Window API Programming- 입문 5

afewhee@gmail.com

- 프로세스
  - 운영체제가 실행 시킨 하나의 프로그램
  - 프로세스가 다른 프로세스를 생성 가능
  - 부모 프로세스(Parent Process)
  - 자식 프로세스(Child Process) : 부모에 의해 생성된 Process

- Thread
  - 운영체제는 Job Scheduling을 통해서 CPU를 여러 프로세스가 사용하도록 함
  - 프로세스의 교환(문맥 전환: Context Switching)은 많은 오버헤드가 발생하게 되는데 Thread는 경량화된 프로세스로 문맥 전환에 대한 오버헤드를 줄임

  - Thread는 프로세스에 의해 생성되는 자식 프로세스
  - 부모 프로세스가 소멸하면 같이 소멸
  - 하나의 프로세스는 하나의 쓰레드를 생성 ➔ 주 쓰레드

  - Stack을 제외한 나머지 메모리 공간은 쓰레드 사이에서 공유

- 쓰레드 함수
  - 생성: CreateThread(...)
  - 소멸:
    - 쓰레드 함수가 리턴
    - 쓰레드 함수 내에서 ExitThread() 함수를 호출
    - TerminateThread() 함수를 외부에서 호출
    - 주 쓰레드가 종료하면 모든 쓰레드가 종료
  - 재개: ResumeThread()
  - 대기: SuspendThread()
  - 우선 순위: SetThreadPriority() / GetThreadPriority()
  - 자원반납: CloseHandle()

- 쓰레드와 같이 사용하는 함수
  - WaitForSingleObject(): 특정 객체가 signaled 상태 또는 타임 아웃이 될 때까지 대기
    ➔ 쓰레드가 종료할 때까지 기다리는 용도로 사용

  - 두 개 이상의 쓰레드가 종료될 때까지 대기하는 함수로 사용되는 것은
    ➔WaitForSingleObject()

- _beginthreadex():
  - 과거의 C 라이브러리는 멀티 프로세싱 환경에 안전하지 않음
  - _beginthreadex() 함수로 생성한 쓰레드는 C 라이브러리 함수를 사용해도 안전
  - 대부분의 게임프로그래머는 쓰레드 생성을 이 함수로 애용

- 동기화
  - 프로세스들의 실행 순서를 정하는 일
  - 자원 사용에 대한 Window의 동기화 방법
    - Critical Section, Mutex. Semaphore, Event

- 자원을 공유하는 멀티 프로세싱 환경에서는 동기화 문제가 꼭 발생
  - 공정한 자원 배분에 대한 경쟁 상태 문제
  - 자원 반납에 대한 교착상태

- 교착상태(Dead Lock)
  - 둘 이상의 프로세스들이 자원을 점유한 상태에서 다른 프로세스가 점유하고 있는 자원을 요구하며 무한정 기다리는 현상
  - 자원의 할당, 해제 등을 진행 할 수 없어 무 한정 기다리게 되는 상황

  - 교착상태 필요충분조건:
    - 상호 배제(Mutual exclusion): 어느 한 시점에 오직 하나의 프로세스만이 하나의 자원을 사용해야 한다는 조건
    - 점유와 대기(Hold and wait): 한 자원 점유하고 있으면서 다른 프로세스가 점유하고 있는 자원을 요청
    - 비선점(Non-preemption): 할당된 자원이 사용이 끝날 때까지 강제로 회수하지 않음
    - 순환 대기(Circular wait): 프로세스 간의 자원 요구가 원형 형태로 존재

2. 동기화

- 예방기법(Prevention)
  - 교착 상태가 발생되지 않도록 사전에 시스템을 제어
  - 교착 상태 발생의 4가지 조건 중에서 상호 배제를 제외한 나머지 중에서 하나를 제거(부정)함으로써 수행
  - 점유 및 대기 부정
    - 프로세스 실행하기 전 필요한 모든 자원을 할당, 프로세스 대기를 제거
    - 프로세서의 자원의 요구는 자원을 점유되지 않은 상태에서만 가능
  - 비선점 부정
    - 다른 자원을 프로세서가 요구하면 자신이 점유하고 있는 자원 반납
  - 환형대기 부정
    - 자원에 고유 번호를 부여하고, 프로세스는 점유한 자원의 고유 번호보다 작거나 큰, 어느 한쪽 방향으로만 자원을 요구하도록 함
- 회피 기법(Avoidance)
  - 교착 상태가 발생할 가능성을 배제하지 않고, 교착 상태가 발생하면 적절히 피해나가는 방법
  - 은행원 알고리즘. Dijkstra가 제안.
  - 안전 상태- 교착이 없는 상태. 불안전 상태 – 교착이 발생할 수 있는 상태
- 발견 기법(Detection)
  - 시스템에 교착 상태가 발생했는지 점검하여 교착 상태에 있는 프로세스와 자원을 발견
  - 교착 상태에 빠진 프로세스 그룹이 있으면 이 들 중 일부를 종료시킴
- 회복 기법(Recovery)
  - 교착 상태를 일으킨 프로세스를 종료하거나 교착상태의 프로세스에 할당된 자원을 선점하여 프로세스나 자원을 회복

- 1. Window의 프로세스 실행 모드
  - 유저 모드(User Mode)
  - 커널 모드(Kernel Mode)

- 2. 유저 모드
  - 동기화가 프로그래머(유저)에 의해 제어
  - 쓰레드 사이의 동기화 제어에 주로 사용
  - 시스템 리소스 접근 없음
  - CRITICAL_SECTION

- 3. 커널 모드
  - 동기화가 운영 체제(커널)에 의해 제어
  - 프로세스 사이의 동기화 제어에 주로 사용
  - 프로세스 제어로 인해 시스템 리소스 접근 필요
  - Mutex, Semarphore, Event

- 유저 모드 동기화

- 객체 생성 / 해제:
  - CRITICAL_SECTION 구조체 이용
  - 생성 함수: InitializeCriticalSection()
  - 해제 함수: DeleteCriticalSection()

- EnterCriticalSection()
  - 임계 영역 설정
  - 같은 자원을 공유하는 다른 프로세스는 대기

- LeaveCriticalSection()
  - 임계 영역 해제

- 뮤텍스
  - 커널 모드 동기화
  - 하나의 공유 자원을 보호하기 위해 사용
  - 하나의 쓰레드에서만 소유가능

- 상태
  - Signaled: 쓰레드의 실행을 허가한 상태. 뮤텍스 소유가 가능한 상태.
  - Non-signaled: 쓰레드의 실행을 허가하지 않은 상태. Signaled 될 때까지 쓰레드는 블록화 ➔ 다른 쓰레드에서 뮤텍스를 소유하고 있는 상태.

- 객체 생성 / 해제:
  - CreateMutex() - 생성
  - CloseHandle() - 해제
  - OpenMutex() – 이름으로 존재하는 뮤텍스 호출

- Lock/ Unlock
  - WaitForSingleObject()
    - 신호 상태가 될 때까지 기다림
    - 뮤텍스가 non-Signaled 상태이면 블록화 시킴.
  - ReleaseMutex()
    - 뮤텍스의 소유를 해제하여 다른 쓰레드가 이 뮤텍스를 소유할 수 있도록 함

5. Semaphore

- 세마포어
  - 커널 모드 동기화
  - 제한된 일정 개수를 가지는 자원을 보호
  - 정해진 숫자의 뮤텍스를 활용하는 것과 유사
  - 세마포어의 숫자를 1로 하면 뮤텍스와 동일

- 상태
  - Signaled: 세마포어 카운트가 0이 아닌 경우
  - Non-signaled: 세마포어 카운트가 0인 경우

- 함수
  - CreateSemaphore() - 생성
  - CloseHandle() - 해제
  - OpenSemaphore() – 이름으로 세마포어 얻기

- Lock/Unlock
  - WaitForSingleObject()
    - 세마포어 카운트를 1증가 시킴
  - ReleaseSemaphore()
    - 세마포어 카운트를 1증가 시킴

6. Event

- 이벤트
  - 커널 모드 동기화
  - 사건을 알리는 동기화 객체
  - 쓰레드 사이의 신호를 보내기 위해 사용. Ex)네트워크 수신이 있을 때.
- 상태
  - Set ➔ Signaled, Reset ➔ Non-signaled
  - Auto Reset Mode: 대기 상태가 종료되면 자동으로 Non-signaled 상태로 돌아감
  - Manual-reset Mode: 수동으로 Reset을 설정
- 함수
  - CreateEvent() – 생성: Auto Reset, Manual-Reset Mode 설정
  - CloseHandle() - 해제:
  - OpenEvent() - 이름 있는 이벤트:
  - SetEvent() – 이벤트를 Signaled 상태로 변경
  - ResetEvent() – 이벤트를 Non-signaled 상태로 변경
- WaitForSingleObject()
  - Auto Reset Mode
    - 자동으로 Non-signaled로 돌아감
  - Manual-reset Mode
    - WaitForSingleObject()함수의 호출에 상관 없이 계속 Signaled