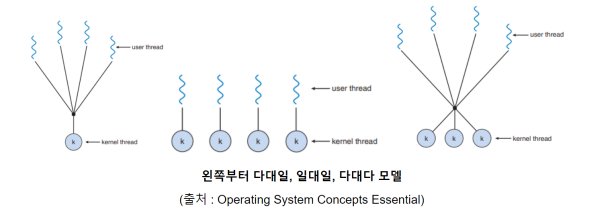
**Thread**

1. **Thread**- 프로세스가 할당 받은 자원을 이용하는 실행 단위  
   - 프로세스는 최소 한 개 이상의 Thread를 가진다.  
   - 프로세스의 메모리 공간을 공유한다.(code, data, heap를 쓰레드가 서로 공유)  
   - 프로세스의 자원공유(한 쓰레드 안에서 file, io 등을 공유)  
   - 비 공유 : 개별적인 PC(program counter), SP(stack pointer), Registers, Stack  
   -> CPU가 p1에 있는 각각의 쓰레드가 가지고 있는 PC값은 다르다.  
   -> 그래서 스위칭 될 때마다 각자가 가리키고 있는 값이 다르니 공유하지 않는다.  
   -> 스택은 리턴값 혹은 파라미터 값을 저장하기 위해 사용, 서로간 호출 메서드 다름  
   -> 즉, 하나의 프로그램은 **코드, 데이터, 스택으로 구성 된다.  
   -> 쓰레드는 코드, 데이터, 힙을 공유하지만 스택은 따로 공유하지 않는다.**
2. **멀티 쓰레드 모델**- 유저 레벨 스레드(User-level thread) : 유저 라이브러리가 제공하는 객체  
   - 커널 레벨 스레드(Kernel-level thread) : 운영체제 커널이 지원하고 관리하는 객체  
   - 유저 레벨 스레드가 커널 레벨 스레드보다 생성과 관리가 빠르다.  
   -> 커널 간 간섭이 없기 때문  
   - 다대일(many to one) : 하나의 커널 스레드에 여러 유저 스레드가 연결  
   - 일대일(one to one) : 각 유저 스레드가 각 하나의 커널 스레드와 대응  
   - 다대다(many to many) : 많은 유저 스레드가 적거나 같은 수의 커널 스레드로 다중화

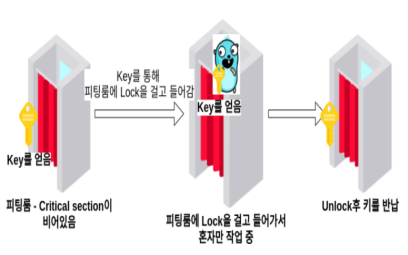


1. **프로세스와의 차이점**

**테이블이(가) 표시된 사진

자동 생성된 설명**

1. **멀티프로세스 vs 멀티스레드  
   - 컨텍스트 전환(Context Switch)**  
   : 다른 프로세스의 수행을 위해 실행되고 있는 프로세스의 현재 상태를 저장하는 절차  
   : 실행되고 있던 프로세스의 내용을 프로세스 제어 블록에 저장  
   : 다른 프로세스의 컨텍스트를 그 프로세스의 제어 블록으로부터 불러와 실행  
   : 컨텍스트 전환이 일어나는 동안 아무런 작업이 수행되지 않아 순수한 오버헤드이다.  
   : 실제 컨텍스트 전환되는 단위는 process 단위가 아니라 thread 단위이다.  
   **- 멀티프로세스와 멀티스레드**  
   : 프로그램을 구현할 때 연산량 증가, 모듈화, 편의성 등의 이유로 하나의 작업을 여러 개의 프로세스로 구성하거나 여러 개의 스레드로 구성한다.  
   **- 멀티프로세스**  
   : 시스템 자원을 많이 사용하고 프로세스 간 통신이 어렵지만, 여러 개의 프로세스 중 하나에 문제가 발생하면 다른 프로세스로 확산이 되지 않는다.  
   **- 좀비 프로세스(Zombie process)**  
   : 자식 프로세스가 종료되었으나 부모 프로세스가 자식의 종료 상태를 회수하지 않음  
   **- 고아 프로세스(Orphan process)**  
   : 부모가 자식보다 먼저 종료되는 경우  
   **- 멀티 스레드**  
   : 시스템 자원은 적게 소모하고 스레드간 통신은 쉽지만, 공유 메모리의 동기화가 문제
2. **임계 영역(Critical Section)**- 공유자원을 한번에 하나만 이용하게끔 보장해줘야 하는 영역  
   - 문제 해결 방안  
   **- 상호 배제(Mutual Exclusion : Mutex)**   
   : 하나의 프로세스가 임계영역에 들어가 있다면 다른 프로세스 접근 금지  
   **- 진행(Progress)**  
   -> 임계 영역에 들어간 프로세스가 없는 상태에서  
   -> 들어가려 하는 프로세스가 여러 개라면 어느 것이 들어갈지 결정  
   **- 한정 대기(Bounded waiting)**  
   -> 다른 프로세스의 기아(Starvation)을 방지하기 위해   
   -> 한 번 임계 구역에 들어간 프로세스는 다음 번 임계 영역에 들어갈 때 제한을 둔다.  
   - 위의 문제들을 해결하기 위한 대표적인 2가지 방법 : **Mutex, Semaphore**
3. **뮤텍스(Mutex)**- Mutual Exclusion의 줄임말, 서로 다른 프로세스가 공유 자원에 접근하는 것을 제한.  
   - Mutex에서의 임계 영역은 Mutex가 Lock을 수행한 뒤 Unlock되기 전까지의 영역.  
   - Lock과 Unlock은 어떠한 경우에도 동시적으로 수행될 수 없다.  
   - EX) 한 프로세스가 Key(공유자원)을 얻고 Lock(사용중)을 알린다.  
   - 다른 프로세스가 acquire(접근)을 시도하지만 Lock이 풀릴때까지 접근불가 -> Block상태  
   - 이 프로세스는 Busy Waiting(spinlock : 일정 주기마다 접근 시도)상태에 빠지고,  
   - 이 현상은 CPU Cycle을 계속해서 낭비한다.



1. **세마포어(Semaphore)**- 임의의 개수를 세는 Counter처럼 동작해 임의의 개수의 프로세스만이 임계 영역에 동시적으로 접근  
   - **Binary Semaphore**(이진 세마포어 : Counter 값이 0, 1두개로만 존재, Mutex와 유사)  
   - **Counting Semaphore** : 정수 값을 가지는 변수로 볼 수 있고, 지정된 정수값 만큼 동시에 접근할 수 있다.  
   - 현재 수행중인 프로세스가 아닌 다른 프로세스가 세마포어를 해제할 수 있다.

