유저 레벨 스레드와
커널 레벨 스레드

커널 스레드와 유저 스레드

커널 스레드

- 커널 스레드는 프로세서에 의해 스케줄링되고 실행될 수 있는 실제 스레드이다.
- 커널 레벨 스레드는 운영체제에서 직접 처리하고 커널에서 스레드 관리를 수행한다.

유저 스레드

- 유저 스레드는 유저가 생성한 스레드이며 커널은 유저 스레드의 존재를 모른다.
- 유저 스레드는 생성한 프로세스의 메모리 공간 안에서만 존재하며, 커널의 개입없이 스레드 라이브러리에 의해 관리된다.
- (스레드 라이브러리는 스레드 생성과 소멸, 그리고 스레드 간에 메세지나 데이터 전달을 위한 코드가 포함되어 있다.)

Light Weight Process(LWP)

- 다대다 모델이나 two-level 모델에서 프로세스에 할당된 적절한 수의 커널 스레드를 유지하기 위해서 커널과 유저 레벨 스레드의 매니저와 통신할 수 있는 방법이 필요하다. 이 때 커널 스레드와 유저 스레드의 징검다리 역할을 하는 계층이다.
- 해당 계층이 없는 모델에서 “Light Weight Process”는 일반적으로 커널 스레드를 의미하고, “Thread”는 유저 스레드를 의미한다.

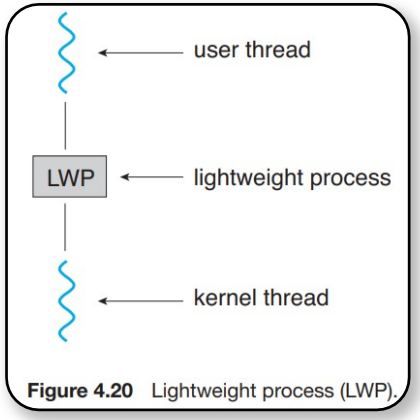
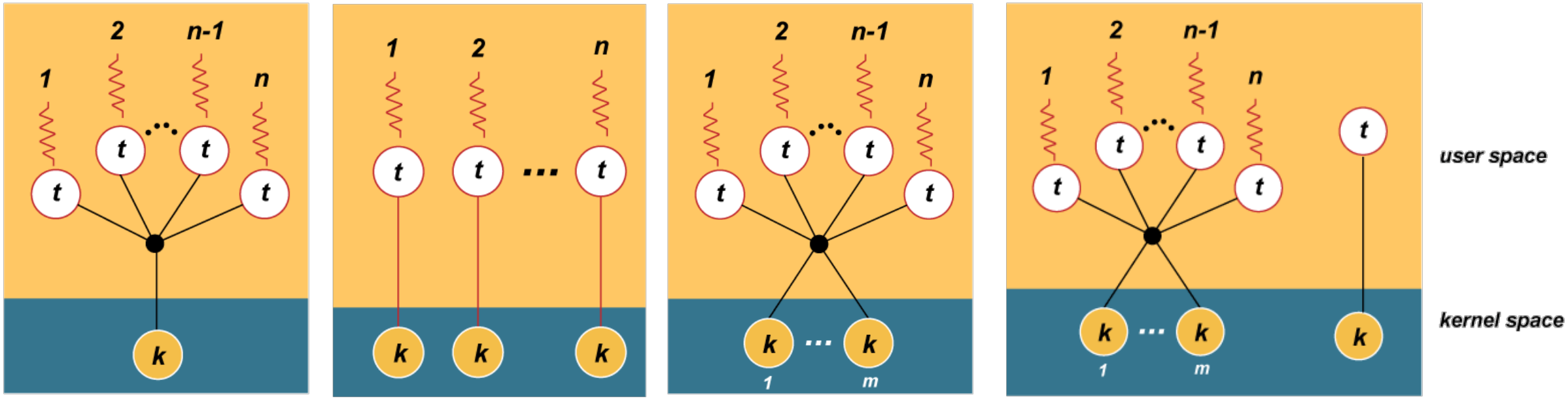


Figure 4.20 Lightweight process (LWP).

커널에서 유저 스레드의 존재를 모른다면, 어떻게 관리될 수 있을까?

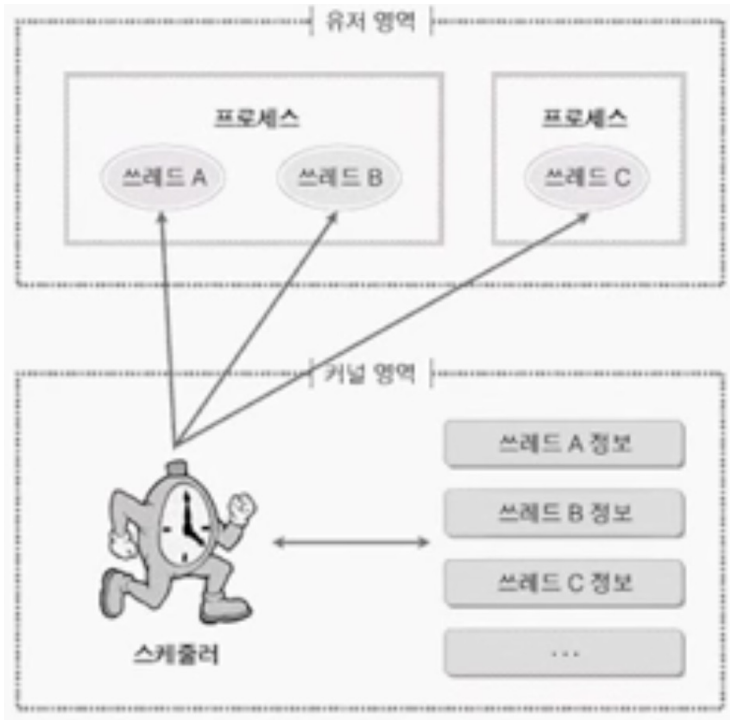
커널 스레드와 유저 스레드의 매핑과 스레드 모델

- 커널 스레드는 프로세서에 의해 스케줄링되고 실행된다. 이는 커널에서 관리가 가능하다는 의미이다.
- 유저 스레드는 커널이 생성에 대한 사실을 모르기 때문에 자체적으로 실행될 수 없다.
- 따라서 유저 스레드를 실행시키기 위해서 사용자 프로그램은 스케줄러가 유저 레벨 스레드를 가져와 **커널 레벨 스레드에서 실행되도록 매핑되어야한다.** 이 때 이 매핑이 수행되는 방식을 “스레드 모델"이라고 한다.
- 스레드 모델은 몇 개의 유저 스레드와 몇 개의 커널 스레드가 매핑되느냐에 따라 1:1(일대일), M:1(다대일), M:N(다대다), two-level 모델로 나뉠 수 있다.

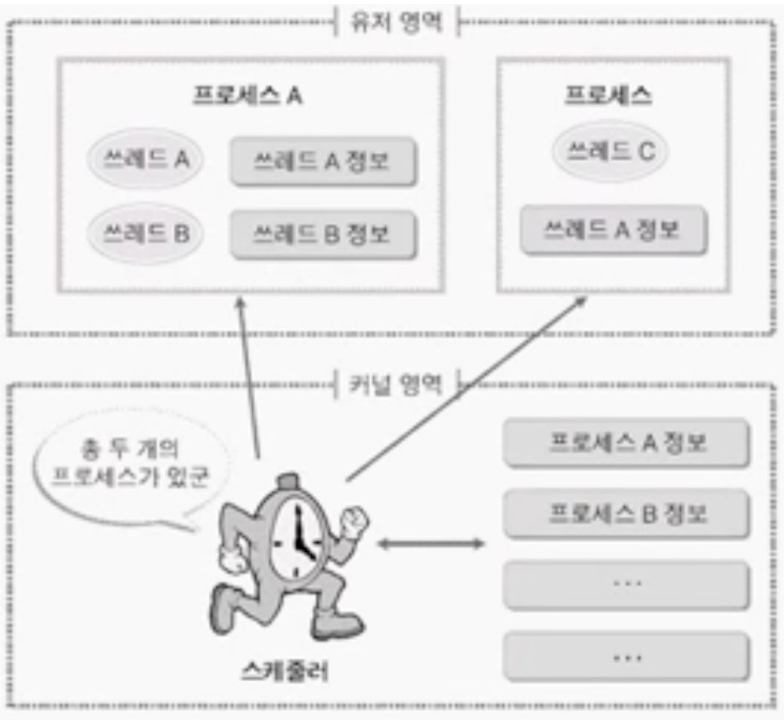


유저 레벨 스레드와 커널 레벨 스레드

- 스레드를 지원하는 운영체제가 있고, 지원하지 않는 운영체제가 있다.
- 스레드를 지원한다는 의미는 커널에서 스레드의 생성과 소멸, 스케줄링을 관리한다는 것으로 이해하면 된다.
- 지원하지 않는 운영체제에서도 라이브러리를 사용하여 스레드를 사용할 수 있다.
- 스레드 라이브러리를 통해 만들어서 사용하는 스레드를 유저 레벨 스레드라고 하고, 커널에서 제공하는 스레드를 커널 레벨 스레드라고 한다.



커널 레벨 스레드



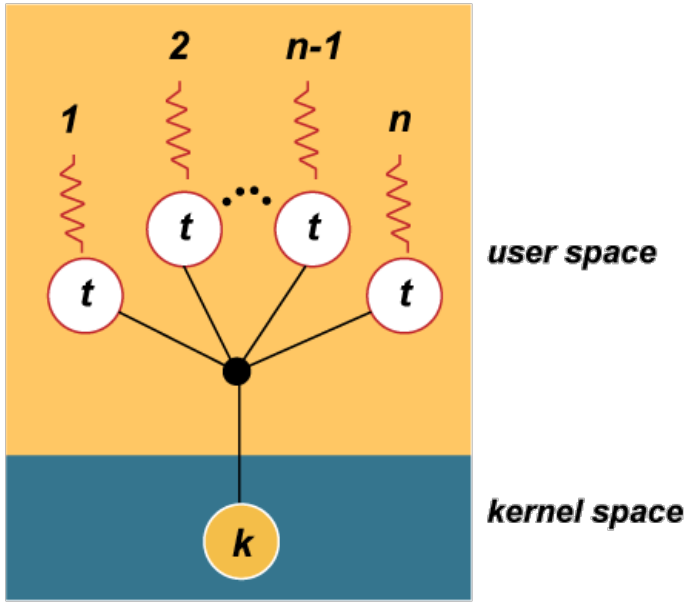
유저 레벨 스레드

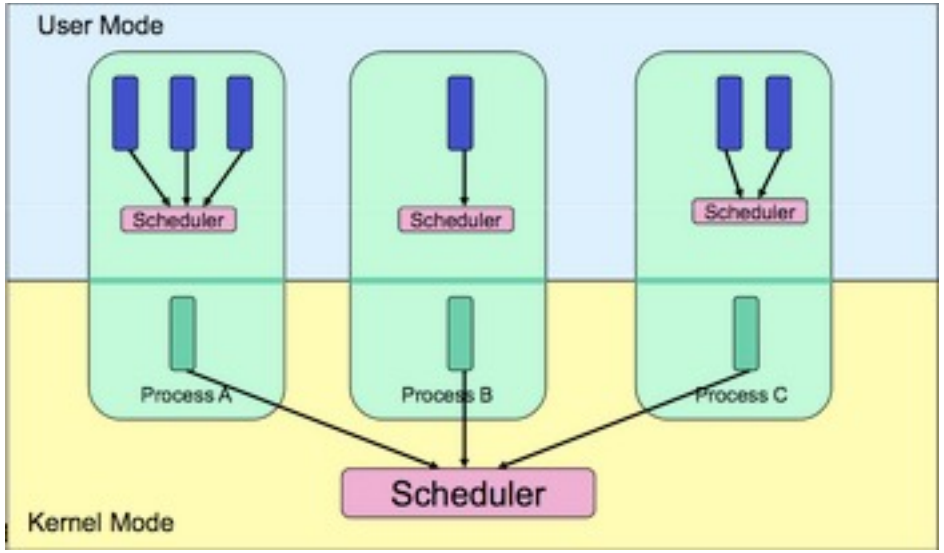
유저 레벨 스레드	커널 레벨 스레드
1. 높은 이식성: 기본 커널을 변경할 필요가 없으므로 모든 운영체제에 적용할 수 있다. 2. 오버헤드 감소: 커널로 전환하지 않고 유저 영역에서 스레드 교환이 가능하다. 따라서 유저 모드와 커널 모드 전환에 따른 오버헤드가 감소한다. 3. 스케줄링의 유연성: 스레드 라이브러리에서 스레드 스케줄링을 제어하기 때문에 스케줄링이 응용 프로그램에 맞게 적절하게 구성된다.	1. 시스템의 동시성 지원: 커널에 의해 직접적으로 스케줄링되고 실행되기 때문에, 하나의 스레드가 대기에 빠져도 같은 프로세스 내의 다른 스레드는 다른 커널 스레드에 매핑되어 있기 때문에 영향을 받지 않는다. 2. 성능 향상: 위와 마찬가지의 이유로 한 스레드가 대기에 빠져도 다른 스레드를 실행시킬 수 있기 때문에 성능을 향상시킬 수 있다.
1. 시스템의 동시성 지원 불가: 프로세스 내에서는 한 번에 하나의 스레드만 실행이 가능하다. 만약 실행 중이던 스레드가 대기 상태에 빠지면 같은 프로세스 내의 모든 스레드가 대기 상태가 된다. 2. 시스템 규모 확장 제약: 커널이 프로세스 내부의 다중 스레드를 프로세스로 하나로 관리한다. 따라서 다중 처리 환경이라도 여러 프로세서에서 분산 처리할 수 없다. 3. 스레드 간 보호가 어려움: 커널의 보호 기법을 사용할 수 없기 때문에, 스레드 라이브러리에서 스레드 간 보호를 제공해야 프로세스 수준에서 보호된다.	1. 오버헤드 증가: 같은 프로세스 내에 다른 스레드로 전환을 하는 경우에도 커널 모드와 유저 모드간 전환에 따른 문맥 교환으로 인해 오버헤드가 증가한다. 2. 낮은 이식성: 시스템을 변경할 때는 제공된 스레드 API를 사용하여 프로그램을 수정해야한다.

이제부턴 스레드 모델에 대해 알아보자!

다대일 모델(유저 레벨 스레드)

- 모든 유저 스레드는 하나의 커널 스레드로 매핑된다. 프로세스는 하나의 커널 스레드와만 연관되어 있기 때문에 한 번에 하나의 유저 스레드만 실행시킬 수 있다.
- 라이브러리 스케줄러는 이 매핑을 처리하고, 사용자 스레드의 프로그래밍 기능을 완전히 처리한다.
- 이 모델은 어떤 시스템에나 적용될 수 있으며, 특히 전통적인 싱글 스레드 시스템에서 사용된다.



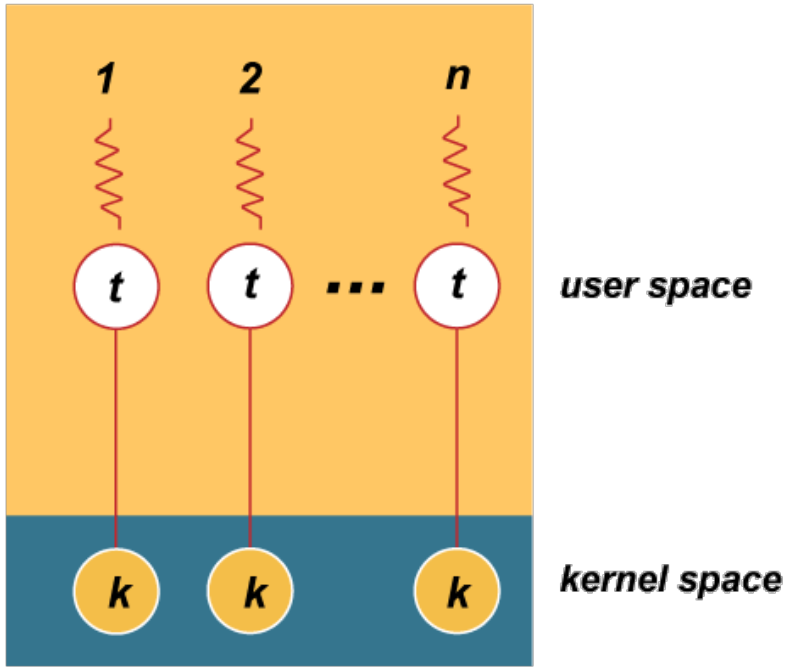


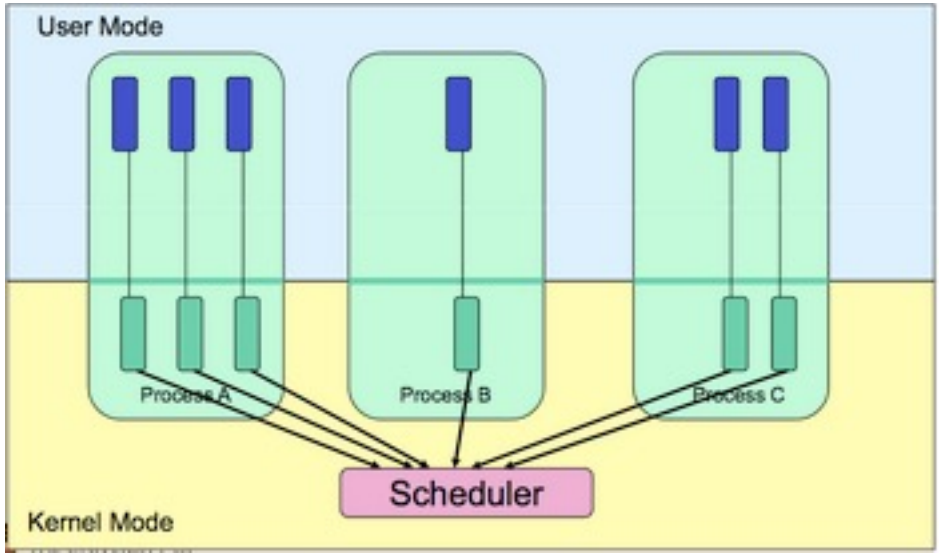
(좌) 간략화 이미지 출처: https://www.javatpoint.com/why-must-user-threads-be-mapped-to-kernel-thread

(우) 이미지 출처: https://elsoc.fandom.com/wiki/Processes_and_Threads

일대일 모델(커널레벨 스레드)

- 각각의 유저 스레드는 하나의 커널 스레드에 매핑된다. 모든 유저 레벨 스레드는 독립적인 커널 레벨 스레드에서 실행된다.
- 대부분의 유저 스레드의 프로그래밍 기능은 커널 스레드에 의해 직접 처리된다. 이 모델은 디폴트 모델이다.



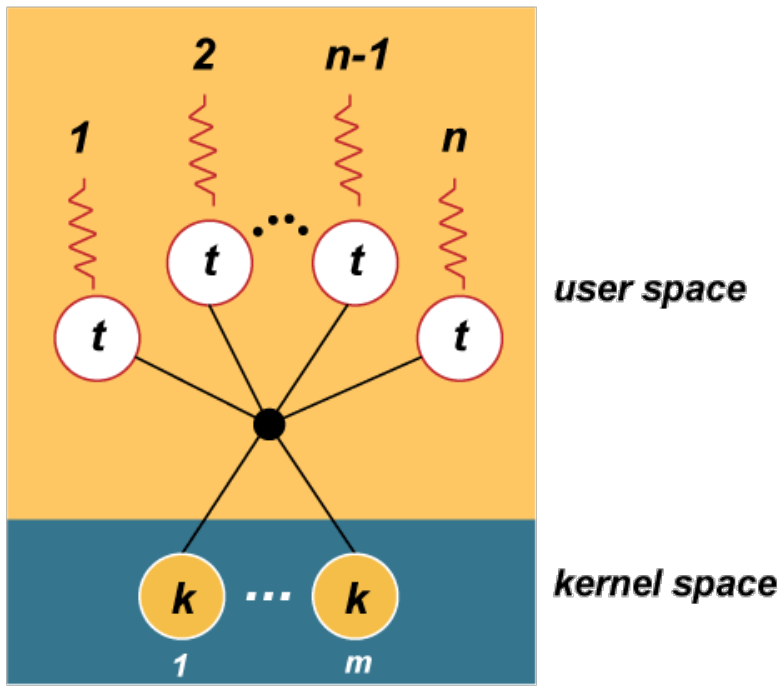


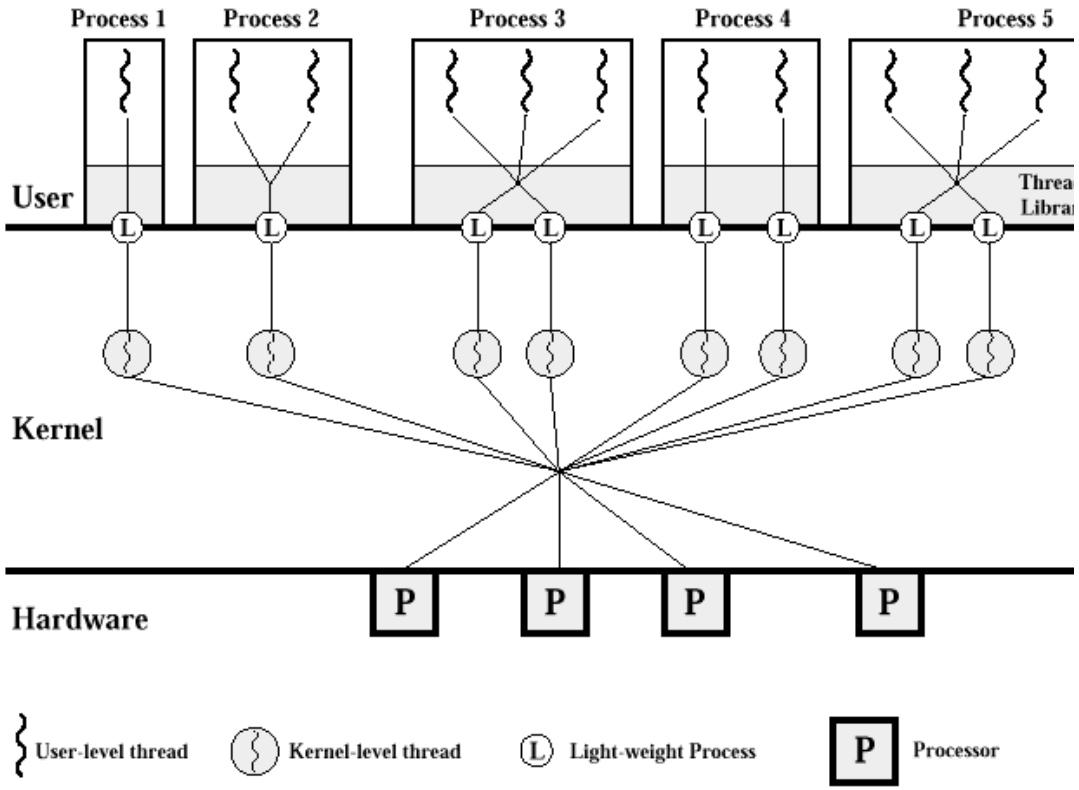
(좌) 간략화 이미지 출처: https://www.javatpoint.com/why-must-user-threads-be-mapped-to-kernel-thread

(우) 이미지 출처: https://elsoc.fandom.com/wiki/Processes_and_Threads

다대다 모델(혼합형 스레드)

- 모든 유저 스레드는 커널의 스레드 풀과 매핑된다.
- 가장 효율적이지만 복잡한 모델이다.



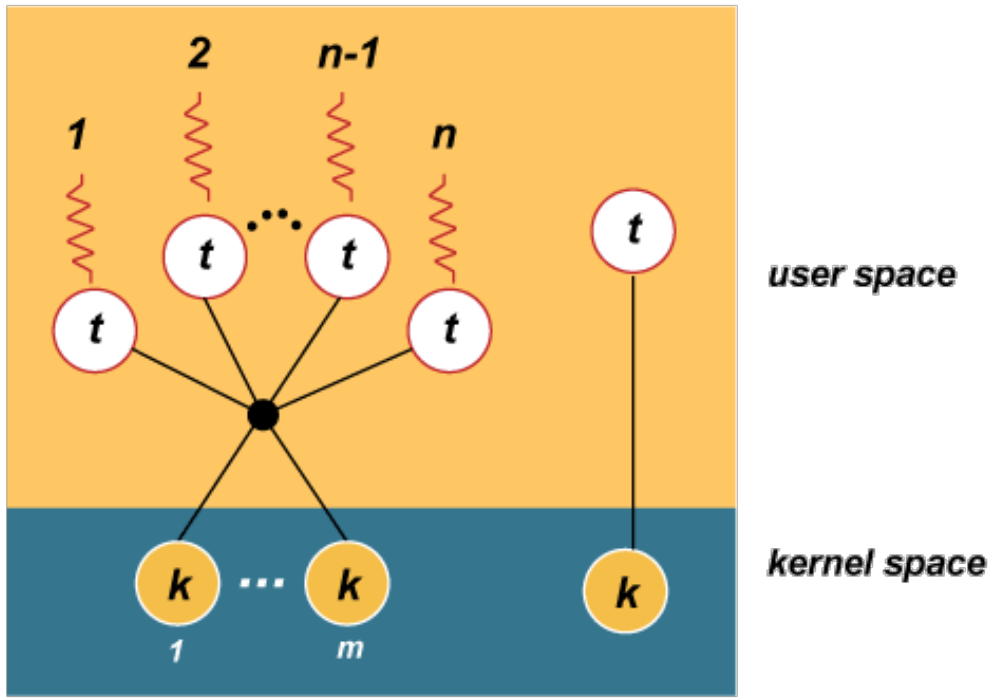


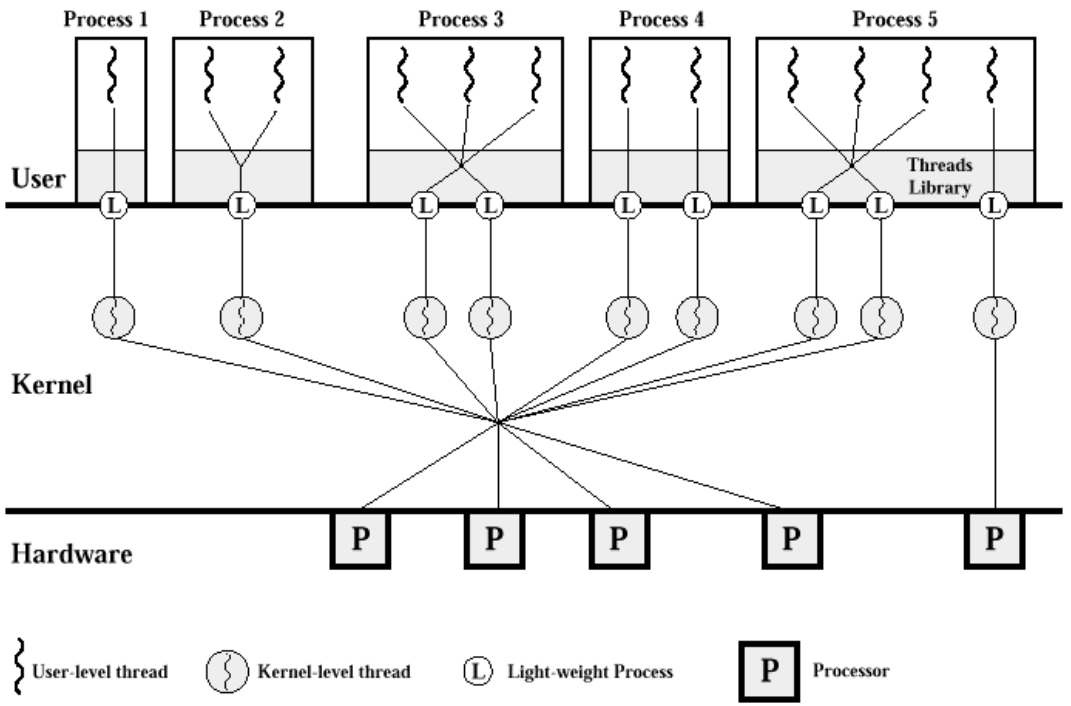
(좌) 간략화 이미지 출처: https://www.javatpoint.com/why-must-user-threads-be-mapped-to-kernel-thread

(우) 이미지 출처: https://elsoc.fandom.com/wiki/Processes_and_Threads

two-level 모델(혼합형 스레드)

- 다대다 모델과 유사하지만, 특정 유저 스레드를 한 개의 커널 스레드에 바인딩하는 것을 허용하는 모델





(좌) 간략화 이미지 출처: https://www.javatpoint.com/why-must-user-threads-be-mapped-to-kernel-thread

(우) 이미지 출처: https://elsoc.fandom.com/wiki/Processes_and_Threads

리눅스에서의 스레드
(feat. 2주차 프로세스 스케줄링)

리눅스 커널 스케줄링 (2.6.23 이전)

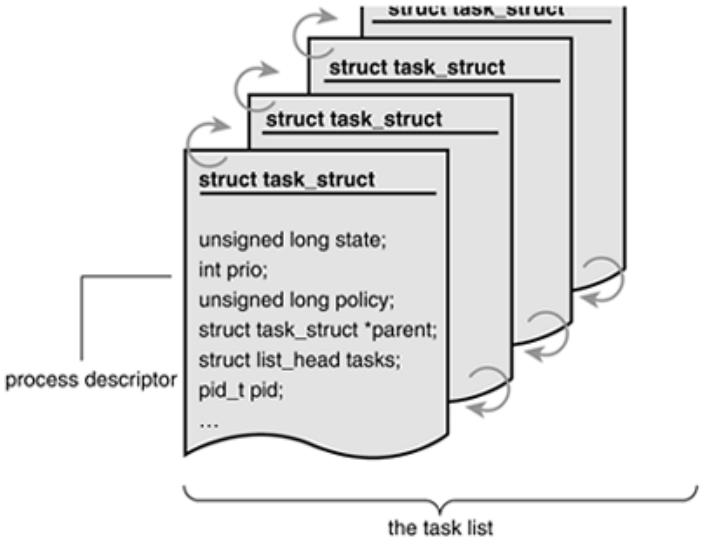
- 리눅스에서는 프로세스와 스레드를 동일한 것으로 간주하여 프로세스 및 스레드 스케줄링을 통합적으로 처리한다.
- 리눅스에서는 run queue에서 다음에 실행할 태스크를 선택하는데, run queue는 active array와 expired array로 구성된다.
- 주어진 time slice를 다 사용하면 expired array로 빠지게 되고, 만약 실행중 다른 프로세스에게 뺏겨서 time slice를 다 사용하지 못했다면 activa array에 다시 추가한다.
- active array에 있는 프로세스를 모두 수행하면 expired array가 active array가 된다. (expired array와 active array 교체)
- 따라서 리눅스에서 프로세스가 CPU를 차지하기 위해서는 남은 time slice가 0보다 크면서 우선순위가 높아야한다.

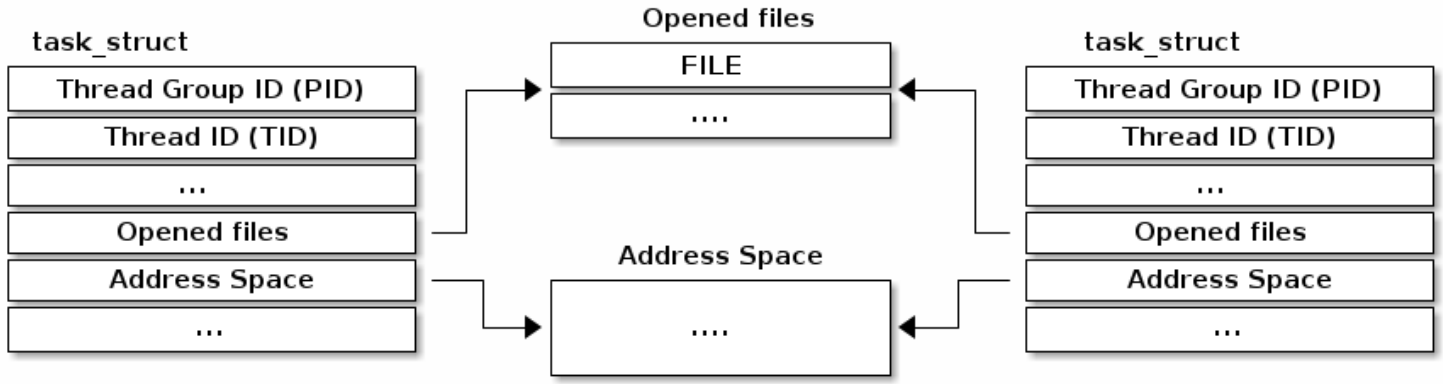
버전	사용한 알고리즘
1.2	• 라운드 로빈 스케줄링 정책에 따라 작동하는 순환형 큐 사용 • 단순하며 빠르지만, 다수의 프로세서나 하이퍼 스레딩이 포함된 대형 아키텍처를 전혀 고려하지 않는다.
2.2	• 스케줄링 클래스라는 개념이 채택되면서 실시간 작업, 우선 순위가 없는 작업 및 비실시간 작업에 대한 스케줄링 정책을 사용할 수 있다. • SMP(Symmetric Multiprocessing)에 대한 지원도 포함되어 있다.
2.4	• O(n) Scheduler 사용 • 시간은 에포크(Epoch) 단위로 나누며 모든 작업은 해당 시간 조각동안 실행할 수 있다. 작업이 주어진 time slice를 다 사용하지 못하고 끝나면 남은 시간의 절반을 추가하여 다음 에포크에서 더 길게 실행할 수 있도록 했다. • 우선순위로 큐가 나눠져있지 않기 때문에, 다음에 실행할 태스크를 찾기 위해서는 대기하고 있는 모든 프로세스의 PCB를 확인해봐야 했기 때문에, O(n)만큼의 시간복잡도를 가진다.
2.6 ~ 2.6.22	• O(1) Scheduler 사용 • 우선 순위 레벨을 두고 우선 순위 레벨별로 두 개의 실행 큐를 사용했다. 따라서 다음 작업을 가져오기 위해서는 우선 순위에 해당하는 큐에 가서 작업을 가져오기만 하면 된다.

서의 스레드
(feat. 2주차 프로세스 스케줄링)

리눅스에서 태스크와 스레드 그룹

- 리눅스는 프로세스와 스레드, 커널 내부에서 사용하는 커널 스레드 모두 태스크로 처리한다.
- PCB와 TCB로 나뉘어져 있지 않고, “task_struct” 를 통해 태스크를 관리한다. 이는 “프로세스 서술자”라고도 부른다.
- 각 태스크를 구분하기 위해 고유한 아이디가 필요하고, 이를 Task ID라는 이름으로 부른다.
- 그런데 POSIX 표준에선 하나의 프로세스 내 스레드는 같은 PID(Process ID)를 가져야 한다고 명시되어 있다.
- 따라서 리눅스를 이를 위해 **스레드 그룹**이라는 개념을 도입했다. 스레드 그룹 또한 고유한 아이디를 가지는데 이를 TGID(Thread Group ID)라고 한다. 스레드 그룹은 경량 프로세스 (Light Weight Process, LWP)의 집합이다.
- 스레드 그룹의 리더는 첫 번째로 생성된 경량 프로세스이다.

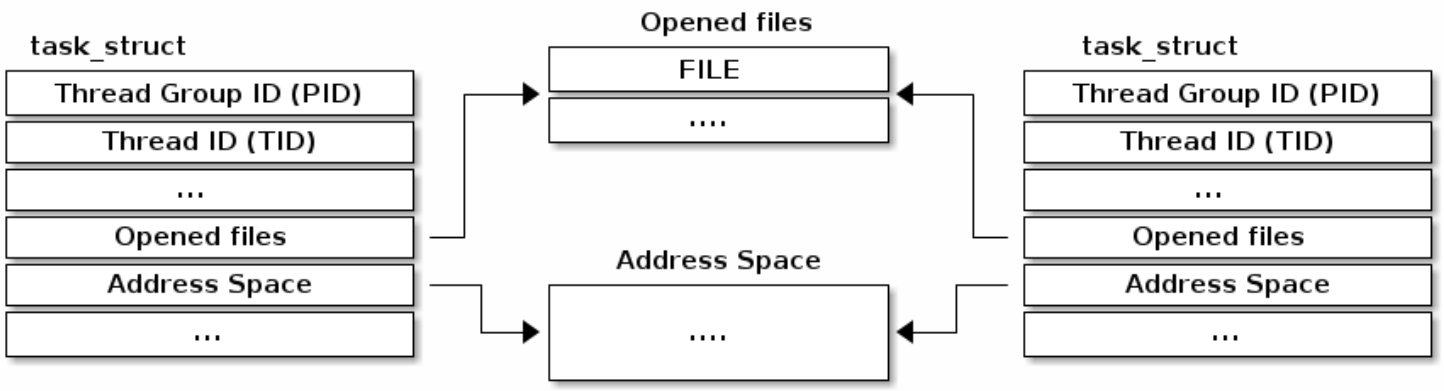


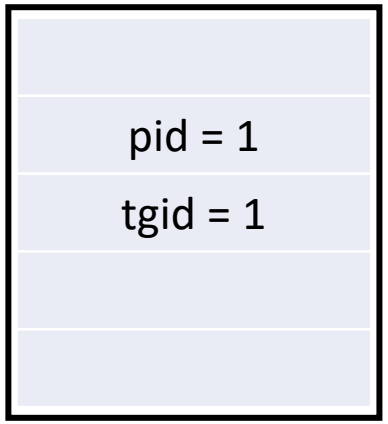


(좌) task_struct 이미지 출처: http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch03lev1sec1.html

(우) 자원을 공유하는 task_struct 이미지 출처: https://linux-kernel-labs.github.io/refs/heads/master/lectures/processes.html

리눅스에서 태스크와 스레드 그룹

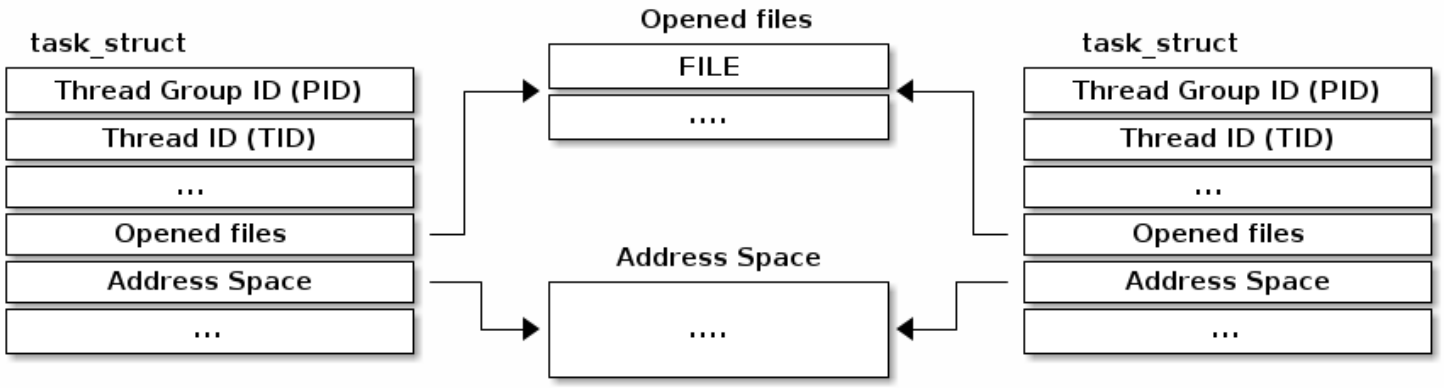


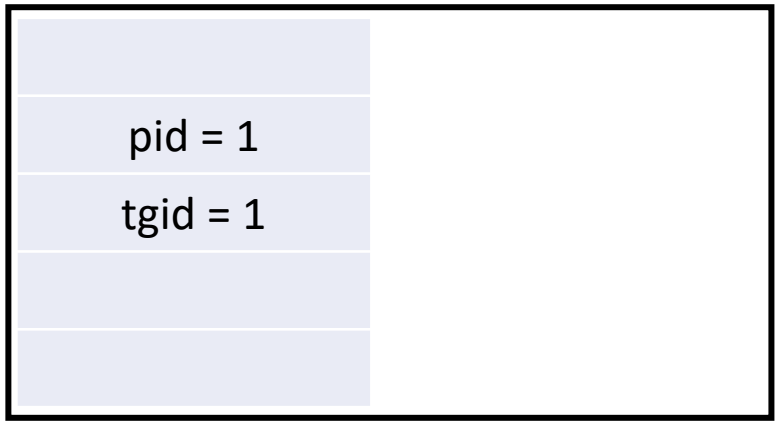


(좌) task_struct 이미지 출처: http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch03lev1sec1.html

(우) 자원을 공유하는 task_struct 이미지 출처: https://linux-kernel-labs.github.io/refs/heads/master/lectures/processes.html

리눅스에서 태스크와 스레드 그룹

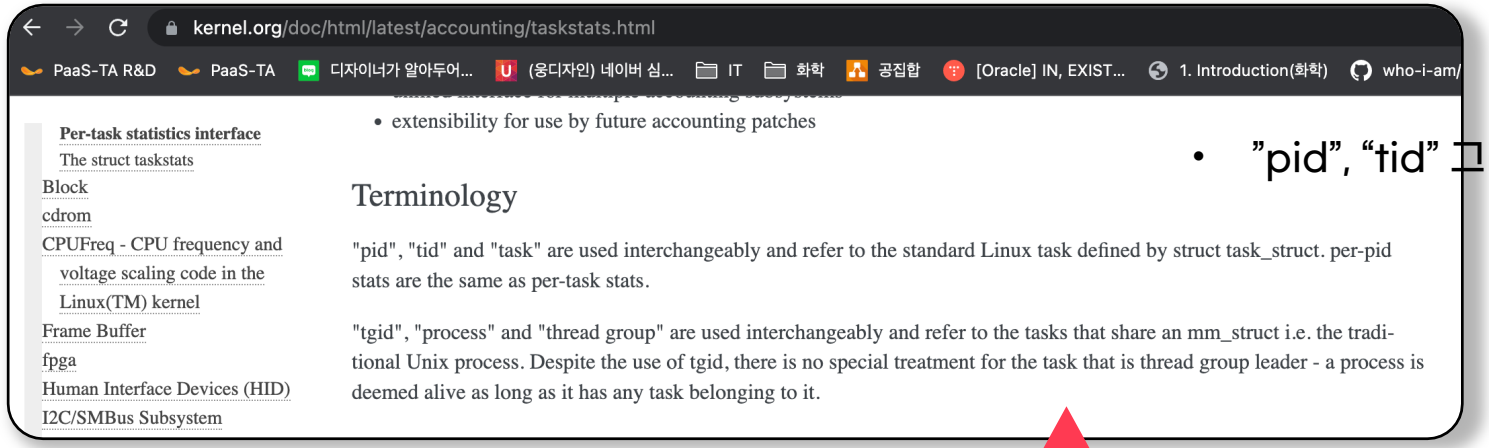




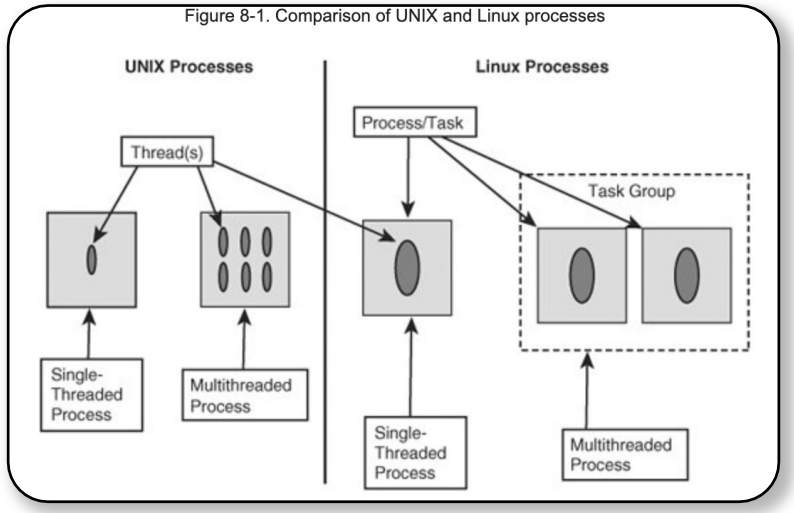
(좌) task_struct 이미지 출처: http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch03lev1sec1.html

(우) 자원을 공유하는 task_struct 이미지 출처: https://linux-kernel-labs.github.io/refs/heads/master/lectures/processes.html

스레드 그룹



- ”pid”, “tid” 그리고 “task”는 상호교환적으로 사용 가능하다.
- ”tgid”, “process” 그리고 “thread group”은 상호교환적으로 사용 가능하다.



(상) 이미지 출처: https://www.kernel.org/doc/html/latest/accounting/taskstats.html

(하) 이미지 출처: https://flylib.com/books/en/1.499.1.54/1/

퀴즈

1. 스레드 모델에 해당하지 않는 모델은?

1️⃣ 1:1 model 2️⃣ 1:M model 3️⃣ N:M model 4️⃣ 3-level model

2. 유저 레벨의 스레드와 커널 레벨의 스레드 중간에 있는 계층으로 둘의 통신을 돕는 개체는 무엇인가요?

3. 리눅스에서 스케줄링 대상은 무엇일까요?

1️⃣ 프로세스 2️⃣ 스레드 3️⃣ 커널 레벨 스레드 4️⃣ 모두 가능

퀴즈

1. 퀴즈 1

1️⃣ 스레드는 자원을 공유하고 있고, 프로세스에 비해 가볍기 때문에 Lightweight process라고도 한다.

2️⃣ Process의 상태에는 New, Ready, Runnable, Run, Wait, Terminated가 있다.

→ Runnable은 Java의 Thread 에서 지원하는 Thread 상태 종류 중 하나이다.

3️⃣ 프로세스는 운영체제로부터 자원을 할당받는 단위이다.

4️⃣ 프로세스가 실행(Run)이 되다가 주어진 시간이 끝나면 대기(Wait) 상태로 전환된다.

→ 대기가 아니라 준비큐에 삽입되고 준비(Ready) 상태로 전환된다.

2. 퀴즈 2

스택은 함수와 관련된 값과 돌아갈 위치가 저장되어 있는 메모리 공간이기 때문에, 독립적인 실행 흐름을 추가하기 위해 스택을 독립적으로 할당한다.

3. 퀴즈 3

1️⃣ 프로세스 상태 2️⃣ Thread control block리스트 3️⃣ 형제 프로세스 아이디 4️⃣ 프로세스 우선순위

참고

1. 커널 스레드와 유저 스레드
https://tutorialwing.com/user-level-thread-and-kernel-level-thread-with-example/
https://www.javatpoint.com/why-must-user-threads-be-mapped-to-kernel-thread

2. 리눅스 스레드 그룹
https://velog.io/@mythos/%EC%BB%A4%EB%84%90-%EC%8A%A4%ED%84%B0%EB%94%94iamroot-18%EA%B8%B0-2%EC%A3%BC%EC%B0%A8-%EB%82%B4%EC%9A%A9-%EC%A0%95%EB%A6%AC-1#pid-tid-%EA%B7%B8%EB%A6%AC%EA%B3%A0-tgid

3. 커널 레벨 스레드와 유저 레벨 스레드
https://www.youtube.com/watch?v=sOt80Kw0Ols&list=RDCMUC31Gc42xzclOOi5Gp1xlpZw&start_radio=1&rv=sOt80Kw0Ols&t=128

4. Light Weight Procss(LWP), Scheduler Activations
https://velog.io/@dandb3/Operating-System-Thread-3