스레드의 동기화 [2016.07.21]

스레드의 동기화

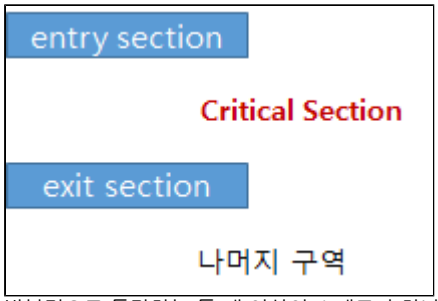
- 공유 데이터에 대한 동시 접근은 데이터의 일관성을 해치는 결과를 낳을 수 있다.
- 데이터의 일관성을 유지하기 위해서는 스레드들이 순차적으로 수행하게 만드는 기법이 필요하다.
  => 스레드 동기화가 필요한 이유이다.

스레드 동기화가 필요한 상황

- 둘 이상의 스레드가 공유 자원에 접근하는 경우
- 한 스레드가 작업을 완료한 후, 기다리고 있는 다른 스레드에 알려주는 경우

동기화 오브젝트의 종류

1. ### 이벤트 (Events)
    a. 순차적으로 수행해야 하는 일을 두 개 이상의 스레드로 나누어 수행할 때 사용

2. ### 크리티컬 섹션 (Critical Sections)
    a. 

    b. 반복적으로 동작하는 두 개 이상의 스레드가 하나의 리소스를 공유할 때 동시에 리소스에 접근하지 못하도록 하기 위해 사용
    c. 단일 프로세스의 스레드에 대해서만 동작

3. ### 뮤텍스 (Mutexes)
    a. 뮤텍스의 용도는 기본적으로 크리티컬 섹션과 동일
    b. 여러 프로세스의 스레드에 대해서 동작

4. ### 세마포어 (Semaphores)
    a. 리소스에 동시에 접근할 수 있는 스레드의 개수를 설정할 때 사용

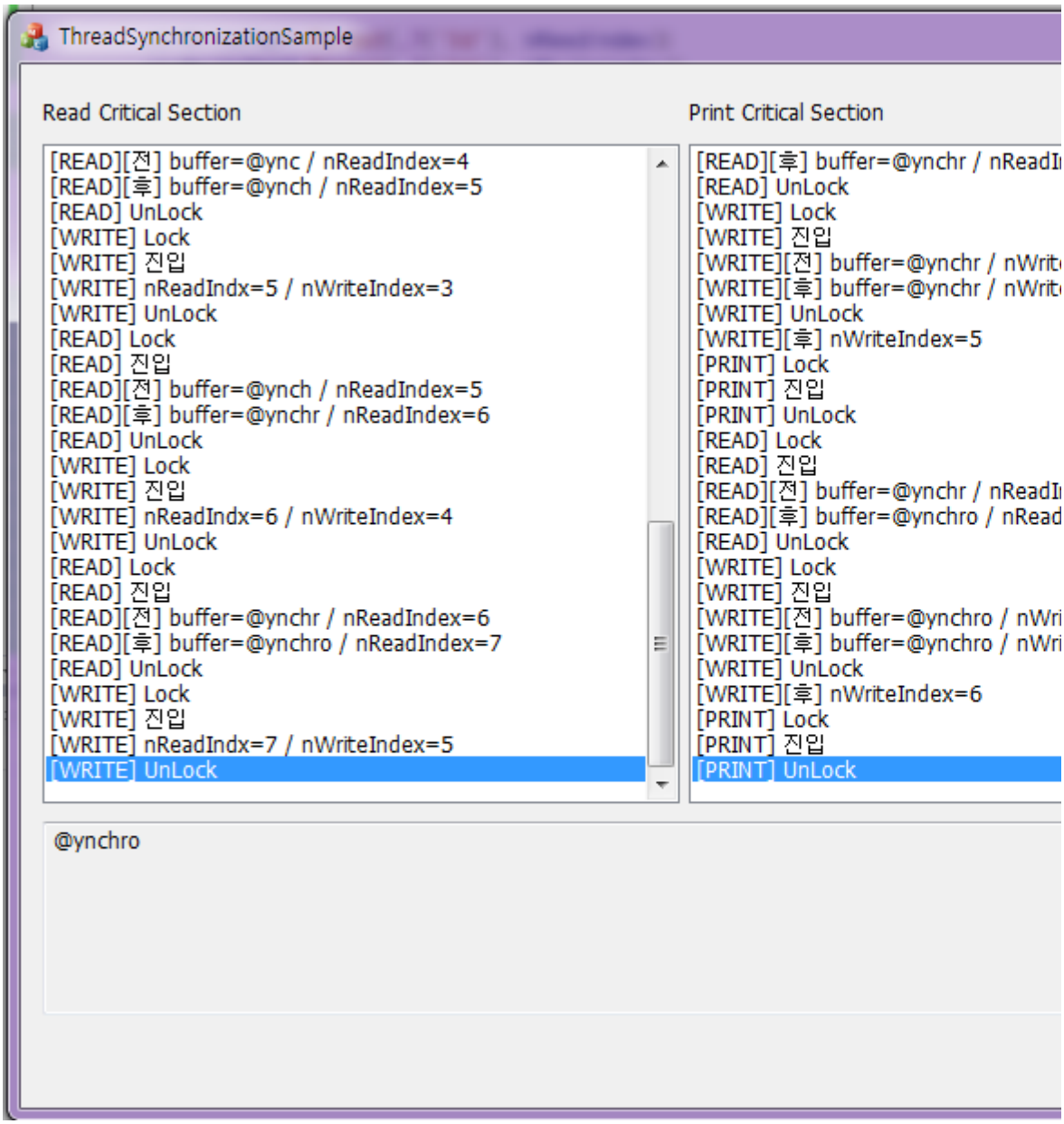
동기화를 할 때 확인해야 하는 부분들

1. **Data consistency가 확보되는지**
2. **Deadlock이 발생 하는지**
    a. Deadlock이란
        i. 두 개 이상의 프로세스들이 끝없이 이벤트를 기다리고 있는 상황
        ii. 그 이벤트는 기다리고 있는 프로세스만이 발생 시킬 수 있는 것
3. **Starvation 가능성이 있는지**
    a. Starvation이란
        i. 자원을 할당 받기 위해 무한정 대기하는 것
4. **Concurrency(병행처리)를 얼마나 제공하는지**

Sample

1. **SVN 주소**
    a. https://tr00129.infraware.net/svn/PolarisOffice7_Engine/POTester/ThreadSample

2. **구동 방법**
    a. ThreadSynchronizationSample을 시작프로젝트로 설정하여 구동한다.
    b. dialog에서 "Run" 버튼을 통해 스레드를 구동한다.
    c. Edit Control 에 글자가 출력되는 것을 확인할 수 있다.
    d. Critical Section 2개와 Event 1개에 대하여 각각 로그창을 확인 할 수 있다.
        i. 각 로그들은 [Thread명] prefix를 가지고 있으며, 해당 Thread Function에서 호출됨을 나타낸다.
        ii. **Read Critical Section** : 1) Read Thread가 Buffer에 데이터를 쓰고 ReadIndex를 증가시킬 때, 2) Write Thread가 ReadIndex를 사용할때
        iii. **Print Critical Section** : 1) Read Thread가 Buffer에 데이터를 쓰고 ReadIndex를 증가시킬 때, 2) Write Thread가 Buffer에 데이터를 쓸 때, 3) Print Thread가 Buffer를 출력할 때
        iv. **Event** : WriteIndex가 ReadIndex보다 크거나 같을 때 사용하는 동기화 오브젝트

e. 

3. Sample 컨셉

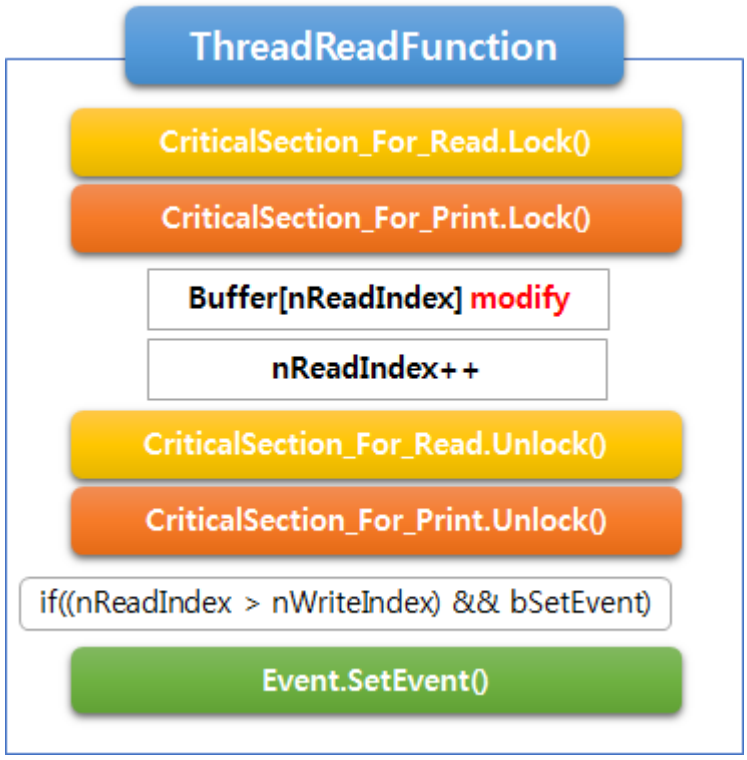
a. 화면에 글자를 출력하는 프로그램이다.
b. buffer를 3개의 스레드(Reader/Writer/Printer)가 공유하며, 화면에 출력되는 글자 중 "S" 를 "@"로 Replace한다.
    i. Reader Thread : buffer에 한 글자씩 적는다. (이미 정해져있는 단어를 한 글자씩 넣는다.)
    ii. Writer Thread : buffer에 적힌 글자들 중 "S"를 "@"로 교체한다.
    iii. Printer Thread : buffer를 화면에 출력한다.
c. read thread와 write thread는 buffer에 접근할 때 nReadIndex, nWriteIndex 를 사용한다.
    i. ex. buffer[nReadIndex], buffer[nWriteIndex]

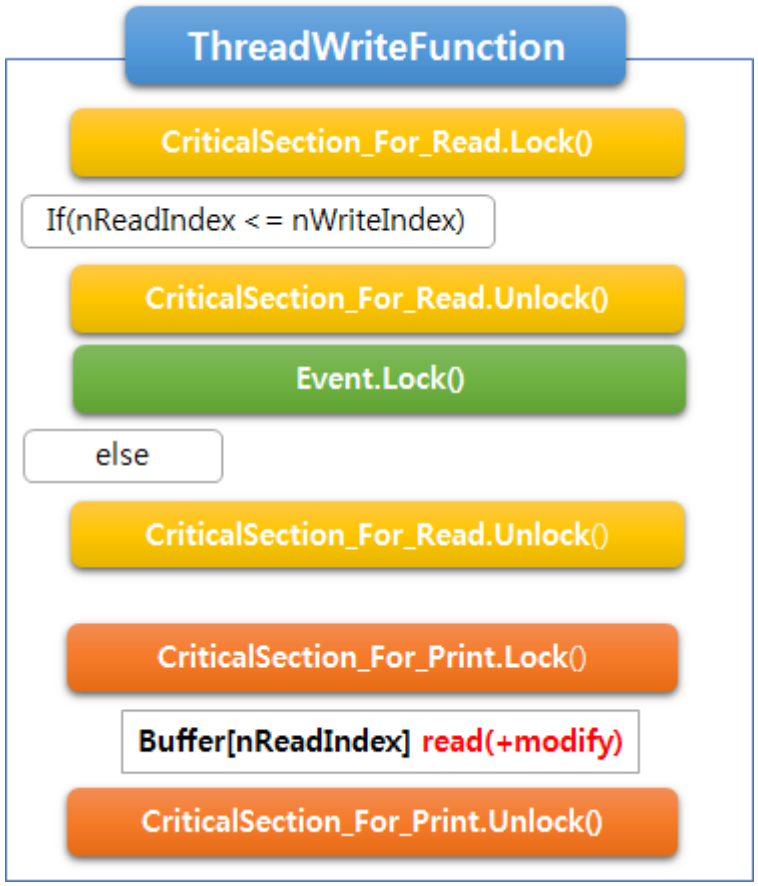
4. Sample에서 해결해야 하는 주요 동기화 문제

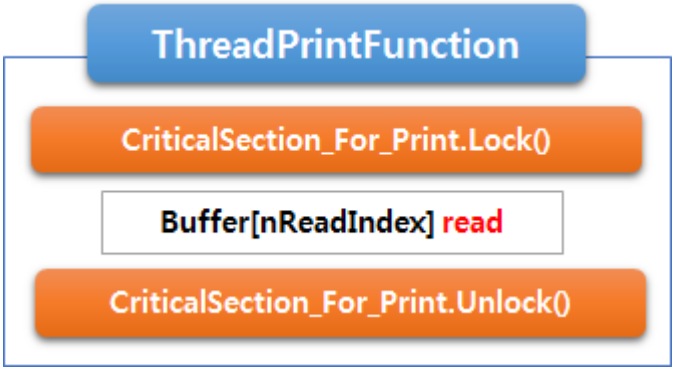
a. 동일한 인덱스의 Buffer[index]에는 한 번에 한 스레드만 접근해야 한다.
    i. 해결책 : Buffer에 접근하는 부분을 Critical Section으로 묶는다.
b. read thread의 index와 buffer에 값을 넣는 부분은 하나의 동작으로 이루어 져야 한다.
    i. 외부에서 index와 buffer를 사용하기 때문에 동기화 이슈가 발생할 수 있다.
    ii. 해결책 : index 증가부분과 buffer에 값을 넣는 부분을 Critical Section으로 묶는다.
c. Write Thread의 index는 Read Thread의 index를 앞지를 수 없다.
    i. 해결책 : Write Thread의 index가 Read Thread index보다 큰 경우 Reader의 작업이 끝날 때까지 기다린다.
    ii. Read 수행을 해야 Write가 수행할 수 있도록 하는 것이 아니라, index를 체크해서 해당 케이스일 때만 Write Thread가 잠시 기다리도록 Event.Lock()을 호출한다.

5. Sample Thread(Read/Write/Print) 요약

a.


ThreadReadFunction
CriticalSection_For_Read.Lock()
CriticalSection_For_Print.Lock()
Buffer[nReadIndex] modify
nReadIndex++
CriticalSection_For_Read.Unlock()
CriticalSection_For_Print.Unlock()
if((nReadIndex > nWriteIndex) && bSetEvent)
Event.SetEvent()


ThreadWriteFunction
CriticalSection_For_Read.Lock()
If(nReadIndex <= nWriteIndex)
CriticalSection_For_Read.Unlock()
Event.Lock()
else
CriticalSection_For_Read.Unlock()
CriticalSection_For_Print.Lock()
Buffer[nReadIndex] read(+modify)
CriticalSection_For_Print.Unlock()



6. 각 Thread 주요 코드

a. Read Thread

```
// Read Thread
UINT ThreadReadFunction(LPVOID lpParam)
{
  g_CriticalSection_For_Read.Lock();
  g_CriticalSection_For_Print.Lock();
  ////////////////////////////// Critical Section 시작
  // Reader Thread가 Buffer에 값을 작성.
  buffer[nReadIndex] = word[i%(SAMPLE_WORD_LENGTH-1)];
  nReadIndex++;
  ////////////////////////////// Critical Section 종료
  g_CriticalSection_For_Read.Unlock();
  g_CriticalSection_For_Print.Unlock();
  if((nReadIndex > nWriteIndex) && bSetEvent) // Writer Thread가 기다리고 있는 경우 event
lock을 해제한다.
  {
   bSetEvent = FALSE;
   g_Event->SetEvent();
   g_Event->ResetEvent();
  }
}
```

b. Write Thread

```
// Write Thread
UINT ThreadWriteFunction(LPVOID lpParam)
{
  g_CriticalSection_For_Read.Lock();
  ////////////////////////////// Critical Section 시작
  // Writer Thread의 index가 Reader Thread의 index보다 크거나 같은 경우
  // Reader Thread의 동작이 끝날 때까지 기다려야 한다.
  if( nReadIndex <= nWriteIndex )
  {
  ////////////////////////////// Critical Section 종료
   g_CriticalSection_For_Read.Unlock();
   bSetEvent = TRUE;
   g_Event->Lock(); // SetEvent()가 호출되기 전까지 멈추고 기다린다.
  }
  else
  ////////////////////////////// Critical Section 종료
   g_CriticalSection_For_Read.Unlock();

  if(nWriteIndex >=0)
  {
   g_CriticalSection_For_Print.Lock();
   ////////////////////////////// Critical Section 시작
   // Writer Thread가 Buffer를 수정하는 동작
   if(buffer[nWriteIndex] == L'S')
    buffer[nWriteIndex] =L'@';
   ////////////////////////////// Critical Section 종료
   g_CriticalSection_For_Print.Unlock();
  }
  nWriteIndex++;
}
```

c. **Print Thread**

```
// Print Thread
UINT ThreadPrintFunction(LPVOID lpParam)
{
  g_CriticalSection_For_Print.Lock();
  ////////////////////////////// Critical Section 시작
  pDlg->SetDlgItemTextW(IDC_EDIT1, buffer);
  ////////////////////////////// Critical Section 종료
  g_CriticalSection_For_Print.Unlock();
}
```