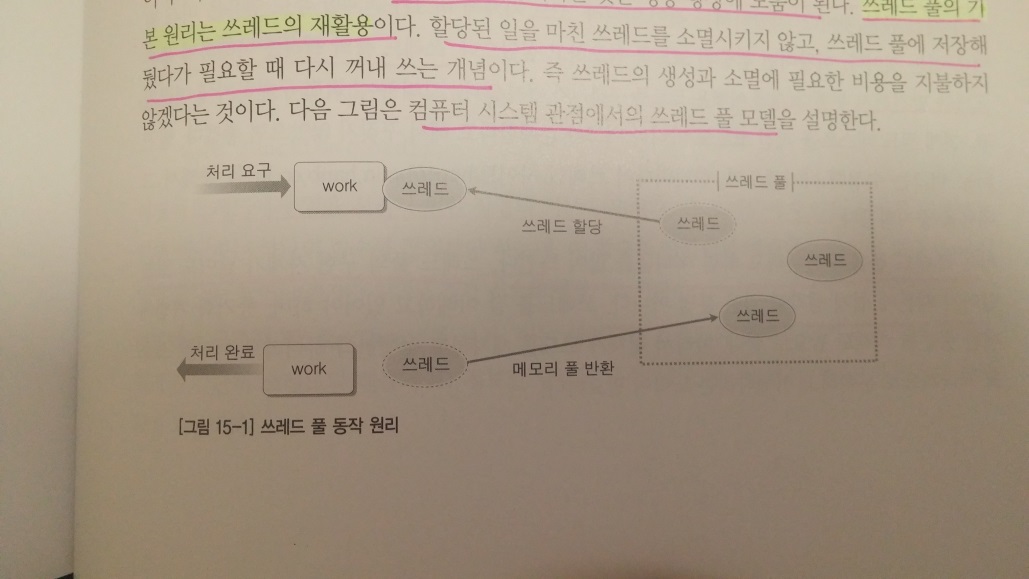
15장 스레드 풀

01. 스레드 풀에 대한 이해

빈번한 스레드의 생성 및 소멸은 우리가 느낄 수 있을 정도로 속도 저하를 가져 온다, 때문에 빈번한 스레드의 생성과 소멸은 피해야 한다. 즉, 스레드 풀을 유지하는 것은 성능 향상에 도움을 준다.

**스레드 풀의 기본 개념은 재활용** 이다. 할당된 일을 마친 스레드를 소멸시키지 않고, 스레드 풀에 저장해 뒀다가 필요할 때 다시 꺼내 쓰는 개념이다. 즉, 스레드의 생성과 소멸에 필요한 비용을 지불하지 않겠다는 것이다.



위 그림은 컴퓨터 시스템 관점에서의 스레드 풀 모델을 보여준다.

스레드 풀은 처리해야 할 일(Work)이 등록 되기 전에 생성되는데, 풀이 생성됨과 동시에 스레드 들도 생성되어 풀에서 대기하게 된다. 지능적인 풀은 처리해야 할 일의 증가 및 감소에 따라서 풀 안의 스레드 개수를 늘리기도 하고, 줄이기도 한다.

02. 스레드 풀의 구현

**스레드 풀 구현의 모듈별 해석**

스레드 풀을 구성하는 데 있어서 정의된 자료형과 함수들이다. 이 자료형과 함수들의 역할 및 관계를 이해히면, 전체 구조를 이해할 수 있다.

**스레드풀 자료구조**

|  |
| --- |
| typedef void (\*WORK) (void); // 수행할 일의 함수 포인터  typedef struct \_\_WorkerThread  {  HANDLE hThread;  DWORD idThread;  } WorkerThread; // 일을 수행할 스레드의 구조체  // Work와 Thread 관리를 위한 구조체.  struct \_\_ThreadPool  {  // Work을 등록하기 위한 배열.  WORK workList[WORK\_MAX];  // Thread 정보와 각 Thread별 Event 오브젝트  WorkerThread workerThreadList[THREAD\_MAX];  HANDLE workerEventList[THREAD\_MAX];    // Work에 대한 정보.  DWORD idxOfCurrentWork; // 대기 1순위 Work Index  DWORD idxOfLastAddedWork; // 마지막 추가 Work Index + 1.  // Number of Thread;  DWORD threadIdx; // Pool에 존재하는 Thread의 개수.  } gThreadPool; |

WORK는 함수 포인터로 선언되어 있다.

이것이 스레드에게 일을 시키기 위한 작업의 기본단위 이다. 즉, 스레드에게 일을 시키기 위해서는 일에 해당하는 하나의 함수를 정의해야 한다는 뜻이다.

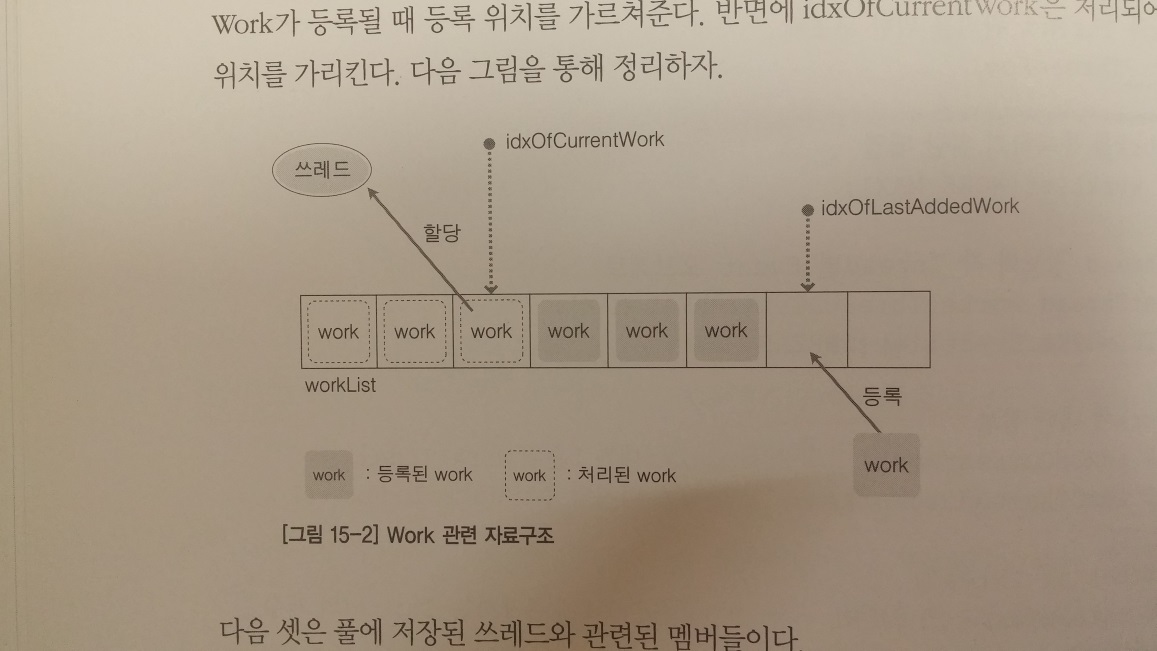
구조체 WorkerThread는 생성되는 스레드의 정보를 담기 위한 구조체 이다. 스레드의 핸들과 ID 값을 들고 있다.

구조체 \_ThreadPool은 typedef 선언 없이, 자료형 선언과 동시에 전역 변수 형태로 gThreadPool을 선언하고 있다. 이것이 스레드 풀을 표현하는 자료 구조 이다.

|  |
| --- |
| // Work을 등록하기 위한 배열.  WORK workList[WORK\_MAX];  // Work에 대한 정보.  DWORD idxOfCurrentWork; // 대기 1순위 Work Index  DWORD idxOfLastAddedWork; // 마지막 추가 Work Index + 1. |

workList는 Work을 등록하는 저장소 이다.

idxOfLastAddedWork는 마지막에 추가된 Work index보다 1많은 값을 유지하면서 새로운 Work가 등록될 때 등록 위치를 가르쳐 준다. 반면에 idxOfCurrentWork은 처리되어야 할 Work의 위치를 가리킨다.



위 그림은 새로운 Work를 등록하고 일이 끝난 스레드에게 다음 Work를 할당하고 있는 그림이다.

다음 셋은 스레드에 관련된 멤버 들이다.

|  |
| --- |
| // Thread 정보와 각 Thread별 Event 오브젝트  WorkerThread workerThreadList[THREAD\_MAX];  HANDLE workerEventList[THREAD\_MAX];  // Number of Thread;  DWORD threadIdx; // Pool에 존재하는 Thread의 개수. |

- workerThreadList : 풀에 저장된 스레드 정보는 workerThreadList에 저장한다.

- workerEvenList : 각 스레드 별로 하나씩 할당되는 이벤트 동기화 오브젝트를 저장하고 있는 배열이다.

- threadIdx : 저장된 스레드의 개수 정보를 담는다.

각 스레드 별로 하나씩 연결 되는 **workerEventList가 필요한 이유**

스레드에게 일이 부여 된다는 것은 스레드가 호출해서 실행할 함수를 지정해 준다는 뜻이다. 만약에 스레드에게 할당된 일이 없다면, 스레드는 WaitFor~ 관련 함수 호출을 통해서 Blocked 상태가 되어야 한다. 그리고 새로운 Work가 들어 왔을떄 잠에서 깨어 일을 실행하려 할 것이다. 이러한 컨트롤 을 위해서 스레드 하나당 하나 씩의 이벤트 오브젝트가 필요 한데, 이를 workerEventList에 저장해 둔 것이다.

**스레드 풀의 함수 관계**

**- WORK GetWorkFromPool(void);**

스레드 풀에서 Work를 가져올 때 호출하는 함수

**- DWORD AddWorkToPool(WORK work);**

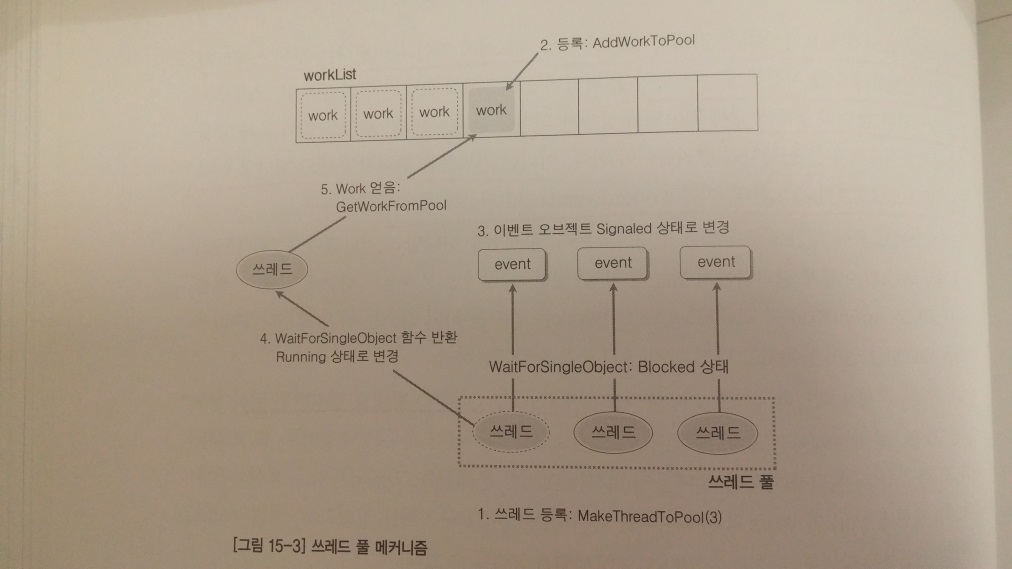
새로운 Work을 등록할 때 호출하는 함수

**- DWORD MakeThreadToPool(DWORD numOfThread);**

스레드 풀이 생성된 이후에 풀에 스레드를 생성(등록)하는 함수. 인자로 전달되는 수 만큼 스레드가 생성

**- void WorkerThreadFunction(LPVOID pParam);**

스레드가 생성되자마자 호출하는 스레드의 main함수이다. 이 함수의 구성을 봐야만 어떻게 Work을 할당받아서 처리하는지, 그리고 Work이 없을 때의 스레드 상태들을 알 수 잇다.



[스레드 풀 메커니즘]

**실행 순서**

단계 1.

스레드 풀에 MakeThreadToPool 함수의 호출을 통해서 스레드를 생성해 등록, 생성된 스레드는 이벤트 오브젝트가 Signaled 상태가 되기를 기다리며 Blocked 상태가 된다.

단계 2.

AddWorkToPool 함수 호출을 통해서 Work를 등록

단계 3.

Work가 등록되면, 스레드 풀에서 Blocked 상태에 있는 모든 이벤트 오브젝트를 Signaled 상태로 변경

단계 4.

모든 이벤트 오브젝트가 Signaled 상태가 되므로, 모든 스레드가 Running 상태가 된다. 그러나 Work 를 할당 받은 하나의 스레드를 제외하고 나머지는 다시 Blocked 상태가 된다. (**비효율적이므로 수정 할 수 있다.**)

단계 5.

Running 상태로 남아 있게 될 하나의 스레드는 GetWorkFromPool 함수 호출을 통해서 Work를 할당 받아서 실행하게 된다.

**전역으로 선언된 스레드 풀 접근 동기화**

스레드 풀은 전역으로 선언 되있고 내부적으로 다수의 스레드에 의해 Work를 할당 받는 동직이 이루어 진다. 따라서 해당 임계 영역에 동기화가 필요하다.

[ThreadPooling.cpp]

**요약**

1. 스레드 풀링의 이해

스레드 풀링은 한번 생성한 스레드를 재활용해서 시스템의 부담을 덜어주기 위한 기법이다. 이는 스레드의 생성과 소멸에 소모되는 리소스가 상당하기 때문에 등장한 것이다. 기본적인 개념은 다음과 같다. 일단 여러 개의 스레드를 생성한다. 그리고 실행해야 할 일이 등록될 때마다 미리 생성해 놓은 스레드 중 하나를 할당한다. 그리고 일이 끝나면 스레드는 소멸 시키지 않고 다음 일을 위해서 보관한다.

2. 스레드 풀링의 동작 원리

스레드 풀링의 동작 원리를 프로그램 코드상에서 공부하는 것은 다양한 의미가 있다. 필요한 상황에 최적의 스레드 풀을 제공할 수도 있고, 스레드 풀의 특성을 정확히 알아서 풀의 도입여부를 결정하는 데도 도움이 된다. 그리고 무엇보다 초보 개발자 들에게는 스레드와 동기화에 관련된 깊이 있는 이해를 경험할 수 있는 기회도 된다.