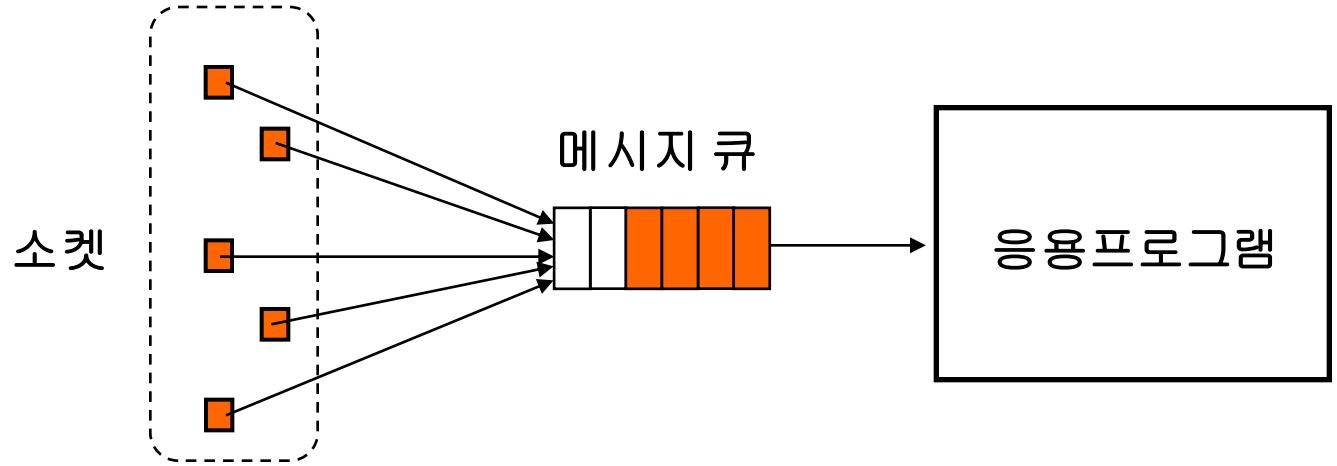
Network Programming - 03

afewhee@gmail.com

- WSAAsyncSelect 모델
- WSAEventSelect 모델

- WSAAsyncSelect 모델
  - WSAAsyncSelect() 함수 사용
  - I/O 멀티 플렉싱을 메시지 큐로 구현
  - 소켓 이벤트 ➔ 메시지로 전환 ➔ 메시지 큐에 저장 ➔ 응용프로그램에 전달
  - select 모델과 마찬가지로 멀티쓰레드를 사용하지 않고 여러 개의 소켓 이벤트를 처리

- 동작 원리



- WSAAsyncSelect의 I/O 절차

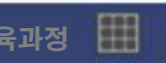
  - WSAAsyncSelect() 함수를 이용 소켓 + 메시지 + 네트워크 이벤트를 연결
    ➔ 성공: 0, 실패: SOCKET_ERROR

  - 네트워크 이벤트 ➔ 윈도우 메시지 발생 ➔ 윈도우 프로시저 순으로 호출 됨
  - 윈도우 프로시저에서 받은 메시지 종류에 따라 적절한 소켓 함수를 호출, 처리

  - 큐의 이용으로 네트워크 이벤트에 대한 동기화처리와, Receive용 쓰레드가 필요 없어 사용하기 편리함. 게임 클라이언트, 접속 인원이 수십 명 정도인 서버에 적합

```
Ex)
// 사용자 정의 윈도우 메시지 정의
#define MSG_NETWORK (WM_USER+1)
...
SOCKET scH ... Create...

// Net Event Binding with Message ID
WSAAsyncSelect(scH
      , hWnd                          // 메시지 프로시저 윈도우 핸들
      , MSG_NETWORK
      , FD_READ|FD_WRITE|FD_ACCEPT|FD_CONNECT|FD_CLOSE);
```

- 네트워크 이벤트 상수 값
  - FD_ACCEPT: 클라이언트가 접속
  - FD_READ: 데이터 수신이 있는 경우
  - FD_WRITE: 데이터 송신이 있는 경우
  - FD_CLOSE: 연결한 상대편이 접속 종료
  - FD_CONNECT: 서버로 접속이 완료
  - FD_OOB: Out-of-Bound 긴급 메시지 수신

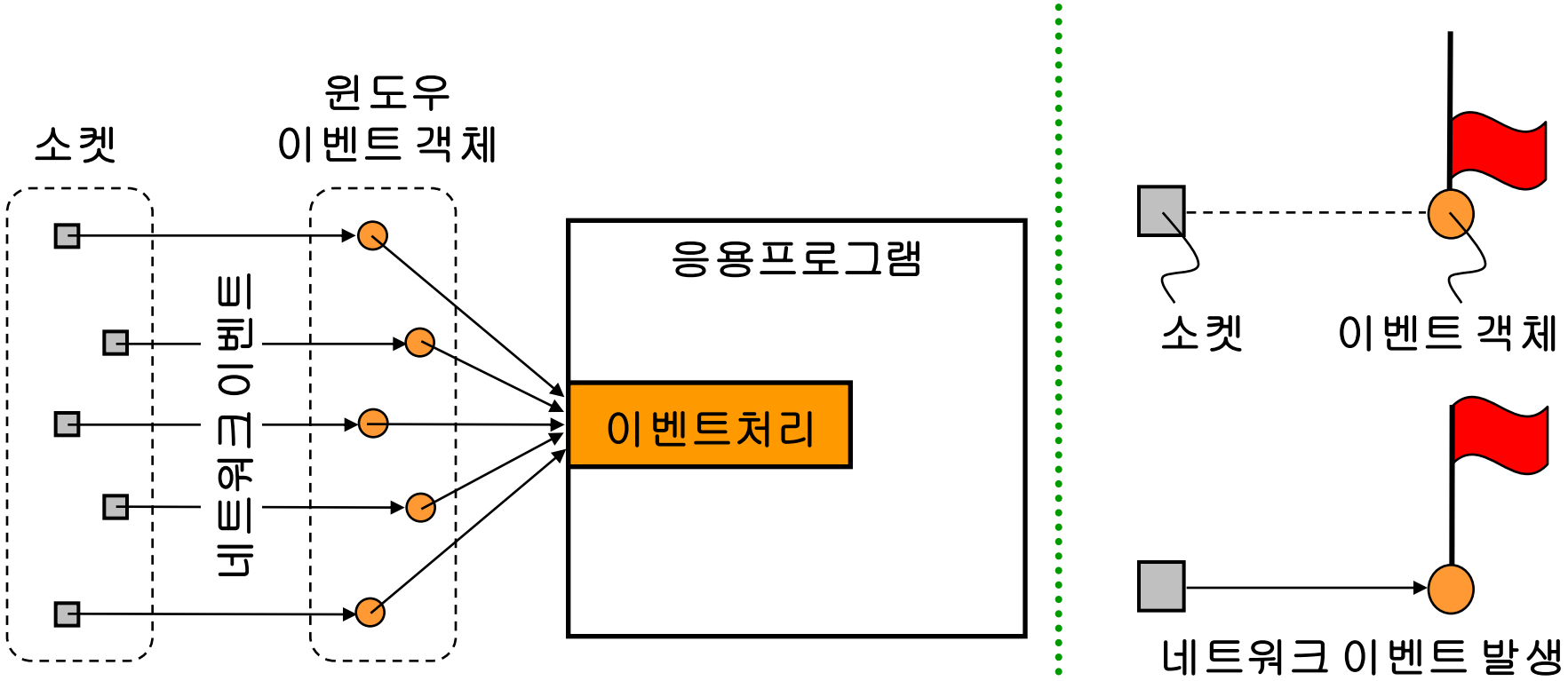


1. WSAAsyncSelect - 윈도우 프로시저 처리 예

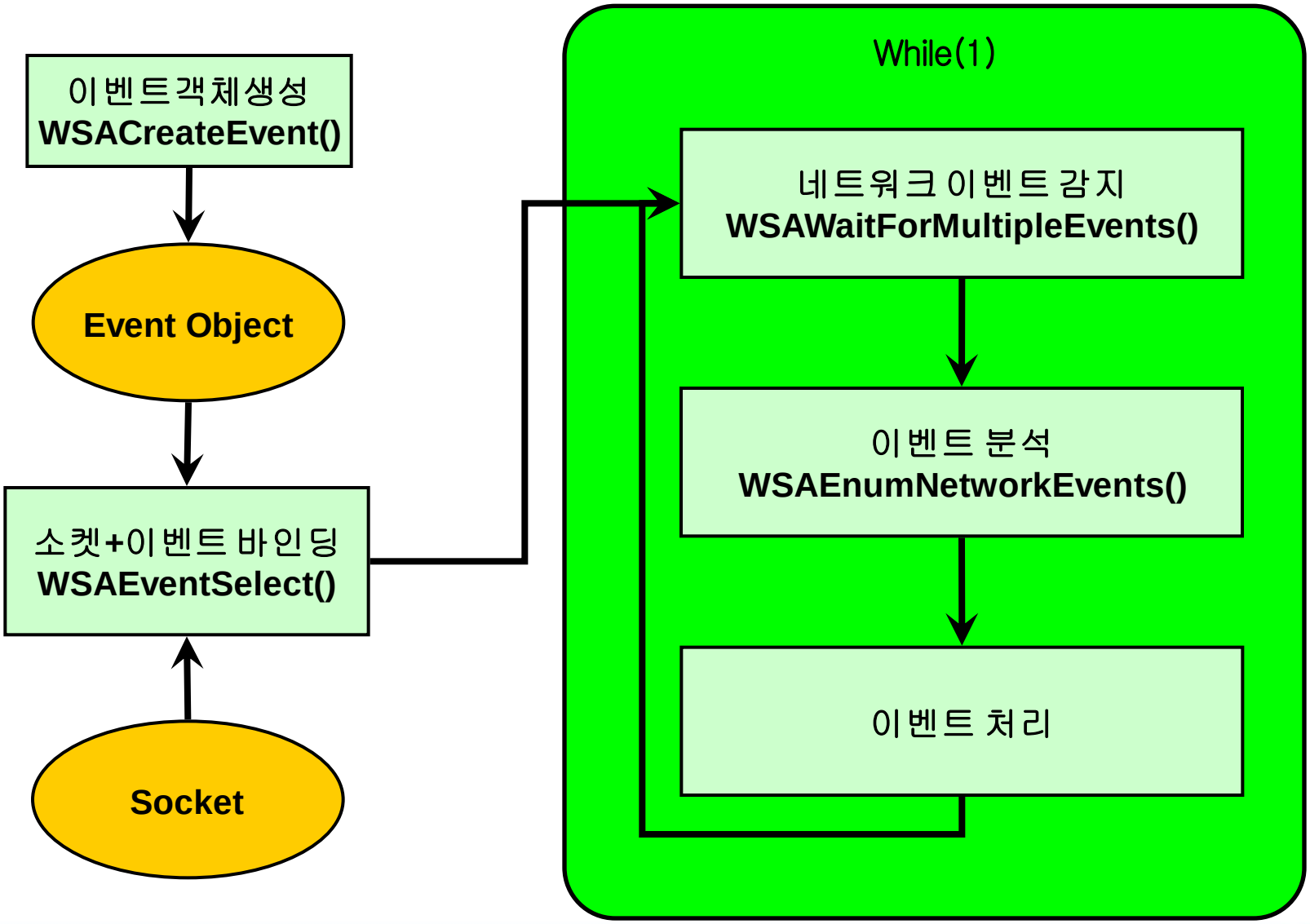
```
LRESULT WINAPI WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    if(MSG_NETWORK == msg )
    {
        SOCKET   scHost   = (SOCKET)wParam;
        DWORD    dError   = WSAGETSELECTERROR(lParam);
        DWORD    dEvent   = WSAGETSELECTEVENT(lParam);

        if(dError)  // 비 정상적인 에러 체크
        ...
        else
            if(FD_ACCEPT == dEvent) // Accept 메시지(서버부분)
            ...
            else if(FD_CONNECT == dEvent)    // connection 메시지(클라이언트부분)
            ...
            else if(FD_WRITE == dEvent) // Sending 메시지
            ...
            else if(FD_READ == dEvent)  // Receive 메시지
            int iRcv=recv(...);
            ...
            else if(FD_CLOSE == dEvent) // 접속 해제 메시지
            ...
```

- WSAEventSelect 모델
  - WSAEventSelect() 함수를 이용 ➔ {소켓 + 이벤트 객체+이벤트 내용}를 바인딩
    - 이벤트 내용은 FD_READ 등, AsyncSelect와 동일
  - 소켓 이벤트 ➔ 이벤트 객체 ➔ 이벤트 처리 순으로 네트워크의 I/O를 처리
  - 소켓에 바인딩 된 이벤트 객체를 관찰함으로써 멀티쓰레드를 사용하지 않고 여러 개의 소켓을 처리
  - 메시지 큐를 이용하는 AsyncSelect 보다 유연
  - 수신을 위한 Work Thread가 필요하나 네트워크 이벤트 발생 ➔ 이벤트 객체 Signaled 상태 일 때만 쓰레드를 사용하므로 효율이 좋음

동작 원리



- 구현 순서



2. WSAEventSelect - 주요 함수 내용

- 이벤트 객체 생성: WSAEVENT WSACreateEvent()
  - 성공: non-signaled 상태, manual-reset 모드 Event를 생성해야 WSAEnumNetworkEvents() 에서 이벤트 해석 가능
  - 실패: WSA_INVALID_EVENT

- 이벤트 객체 해제: BOOL WSACloseEvent (WSAEVENT hEvent)
  - 성공: TRUE, 실패: FALSE

- {소켓 + 이벤트 객체 + 이벤트 내용} 바인딩:

```
int WSAEventSelect(
        SOCKET s,                       // 감시 대상이 되는 소켓
        WSAEVENT hEventObject,          // 소켓의 변화를 확인하기 위한 Event 객체 핸들
        long lNetworkEvents             // 확인하고자 하는 이벤트 종류들
)
```

  - 성공: 0, 실패: SOCKET_ERROR

- 네트워크 이벤트 감지:

```
DWORD WSAWaitForMultipleEvents(
        DWORD cEvents,                  // 검사 대상의 수
        const WSAEVENT FAR *lphEvents, // 검사 대상 배열
        BOOL fWaitAll,                  // FALSE 전달 시 하나의 변화로도 리턴
        DWORD dwTimeout,                // 타임-아웃 설정
        BOOL fAlertable                 // For Completion Routine...
)
```

  - 성공: WSA_WAIT_EVENT_0 ~ WSA_WAIT_EVENT_0 + cEvents-1또는 WSA_WAIT_TIMEOUT. 실패: WSA_WAIT_FAILED

- 이벤트 내용 분석:

```
int WSAEnumNetworkEvents(
          SOCKET s,                             // 변화가 생긴 소켓의 핸들
          WSAEVENT hEventObject,                // 변화가 생긴 소켓의 Event 오브젝트 핸들
          LPWSANETWORKEVENTS lpNetworkEvents // 발생한 변화에 대한 정보를 채우기 위한 포인터
)
```

  - 해당 이벤트 객체를 non-signaled 상태로 전환. 성공: 0, 실패: SOCKET_ERROR

- 소켓을 생성할 때마다 WSACreateEvent() 함수를 이용하여 이벤트 객체를 생성
- WSAEventSelect() 함수를 이용 {소켓+이벤트 객체+이벤트 내용들}을 바인딩
- WSAWaitForMultipleEvents() 함수를 호출하여 Signaled 상태인 이벤트들을 기다림
- WSAEnumNetworkEvents() 함수를 이용 네트워크 이벤트를 파악하고 적절한 소켓 함수를 호출하여 처리

```
// 소켓: SOCKET s;
WSAEVENT hEvent = WSACreateEvent();                                    // 이벤트 객체 생성
WSAEventSelect(s, hEvent, (FD_ACCEPT|FD_READ|FD_WRITE|FD_CLOSE));  // 소켓에 이벤트 객체 바
    인딩.
...
while(1)
    nEv = WSAWaitForMultipleEvents(..., FALSE, WSA_INFINITE, FALSE);    // 네트워크 이벤트를 기
    다림
    nEv -= WSA_WAIT_EVENT_0;
    for(i= nEv; i<MAX_CLIENT; ++i)
    {
        nEv = WSAWaitForMultipleEvents(1, &vEv[i], TRUE, 0, FALSE);
        if((nEv==WSA_WAIT_FAILED || nEv=WSA_WAIT_TIMEOUT))
            continue;
        scHost = vSc[i];
        hr = WSAEnumNetworkEvents(scHost, vEv[i], &wtEv);       // 이벤트 분해
        if( FD_ACCEPT & wtEv.lNetworkEvents)                    // Accept 이벤트
            WSAEventSelect(rmClnH, wsEv, (FD_READ|FD_WRITE|FD_CLOSE)); // 소켓에 이벤트 객체
    바인딩.
        ...
        else if( FD_CONNECT & wtEv.lNetworkEvents)              // Connection 이벤트
        ...
        else if( FD_WRITE & wtEv.lNetworkEvents)                // Send 이벤트
        ...
        else if( FD_READ & wtEv.lNetworkEvents)                 // Receive 이벤트
            recv(...);
        ...
        else if( FD_CLOSE & wtEv.lNetworkEvents)                // Close 이벤트
        ...
```