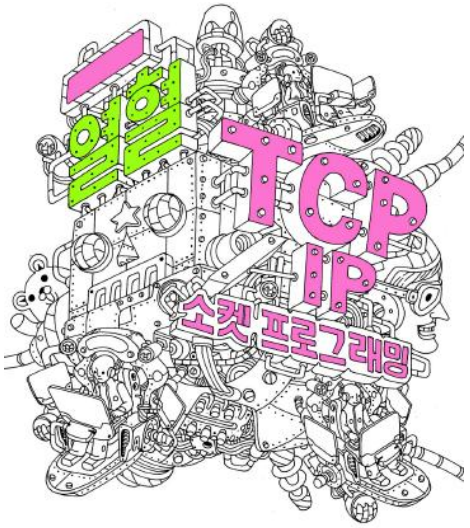
윤성우의 열혈 TCP/IP 소켓 프로그래밍

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

Chapter 20. Windows에서의 쓰레드 동기화



Chapter 20-1. 동기화 기법의 분류와 CRITICAL_SECTION 동기화

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

유저모드(User mode)와 커널모드(Kernel mode)

유저모드 VS 커널모드

- 유저모드 응용 프로그램이 실행되는 기본모드로, 물리적인 영역으로의 접근이 허용되지 않으며, 접근할 수 있는 메모리의 영역에도 제한이 따른다.
- 커널모드 운영체제가 실행될 때의 모드로, 메모리뿐만 아니라, 하드웨어의 접근에도 제한이 따르지 않는다.

간단히 유저모드는 응용 프로그램의 실행모드, 커널모드는 운영체제의 실행모드로 구분하기도 한다.

유저모드는 프로그램의 실행에 제한을 두기 위한 모드이다. 즉, 유저모드로 동작을 하면, 제한된 메모리 및 리소스에만 접근이 가능하다. 하지만 쓰레드의 생성과 **같은 운영체제에 의해서 완성되는 연산을 위해서는 유저모드에서 커널모드로의 전환이 이뤄져야 하며,** 또 반대로 커널모드에서 유저모드로의 전환도 이뤄져야 한다.

유저모드 동기화와 커널모드 동기화

유저모드 동기화

- 운영체제에 의해서 이뤄지는 동기화가 아닌, 순수 라이브러리에 의존해서 완성되는 동기화 기법이다.
- 운영체제에 의해서 제공되지 않으므로 커널모드로의 전환이 불필요하다. 따라서 상대적으로 가볍고 속도가 빠르다.

커널모드 동기화

- 커널모드 동기화는 커널에 의해서 제공이 되는 동기화 기법이다.
- 커널에 의해 제공되는 동기화이다 보니, 다양한 기능이 제공된다.
- 한가지 예로 Dead-lock에 걸리지 않도록 타임아웃을 지정할 수 있다.

CRITICAL_SECTION 기반의 동기화

```
#include <windows.h>

void InitializeCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
void DeleteCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

- lpCriticalSection — Init... 함수에서는 초기화 할 CRITICAL_SECTION 오브젝트의 주소 값 전달, 반면 Del... 함수에서는 해제할 CRITICAL_SECTION 오브젝트의 주소 값 전달.

CRITICAL_SECTION 오브젝트의 생성과 소멸

```
#include <windows.h>

void EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
void LeaveCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

- lpCriticalSection — 획득(소유) 및 반납할 CRITICAL_SECTION 오브젝트의 주소 값 전달.

CRITICAL_SECTION 오브젝트의 획득과 반납

CRITICAL_SECTION 오브젝트는 커널 오브젝트가 아니다. 따라서 유저모드에서만 동작한다.

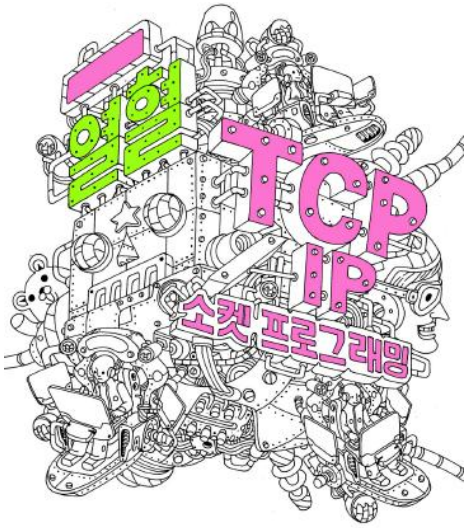
CRITICAL_SECTION 기반 동기화의 예

예제 SyncCS_win.c

```
long long num=0;
CRITICAL_SECTION cs;

int main(int argc, char *argv[])
{
    HANDLE tHandles[NUM_THREAD];
    int i;

    InitializeCriticalSection(&cs);
    for(i=0; i<NUM_THREAD; i++)
    {
        if(i%2)
            tHandles[i]=(HANDLE)_beginthreadex(NULL, 0, threadInc, NULL, 0, NULL);
        else
            tHandles[i]=(HANDLE)_beginthreadex(NULL, 0, threadDes, NULL, 0, NULL);
    }

    WaitForMultipleObjects(NUM_THREAD, tHandles, TRUE, INFINITE);
    DeleteCriticalSection(&cs);
    printf("result: %lld \n", num);
    return 0;
}
```

```
unsigned WINAPI threadInc(void * arg)
{
    int i;
    EnterCriticalSection(&cs);
    for(i=0; i<50000000; i++)
        num+=1;
    LeaveCriticalSection(&cs);
    return 0;
}
unsigned WINAPI threadDes(void * arg)
{
    int i;
    EnterCriticalSection(&cs);
    for(i=0; i<50000000; i++)
        num-=1;
    LeaveCriticalSection(&cs);
    return 0;
}
```

임계영역을 EnterCri~ 함수와 LeaveCri~ 함수로 감싼다.



Chapter 20-2. 커널모드 동기화 기법

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

Mutex 오브젝트의 생성과 소멸

```
#include <windows.h>

HANDLE CreateMutex(
    LPSECURITY_ATTRIBUTES lpMutexAttributes, BOOL bInitialOwner, LPCTSTR lpName);
```

➜ 성공 시 생성된 Mutex 오브젝트의 핸들, 실패 시 NULL 반환

- lpMutexAttributes — 보안관련 특성 정보의 전달, 디폴트 보안설정을 위해서 NULL 전달.
- bInitialOwner — TRUE 전달 시, 생성되는 Mutex 오브젝트는 이 함수를 호출한 쓰레드의 소유가 되면서 non-signaled 상태가 된다. 반면 FALSE 전달 시, 생성되는 Mutex 오브젝트는 소유자가 존재하지 않으며, signaled 상태로 생성된다.
- lpName — Mutex 오브젝트에 이름을 부여할 때 사용된다. NULL을 전달하면 이름 없는 Mutex 오브젝트가 생성된다.

뮤텍스는 auto-reset 모드 커널 오브젝트이다!

```
#include <windows.h>

BOOL CloseHandle(HANDLE hObject);
```

➜ 성공 시 TRUE, 실패 시 FALSE 반환

- hObject — 소멸하고자 하는 커널 오브젝트의 핸들 전달.

Mutex 오브젝트의 획득과 반납

```
#include <windows.h>

BOOL ReleaseMutex(HANDLE hMutex);
```

➜ 성공 시 TRUE, 실패 시 FALSE 반환

- hMutex　반납할, 다시 말해서 소유를 해제할 Mutex 오브젝트의 핸들 전달.

Mutex 오브젝트는 커널 오브젝트이기 때문에 획득을 위해서 별도로 정의된 함수가 없고, 대신 WaitForSingleObject 함수를 이용한다. 즉, **Mutex 오브젝트는 획득 가능한 상태가 되면, signaled 상태가 된다.**

```
WaitForSingleObject(hMutex, INFINITE);
// 임계영역의 시작
// . . . . .
// 임계영역의 끝
ReleaseMutex(hMutex);
```

따라서 왼쪽에 보이는 구조로 임계영역을 감싸면 된다.

Mutex 기반의 동기화 예

예제 SyncMutex_win.c

```
long long num=0;
HANDLE hMutex;

int main(int argc, char *argv[])
{
    HANDLE tHandles[NUM_THREAD];
    int i;

    hMutex=CreateMutex(NULL, FALSE, NULL);
    for(i=0; i<NUM_THREAD; i++)
    {
        {
            if(i%2)
                tHandles[i]=(HANDLE)_beginthreadex(NULL, 0, threadInc, NULL, 0, NULL);
            else
                tHandles[i]=(HANDLE)_beginthreadex(NULL, 0, threadDes, NULL, 0, NULL);
        }
    }

    WaitForMultipleObjects(NUM_THREAD, tHandles, TRUE, INFINITE);
    CloseHandle(hMutex);
    printf("result: %lld \n", num);
    return 0;
}
```

```
unsigned WINAPI threadInc(void * arg)
{
    int i;
    WaitForSingleObject(hMutex, INFINITE);
    for(i=0; i<50000000; i++)
        num+=1;
    ReleaseMutex(hMutex);
    return 0;
}
unsigned WINAPI threadDes(void * arg)
{
    int i;
    WaitForSingleObject(hMutex, INFINITE);
    for(i=0; i<50000000; i++)
        num-=1;
    ReleaseMutex(hMutex);
    return 0;
}
```

Semaphore 오브젝트 기반 동기화


```
#include <windows.h>

HANDLE CreateSemaphore(
    LPSECURITY_ATTRIBUTES lpSemaphoreAttributes, LONG lInitialCount, LONG lMaximumCount,
    LPCTSTR lpName);
    ➜ 성공 시 생성된 Semaphore 오브젝트의 핸들, 실패 시 NULL 반환
```

- lpSemaphoreAttributes — 보안관련 정보의 전달, 디폴트 보안설정을 위해서 NULL 전달.
- lInitialCount — 세마포어의 초기 값 지정, 매개변수 lMaximumCount에 전달된 값보다 크면 안되고, 0 이상이어야 한다.
- lMaximumCount — 최대 세마포어 값을 지정한다. 1을 전달하면 세마포어 값이 0, 또는 1이 되는 바이너리 세마포어가 구성된다.
- lpName — Semaphore 오브젝트에 이름을 부여할 때 사용한다. NULL을 전달하면 이름없는 Semaphore 오브젝트가 생성된다.

Semaphore 오브젝트의 소멸도 CloseHandle 함수호출을 통해서 이뤄진다. 그리고 세마포어 값이 0인 경우 non-signaled 상태가 되고, **0보다 큰 경우에 signaled 상태가 된다.** 그리고 세마포어 값은 0보다 작아질 수 없다. 따라서 세마포어의 초기값 설정을 통해서 임계영역에 접근가능 한 쓰레드의 수를 제한할 수 있다.

세마포어 기반 동기화의 형태

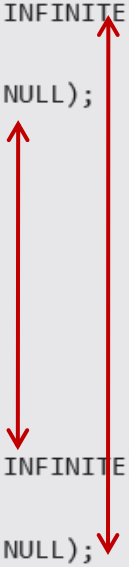
```
WaitForSingleObject(hSemaphore, INFINITE);
// 임계영역의 시작
// . . . . .
// 임계영역의 끝
ReleaseSemaphore(hSemaphore, 1, NULL);
```

```
#include <windows.h>

BOOL ReleaseSemaphore(HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount);
    ➜ 성공 시 TRUE, 실패 시 FALSE 반환
```

- hSemaphore — 반납할 Semaphore 오브젝트의 핸들 전달.
- lReleaseCount — 반납은 세마포어 값의 증가를 의미하는데, 이 매개변수를 통해서 증가되는 값의 크기를 지정할 수 있다. 그리고 이로 인해서 세마포어의 최대 값을 넘어서게 되면, 값은 증가하지 않고 FALSE가 반환된다.
- lpPreviousCount — 변경 이전의 세마포어 값 저장을 위한 변수의 주소 값 전달, 불필요하다면 NULL 전달.

세마포어 동기화의 예

```
unsigned WINAPI Read(void * arg)
{
    int i;
    for(i=0; i<5; i++)
    {
        fputs("Input num: ", stdout);
        WaitForSingleObject(semTwo, INFINITE);
        scanf("%d", &num);
        ReleaseSemaphore(semOne, 1, NULL);
    }
    return 0;
}
unsigned WINAPI Accu(void * arg)
{
    int sum=0, i;
    for(i=0; i<5; i++)
    {
        WaitForSingleObject(semOne, INFINITE);
        sum+=num;
        ReleaseSemaphore(semTwo, 1, NULL);
    }
    printf("Result: %d \n", sum);
    return 0;
}
```



```
int main(int argc, char *argv[])
{
    HANDLE hThread1, hThread2;
    semOne=CreateSemaphore(NULL, 0, 1, NULL);
    semTwo=CreateSemaphore(NULL, 1, 1, NULL);

    hThread1=(HANDLE)_beginthreadex(NULL, 0, Read, NULL, 0, NULL);
    hThread2=(HANDLE)_beginthreadex(NULL, 0, Accu, NULL, 0, NULL);

    WaitForSingleObject(hThread1, INFINITE);
    WaitForSingleObject(hThread2, INFINITE);

    CloseHandle(semOne);
    CloseHandle(semTwo);
    return 0;
}
```

```
Input num: 1
Input num: 2
Input num: 3
Input num: 4
Input num: 5
Result: 15
```

예제 SyncSema_win.c

실행결과

두 개의 세마포어 오브젝트를 생성해서 두 쓰레드의 실행 순서를 동기화하고 있다.

Event 오브젝트 기반 동기화

```
#include <windows.h>

HANDLE CreateEvent(
    LPSECURITY_ATTRIBUTES lpEventAttributes, BOOL bManualReset, BOOL bInitialState, LPCTSTR lpName);
```

➜ 성공 시 생성된 Event 오브젝트의 핸들, 실패 시 NULL 반환

- lpEventAttributes: 보안관련 정보의 전달, 디폴트 보안설정을 위해서 NULL 전달.
- bManualReset: TRUE 전달 시 manual-reset 모드 Event, FALSE 전달 시 auto-reset 모드 Event 오브젝트 생성.
- bInitialState: TRUE 전달 시 signaled 상태의 Event 오브젝트 생성, FALSE 전달 시 non-signaled 상태의 Event 오브젝트 생성.
- lpName: Event 오브젝트에 이름을 부여할 때 사용된다. NULL을 전달하면 이름없는 Event 오브젝트가 생성된다.

이벤트 오브젝트의 가장 큰 특징은, 자동으로 non-signaled 상태로 돌아가지 않는 **manual-reset 모드 커널 오브젝트로 생성이 가능**하다는 점이다.

```
#include <windows.h>

BOOL ResetEvent(HANDLE hEvent);   // to the non-signaled
BOOL SetEvent(HANDLE hEvent);     // to the signaled
```

➜ 성공 시 TRUE, 실패 시 FALSE 반환

따라서 manual-reset 모드로 이벤트 오브젝트를 생성했다면, 왼쪽의 두 함수를 통해서 이벤트의 상태를 변경시켜야 한다.

이벤트 기반 동기화의 예

```
static char str[STR_LEN];
static HANDLE hEvent;

int main(int argc, char *argv[])
{
    HANDLE hThread1, hThread2;
    hEvent=CreateEvent(NULL, TRUE, FALSE, NULL);
    hThread1=(HANDLE)_beginthreadex(NULL, 0, NumberOfA, NULL, 0, NULL);
    hThread2=(HANDLE)_beginthreadex(NULL, 0, NumberOfOthers, NULL, 0, NULL);

    fputs("Input string: ", stdout);
    fgets(str, STR_LEN, stdin);
    SetEvent(hEvent);

    WaitForSingleObject(hThread1, INFINITE);
    WaitForSingleObject(hThread2, INFINITE);
    ResetEvent(hEvent);
    CloseHandle(hEvent);
    return 0;
}
```

main 쓰레드가 문자열을 읽어 들인 후 대기 중에 있는 두 쓰레드를 깨어나게 한다.

manaul-reset 모드로 이벤트 오브젝트를 생성했으므로, 두 쓰레드는 동시에 깨어나게 된다.

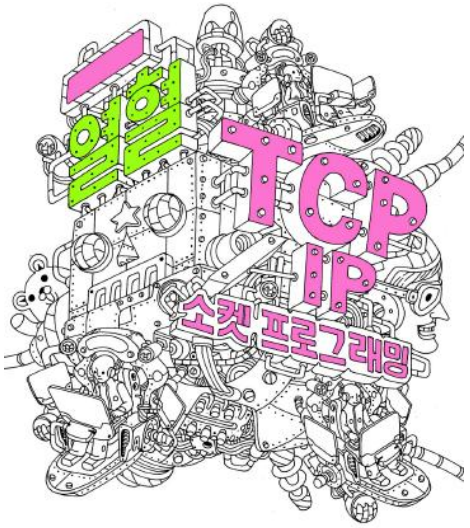
```
Input string: ABCDABC
Num of A: 2
Num of others: 5
```

실행결과

```
unsigned WINAPI NumberOfA(void *arg)
{
    int i, cnt=0;
    WaitForSingleObject(hEvent, INFINITE);
    for(i=0; str[i]!=0; i++)
    {
        if(str[i]=='A')
            cnt++;
    }
    printf("Num of A: %d \n", cnt);
    return 0;
}
```

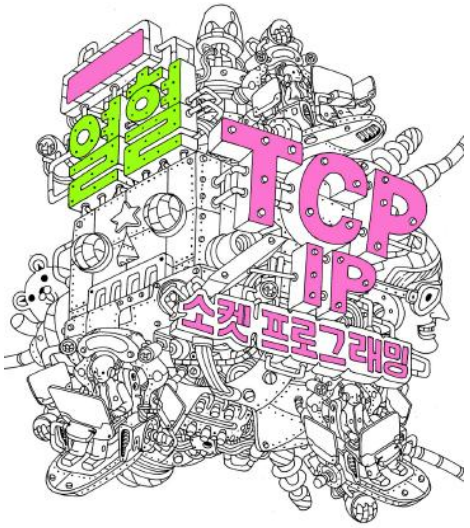
```
unsigned WINAPI NumberOfOthers(void *arg)
{
    int i, cnt=0;
    WaitForSingleObject(hEvent, INFINITE);
    for(i=0; str[i]!=0; i++)
    {
        if(str[i]!='A')
            cnt++;
    }
    printf("Num of others: %d \n", cnt-1);
    return 0;
}
```

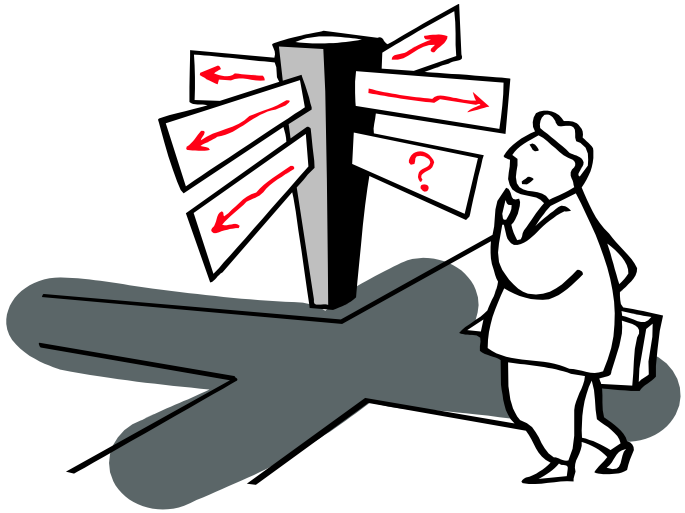
여기서 제시하는 채팅 클라이언트와 서버는 Chapter 18의 chat_serv.c와 chat_clnt.c를 단순히 윈도우 버전으로 변경한 것이니, 별도의 PPT 제공은 생략합니다.



Chapter 20-3. 윈도우 기반의 멀티 쓰레드 서버 구현

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판





Chapter 20이 끝났습니다. 질문 있으신지요?