성화하여 실행되며, 실행이 끝나면 다시 스레드 풀에서 대기한다. 만약 스레드 풀에 더 이상 활성화할 스레드가 없으면 처리 요청은 있을 때까지 대기하게 된다.
- 스레드 풀의 장점
  - 보다 빠르게 처리할 수 있다.
  - 스레드 수를 제한하는 효과적인 방법이다.

5.3.5 스레드 전용 데이터

- 스레드는 보통 같은 프로세스에 속한 스레드와 데이터를 공유한다.
- 스레드가 자신만의 데이터가 필요할 수 있다. 이런 데이터를 스레드 전용 데이터(thread-specific data)라 하며, 대부분 스레드 라이브러리는 이를 지원한다.

5.4 Pthread

- Pthread는 thread 생성과 관리를 위한 API를 정의하는 POSIX 표준(IEEE 1003.1c)이다.
- 이 표준은 API의 행위만 정의하고 있으며, 이것의 구현은 운영체제의 따라 다르다.



<그림 5.7> 솔라리스 2 스레드

- Pthread API의 사용의 예: 그림 5.6
  - 이 프로그램이 실행되면 초기에는 하나의 스레드를 이용하여 수행된다.
  - pthread_create 호출을 이용하여 스레드를 생성하면 두 개의 스레드가 병행으로 수행된다.
  - pthread_join 호출을 이용하여 생성한 스레드의 종료를 기다릴 수 있다.
  - 스레드는 pthread_exit 호출을 이용하여 종료한다.

5.5 솔라리스 2 스레드

- 솔라리스 2는 사용자 스레드와 커널 스레드 사이에 LWP(LightWeight Process)라고 하는 중간 스레드를 사용한다.
- 솔라리스 2는 다대다 모델을 사용한다.
- 하나의 프로세스는 LWP 풀을 가진다. 프로세스의 각 사용자 스레드는 LWP를 통해 수행되며, 각 LWP는 하나의 커널 스레드와 연관되어 있다.
- 사용자 스레드는 영구적으로 하나의 LWP와 연관되어 실행될 수 있고, 그렇지 않을 수 있다. 전자와 같은 스레드를 **바인드된 스레드(bound thread)**라 하고 후자를 **바인드되지 않은 스레드(unbound thread)**라 한다.
- 기본적으로 사용자 스레드는 바인드되지 않는다.
- LWP 풀에 있는 LWP의 수는 동적으로 변화하며, 오래 동안 사용되지 않는 LWP는 시스템에서 제거한다.

5.6 윈도우즈 2000 스레드

- 윈도우즈 2000은 일대일 다중 스레드 모델을 사용한다.

- 윈도우즈 2000은 기본적으로 일대일 모델이지만 fiber 라이브러리를 통해 다대다 모델의 기능도 제공한다.

5.7 리눅스 스레드

- 리눅스는 기본적으로 스레드를 지원하지는 않는다. 그러나 스레드와 유사한 형태의 프로세스를 생성하여 사용할 수 있도록 한다.
- 기존 fork 시스템 호출 대신에 clone 시스템 호출을 사용하면 주소 공간을 부모 프로세스와 공유할 수 있다.

5.8 자바 스레드

- 자바 스레드는 자바 가상 기계에서 처리하므로 자바 스레드를 사용자 스레드 또는 커널 스레드로 분류하는 것이 어렵다.
- 스레드의 생성
  - Thread 클래스를 상속받는 새 클래스를 정의하여 사용
  - 실제 스레드의 생성은 이 클래스의 start 메소드를 호출할 때 이루어진다.
  - start 메소드가 호출되면 다음과 같은 일이 수행된다.
    - 새 메모리를 할당하고 새 스레드를 초기화한다.
    - 클래스의 run 메소드를 호출한다.
- 자바 스레드는 JVM에 의해 커널 스레드로 매핑된다. 윈도우즈용 JVM은 일대일 모델을 사용하며, 솔라리스는 다대다 모델을 사용한다.