1. 운영체제는 사용자가 하드웨어를 효율적으로 사용할 수 있도록 도와주는 역할을 한다.
2. User mode는 사용자가 일반적으로 컴퓨터를 사용할 때의 모드이고 Kernel mode는 운영체제가 제어를 하는 모드이다. User mode에서 kernel mode로 접근하기 위해서는 system call이 필요하다. Read()와 같은 system call 함수를 통해 kernel mode로 진입 후 운영체제에서 이를 처리해준다. 위와 같은 방식으로 user mode와 kernel mode는 system call interface를 통해 정보를 처리한다.
3. Process와 다르게 thread는 한 process내에서 서로 data, code 영역을 서로 공유한다. 또 각각의 thread는 고유한 register, stack을 가지고 있다. process간에는 정보를 공유하려면 IPC방법이 필요해서 오버헤드가 크다는 단점이 있다. 하지만 thread는 data, code영역을 서로 공유하기 때문에 비교적 오버헤드가 적고 빠른 속도로 공유를 할 수 있다. 이것이 multi threading의 장점이다.
4. Interrupt는 I/O 요청과 같은 작업이 들어왔을 때 cpu에게 해당 사항을 요청하는방식이다. 이 방식에는 polling과 hardware interrupt 두가지가 있는데, hardware interrupt는 위와 같은 방식으로 I/O작업이 끝났다면 cpu에게 알리는것이고, polling은 cpu에게 I/O작업이 끝났는지 지속적으로 물어보는 방식이다. Polling은 계속해서 cpu에게 요청하는 것이므로 오버헤드가 크다는 단점이 있다.

비슷한 개념으로 Exception은 cpu가 자기 자신에게 interrupt를 발생시키는 것이다. Exception에는 trap과 fault 두가지로 나눌 수 있는데 trap은 의도적인 발생이고 fault는 예상치 못한 발생이다. Timer interrupt와 같은 것을 trap의 예로 들 수 있고, 컴파일 에러에러 같은 fault의 예로 들 수 있다.

1. 두개의 프로세스가 자원을 공유하려면 서로 데이터를 주고 받을 수 있는 방식이 필요하다. 직접 서로의 메모리에 접근하는 것은 Memory protection때문에 불가능하다. 그래서 Inter-process communication이 등장했는데, IPC 방식에는 Message passing과 Shared memory 두가지 방법이 있다. Message passing은 두 개의 process가 서로 정보를 전달할 때 kernel-mode를 거쳐서 전달하는 방식이고 shared-memory는 같은 상황일 때 user-mode에서 서로 정보를 전달할 공유 메모리를 만들어서 보다 직접적으로 통신하는 방식이다. Message passing은 운영체제 kernel을 거쳐서 가기 때문에 오버헤드가 크고 느리다는 단점이 있지만 데이터를 전달만 하면 운영체제가 알아서 해준다는 장점이 있다. shared-memory는 반대로 kernel을 거치지 않기 떄문에 오버헤드가 적고 속도가 비교적 빠르다는 장점이 있지만 공유 자원인 shared-memory 공간을 직접 개발자가 구현해야 한다는 번거로움이 있다.
2. Preemtive shortest job first scheduling

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P3 | P2 | P4 | P2 |
| 0 | 3 | 4 | 5 | 9 |

Waiting time: P1=0, P2=2+4, P3=0, P4=0

Average waiting time = (0+6+0+0)/4 = 1.5ms

1. Critical-section은 두개의 process가 공유자원에 접근했을 경우 문제가 일어나는 부분의 코드section이다. 만약 critical-section에서 하나의 process가 접근해서 작업을 수행중인데 context switching이 일어나서 다른 process가 접근한다면 작업을 끝냈을 때 예상치 못했던 결과가 나올 수 있다. 이것을 synchronization problem이라고 하고 이를 해결하기 위해선 3가지 조건이 필요하다. 첫번째는 mutual exclusion이고 한 개의 process가 critical section을 수행 중일 때 다른 process가 들어오면 안된다. 두번째는 progress이고 수행중인 프로세스가 없다면 바로 다음 process를 수행해야 한다는것이다. 이것은 cpu utilization을 최대로 해야한다는 조건이다. 세번째는 waited-bound이고 wait bound를 두어서 starvation을 방지해야 한다는 조건이다.
2. 해당 solution은 mutual exclusion을 보장한다. 왜냐하면 어떠한 순간에 context swichting이 일어나더라도 flag와 turn두가지 변수에 의해 하나의 프로세스가 모두 수행된 다음 다른 프로세스가 수행되기 때문이다.
3. Concurrent 는 하나의 process를 수행하지만 context switching이 매우 빠르게 일어나서 여러개의 process를 수행하는 것같이 보이는 것을 말한다.  
   따라서 하나의 process가 critical section에 들어갔을 때 context switching이 일어나서 다른 프로세스가 접근하는 문제를 일으키면 안되기 때문에 상호 배제가 필요하다. 예로는 producer-consumer problem, Dining philosophy 등이 있다.
4. Semaphore는 Higher level synchronization이고 integer 변수를 생성해서 동기화를 제어한다.  
   예를들어 process1의 semaphore x변수의 초기값은 1로 설정 후 critical section에 진입하기 전에 atomic단위의 wait함수에 전달한다. 그러면 wait함수는 내부적으로 x변수가 0이하이면 아무것도 수행하지 않는 조건을 건 후 변수를 0으로 변경한다. Process1은 critical section을 모두 수행 후 x변수를 signal함수에 전달한다. 그러면 signal 함수는 내부적으로 x변수를 +1 해준다.