**Ch4) Multithreaded Programming**

1. What is the difference between Thread and Process? What are the advantages of using multithreading?

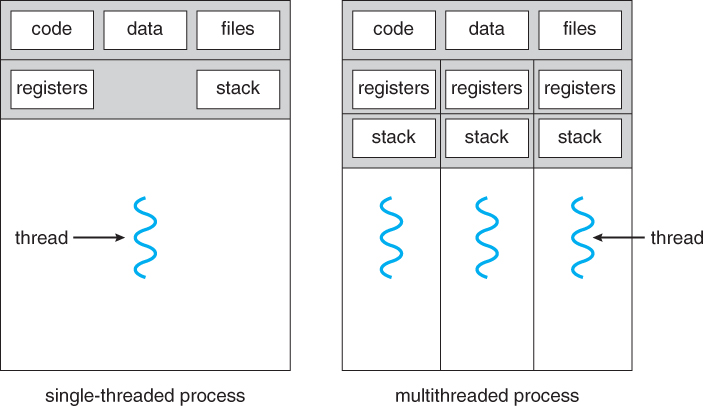
* Process와 다르게 thread는 각각이 고유한 register, stack을 가지고 있고 한 process내에서 서로 data, code 을 서로 공유한다. process간에는 정보를 공유하려면 IPC방법이 필요해서 오버헤드가 크다는 단점이 있다. 하지만 thread는 data, code영역을 서로 공유하기 때문에 비교적 오버헤드가 적고 빠른 속도로 공유를 할 수 있다. 이것이 multi threading의 장점이다.

1. Processes(멀티 프로세스)

* 여러 개의 process를 만들어서 동시에 실행
* Creating a new process is costly
* Inter-process communication is costly
* Heavy-weight, 오버헤드 발생
* Thread concept 등장

1. Thread

* Lightweight process라고도 함
* Cpu utilization의 기본 단위
* Program counter, register set, stack space로 구성



1. \*\*Processes vs Threads

* Process는 독립된 공간에서 다른 process의 영역을 확인하지 못하기 때문에 IPC(Inter-process communication)필요
* Thread는 code와 data는 서로 공유하고 register와 stack은 초기화해서 영역을 따로 할당받는다.
* Process는 은행 지점, thread는 은행의 창구
* Thread의 장점: thread간의 통신, 소통 가능
* Thread의 단점: 한 thread가 다른 thread에 영향

1. User Threads vs Kernel Threads (지원하는 주체에 따라)

* User Threads ex) POSIX Pthreads, Mach C-threads, Solaris threads 등
* Kernel Threads ex) Windows 95/98/NT/2000, Solaris, Linux, Tru64 UNIX

1. Multithreading Models
2. Many to one
3. One to one
4. Many to many
5. \*\* Threading Issues

* UNIX가 만들어진 10년 후 thread개념이 등장. 이슈가 많음

1. Semantics of fork() and exec() system calls - two versions of fork()

Ex) thread를 5개 생성했다. 그 중 1개를 fork()를 통해 생성한다면, thread가 6개가 되어야 할까? 아니면 전체 복사를 통해 process 2개, thread 10개가 생성되어야 할까?

* + 답이 없다. 운영체제마다 다름, thread에서는 fork()를 하지 않도록 만드는 것이 일반적임.

1. Thread cancellation
2. Asynchronous cancellation(비동기식 취소): 즉시 취소해야 하는 thread 강제 종료
3. Deferred cancellation(지연 취소): thread가 주기적으로 자기가 강제 종료되어야 할지를 확인한 후 강제 종료
   * Thread 취소를 어렵게 만드는 것? – 취소하는 thread에 할당된 자원문제. 갑자기 thread를 강제 종료하면 thread의 자원, 공유, 자료구조에 관한 내용의 수정이 일어나야 함.
4. Signal handling
5. Thread pools
6. Thread specific data
7. Pthreads

* POSIX는 UNIX계열의 system call 표준이다.
* Pthreads는 thread 표준임