

## 2024 스마트네트워크서비스 기말 프로젝트

- 스마트네트워크서비스 기말 프로젝트
  - 10개조 \* 10분 = 100분/ → 2시간
  - 조편성 : 2~3 명
    - <https://docs.google.com/spreadsheets/d/1RS-bSNHCf72tvhnMthSh96VXjlggn4ij7O2BC6gangQ/edit?usp=sharing>
  - 구현내용
    - 16개 재료 이용 → 시나리오 작성 → ui 프로그램 작성
    - 16개 재료는 교재에 이벤트 핸들러 함수가 있는 내용임
    - 그 외 필요하다고 생각되는 재료 추가 가능
    - 모든 플랫폼에서 구현 가능 (모바일 플랫폼 포함)
- 스마트네트워크서비스 기말 프로젝트 제출내용
  - 제출 : 12월 2일 (월) 수업시작전,
    - 단 수업 진행시 12월 4일 (수) 제출 연기 할 수도 있음
    - 수업 상황에 따라 다름
  - 배점 : 구현 내용의 품질 및 재료의 구현 갯수에 따라 A(100)/B(90)/C(80)
  - 코드: zip해서 가상 강의실에 업로드
  - 발표 영상 4분 이내 : 유튜브에 올려서 발표시간에 영상 재생 가능함
  - 발표 자료 (ppt등): 직접 발표시
  - 자료 : 조원, 성명, 학번 정보가 꼭 있어야 함
  - 모든 조원이 같은 내용을 zip해서 각자 올리기 바랍니다

### 1. ifconfig로 ip 구성 내용 확인

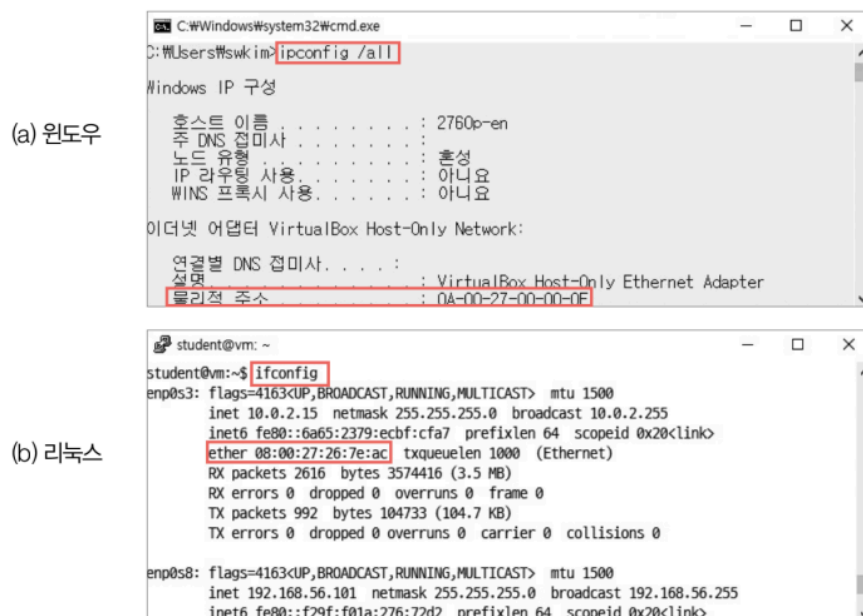


그림 1-4 물리적 주소 확인

### 2. 바이트 정렬 함수 : 호스트 바이트와 네트워크 바이트 주소 변환 과정

## ByteOrder.cpp

```
1 #include "..\..\Common.h"
2
3 int main(int argc, char *argv[])
4 {
5     // 윈속 초기화
6     WSADATA wsa;
7     if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
8         return 1;
9
10    u_short x1 = 0x1234;
11    u_long y1 = 0x12345678;
12    u_short x2;
13    u_long y2;
14
15    // 호스트 바이트 -> 네트워크 바이트
16    printf("[호스트 바이트 -> 네트워크 바이트]\n");
17    printf("%#x -> %#x\n", x1, x2 = htons(x1));
18    printf("%#x -> %#x\n", y1, y2 = htonl(y1));
19
20    // 네트워크 바이트 -> 호스트 바이트
21    printf("\n[네트워크 바이트 -> 호스트 바이트]\n");
22    printf("%#x -> %#x\n", x2, ntohs(x2));
23    printf("%#x -> %#x\n", y2, ntohl(y2));
24
25    // 잘못된 사용 예
26    printf("\n[잘못된 사용 예]\n");
27    printf("%#x -> %#x\n", x1, htonl(x1));
28
29    // 윈속 종료
30    WSACleanup();
31    return 0;
32 }
```



```
Microsoft Visual Studio 디버그 콘솔
[호스트 바이트 -> 네트워크 바이트]
0x1234 -> 0x3412
0x12345678 -> 0x78563412
[네트워크 바이트 -> 호스트 바이트]
0x3412 -> 0x1234
0x78563412 -> 0x12345678
[잘못된 사용 예]
0x1234 -> 0x34120000
```

### 3. IP 주소 변환 함수 : inet\_pton (), inet\_ntop (), hton(), ntoh()

- ① 응용 프로그램이 소켓 주소 구조체를 초기화하고 소켓 함수에 넘겨준다. 단, SocketFunc() 함수는 임의의 소켓 함수를 나타낸다.

```
// 소켓 주소 구조체를 초기화한다.
struct sockaddr_in addr;           // IPv4용 소켓 주소 구조체
memset(&addr, 0, sizeof(addr));    // 0으로 채운다.
addr.sin_family = AF_INET;         // 주소 체계: IPv4
inet_pton(AF_INET, "147.46.114.70", &addr.sin_addr); // IP 주소: 문자열 → 숫자
addr.sin_port = htons(9000);       // 포트 번호: 호스트 바이트 정렬 → 네트워크 바이트 정렬

// 소켓 함수를 호출한다.
SocketFunc(..., (struct sockaddr *)&addr, sizeof(addr), ...); // 소켓 함수 호출
```

```
// 소켓 함수를 호출한다.
struct sockaddr_in addr; // IPv4용 소켓 주소 구조체
int addrlen = sizeof(addr); // 리눅스에서는 int 대신 socklen_t 타입을 사용해야 한
SocketFunc(..., (struct sockaddr *)&addr, &addrlen, ...); // 소켓 함수 호출

// 소켓 주소 구조체를 사용한다.
char ipaddr[INET_ADDRSTRLEN]; // 문자열 형태의 IPv4 주소를 담을 버퍼
inet_ntop(AF_INET, &clientaddr.sin_addr,
          ipaddr, sizeof(ipaddr)); // IP 주소: 숫자 → 문자열
printf("\n[TCP 서버] 클라이언트 접속: IP 주소=%s, 포트 번호=%d\n",
       ipaddr, ntohs(clientaddr.sin_port)); // 소켓 주소 구조체 사용
```

```
47
48 // 원속 종료
49 WSACleanup();
50 return 0;
51 }
```

Microsoft Visual Studio 디버그 콘솔

IPv4 주소(변환 전) = 147.46.114.70  
IPv4 주소(변환 후) = 0x46722e93  
IPv4 주소(다시 변환 후) = 147.46.114.70

IPv6 주소(변환 전) = 2001:0230:abcd:ffab:0023:eb00:ffff:1111  
IPv6 주소(변환 후) = 0x20010230abcdffab0023eb00ffff1111  
IPv6 주소(다시 변환 후) = 2001:230:abcd:ffab:23:eb00:ffff:1111

1행은 원래의 IPv4 주소를, 2행은 inet\_pton() 함수를 사용하여 32비트 숫자(네트워크 바이트 정렬)로 변환한 결과를, 3행은 inet\_ntop() 함수를 사용하여 문자열로 변환한 결과를 보여준다.

## NameResolution.cpp

```
1 #include "..\..\Common.h"
2
3 #define TESTNAME "www.google.com"
4
5 // 도메인 이름 -> IPv4 주소
6 bool GetIPAddr(const char *name, struct in_addr *addr)
7 {
8     struct hostent *ptr = gethostbyname(name);
9     if (ptr == NULL) {
10         err_display("gethostbyname()");
11         return false;
12     }
13     if (ptr->h_addrtype != AF_INET)
14         return false;
15     memcpy(addr, ptr->h_addr, ptr->h_length);
16     return true;
17 }
18
19 // IPv4 주소 -> 도메인 이름
20 bool GetDomainName(struct in_addr addr, char *name, int namelen)
21 {
22     struct hostent *ptr = gethostbyaddr((const char *)&addr,
23         sizeof(addr), AF_INET);
24     if (ptr == NULL) {
25         err_display("gethostbyaddr()");
26         return false;
27     }
28     if (ptr->h_addrtype != AF_INET)
29         return false;
```

5. Server 상태 확인 netstat -a -n -p tcp | findstr 9000 명령어 GUI

```

7 {
8     int retval;
9
10    // 윈속 초기화
11    WSADATA wsa;
12    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
13        return 1;
14
15    // 소켓 생성
16    SOCKET listen_sock = socket(AF_INET, SOCK_STREAM, 0);
17    if (listen_sock == INVALID_SOCKET) err_quit("socket()");
18
19    // bind()
20    struct sockaddr_in serveraddr;
21    memset(&serveraddr, 0, sizeof(serveraddr));
22    serveraddr.sin_family = AF_INET;
23    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
24    serveraddr.sin_port = htons(SERVERPORT);
25    retval = bind(listen_sock, (struct sockaddr *)&serveraddr, sizeof(serveraddr));
26    if (retval == SOCKET_ERROR) err_quit("bind()");
27
28    // listen()
29    retval = listen(listen_sock, SOMAXCONN);
30    if (retval == SOCKET_ERROR) err_quit("listen()");
31
32    // 데이터 통신에 사용할 변수
33    SOCKET client_sock;
34    struct sockaddr_in clientaddr;
35    int addrlen;

```



**NOTE** 리눅스에서는 'netstat -a -n --tcp | grep 9000' 명령을 실행한다.

## 6. TCP 서버 함수 상태 GUI 표시

### 1 TCP 서버 함수

일반적으로 TCP 서버는 [그림 4-7]과 같은 순서로 소켓 함수를 호출한다.

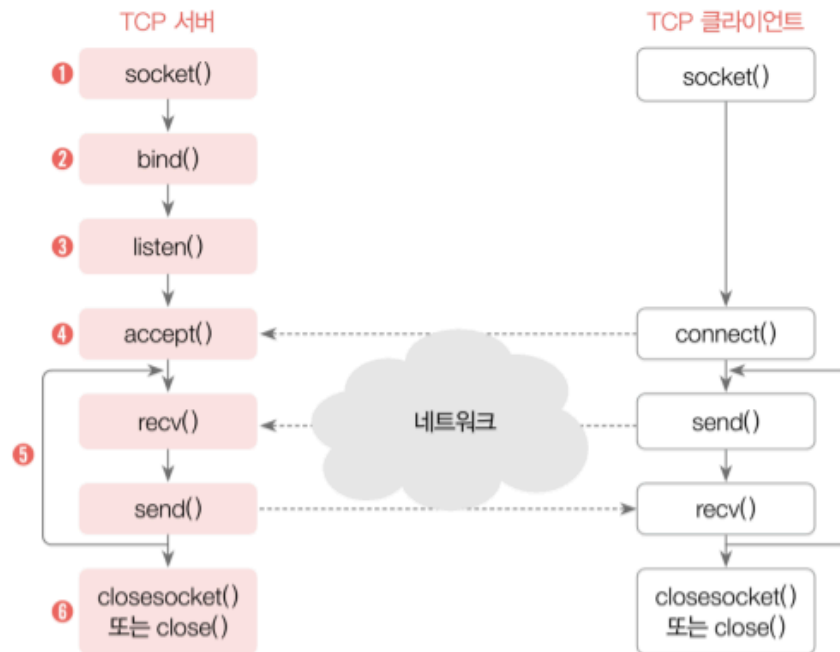


그림 4-7 TCP 서버 핵심 함수

## 7. TCP 클라이언트 함수 상태 GUI 표시

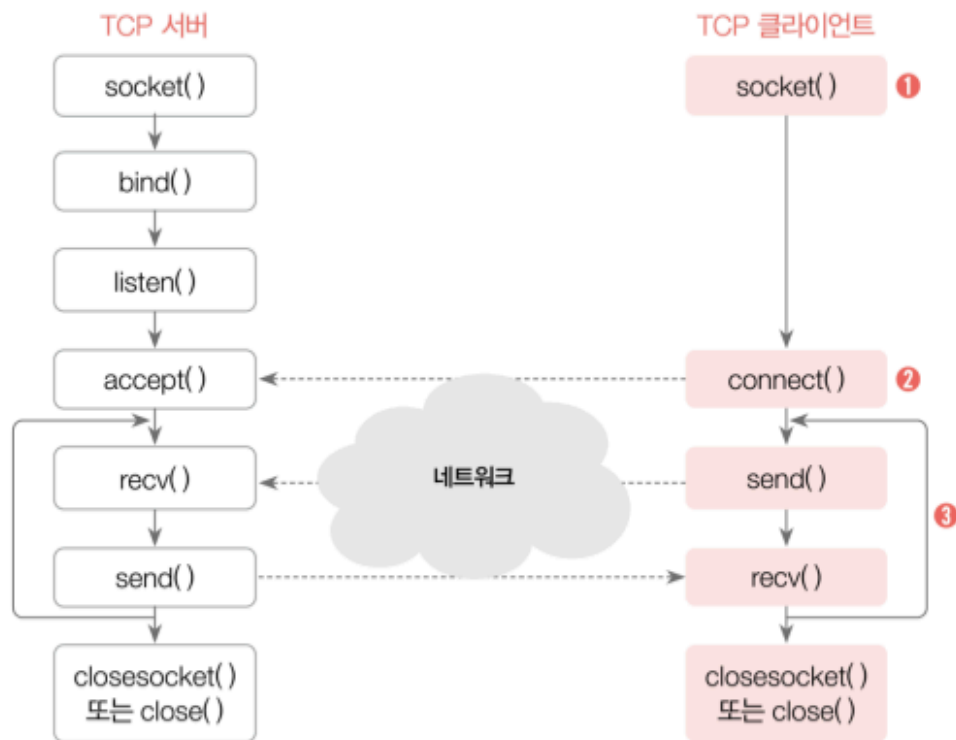


그림 4-8 TCP 클라이언트 핵심 함수

## 8. 소켓 데이터 구조체 상태 GUI 표시 (수신 버퍼, 송신 버퍼 상태)

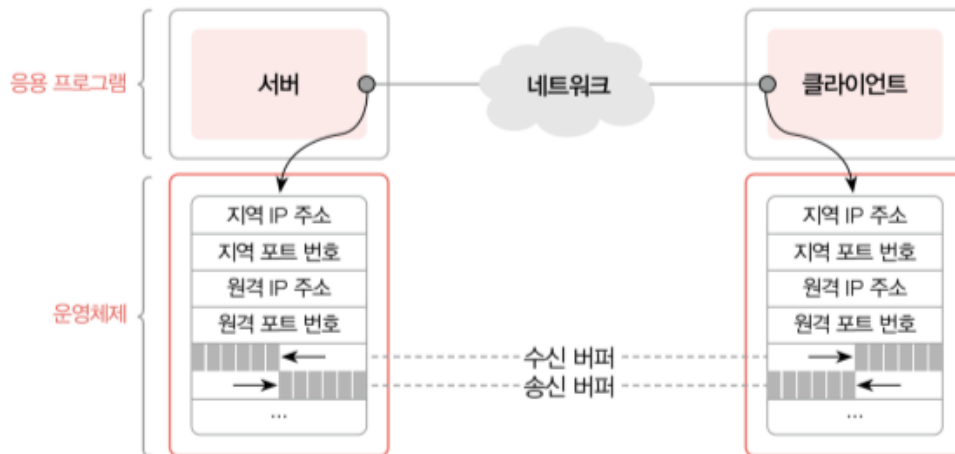


그림 4-9 소켓 데이터 구조체(2)

## 9. 네트워크 그림판 응용 프로그램

기능을 제공한다.



그림 5-1 네트워크 그림판 프로그램

## 10. 고정길이 데이터 전송

```
C:\Source\Windows\Chapter05\TCPServer_Fixed\x64\Debug\TCPServer_Fixed.exe
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=13574
[TCP/127.0.0.1:13574] 안녕하세요#####
[TCP/127.0.0.1:13574] 반가워요#####
[TCP/127.0.0.1:13574] 오늘따라 할 이야기가 많을 것 같네요#####
[TCP/127.0.0.1:13574] 저도 그렇네요#####
[TCP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=13574

Microsoft Visual Studio 디버그 콘솔
[TCP 클라이언트] 50바이트를 보냈습니다.
[TCP 클라이언트] 50바이트를 보냈습니다.
[TCP 클라이언트] 50바이트를 보냈습니다.
[TCP 클라이언트] 50바이트를 보냈습니다.

C:\Source\Windows\Chapter05\TCPClient_Fixed\x64\Debug\TCPClient_Fixed.exe(9580
프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

TCPServer\_Fixed.cpp

## 11. 가변길이 데이터 전송

### 실습 5-2 가변 길이 데이터 전송 연습

TCPServer\_Variable과 TCPClient\_Variable이라는 콘솔 응용 프로그램 프로젝트 두 개를 생성한다. 그런 후에 다음 실행 결과와 같이 클라이언트가 가변 길이 데이터를 보내면, 서버가 데이터를 정확히 구분하여 읽도록 구현해보자.

```
C:\Source\Windows\Chapter05\TCPServer_Variable\x64\Debug\TCPServer_Variable...
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=11265
[TCP/127.0.0.1:11265] 안녕하세요
[TCP/127.0.0.1:11265] 반가워요
[TCP/127.0.0.1:11265] 오늘따라 할 이야기가 많을 것 같네요
[TCP/127.0.0.1:11265] 저도 그렇네요
[TCP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=11265

Microsoft Visual Studio 디버그 콘솔
[TCP 클라이언트] 11바이트를 보냈습니다.
[TCP 클라이언트] 9바이트를 보냈습니다.
[TCP 클라이언트] 36바이트를 보냈습니다.
[TCP 클라이언트] 14바이트를 보냈습니다.

C:\Source\Windows\Chapter05\TCPClient_Variable\x64\Debug\TCPClient_Variable.exe(
2936 프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

TCPServer\_Variable.cpp



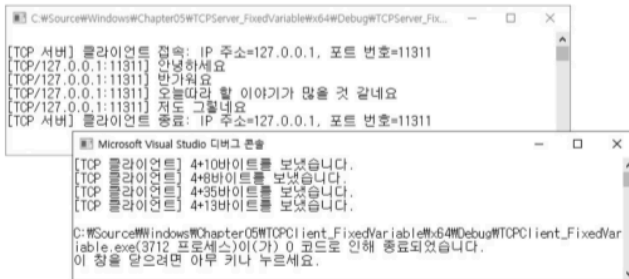
## 12. 고정 길이 + 가변 길이 데이터 전송

### ③ 고정 길이 + 가변 길이 데이터 전송

송신 측에서 가변 길이 데이터의 크기를 미리 계산할 수 있다면 '고정 길이 데이터 + 가변 길이 데이터' 전송이 효과적이다. 수신 측에서는 '① 고정 길이 데이터 수신, ② 가변 길이 데이터 수신', 단 두 번의 데이터 읽기 작업으로 가변 길이 데이터의 경계를 구분하여 읽을 수 있다.

#### 실습 5-3 고정 길이 + 가변 길이 데이터 전송 연습

TCPServer\_FixedVariable과 TCPClient\_FixedVariable이라는 콘솔 응용 프로그램 프로젝트 두 개를 생성한다. 그런 후에 다음 실행 결과와 같이 클라이언트가 [고정 길이 데이터 + 가변 길이 데이터]를 보내면, 서버가 데이터를 정확히 구분하여 읽도록 구현해보자.



TCPServer\_FixedVariable.cpp

## 13. 데이터 전송 후 종료

## 4 데이터 전송 후 종료

### 실습 5-4 데이터 전송 후 종료 연습

데이터 전송 후 종료 방식은 EOR로 특별한 데이터 패턴 대신 연결 종료를 사용하는 일종의 가변 길이 데이터 전송 방식이다. TCPServer\_CloseOnTransfer와 TCPClient\_CloseOnTransfer라는 콘솔 응용 프로그램 프로젝트 두 개를 생성해 다음 실행 결과와 같이 클라이언트가 데이터를 하나 보낼 때마다 접속과 접속 종료를 반복하도록 구현해보자.



The image shows two overlapping console windows from Microsoft Visual Studio. The top window, titled 'C:\Source\Windows\Chapter05\TCPServer\_CloseOnTransfer\x64\Debug\TCPServer\_...', displays the server's log. It shows three successful connections from IP 127.0.0.1 on ports 11408, 11410, and 11411. Each connection is followed by a '종료' (End) message. The bottom window, titled 'C:\Source\Windows\Chapter05\TCPClient\_CloseOnTransfer\x64\Debug\TCPClient\_CloseOnTransfer.exe(9316 프로세스)이(가) 0 코드로 인해 종료되었습니다.', shows the client's log. It displays four messages: '10바이트를 보냈습니다.', '8바이트를 보냈습니다.', '35바이트를 보냈습니다.', and '13바이트를 보냈습니다.', indicating the data sent in each connection.

TCPServer\_CloseOnTransfer.cpp

## 14. 멀티 스레드 동작 (멀티스레드 TCP 서버 작성과 테스트)

### 실습 6-4 멀티스레드 TCP 서버 작성과 테스트

4장 TCPServer 예제에 멀티스레드 기법을 적용하여, 접속한 클라이언트마다 스레드를 하나씩 생성하여 처리하도록 만들어보자. 이렇게 하면 각 클라이언트는 이전에 접속한 다른 클라이언트와 관계없이 독립적인 서비스를 받을 수 있다는 장점이 있다. 이때 기존 TCPClient 예제는 전혀 변경할 필요가 없다는 점도 알아두자.

- 1 ThreadTCPServer라는 콘솔 응용 프로그램 프로젝트를 생성하고, 4장 TCPServer 코드를 복사해서 추가한다. 변경할 부분을 굵게 표시했으니 참고하여 코드를 수정하고 분석해보자.

ThreadTCPServer.cpp

2 실행 결과는 다음과 같다. 4장 TCPServer 예제와 달리 두 개 이상의 클라이언트가 접속해도 독립적으로 처리가 진행된다.



## 15. 임계영역 연습

### 실습 6-5 임계 영역 연습

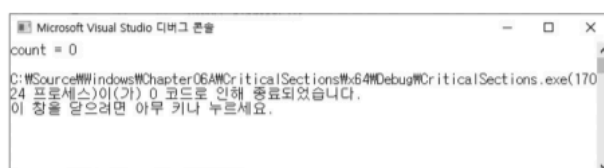
두 스레드가 공유 자원에 접근할 때 문제가 발생하는 상황을 만들고, 이를 임계 영역으로 해결해보자.

1 CriticalSections라는 콘솔 응용 프로그램 프로젝트를 생성하고 다음 코드를 입력한다.

CriticalSections.cpp

```
1 #include <windows.h>
2 #include <stdio.h>
3
4 #define MAXCNT 100000000
5 int count = 0;
6
7 DWORD WINAPI MyThread1(LPVOID arg)
8 {
9     for (int i = 0; i < MAXCNT; i++) {
10         count += 2;
11     }
12     return 0;
13 }
14
15 DWORD WINAPI MyThread2(LPVOID arg)
```

극단적인 상황에서 지나치게 세밀한 단위로 스레드 동기화를 하고 있으므로 성능 저하가 두드러진다. 하지만 실전에서 이와 같은 상황은 잘 발생하지 않으므로 스레드 동기화의 성능은 크게 문제되지 않는다.



## 16. 이벤트 연습

### 실습 6-6 이벤트 연습

데이터를 생성하여 공유 버퍼에 저장하는 한 개의 스레드와 공유 버퍼에서 데이터를 읽어서 처리하는 두 개의 스레드를 생성하자(그림 6-12). 이 경우 한 스레드만 버퍼에 접근할 수 있도록 해야 하며, 접근 순서도 정해야 한다. 스레드 실행 순서에 대한 제약 사항은 다음과 같다.

- 스레드 ①이 쓰기를 완료한 후 스레드 ②나 스레드 ③이 읽을 수 있다. 이때 스레드 ②와 스레드 ③ 중 한 개만 버퍼 데이터를 읽을 수 있으며, 일단 한 스레드가 읽기 시작하면 다른 스레드는 읽을 수 없다.
- 스레드 ②나 스레드 ③이 읽기를 완료하면 스레드 ①이 다시 쓰기를 할 수 있다.

이 문제는 공유 데이터에 접근하는 순서가 중요하므로 이벤트를 사용하는 것이 좋다. 쓰기 완료와 읽기 완료를 알릴 목적으로 이벤트 두 개(hWriteEvent, hReadEvent)를 생성하여 사용한다.

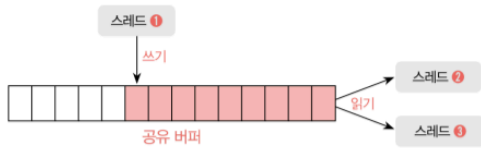


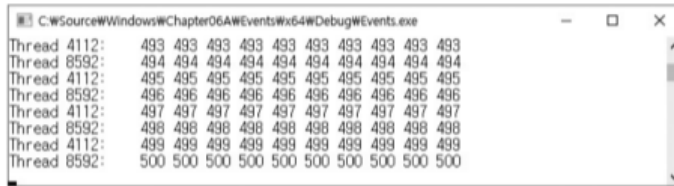
그림 6-12 예제 프로그램 구조

1 Events라는 콘솔 응용 프로그램 프로젝트를 생성하고 다음 코드를 입력한다.

Events.cpp

```
1 #include <windows.h>
2 #include <stdio.h>
3
4 #define BUFSIZE 10
5
6 HANDLE hWriteEvent;
7 HANDLE hReadEvent;
8 int buf[BUFSIZE];
9
10 DWORD WINAPI WriteThread(LPVOID arg)
11 {
```

- 2 실행 결과는 다음과 같다. 총 500번에 걸쳐 버퍼에 데이터를 저장할 때마다 두 스레드 중 하나가 대기 상태에서 깨어나 버퍼 데이터를 읽고 화면에 출력한다. 출력 행마다 스레드 ID를 표시하므로 어느 스레드가 데이터를 읽어서 처리했는지 알 수 있다.



```
C:\Source\Windows\Chapter06\AWEvents\Wx64\Debug\WEvents.exe
Thread 4112: 493 493 493 493 493 493 493 493 493 493
Thread 8592: 494 494 494 494 494 494 494 494 494 494
Thread 4112: 495 495 495 495 495 495 495 495 495 495
Thread 8592: 496 496 496 496 496 496 496 496 496 496
Thread 4112: 497 497 497 497 497 497 497 497 497 497
Thread 8592: 498 498 498 498 498 498 498 498 498 498
Thread 4112: 499 499 499 499 499 499 499 499 499 499
Thread 8592: 500 500 500 500 500 500 500 500 500 500
```