

Chapter 06 멀티스레드: 윈도우

- 멀티스레드 프로그래밍의 필요성을 이해하고 기본 개념을 익힌다.
- 윈도우에서 멀티스레드를 이용한 다중 처리 서버를 작성할 수 있다.
- 윈도우의 스레드 동기화 기법을 이해하고 활용할 수 있다.

목차

01 스레드 기초

02 스레드 API

03 멀티스레드 TCP 서버

04 스레드 동기화

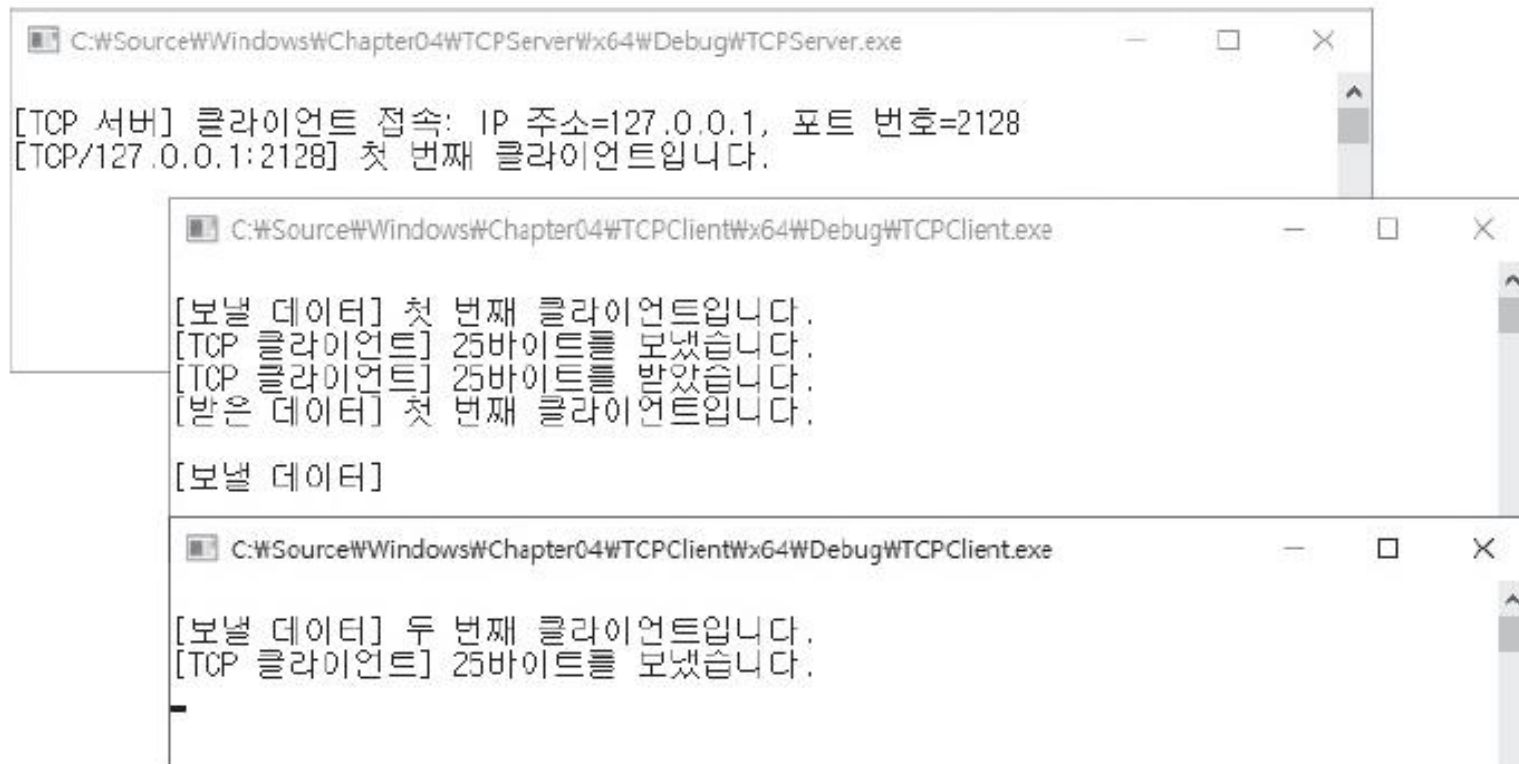
01 스레드 기초



스레드 기초 (1)

■ 소켓 응용 프로그램과 멀티스레드

- 4장에서 작성한 TCP 서버-클라이언트 예제에서 발생하는 문제 ①
 - 다중 처리 문제



The image shows three overlapping console windows from a Windows operating system. The top window is titled 'C:\Source\Windows\Chapter04\TCPServer\Wx64\Debug\TCPServer.exe' and contains the text: '[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=2128' and '[TCP/127.0.0.1:2128] 첫 번째 클라이언트입니다.'. The middle window is titled 'C:\Source\Windows\Chapter04\TCPClient\Wx64\Debug\TCPClient.exe' and contains the text: '[보낼 데이터] 첫 번째 클라이언트입니다.', '[TCP 클라이언트] 25바이트를 보냈습니다.', '[TCP 클라이언트] 25바이트를 받았습니다.', and '[받은 데이터] 첫 번째 클라이언트입니다.'. The bottom window is also titled 'C:\Source\Windows\Chapter04\TCPClient\Wx64\Debug\TCPClient.exe' and contains the text: '[보낼 데이터] 두 번째 클라이언트입니다.' and '[TCP 클라이언트] 25바이트를 보냈습니다.'. The windows are overlapping, with the bottom window partially obscured by the middle one, and the middle one partially obscured by the top one.

```
C:\Source\Windows\Chapter04\TCPServer\Wx64\Debug\TCPServer.exe
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=2128
[TCP/127.0.0.1:2128] 첫 번째 클라이언트입니다.

C:\Source\Windows\Chapter04\TCPClient\Wx64\Debug\TCPClient.exe
[보낼 데이터] 첫 번째 클라이언트입니다.
[TCP 클라이언트] 25바이트를 보냈습니다.
[TCP 클라이언트] 25바이트를 받았습니다.
[받은 데이터] 첫 번째 클라이언트입니다.

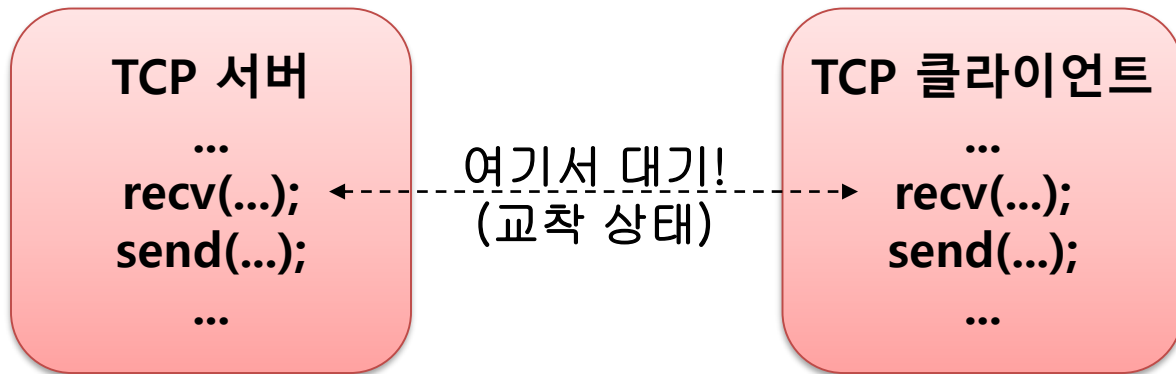
[보낼 데이터]

C:\Source\Windows\Chapter04\TCPClient\Wx64\Debug\TCPClient.exe
[보낼 데이터] 두 번째 클라이언트입니다.
[TCP 클라이언트] 25바이트를 보냈습니다.
```

스레드 기초 (2)

■ 소켓 응용 프로그램과 멀티스레드

- 4장에서 작성한 TCP 서버-클라이언트 예제에서 발생하는 문제 ②
 - 교착 상태 발생



스레드 기초 (3)

■ 다중 처리 문제 해결책

- 서버가 각 클라이언트와 통신하는 시간을 짧게 줄임
 - 장점: 구현이 쉬움, 가장 적은 시스템 자원 사용
 - 단점: 대용량 데이터를 전송하는 응용 프로그램을 구현하는 데 적합하지 않음, 각 클라이언트의 처리 지연 시간이 길어질 수 있음
- 각 클라이언트를 스레드를 이용해 독립적으로 처리
 - 장점: 소켓 입출력 모델에 비해 구현이 쉬움
 - 단점: 서버의 시스템 자원을 많이 사용
- 소켓 입출력 모델 사용(11~12장)
 - 장점: 소수의 스레드를 이용해 다수의 클라이언트를 처리
⇒ 상대적으로 적은 시스템 자원 사용
 - 단점: 구현이 어려움

스레드 기초 (4)

■ 교착 상태 발생 해결책

- 데이터 송수신 부분 잘 설계하기
 - 장점: 특별한 기법 없이 곧바로 구현 가능
 - 단점: 모든 경우에 대한 해결책은 될 수 없음
- 소켓에 타임아웃 옵션 적용하기
 - 장점: 구현이 쉬움
 - 단점: 다른 방법에 비해 성능이 떨어짐
- 년블로킹 소켓 사용하기(11~12장)
 - 장점: 조건을 만족하지 않더라도 소켓 함수가 즉시 리턴해 교착 상태 해결
 - 단점: 구현이 복잡, 시스템 자원(특히 CPU 시간) 낭비
- 소켓 입출력 모델 사용(11~12장)
 - 장점: 년블로킹 소켓의 단점을 보완하면서 교착 상태 해결
 - 단점: 첫 번째나 두 번째 해결책보다 구현이 어려움

스레드 기초 (5)

■ 스레드 기본 개념

■ 프로세스

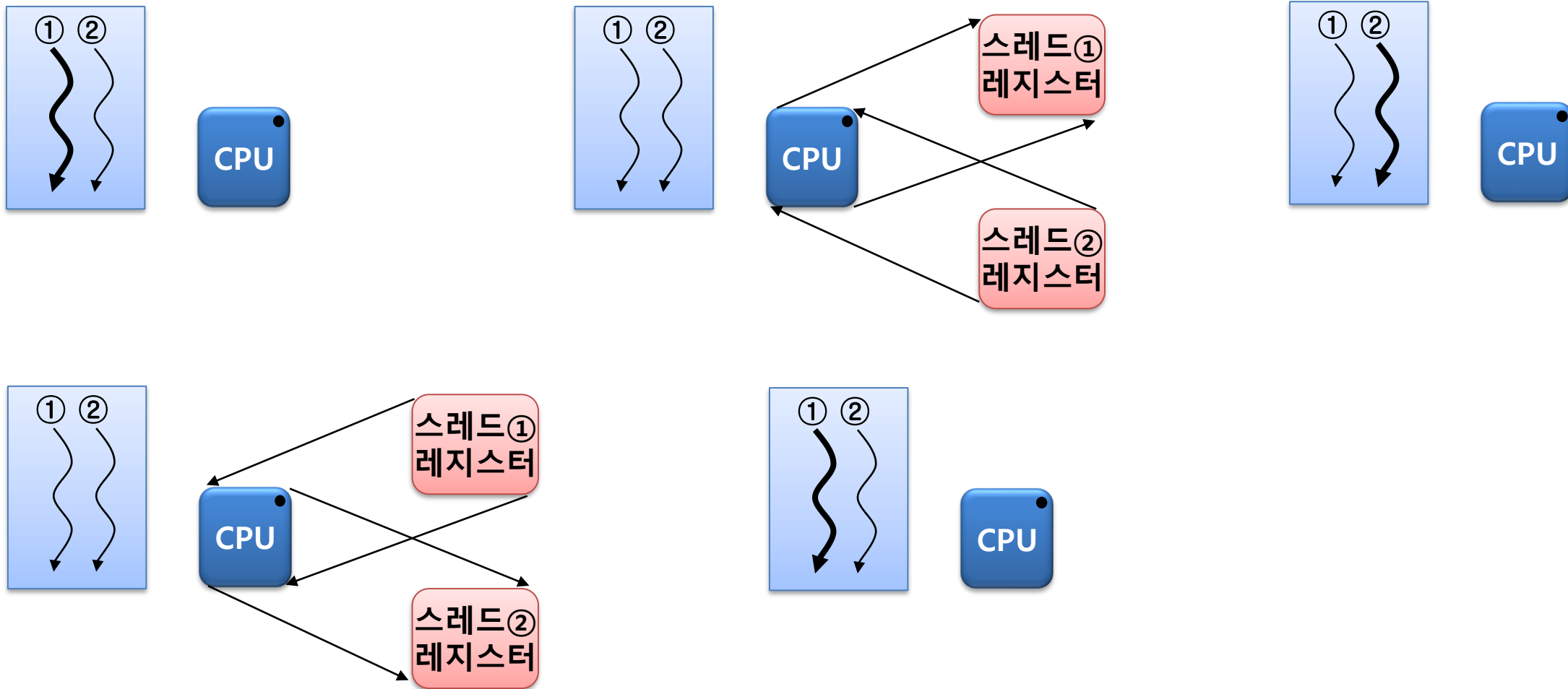
- 운영체제의 프로세스는 CPU 시간을 할당받아 실행 중인 프로그램
- 코드, 데이터, 리소스를 파일에서 읽어들이 작업을 수행하는 동적인 개념
- 일반 운영체제의 프로세스 = 윈도우 운영체제의 프로세스 + 스레드
 - 프로세스 : 코드, 데이터, 리소스를 파일에서 읽어들이 메모리 영역에 담고 있는 일종의 컨테이너(정적 개념)
 - 스레드 : CPU 시간을 할당받아 프로세스 메모리 영역에 있는 코드를 수행하고 데이터를 사용하는 실행 흐름(동적 개념)

■ 주 스레드 or 메인 스레드

- 응용 프로그램 실행 시 최초로 생성되는 스레드
- `main()` 함수 또는 `WinMain()` 함수에서 실행 시작

스레드 기초 (6)

■ 멀티스레드 동작 원리



스레드 기초 (7)

■ 스레드 수 확인

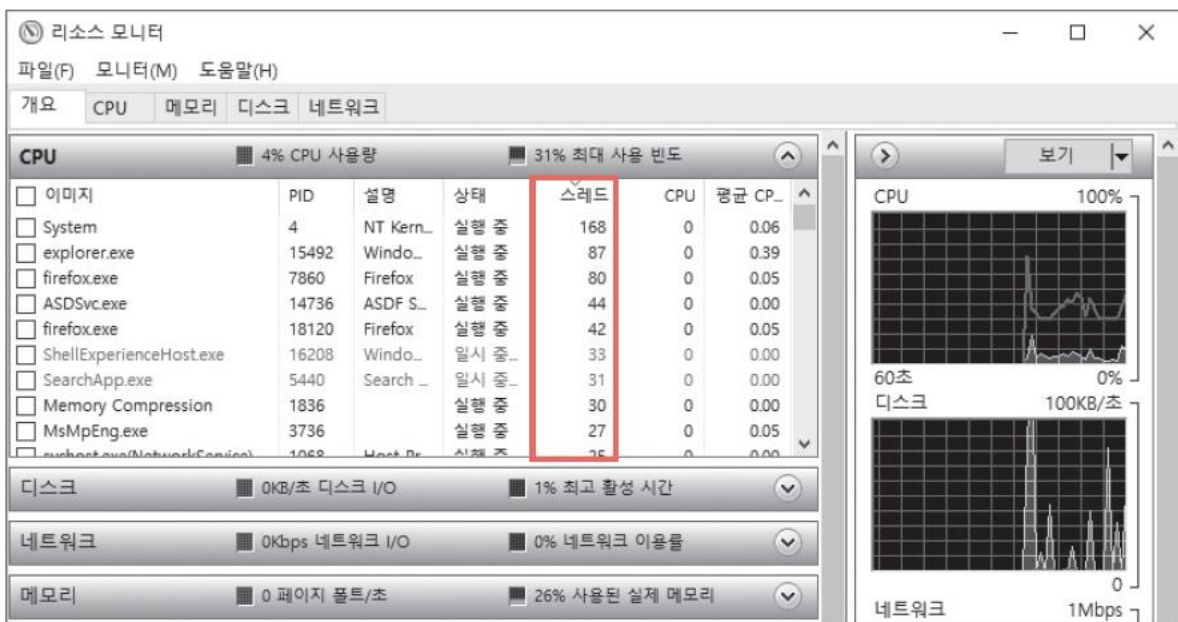


그림 6-4 윈도우 스레드 수 확인

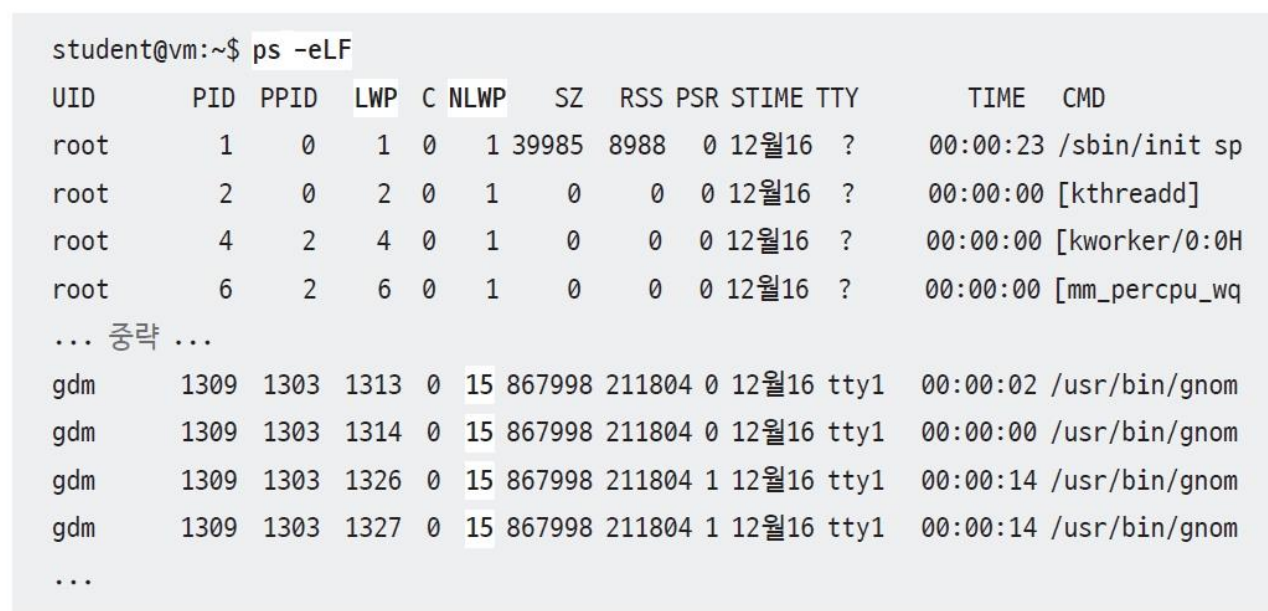


그림 6-5 리눅스 스레드 수 확인

02 스레드 API



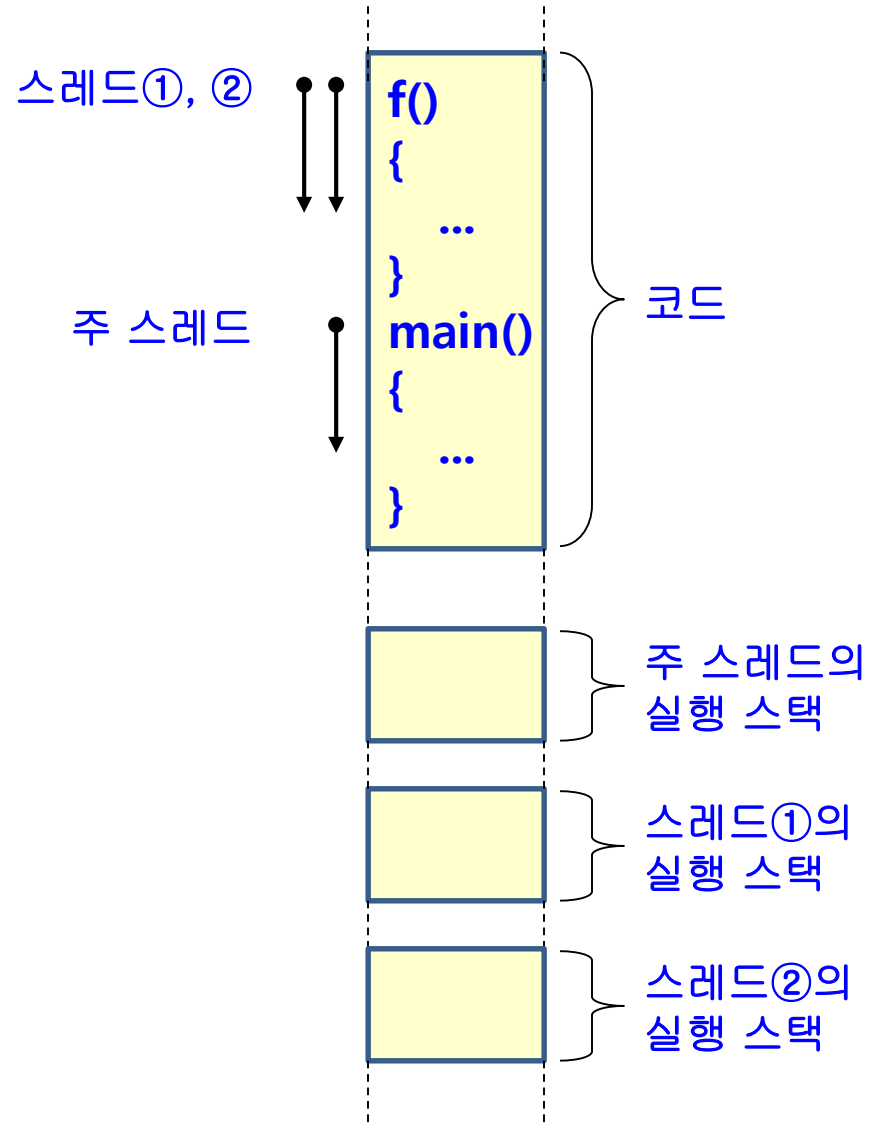
스레드 생성과 종료 (1)

■ 스레드 생성에 필요한 요소

- f() 함수의 시작 주소
 - C/C++ 프로그램에서는 함수 이름이 곧 그 함수의 시작 주소를 의미
 - 스레드 함수 : f() 함수와 같이 스레드 실행 시작점이 되는 함수
- f() 함수 실행 시 사용할 스택의 크기
 - 스레드 실행에 필요한 스택 생성은 운영체제가 자동으로 해주므로 응용 프로그램은 스택 크기만 알려주면 됨

스레드 생성과 종료 (2)

- 함수 두 개로 구성된 응용 프로그램에 스레드 세 개가 존재하는 상황



스레드 생성과 종료 (3)

■ CreateThread() 함수

- 스레드 생성 후 스레드 핸들을 리턴

```
#include <windows.h>

HANDLE CreateThread(
    ① LPSECURITY_ATTRIBUTES lpThreadAttributes,
    ② SIZE_T dwStackSize,
    ③ LPTHREAD_START_ROUTINE lpStartAddress,
    ④ LPVOID lpParameter,
    ⑤ DWORD dwCreationFlags,
    ⑥ LPDWORD lpThreadId
);
```

성공: 스레드 핸들, 실패: NULL

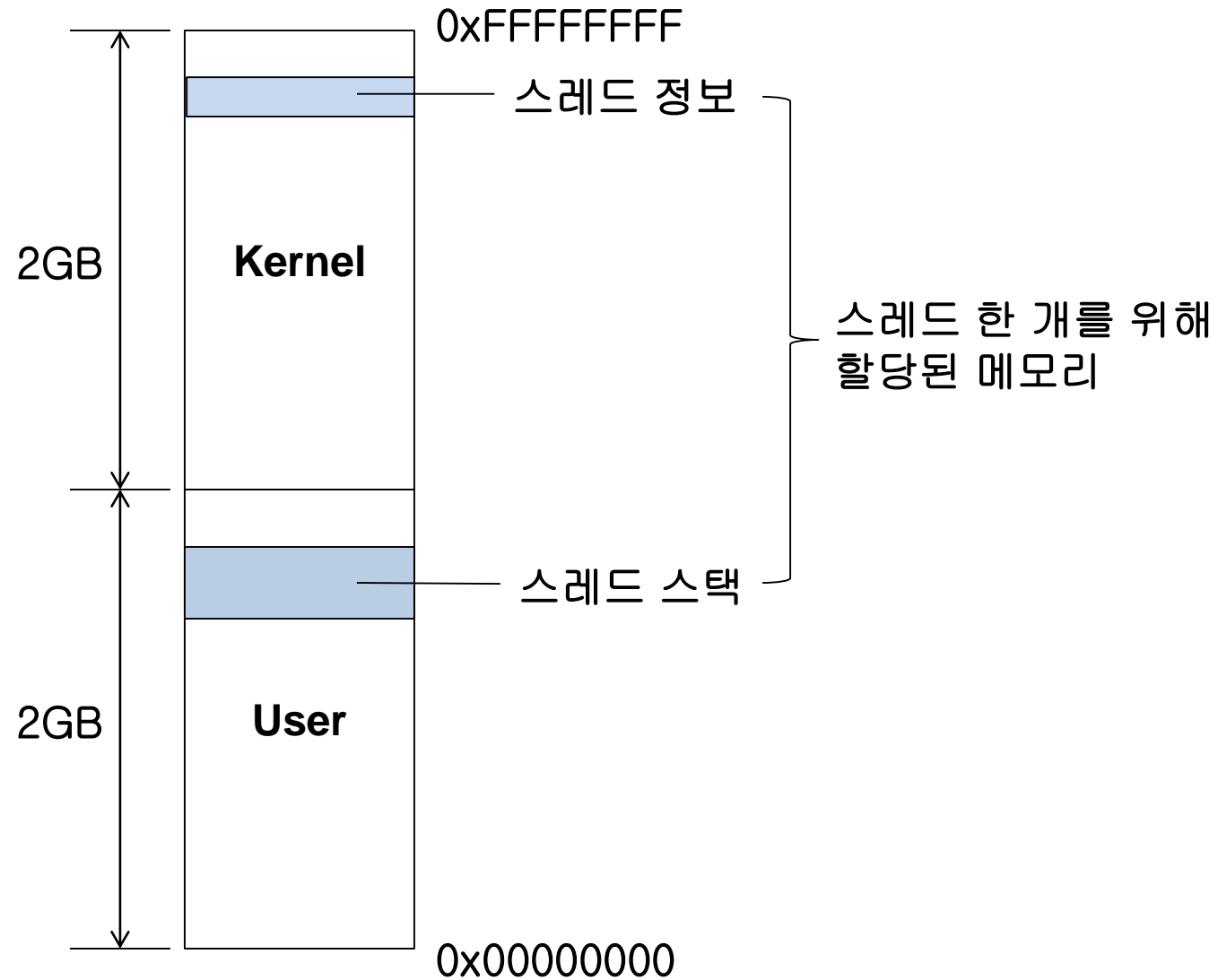
스레드 생성과 종료 (4)

■ 스레드 함수의 형태

```
DWORD WINAPI ThreadProc(LPVOID lpParameter)
{
    ...
}
```


스레드 생성과 종료 (5)

■ 스레드의 최대 개수 [32비트 윈도우]



스레드 생성과 종료 (6)

■ 스레드 종료 방법

- ① 스레드 함수가 리턴
- ② 스레드 함수 안에서 `ExitThread()` 함수를 호출
- ③ 다른 스레드가 `TerminateThread()` 함수를 호출해 스레드 강제 종료
- ④ 메인 스레드가 종료하면 프로세스 내 다른 모든 스레드가 강제 종료

스레드 생성과 종료 (7)

■ ExitThread() 함수와 TerminateThread() 함수

```
#include <windows.h>
void ExitThread(
    DWORD dwExitCode // 종료 코드
);
```

```
#include <windows.h>
BOOL TerminateThread(
    HANDLE hThread, // 종료할 스레드를 가리키는 핸들
    DWORD dwExitCode // 종료 코드
);
```

성공: 0이 아닌 값, 실패: 0

스레드 생성과 종료 (8)

■ 실습 6-1 스레드 생성과 종료, 인수 전달 연습

- ThreadTest1.cpp
- <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter06/ThreadTest1/ThreadTest1.cpp>

스레드 제어 (1)

■ 스레드 우선순위 변경하기

- CPU 스케줄링 or 스레드 스케줄링
 - 윈도우 운영체제가 각 스레드에 CPU 시간을 적절히 분배하기 위한 정책
- 스레드의 우선순위를 결정하는 요소
 - 프로세스 우선순위 : 우선순위 클래스
 - 스레드 우선순위 : 우선순위 레벨

스레드 제어 (2)

■ 우선순위 클래스

- REALTIME_PRIORITY_CLASS(실시간)
- HIGH_PRIORITY_CLASS(높음)
- ABOVE_NORMAL_PRIORITY_CLASS(높은 우선순위; 윈도우2000 이상)
- NORMAL_PRIORITY_CLASS(보통)
- BELOW_NORMAL_PRIORITY_CLASS(낮은 우선순위; 윈도우2000 이상)
- IDLE_PRIORITY_CLASS(낮음)

스레드 제어 (3)

- 프로세스의 우선순위 클래스 보기/변경하기

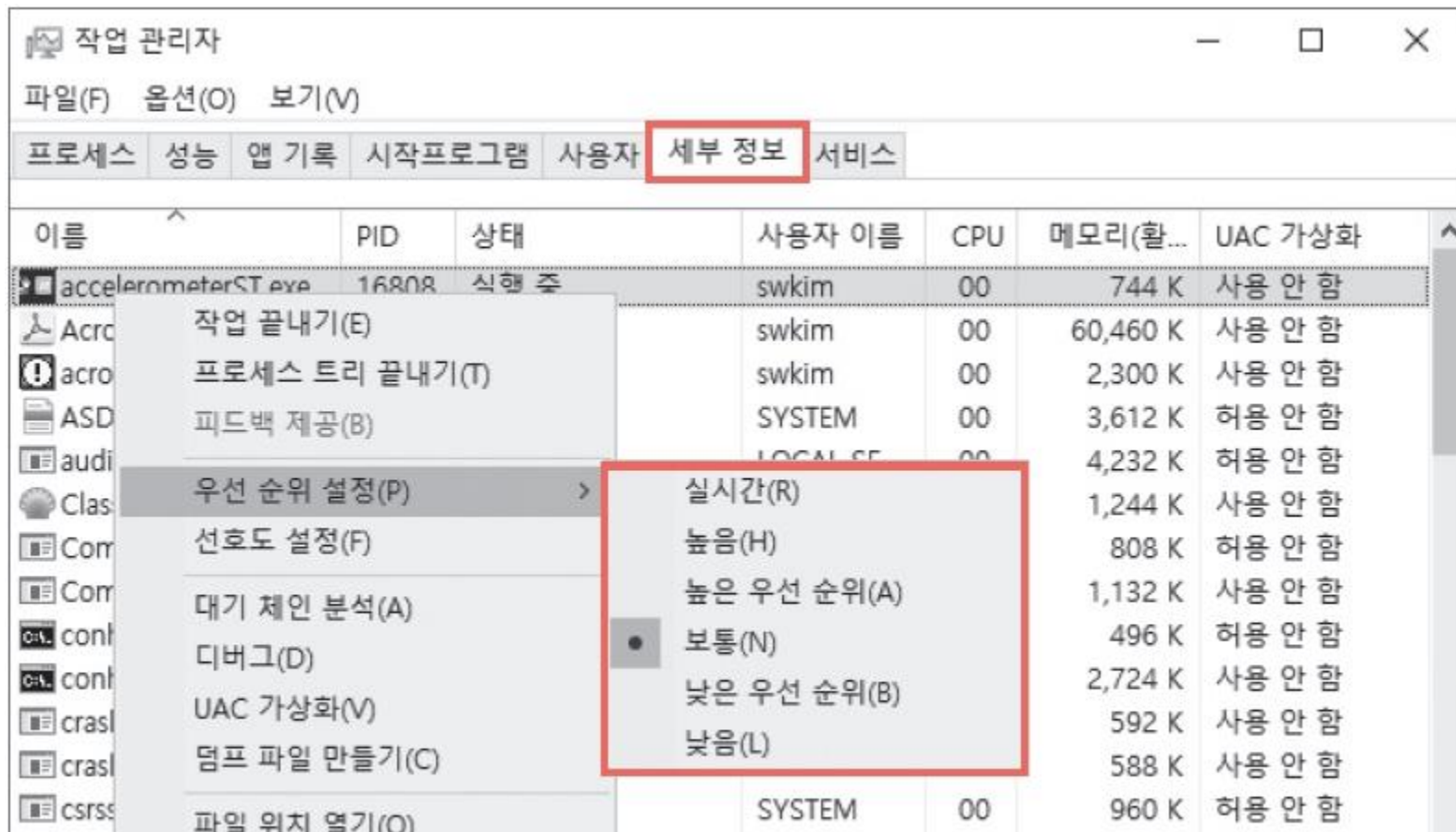


그림 6-7 프로세스의 우선순위 클래스 보기/변경하기

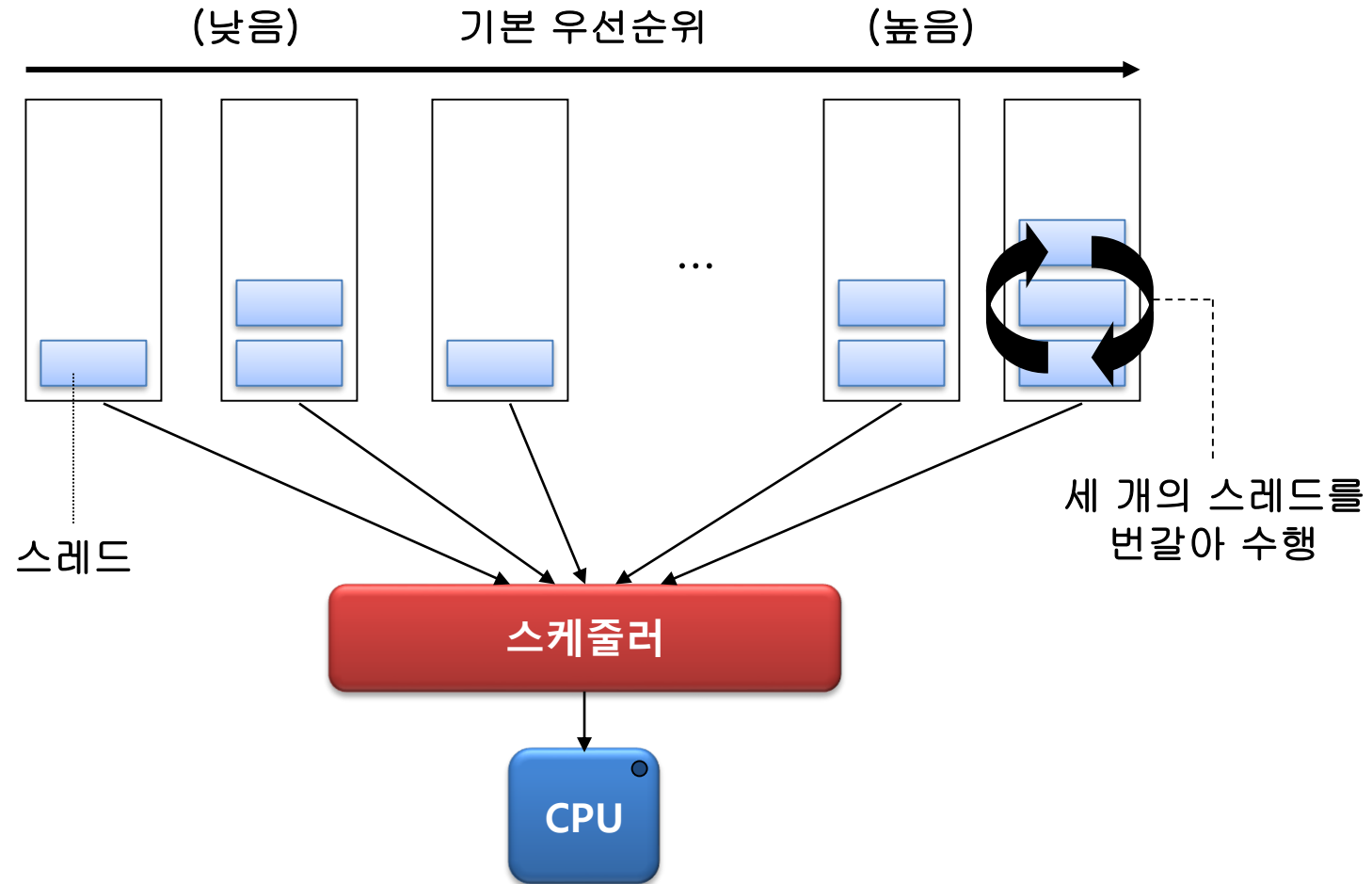
스레드 제어 (4)

■ 우선순위 레벨

- THREAD_PRIORITY_TIME_CRITICAL
- THREAD_PRIORITY_HIGHEST
- THREAD_PRIORITY_ABOVE_NORMAL
- THREAD_PRIORITY_NORMAL
- THREAD_PRIORITY_BELOW_NORMAL
- THREAD_PRIORITY_LOWEST
- THREAD_PRIORITY_IDLE

스레드 제어 (5)

■ 윈도우의 스레드 스케줄링 방식



스레드 제어 (6)

■ 우선순위 레벨 관련 API 함수

- SetThreadPriority() 함수는 우선순위 레벨을 변경할 때 사용
- GetThreadPriority() 함수는 현재의 우선순위 레벨을 얻을 때 사용

```
#include <windows.h>
BOOL SetThreadPriority(
    HANDLE hThread, // 스레드 핸들
    int nPriority    // 우선순위 레벨
);
```

성공: 0이 아닌 값, 실패: 0

```
#include <windows.h>
int GetThreadPriority(
    HANDLE hThread // 스레드 핸들
);
```

성공: 우선순위 레벨, 실패: THREAD_PRIORITY_ERROR_RETURN

스레드 제어 (7)

■ 실습 6-2 스레드 우선순위 변경 연습

- ThreadTest2.cpp
- <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter06/ThreadTest2/ThreadTest2.cpp>

스레드 종료 기다리기 (1)

■ WaitForSingleObject() 함수

- 특정 스레드가 종료할 때까지 기다리기

```
#include <windows.h>
DWORD WaitForSingleObject(
    ❶ HANDLE hHandle,
    ❷ DWORD dwMilliseconds
);
```

성공: WAIT_OBJECT_0 또는 WAIT_TIMEOUT, 실패: WAIT_FAILED

■ WaitForSingleObject() 함수 사용 예

```
HANDLE hThread = CreateThread(...);
DWORD retval = WaitForSingleObject(hThread, 1000);
if (retval == WAIT_OBJECT_0) { ... }      // 스레드 종료
else if (retval == WAIT_TIMEOUT) { ... }  // 타임아웃(스레드는 아직 종료 안 함)
else { ... }                             // 에러 발생
```

스레드 종료 기다리기 (2)

■ WaitForMultipleObjects() 함수

```
#include <windows.h>

DWORD WaitForMultipleObjects(
    ❶ DWORD nCount, const HANDLE *lpHandles,
    ❷ BOOL bWaitAll,
    ❸ DWORD dwMilliseconds
);
```

성공: $WAIT_OBJECT_0 \sim WAIT_OBJECT_0 + nCount - 1$ 또는 $WAIT_TIMEOUT$
실패: $WAIT_FAILED$

■ WaitForMultipleObjects() 함수 사용 예 ①

```
// 모든 스레드의 종료를 기다린다.
HANDLE hThread[2];
hThread[0] = CreateThread(...);
hThread[1] = CreateThread(...);
WaitForMultipleObjects(2, hThread, TRUE, INFINITE);
```

스레드 종료 기다리기 (3)

■ WaitForMultipleObjects() 함수 사용 예 ②

```
// 스레드 하나의 종료를 기다린다.  
HANDLE hThread[2];  
hThread[0] = CreateThread(...);  
hThread[1] = CreateThread(...);  
DWORD retval = WaitForMultipleObjects(2, hThread, FALSE, INFINITE);  
switch (retval) {  
    case WAIT_OBJECT_0:      // hThread[0] 종료  
        ...  
        break;  
    case WAIT_OBJECT_0 + 1: // hThread[1] 종료  
        ...  
        break;  
    case WAIT_FAILED:       // 오류 발생  
        ...  
        break;  
}
```

스레드 실행 일시 중지 및 재시작하기 (1)

■ 실행 일시 중지 함수 ①

```
#include <windows.h>
DWORD SuspendThread(
    HANDLE hThread // 스레드 핸들
);
```

성공: 중지 횟수, 실패: -1

■ 재실행 함수

```
#include <windows.h>
DWORD ResumeThread(
    HANDLE hThread // 스레드 핸들
);
```

성공: 중지 횟수, 실패: -1

스레드 실행 일시 중지 및 재시작하기 (2)

■ 실행 일시 중지 함수 ②

```
void Sleep(  
    DWORD dwMilliseconds // 밀리초  
);
```

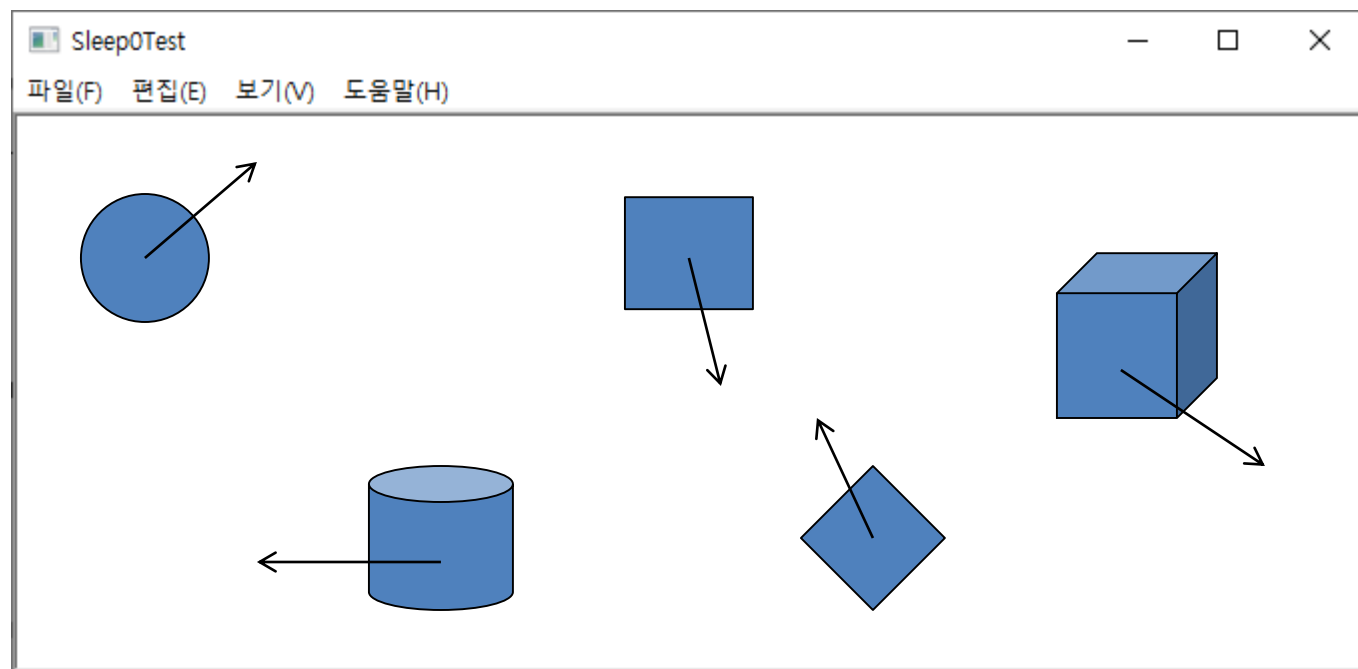

스레드 실행 일시 중지 및 재시작하기 (3)

■ 실습 6-3 스레드 실행 제어와 종료 기다리기 연습

- ThreadTest3.cpp
- <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter06/ThreadTest3/ThreadTest3.cpp>

스레드 실행 일시 중지 및 재시작하기 (4)

■ 빠르게 컨텍스트 전환하기



03 멀티스레드 TCP 서버



멀티스레드 TCP 서버 (1)

■ 기본 형태

```
DWORD WINAPI ProcessClient(LPVOID arg)
{
    // ③ 전달된 소켓 저장
    SOCKET client_sock = (SOCKET)arg;
    // ④ 클라이언트 정보 얻기
    addrlen = sizeof(clientaddr);
    getpeername(client_sock, (struct sockaddr *)&clientaddr, &addrlen);
    // ⑤ 클라이언트와 데이터 통신
    while (1) {
        ...
    }
    ...
}
```

멀티스레드 TCP 서버 (2)

■ 기본 형태(계속)

```
int main(int argc, char *argv[])
{
    ...
    while (1) {
        // ❶ 클라이언트 접속 수용
        client_sock = accept(listen_sock, ...);
        ...
        // ❷ 스레드 생성
        CreateThread(NULL, 0, ProcessClient, (LPVOID)client_sock, 0, NULL);
    }
    ...
}
```

멀티스레드 TCP 서버 (3)

■ 소켓에서 주소 정보 얻기

```
#include <windows.h>
int getpeername(
    SOCKET sock,
    struct sockaddr *addr,
    int *addrlen
);
```

성공: 0, 실패: SOCKET_ERROR

```
#include <windows.h>
int getsockname(
    SOCKET sock,
    struct sockaddr *addr,
    int *addrlen
);
```

성공: 0, 실패: SOCKET_ERROR

멀티스레드 TCP 서버 (4)

■ 실습 6-4 멀티스레드 TCP 서버 작성과 테스트

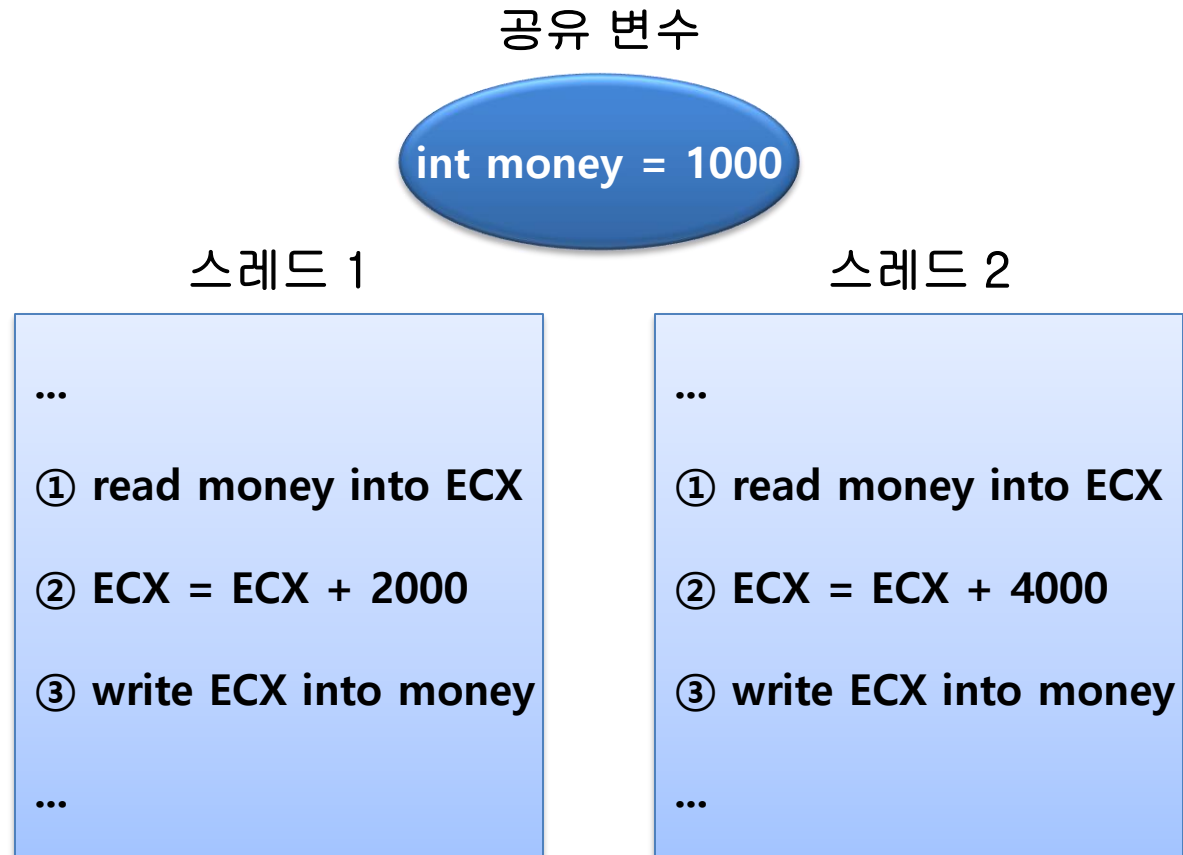
- ThreadTCPServer.cpp
- <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter06/ThreadTCPServer/ThreadTCPServer.cpp>

04 스레드 동기화



스레드 동기화 (1)

■ 스레드 동기화 필요성



스레드 동기화 (2)

■ 스레드 동기화 기법

종류	기능
임계 영역	공유 자원에 오직 한 스레드의 접근만 허용 (한 프로세스에 속한 스레드 간에만 사용 가능)
뮤텍스	공유 자원에 오직 한 스레드의 접근만 허용 (서로 다른 프로세스에 속한 스레드 간에도 사용 가능)
이벤트	사건 발생을 알려서 대기 중인 스레드를 깨움
세마포어	한정된 개수의 자원에 여러 스레드가 접근할 때, 자원을 사용할 수 있는 스레드 개수를 제한
대기 가능 타이머	시간과 관련된 조건이 만족되면 대기 중인 스레드를 깨움

스레드 동기화 (3)

■ 스레드 동기화가 필요한 상황

- ① 둘 이상의 스레드가 공유 자원에 접근
- ② 한 스레드가 작업을 완료한 후, 기다리고 있는 다른 스레드에 알려줌

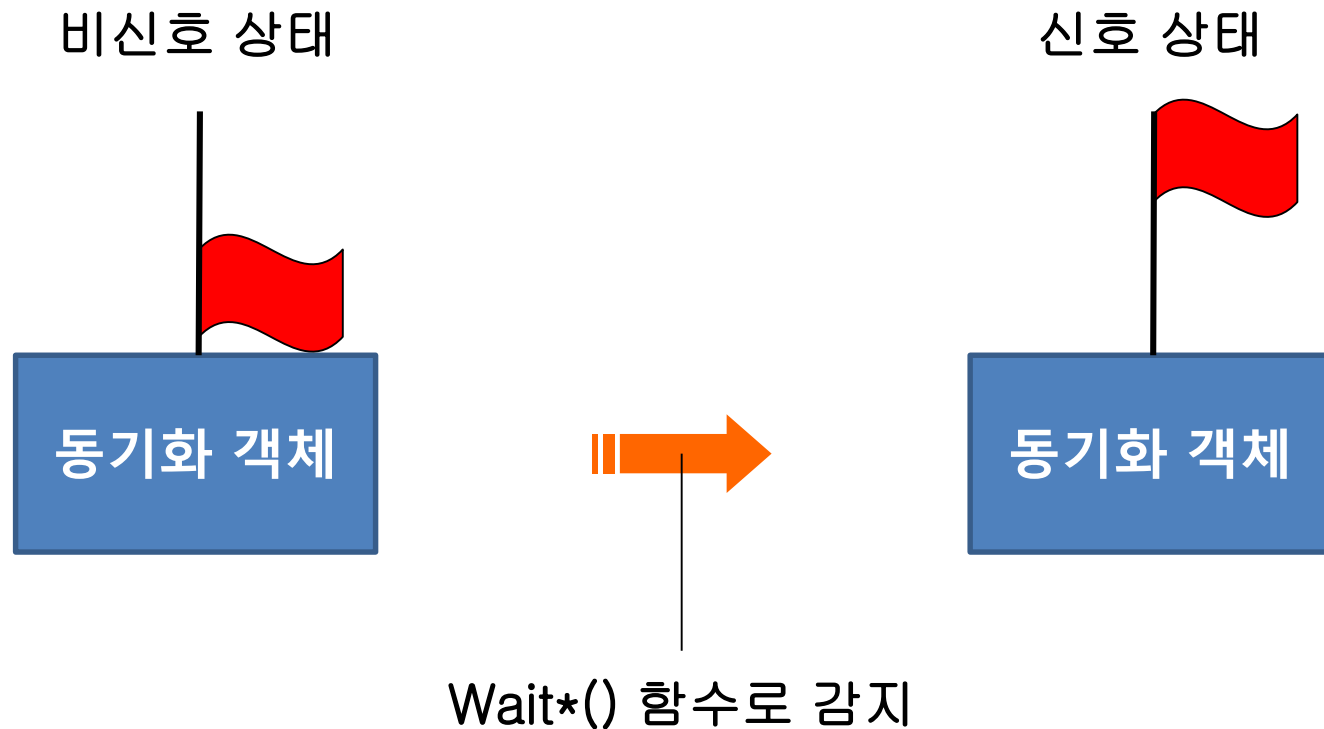
■ 스레드 동기화 원리



스레드 동기화 (4)

■ 동기화 객체의 특징

- Create*() 함수를 호출하면 커널 메모리 영역에 동기화 객체가 생성되고, 이에 접근할 수 있는 핸들이 리턴
- 평소에는 비신호 상태로 있다가 특정 조건이 만족되면 신호 상태가 됨. 비신호 상태에서 신호 상태로 변화 여부는 Wait*() 함수를 사용해 감지
- 사용이 끝나면 CloseHandle() 호출



임계 영역 (1)

■ 임계 영역

- 두 개 이상의 스레드가 공유 자원에 접근할 때, 오직 한 스레드만 접근을 허용해야 하는 경우에 사용

■ 특징

- 프로세스의 사용자 메모리 영역에 존재하는 단순한 구조체이므로 한 프로세스에 속한 스레드 간 동기화에만 사용
- 다른 동기화 객체보다 빠르고 효율적

임계 영역 (2)

■ 임계 영역 사용 예

```
#include <windows.h>

CRITICAL_SECTION cs;

DWORD WINAPI MyThread1(LPVOID arg)
{
    ...

    EnterCriticalSection(&cs);
    // 공유 자원 접근
    LeaveCriticalSection(&cs);
    ...
}

DWORD WINAPI MyThread2(LPVOID arg)
{
    ...

    EnterCriticalSection(&cs);
    // 공유 자원 접근
    LeaveCriticalSection(&cs);
    ...
}

int main(int argc, char *argv[])
{
    ...

    InitializeCriticalSection(&cs);
    // 스레드를 둘 이상 생성하여 작업을 진행한다.
    // 생성한 모든 스레드가 종료할 때까지 기다린다.
    DeleteCriticalSection(&cs);
    ...
}
```

CRITICAL_SECTION 구조체 타입의 전역 변수를 선언한다. 일반 동기화 객체는 Create*() 함수를 호출하여 커널 메모리 영역에 생성하지만, 임계 영역은 사용자 메모리 영역에 (대개는 전역 변수 형태로) 생성한다.

공유 자원에 접근하기 전에 호출한다. 공유 자원을 사용하고 있는 스레드가 없다면 곧바로 리턴한다. 하지만 공유 자원을 사용 중인 스레드가 있다면 리턴하지 못하고 호출 스레드는 곧바로 대기 상태가 된다.

공유 자원 사용을 마치면 호출한다. 이때 EnterCriticalSection() 함수에서 대기 중인 다른 스레드가 있다면 하나만 선택되어 깨어난다.

임계 영역을 사용하기 전에 호출하여 초기화한다.

임계 영역을 사용하는 모든 스레드가 종료하면 이 함수를 호출하여 삭제한다.

임계 영역 (3)

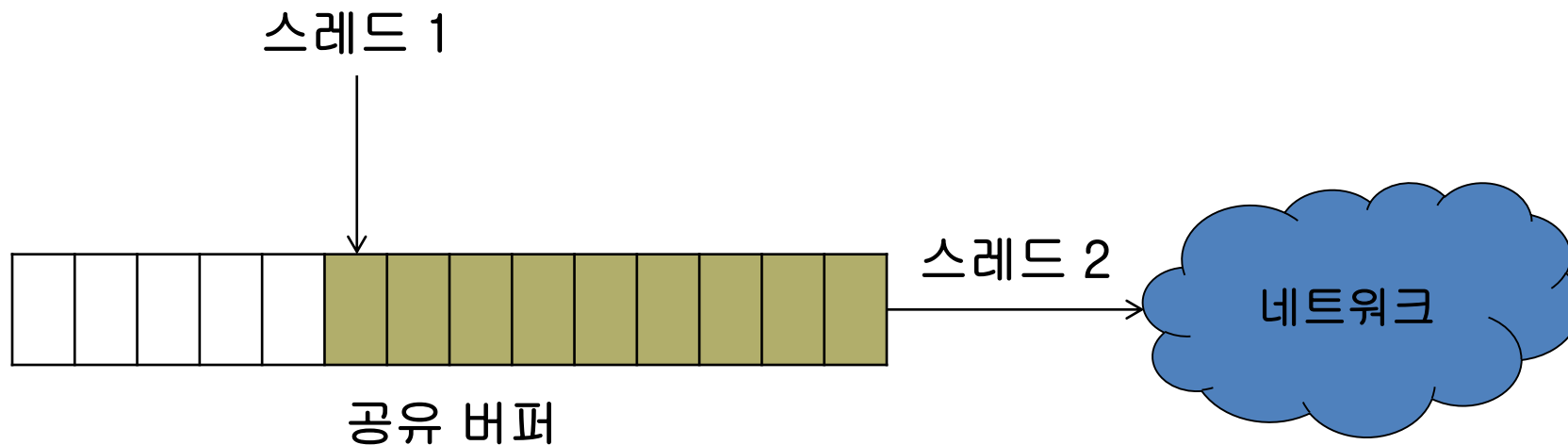
■ 실습 6-5 임계 영역 연습

- CriticalSections.cpp
- <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter06/CriticalSections/CriticalSections.cpp>

임계 영역 (4)

■ 임계 영역 사용 시 주의 사항

- 임계 영역만으로는 어느 스레드가 먼저 리소스를 사용할지 결정할 수 없음. 어느 스레드가 먼저 EnterCriticalSection() 함수를 호출할지 알 수 없음



↑ 스레드 실행 순서 제어가 필요한 상황

이벤트 (1)

■ 이벤트

- 사건 발생을 다른 스레드에 알리는 동기화 기법

■ 이벤트를 사용하는 전형적인 절차

- ❶ 이벤트를 비신호 상태로 생성
- ❷ 한 스레드가 작업을 진행하고, 나머지 스레드는 이벤트에 대해 `Wait*()` 함수를 호출해 이벤트가 신호 상태가 될 때까지 대기
- ❸ 스레드가 작업을 완료하면 이벤트를 신호 상태로 변경
- ❹ 대기 중인 스레드 중 한 개 혹은 전부가 깨어남. 깨어난 스레드는 후속 작업을 함
- ❺ 이벤트가 필요하지 않으면 `CloseHandle()` 함수를 호출해 이벤트를 제거

이벤트 (2)

■ 이벤트 상태 변경

```
BOOL SetEvent(HANDLE hEvent);    // 비신호 상태 → 신호 상태  
BOOL ResetEvent(HANDLE hEvent); // 신호 상태 → 비신호 상태
```

■ 이벤트의 종류

■ 자동 리셋 이벤트

- 이벤트를 신호 상태로 바꾸면, 대기 중인 스레드 중 하나만 깨운 후 자동으로 비신호 상태가 됨

■ 수동 리셋 이벤트

- 이벤트를 신호 상태로 바꾸면, 대기 중인 스레드를 모두 깨운 후 계속 신호 상태를 유지함

이벤트 (3)

■ 이벤트 생성

```
#include <windows.h>
HANDLE CreateEvent(
    ① LPSECURITY_ATTRIBUTES lpEventAttributes,
    ② BOOL bManualReset,
    ③ BOOL bInitialState,
    ④ LPCTSTR lpName
);
```

성공: 이벤트 핸들, 실패: NULL

이벤트 (4)

■ 실습 6-6 이벤트 연습

- Events.cpp
- <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter06/Events/Events.cpp>

