[Chapter 4]

경영학과/2018100233/김정민

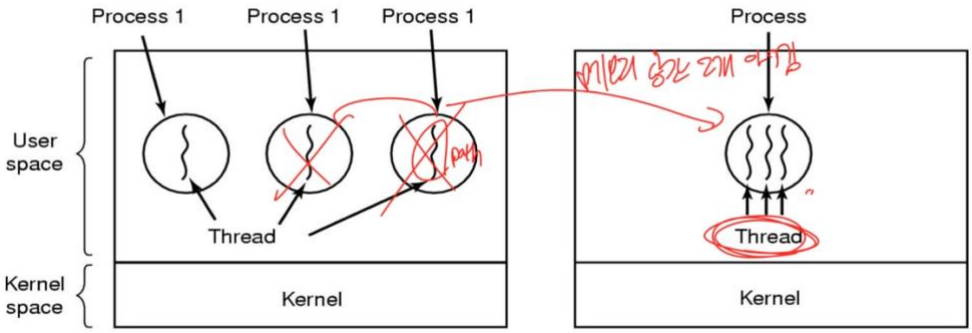
I. Process -> Thread

A. (멀티) Process의 단점

1. 하드웨어 부담이 크다.
   - 프로세스마다
     - 저장 공간이 여러 개
     - 오픈 파일과 같이 OS의 자원을 많이 잡아먹는다.
     - 상태를 저장하는 PCB 정보 필요(PC, SP, 레지스터, 등)
2. 새로운 프로세스를 만드는 것은 비용이 크다. (감수해야 할 게 많다.)
   - 프로세스마다 데이터 구조가 할당되고, 초기화된다.
3. IPC 역시 비용이 크다.
   - OS 커널의 개입이 필요
   - 시스템 콜 및 데이터를 복사하는 과정에서 커널 오버헤드 증가

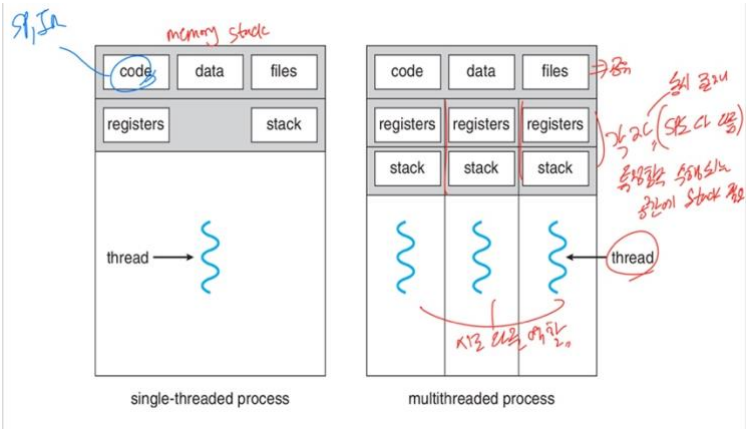
B. Thread의 개념

1. 프로세스를 실행 상태와 분리한다.
   - 프로세스는 주소 공간(로지컬 메모리), 우선순위 등의 여러 특성 및 리소스를 포함
2. 실행 상태란?
   - PC, SP, 레지스터 등의 프로세스의 실행 중인 부분과 관련된 정보
3. 이러한 실행 상태를 쓰레드라고 부르기로 한다.
   - A lightweight process (LWP)
     - Sun Microsystems에서 제시한 이름
     - 후에 thread로 정착
4. 새로운 프로세스를 만드는 데 필요한 부담을 줄일 수 있다. (하나의 프로세스 안에 여러 Path)

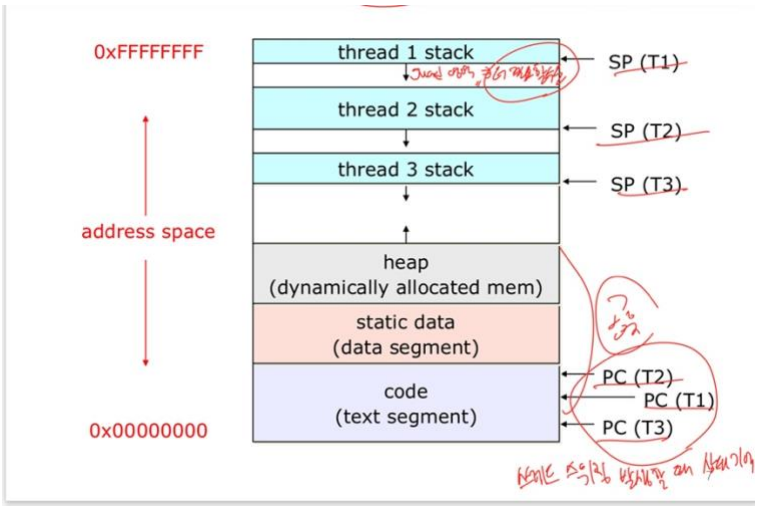


C. Single and Multithreaded Processes

1. Single-threaded
   - main 함수 내에서 두 개의 유저 함수를 호출할 경우, 코드에 따라서 순차적으로 작업을 수행한다.
2. Multithreaded
   - main 함수 내에서 두 개의 유저 함수를 호출할 경우, 새로운 path가 두 개 생성된다.
   - 따라서 두 개의 작업을 병렬적으로 처리할 수 있다.
   - 여전히 코드, 데이터, 파일은 공유하지, 각 path(작업)마다 필요한 데이터나 코드가 다르기 때문에 PC, SP, 레지스터 값은 각 path마다 다른 값을 가진다.



3. 멀티쓰레드 환경에서의 주소 공간 (가상, 로지컬 메모리)



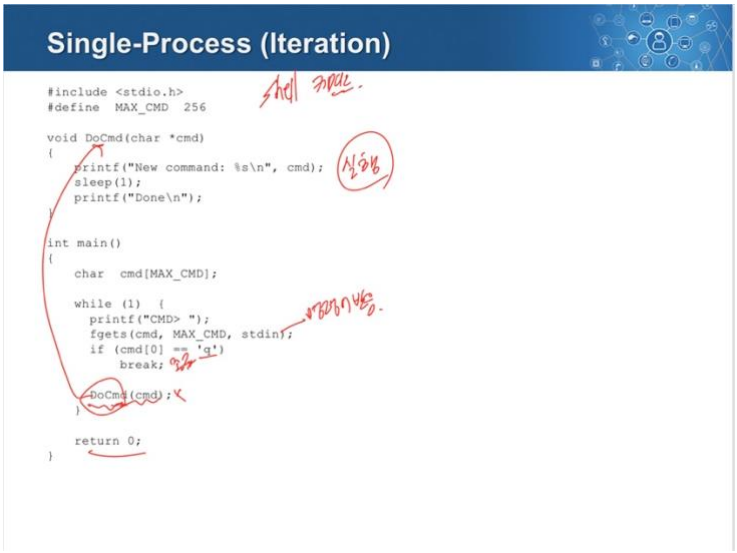
4. Static data
   - 정적 변수들을 저장해놓는 공간
   - 글로벌 변수, 상수 데이터 등
5. Code
   - 이진수로 컴파일된 코드 덩어리를 저장해놓는 공간

D. Multiprocess vs Multithread 비교

1. Multiprocess
   - single path의 경우
   - 부모 프로세스는 소켓 연결을 한 후에 요청을 기다리다가(while문), 요청이 들어오면 fork()를 통해 자식 프로세스를 생성
   - 이 과정에서 자식 프로세스를 생성하는 데 드는 시간보다 자식 프로세스가 작업을 수행하는 데 걸리는 시간이 짧다면?
     - 비효율적이다.
     - 쓰레드로 구현해보자.
   - Linux, Apache는 멀티 프로세스 기법을 활용
2. Multithread
   - 메인 함수의 path가 소켓 연결을 한 후에 요청을 기다리다가(while문), 요청이 들어오면 thread를 새로 만들어서 해당 thread에서 요청을 핸들링한다.
   - 핸들링이 끝났으면 소켓 연결을 해제한다.

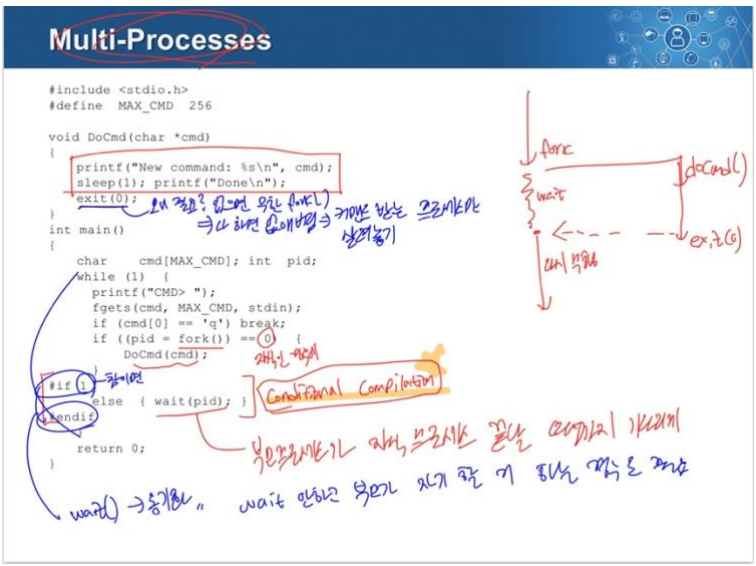
E. 코드 비교

1. Single-Process



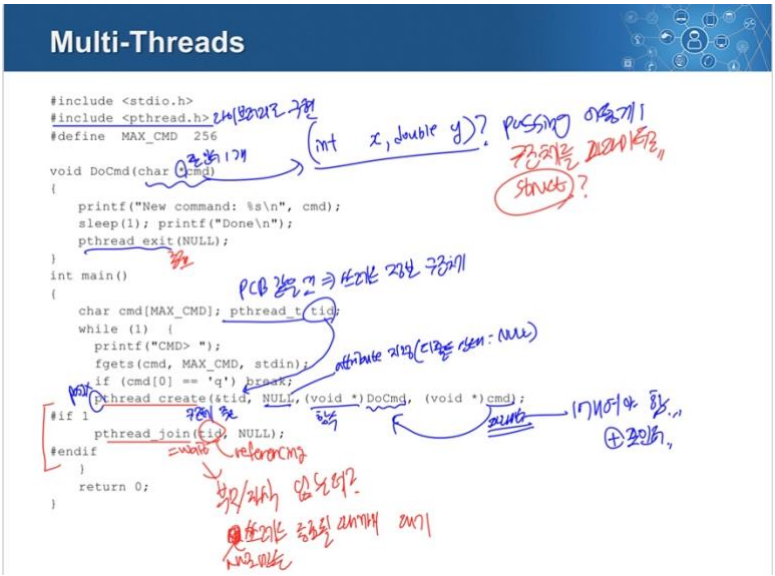
- fgets()를 통해 cmd를 받음
- cmd (입력값)으로 'q'가 들어오면 종료
- 그 외의 경우에선 DoCmd() 함수를 호출해서 실행한다.
- 프로세스 생성 개념 X

2. Multi-Processes



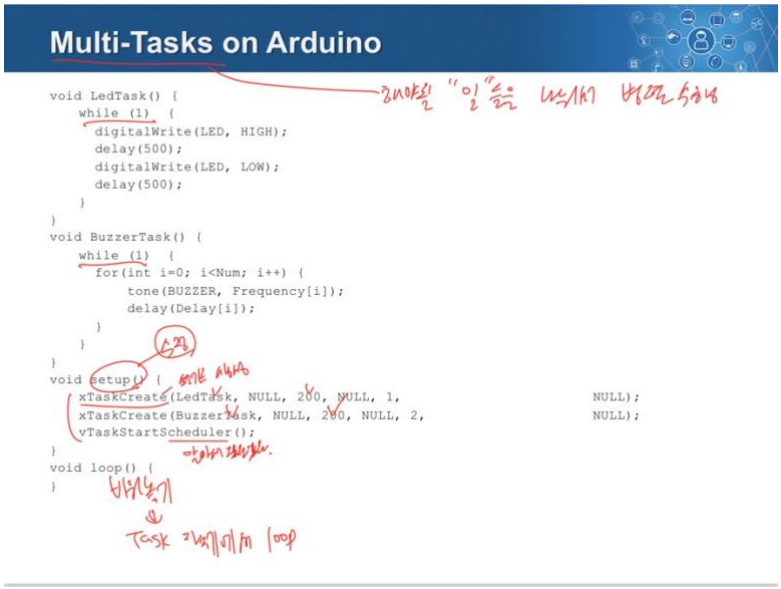
- cmd 입력받는 과정은 위와 똑같음
- fork()의 리턴 값이 0인 경우(자식 프로세스인 경우)에 DoCmd() 함수를 실행해서 작업을 수행한다.
- 부모 프로세스인 경우엔 wait(pid)를 통해 자식 프로세스가 끝날 때까지 기다린다.
  - wait() 함수는 동기화를 맞춰준다.
- **#if 1, #endif**
  - conditional compilation
  - 1을 0으로 바꿔야 할 때도 있다.
  - 부모가 자식을 기다리지 않고 자기 할 거 하는 경우
- exit(0)
  - DoCmd()에 싱글 프로세스에는 없는 exit(0); 코드 추가
  - exit()를 통해 작업 수행을 완료한 프로세스는 없어져야 한다.
  - 해당 코드 없으면 작업을 완료한 프로세스가 사라지지 않고 무한 fork() 발생
  - 커맨드 받아서 수행 중인 프로세스만 살아 있도록 !

3. Multi-Threads



- <pthread.h> 라이브러리로 구현
- 커맨드 입력받는 과정은 위와 같다.
- pthread_t tid;
  - pthread_t : 프로세스의 PCB 같은 것 -> 쓰레드 정보를 담고있는 구조체
- pthread_create(&tid, NULL, (void *) DoCmd, (void *) cmd) ;
  - &tid: pthread_t의 주소를 받음 (포인터 형태)
  - NULL : attributes 지정
  - (void *) DoCmd : 실행할 함수
  - (void *) cmd : 파라미터 (1개, 포인터)
    - 여러 개의 파라미터를 패싱하려면?
    - 구조체를 파라미터로 넘겨주면 된다.
- pthread_join(tid, NULL)
  - wait()과 같은 기능
  - 새로 만든 쓰레드가 종료될 때까지 대기한다.
- pthread_exit(NULL)
  - 쓰레드 종료

4. Arduino
- 아두이노에서 두 가지 프로그램을 동시에 실행하려면?
- 아두이노는 MMU가 없다.
  - 쓰레드 형태로 멀티 태스크(쓰레드) 수행
- Single-Task
  - 순차적으로 수행되기 때문에 두 프로그램의 딜레이 주기가 들쭉날쭉하다.
- Multi-Tasks



- setup() 함수 내에서 태스크를 생성한다.
  - xTaskCreate() : 쓰레드 생성
  - vTaskStartScheduler() : 스케줄링
- loop() : 비워놓기
  - Task 자체에서 loop 수행

II. Multicore Programming

A. Single-core vs Multi-core

1. Single-core
   - 순차적으로 태스크 실행
2. Multi-core
   - 의존성이 없는 태스크들을 병렬적으로 수행
3. Data vs. Task Parallelism
   - Data parallelism
     - 코어마다 각각의 데이터 공간을 가진다.
   - Task parallelism
     - 하나의 데이터를 모든 태스크가 공유한다.
     - 태스크는 코어별로 병렬 수행

B. Parallel Programming

1. Pthreads (POSIX threads)
   - Task parallelism에 중점
2. Data parallelism에 중점
   - OpemMP (Open Multi-Processing)
   - Open MPI (Message Passing Interface)
   - SIMD (Single Instruction Passing Interface)
   - GPGPU (General Purpose computing on GPUs)
     - CUDA (Computed Unified Device Architecture)
     - OpenCL (Open Computing Language)

III. Threads 종류 및 이슈

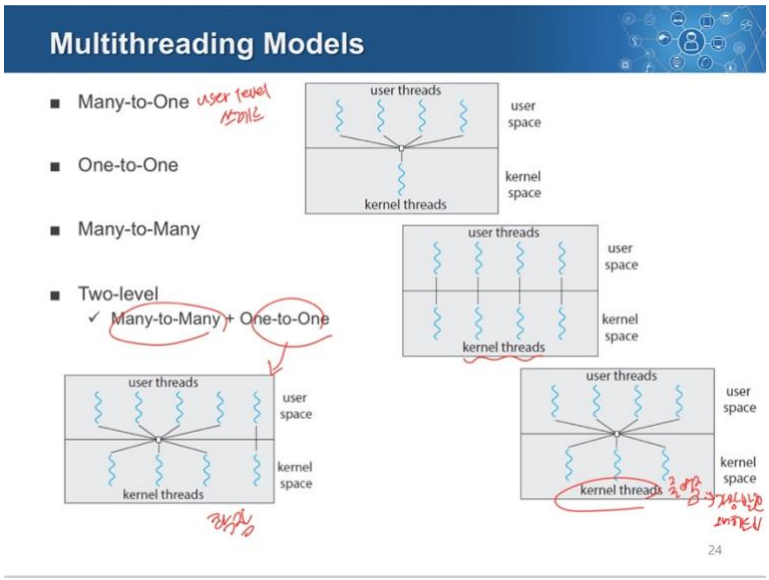
A. User Threads

1. 유저 레벨의 라이브러리를 통해 쓰레드를 관리
2. 작고 빠르다.

3. 예시
   - POSIX Pthreads
   - Mach C-thread
   - Solaris threads
4. 초창기 POSIX 쓰레드
   - 초창기 커널 – Monolithic
   - 쓰레드까지 관리하는 건 너무 헤비하다.
   - 라이브러리 형태로 유저 애플리케이션 단에서 관리
5. 문제점
   - 커널은 쓰레드의 관해 아예 모른다.
     - 커널은 프로세스 테이블만 가지고 있고, 쓰레드 테이블은 프로세스 내에 존재
   - 프로세스 1번의 A 쓰레드가 시작하자마자 I/O 요청하면? (blocking I/O)
     - 1번 프로세스의 모든 쓰레드는 다 같이 wait해야 한다.
     - 효율성 떨어진다.
   - 쓰레드 스케줄링이 프로세스 스케줄링부터 진행한 후에 수행된다.
     - Non-preemptive scheduling : yield()
       - 스레드는 스스로 수행 가능한 권한을 넘기지 않는다.
     - Preemptive scheduling : timer through signal
       - 일정 시간이 지나면 신호를 보내서 다른 쓰레드에게 주도권을 넘기도록 강제한다.

B. Kernel Threads
1. 커널이 쓰레드를 관리한다.
   - 프로세스 테이블 뿐만 아니라 쓰레드 테이블까지도 커널 내에 존재
   - 모든 쓰레드 operation은 커널 안에서 수행된다.
   - 커널이 시스템 안의 모든 쓰레드들의 스케줄링을 수행한다.
2. 시스템 콜에 의해 쓰레드 생성 및 관리
3. 커널 쓰레드는 프로세스들보다 비용이 적다.
   - IPC 줄어든다.
   - 같은 프로세스 내에서 메모리 공유
   - 프로세스 생성 및 종료로 인한 커널 오버헤드 감소
   - 스케줄링 오버헤드 감소
4. 예시
   - Windows 95/98/NT/2000
   - 발전된 Solaris
   - Tru64 UNIX
   - BeOS
   - Linux
5. 문제점
   - 커널이 너무 할 일이 많아서 비용이 크다.
     - 초기에는 멀티 프로세스 방식보다 비용이 컸다.
6. 만약에 A 프로세스의 5번째 쓰레드에서 fork()를 수행한다면, A 프로세스에 쓰레드 추가 or B 프로세스 생성?
   - 둘 중에 뭐가 나은지에 대한 해결 X
   - fork()를 통해 프로세스를 먼저 만들고 해당 프로세스에 쓰레드 생성 (불문율)
     - 쓰레드를 만들고나서 fork()하는 케이스는 사용하지 말자.

C. Multithreading Models



1. Many-to-One
   - 커널 쓰레드는 1개, 유저 쓰레드는 여러 개
   - 유저레벨 쓰레드와 같은 개념
   - 커널은 유저 레벨에서 쓰레드 여러 개가 어떻게 돌아가는지 알지 못한다.
2. One-to-One
   - 유저 쓰레드 1개 당 커널 쓰레드 1개 매핑
   - 커널 레벨 쓰레드와 같은 개념
   - 커널 오버헤드 증가
3. Many-to-Many
   - 다대다 관계
   - 하지만 유저 쓰레드에 비해 커널 쓰레드의 수가 적다.
     - 커널 오버헤드를 줄일 수 있다.
4. Two-level
   - Many-to-Many와 One-to-One을 동시에 가지고 있는 형태
   - 이전 개념들의 확장판

D. Threading Issues

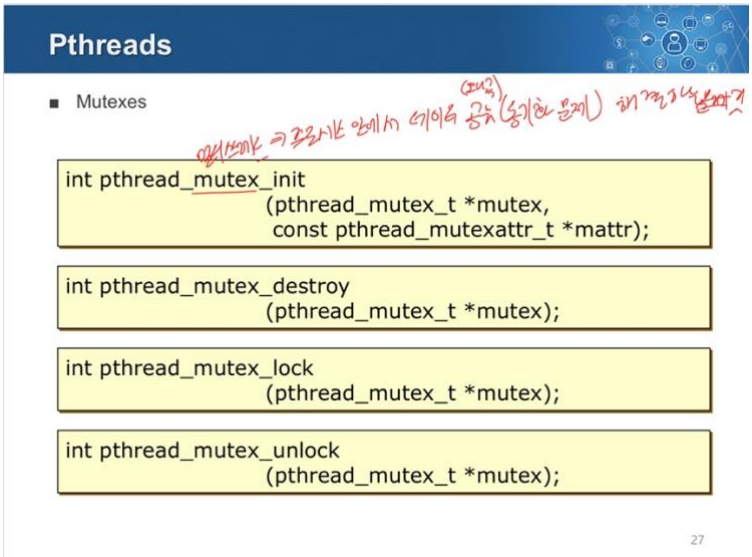
1. fork(), exec() 시스템 콜의 의미?
   - 사실상 두 가지 버전의 fork()라고 볼 수 있다.
2. Thread 취소
   - 비동기 취소 : 현재 수행 중인 작업과 별개로 쓰레드 즉시 종료
   - 지연 취소 : 예약한 다음에, 순서가 오면 쓰레드 종료
   - 해결책은? pthread_join()
     - 멀티프로세스에서 wait()과 같은 기능
     - 주어진 스레드가 종료될 때까지 현재 스레드의 실행을 wait 상태로 만든다.
3. 시그널 처리
   - 시그널 : 특정 이벤트 발생을 알리는 운영체제의 방법
   - 프로세스의 모든 쓰레드, 혹은 특정 쓰레드 등 개발자가 구현하기 나름대로 설정할 수 있다.
4. Thread Pools
   - 위에서 말한 불문율
   - 프로세스가 생성할 때 미리 많은 쓰레드를 생성해놔야 한다.

5. Thread specific data
   - 쓰레드 간의 데이터 교환은 MMU의 능력 밖이다.
   - 각 스레드는 개별적으로 스택과 레지스터 값을 가진다.

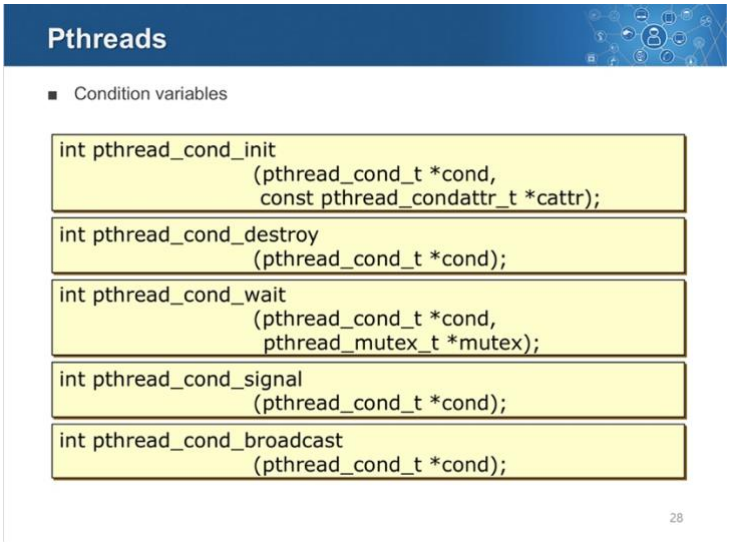
IV. 표준별 thread 라이브러리 및 형태(POSIX / Windows / Java 등)

A. Pthreads (POSIX)

1. pthread_create
   - 쓰레드 생성
   - 프로세스로 치면 fork()와 같은 역할
2. pthread_exit
   - exit()와 같은 역할
   - 호출되면 스레드는 즉시 종료
   - 종료되기. 전에 사후처리 작업을 할 수 있음
3. pthread_join
   - wait()과 같은 역할
4. Mutexes
   - 멀티 쓰레드 -> 프로세스 안에서 여러 쓰레드가 데이터를 공유하는 과정에서 동기화 문제를 해결하는 것
   - 한 번에 하나의 쓰레드만 특정 자원에 접근할 수 있도록 (Lock, unlock 등)



5. Conditional variables
   - 멀티 쓰레드 -> 프로세스 안에서 쓰레드 간 통신과 동기화를 위한 개념



- wait
  - 조건 변수를 대기 상태로 두고, 다른 스레드가 특정 조건을 만족할 때까지 기다림
- signal
  - 기다리고 있는 한 쓰레드한테 조건 변수가 충족됐음을 알림
- broadcast
  - 기다리고 있는 모든 쓰레드한테 조건 변수가 충족됐음을 알림

B. Windows

1. CreateThread
   - 쓰레드 생성
2. ExitThread
   - 쓰레드 종료 (리소스 해제까지)

C. Java

1. 자바는 VM 상에서 동작
2. 쓰레드 생성 및 실행에는 2가지 방법 존재
   - Create a new class derived from Thread class

     Override run() method
     - Thread 클래스로부터 파생된 클래스 생성 (상속)
     - Run() 메서드를 오버라이드(override) -> 실행할 코드를 작성
   - Create a new class that implements the runnable interface
     - Runnable 인터페이스를 구현한 클래스 생성
     - Run() 메서드에 실행할 코드를 작성

D. OS 회사별 쓰레드 디자인 형태 (싱글 프로세스, 멀티 프로세스, 멀티 쓰레드, 멀티 프로세스 + 쓰레드)