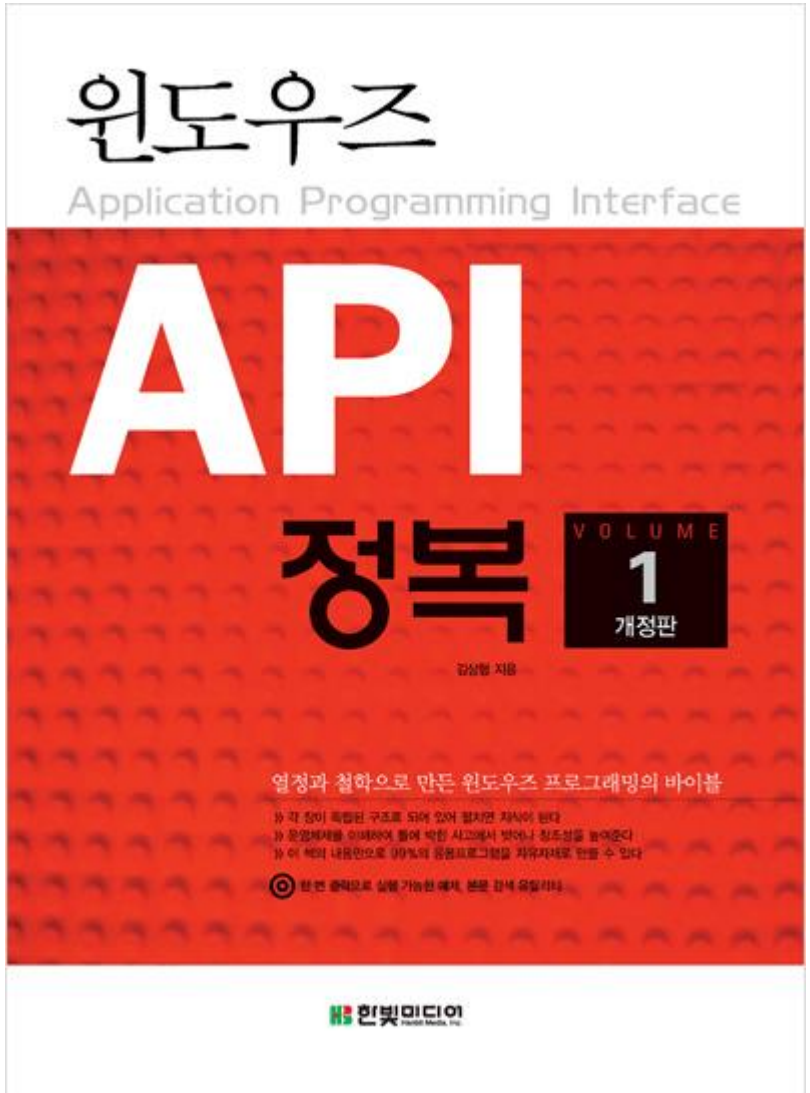
잘 알려지지 않은 숨은 진주, Winsock API - WSAPoll, Fast Loopback

NHN Next
최흥배
https://jacking75.github.io/

강연 자료는github에

https://github.com/jacking75/conf_cpp_korea_201702



Microsoft® Windows® xp


Windows Vista


Windows 7


Windows 8


Windows 10


Microsoft
Linux

Google Play

검색

앱

게임 하위 카테고리 홈 인기 순위 새 출시작

내 앱

쇼핑하기

게임

키즈

에디터 추천

계정

기프트 카드 사용

기프트 카드 구매

내 위시리스트

내 게임 활동

자녀 보호 안내

최고 매출 - 게임

1. 리니지2 레볼루션
Netmarble Games

2. Pokémon GO
Niantic, Inc.

3. 모두의마블 for Ka
Netmarble Games

4. 세븐나이츠 for Ka
Netmarble Games

5. FIFA ONLINE 3 M
NEXON Company

6. 클래시 로얄
Supercell

7. 삼국블레이드
4:33

8. 리니지 레드나이츠
NCSOFT Corporation

9. 원피스 트레저 크루
BANDAI NAMCO Entert

10. 뮤오리진
Webzen Inc.

11. 모바일 스트라이
Epic War

12. 아덴
ITSGAMES

13. 클래시 오브 클랜
Supercell

14. 피망 포커: 카지노
NEOWIZ

WSAPoll

Fast Loopback

poll

Unix Domain Socket

EN 이 콘텐츠는 한국어로 제공되지 않아 영어 버전으로 표시됩니다.

WSAPoll function

The **WSAPoll** function determines status of one or more sockets.

Syntax

C++

```
int WSAAPI WSAPoll(
  _Inout_ WSAPOLLFD fdarray[],
  _In_    ULONG     nfds,
  _In_    INT       timeout
);
```

WSAPoll

- **Windows Vista**부터 새로 생긴 네트워크 API.
- Linux(Unix) OS의 **poll**과 비슷한 것이다.
- 복수의 파일 디스크립터를 감시하는 API.
- 지정한 소켓의 상태가 변화했는지 확인하는 기능을 제공한다.
  기능적으로는 select와 유사하다.

잠시 Linux의 poll에 대해...

poll은 거의 select와 비슷하지만 아래와 같은 차이가 있다.

- 관리할 수 있는 **디스크립터(네트워크에서는 소켓) 수가 무제한**이 된다.

- poll 시스템 콜 자체를 구현하고 있는 시스템이 select 보다 적기 때문에 이식성 등에 좋지 않다(오래된 이야기..)

잠시 select 에 대해서...

```
/* According to POSIX.1-2001 */
#include <sys/select.h>

/* According to earlier standards */
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
```

출처: http://downman.tistory.com/79

소켓 멀티플렉싱 함수 select()
http://eastroot1590.tistory.com/entry/%EC%86%8C%EC%BC%93-%EB%A9%80%ED%8B%B0%ED%94%8C%EB%A0%89%EC%8B%B1-%ED%95%A8%EC%88%98-select

Non-Block 네트워크 프로그램 - 2 (select 사용하기)
http://blog.naver.com/cestlavie_01/220908793329

Non-Block 네트워크 프로그램 - 3 (select를 이용한 echo server)
http://blog.naver.com/cestlavie_01/220909732913

select와 poll의 차이는?

Programming UNIX Sockets in C - Frequently Asked Questions
Created by Vic Metcalfe, Andrew Gierth and other contributers (Transrated into Japanese by: Keisuke Mori)
May 21, 1998 **주의: 오래된 이야기 입니다.**

Richard Stevens(rstevens@noao.edu):

기본적 차이는 select()의 fd_set은 비트 마스크로 되어 있어서 고정 크기라는 것이다. **커널 컴파일 때 이 사이즈 제한을 제외하여 애플리케이션에 필요한 만큼 FD_SETSIZE로 정의할 수 있지만, 많이 작업이 필요**하다.

4.4 BSD의 커널과 Solaris 라이브러리 함수 양쪽에는 이 제한이 있다. 그러나 BSD/OS 2.1에는 이 제한을 피하도록 코딩 되어 있는 것을 찾았다. 그래서 이것은 가능하다.

그러나 poll()에서는 유저는 pollfd 구조체 배열을 나누 맞추어야 한다. 그리고 **이 배열의 엔트리 수를 주기 때문에 근본적으로는 상한은 없다**.

Casper가 언급했듯이 poll()을 가진 시스템은 select 보다 적기 때문에 후자가 이식성은 높다. 또 오리지날의 구현(SVR3)에서는 디스크립터에 -1를 설정하여 커널에 pollfd 구조체 속의 엔트리를 무시하게 할 수 없었다.

이것은 배열 중에서 엔트리를 삭제하는 것이 귀찮게 된다. SVR4에서는 이것은 회피 되었다.

개인적으로는 나는 항상 select()를 쓰고, poll()은 좀처럼 쓰지 않는다.
그것은 나의 코드를 BSD 환경에도 이식 하기 때문이다. 누군가가 select()을 사용한 poll() 구현을 쓰는지 모르겠지만, 나는 본 적이 없다.

select()와 poll()은 모두 POSIX 1003.1g에 의해서 표준화되고 있다.

Windows는 이전부터 select가 있었고, Vista부터 poll 이 생겼다.

poll 과 select 의 차이

분류없음 2008.01.16 13:00

갑자기 생각이 나서 찾아봤다. 차이는...

" poll 은 select 를 개량하여 나온 것.... 성능상의 약간의 이점은 있지만 절대적이라고 말할 정도는 아닙니다. 하지만 될 수 있으면 select 보다는 poll을 사용하는 것이 좋습니다.

우선 poll 은 ...자신이 관리하는 파일 기술자 번호를 struct pollfd 구조체에 넣어서 관리하므로 쓸데없이 감시하지 않는 파일 기술자까지 루프 문에서 검사할 필요는 없어집니다.. 하지만, 이도 기본적으로는 단번에 입출력 이벤트가 발생한 파일 기술자를 알 수는없으므로 루프 문을 돌면서 각 파일 기술자에 대해서 일일이 이벤트가 발생했는지 검사해야 하는 오버헤드는 피할 수 없습니다. 다만 select 에 비해서 루프의 횟수를 줄일 수 있는 .. 장점은..."
((Advanced!)리눅스 시스템 네트워크 프로그래밍 / 김선영 저, 286쪽)

아래 참조는 명쾌하지 않다.

http://kldp.org/node/35279
http://www.joinc.co.kr/modules/moniwiki/wiki.php/RTS

출처: http://eenn.tistory.com/entry/poll-%EA%B3%BC-select-%EC%9D%98-%EC%B0%A8%EC%9D%B4

select 와 poll 비교

poll 은 가장 높은 fd에 +1을 할 필요가 없다.

poll은 fd가 클 경우 좋다. select는 모든 fd의 비트를 검사한다. (select는 for loop에서 set 된 정보를 찾음)
-> 그렇다고 select가 안좋다고 말할 수 없다. 이벤트가 자주 발생하고 연속적인 시스템에서는 select를 사용 (apache http)

poll은 어느 정도 분산되어 있거나 크기 제한이 없는 여러 개의 array 형태로 넘겨서 사용할 때 유용하다. 그리고, 필요한 것만 비교할 경우가 효과적이다.

- select는 fd set을 초기화를 해야 하지만, poll은 입력과 결과를 분리할 수 있다.
- select는 사이즈 제한이 있다.
- select가 이식성이 좋음. 어떤 시스템은 poll을 쓰지 않기도 함
- select의 timeout이 poll의 timeout보다 안정적임

출처: http://knight76.tistory.com/entry/select-poll-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%BD%9C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0

이제 Windows의 WSAPoll 로

https://blogs.msdn.microsoft.com/wndp/2006/10/26/wsapoll-a-new-winsock-api-to-simplify-porting-poll-applications-to-winsock/

If you have experience developing applications using poll(), WSAPoll will be very familiar. It is designed to behave just like poll(). In fact, I recently ported the Winsock WSAPoll SDK sample to run on BSD Unix. I know that sounds backwards from what you would expect but, we all like sample code after all. As it turns out it didn't matter which direction I ported. I had really hoped to have some clever insight to give you, or some real handy tip about how to work around some special difference. The truth is, the portions of the code dealing with poll() were unchanged from the sample. Except, of course, for changing the name of the API. Frankly, porting the poll() code went so smoothly, it made converting Windows threads to pthreads seem like the lion's share of the work.

poll()을 사용하여 응용 프로그램을 개발 한 경험이 있다면 WSAPoll은 매우 친숙하다. **poll()과 똑같이 동작하도록 설계되었다**. 실제로 BSD Unix에서 실행되도록 Winsock WSAPoll SDK 샘플을 이식했다.
.....

WSAPoll function

MSDN
https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms741669(v=vs.85).aspx

```
int WSAAPI WSAPoll(
  WSAPOLLFD fdarray[],
  ULONG nfds,
  INT timeout
);
```

```
typedef struct pollfd {
  SOCKET fd;
  short  events;
  short  revents;
} WSAPOLLFD
```

- fdarray는 WSAPOLLFD 구조체 배열을 지정한다.
- nfds는 fdarray의 요소 수를 지정한다.
- timeout은 함수 호출 후 대기할 시간이다(어떤 이벤트도 없으면 이 시간까지 대기한다).
  - 0 보다 클 경우 그 값만큼 함수가 대기하고 0의 경우는 즉시 제어를 반환한다.
  - 마이너스 값의 경우는 소켓이 지정된 위상에 변화가 있을 때까지 제어를 반환하지 않는다.

```
typedef struct pollfd {
  SOCKET fd;
  short  events;
  short  revents;
} WSAPOLLFD
```

events는 어떤 이벤트 발생을 원하는지를 뜻하는 플래그 셋이다.
이것은 꼭 하나 이상 설정이 되어야 한다.

Flag	Description
POLLPRI	Priority data를 블러킹 없이 읽을 수 있다. 이 플로그는 MS winsock 에서는 지원하지 않는다.
POLLRDBAND	Priority band(out-of-band) data를 블럭킹 읽을 수 있다.
POLLRDNORM	보통의 데이터를 블럭킹 없이 읽을 수 있다.
POLLWRNORM	보통의 데이터를 블럭킹 없이 쓸 수 있다.

POLLIN 플러그는 POLLRDNORM과 POLLRDBAND 플러그의 값을 합친 것이다.

POLLOUT 플러그는 POLLWRNORM 플러그 값과 같은 정의이다.

```
typedef struct pollfd {
  SOCKET fd;
  short  events;
  short  revents;
} WSAPOLLFD
```

revents는 WSAPoll 함수 호출 후 반환 되었을 때 상태 쿼리 결과를 나타내는 플러그 셋이다. 이것은 아래의 플러그를 조합할 수 있다.

Flag	**Description**
POLLERR	에러가 발생하였다.
POLLHUP	스트림 지향 접속의 경우 접속 종료 및 중지 되었다
POLLNVAL	무효한 소켓이 사용되었다.
POLLPRI	Priority data를 블럭킹 없이 읽을 수 있다. 이 플로그는 MS winsock에서는 지원하지 않는다.
POLLRDBAND	Priority band(out-of-band) data를 블럭킹 없이 읽을 수 있다
POLLRDNORM	보통의 데이터를 블럭킹 없이 읽을 수 있다.
POLLWRNORM	보통의 데이터를 블럭킹 없이 쓸 수 있다

예제 코드 분석

```
SOCKET     listenSocket = *((SOCKET *)lpParamater);
WSAPOLLFD fdArray[clientMaxCount];

for (int i = 0; i < clientMaxCount; i++) {
    fdArray[i].fd = INVALID_SOCKET;
    fdArray[i].events = 0;
}

fdArray[0].fd = listenSocket;
fdArray[0].events = POLLRDNORM;
```

fdArray의 요소 수가 16이라면 이것은 1+15로 **1개의 리슨 소켓과 15개의 서버 소켓의 상태의 변경을 검출하는 목적이다**. 즉 접속할 수 있는 클라이언트의 최대 크기는 15이다.
초기 상태에서는 서버 소켓은 만들어지 않았으므로(접속한 클라이언트가 없으니) NULL을 지정하고 리슨 소켓은 배열의 선두에 지정한다.
events는 검출하고 싶은 이벤트를 지정하고 POLLRDNORM은 접속 또는 데이터 읽기 검출을 한다.

```
int nResult = WSAPoll(fdArray, clientMaxCount, 500);
if (nResult < 0) {
    SendMessage(g_hwndListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("WSAPOLLFD 실행이 실패하였다"));
    break;
}
else if (nResult == 0) {
    continue;
}
```

WSAPoll이 0 이하의 값을 반환한 경우에는 함수가 실패했음을 의미하므로 그 뜻을 표시한다.
0이 반환된 경우에는 타임 아웃이 발생한 것을 의미하며 이 경우 후속 처리를 실행하지 않는다.

위 WSAPoll 호출에서는 **500밀리 초 타임 아웃 값을 지정**하고 있지만 본래라면 -1 등의 마이너스 값을 지정하여 이벤트가 발생할 때까지 대기하는 것이 이상적이다.

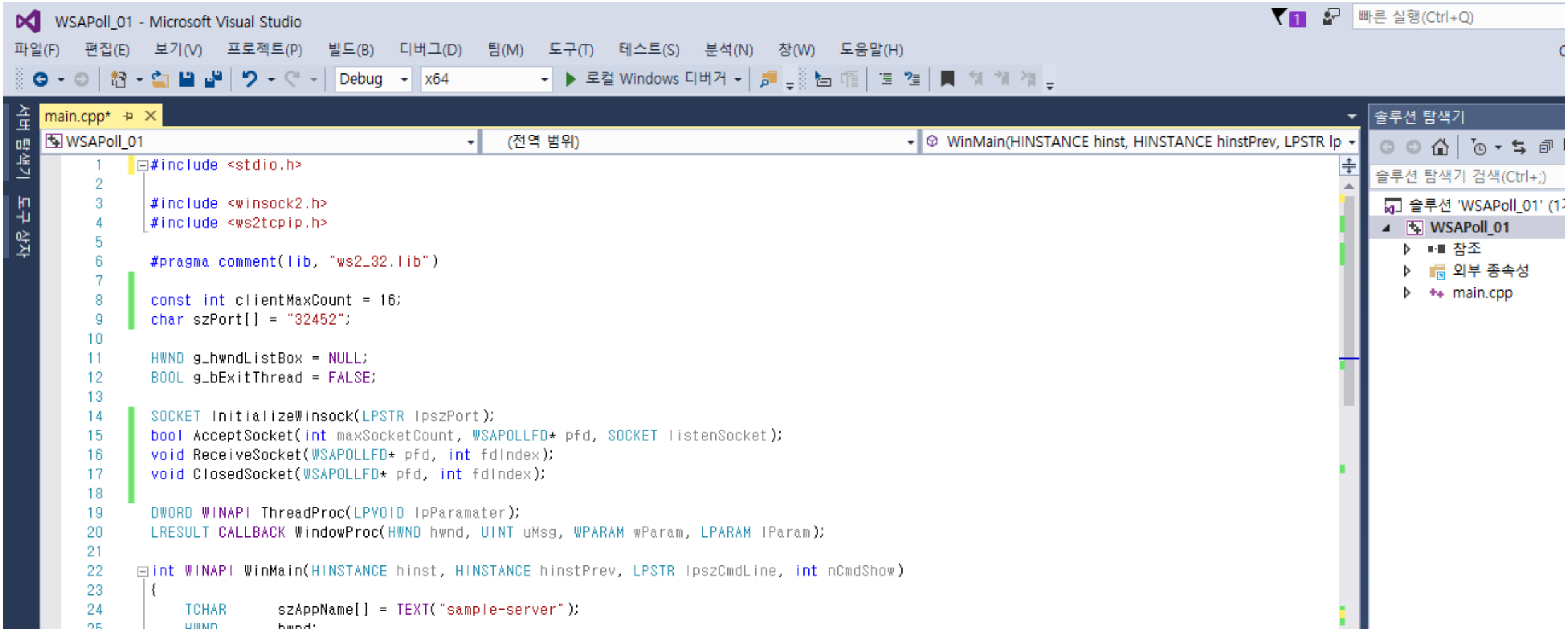
0 보다 큰 값이 반환된 경우 배열 내의 어떤 소켓에 상태의 변경이 생겼는지 다음 처리에서 확인한다.

```
for (int i = 0; i < clientMaxCount; i++)
{
    if (fdArray[i].revents & POLLRDNORM)
    {
        if (fdArray[i].fd == listenSocket)
        {
            AcceptSocket(clientMaxCount, fdArray, listenSocket);
        }
        else
        {
            ReceiveSocket(&fdArray[i], i);
        }
    }
    else if (fdArray[i].revents & POLLHUP)
    {
        ClosedSocket(&fdArray[i], i);
    }
}
```

소켓의 현재 상태는 revents 멤버에 저장되어 있다. 이 값이 **POLLRDNORM 라는 것은 접속 또는 읽기가 발생한 것**을 의미한다.

POLLHUP은 접속 종료를 의미한다.

리슨 소켓에 대해서 POLLRDNORM이 저장되어 있다면 접속을 뜻하므로 accept를 호출, 그렇지 않다면 서버 소켓에 클라이언트의 데이터를 받을 수 있으므로 recv를 호출하여 수신한다.


WSAPoll_01 - Microsoft Visual Studio
빠른 실행(Ctrl+Q)
파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) 도구(T) 테스트(S) 분석(N) 창(W) 도움말(H)
Debug
x64
로컬 Windows 디버거
서버 탐색기
도구 상자
main.cpp*
WSAPoll_01
(전역 범위)
WinMain(HINSTANCE hinst, HINSTANCE hinstPrev, LPSTR lp
#include <stdio.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "ws2_32.lib")
const int clientMaxCount = 16;
char szPort[] = "32452";
HWND g_hwndListBox = NULL;
BOOL g_bExitThread = FALSE;
SOCKET InitializeWinsock(LPSTR lpszPort);
bool AcceptSocket(int maxSocketCount, WSAPOLLFD* pfd, SOCKET listenSocket);
void ReceiveSocket(WSAPOLLFD* pfd, int fdIndex);
void ClosedSocket(WSAPOLLFD* pfd, int fdIndex);
DWORD WINAPI ThreadProc(LPVOID lpParamater);
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev, LPSTR lpszCmdLine, int nCmdShow)
{
TCHAR szAppName[] = TEXT("sample-server");
솔루션 탐색기
솔루션 탐색기 검색(Ctrl+;)
솔루션 'WSAPoll_01'
WSAPoll_01
참조
외부 종속성
main.cpp

linux의 poll

poll 함수 예제
http://sfixer.tistory.com/entry/%EA%B1%B0%EA%BE%B8%EB%A1%9C-%EC%8B%9C%EC%9E%91%ED%95%98%EB%8A%94-%EC%8B%9C%EC%8A%A4%ED%85%9C-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-poll-%ED%95%A8%EC%88%98-%EC%98%88%EC%A0%9C

Poll을 이용한 다중서버 예제
http://robelias.tistory.com/2

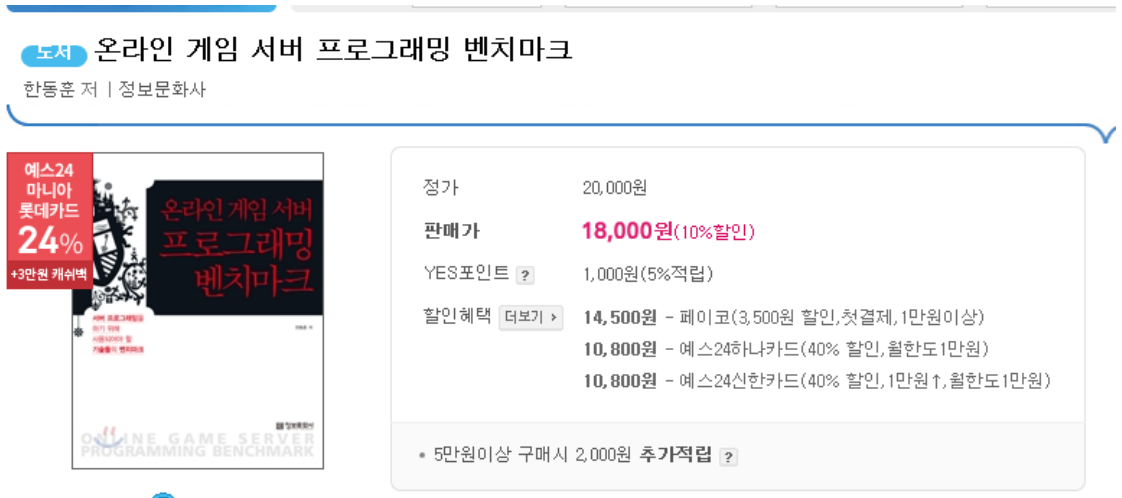
.NET의 Sokcet.Poll 이 있지만...

당연하게 닷넷에서도 poll 함수를 지원한다.
https://msdn.microsoft.com/ko-kr/library/system.net.sockets.socket.poll(v=vs.110).aspx

C# | C++ | VB

```
//Creates the Socket for sending data over TCP.
Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
   ProtocolType.Tcp );

// Connects to host using IPEndPoint.
s.Connect(EPhost);
if (!s.Connected)
{
   strRetPage = "Unable to connect to host";
}
// Use the SelectWrite enumeration to obtain Socket status.
 if(s.Poll(-1, SelectMode.SelectWrite)){
      Console.WriteLine("This Socket is writable.");
 }
 else if (s.Poll(-1, SelectMode.SelectRead)){
           Console.WriteLine("This Socket is readable." );
 }
 else if (s.Poll(-1, SelectMode.SelectError)){
      Console.WriteLine("This Socket has an error.");
 }
```

IOCP, select, poll



http://www.yes24.com/24/goods/3097854?scode=032&OzSrank=8

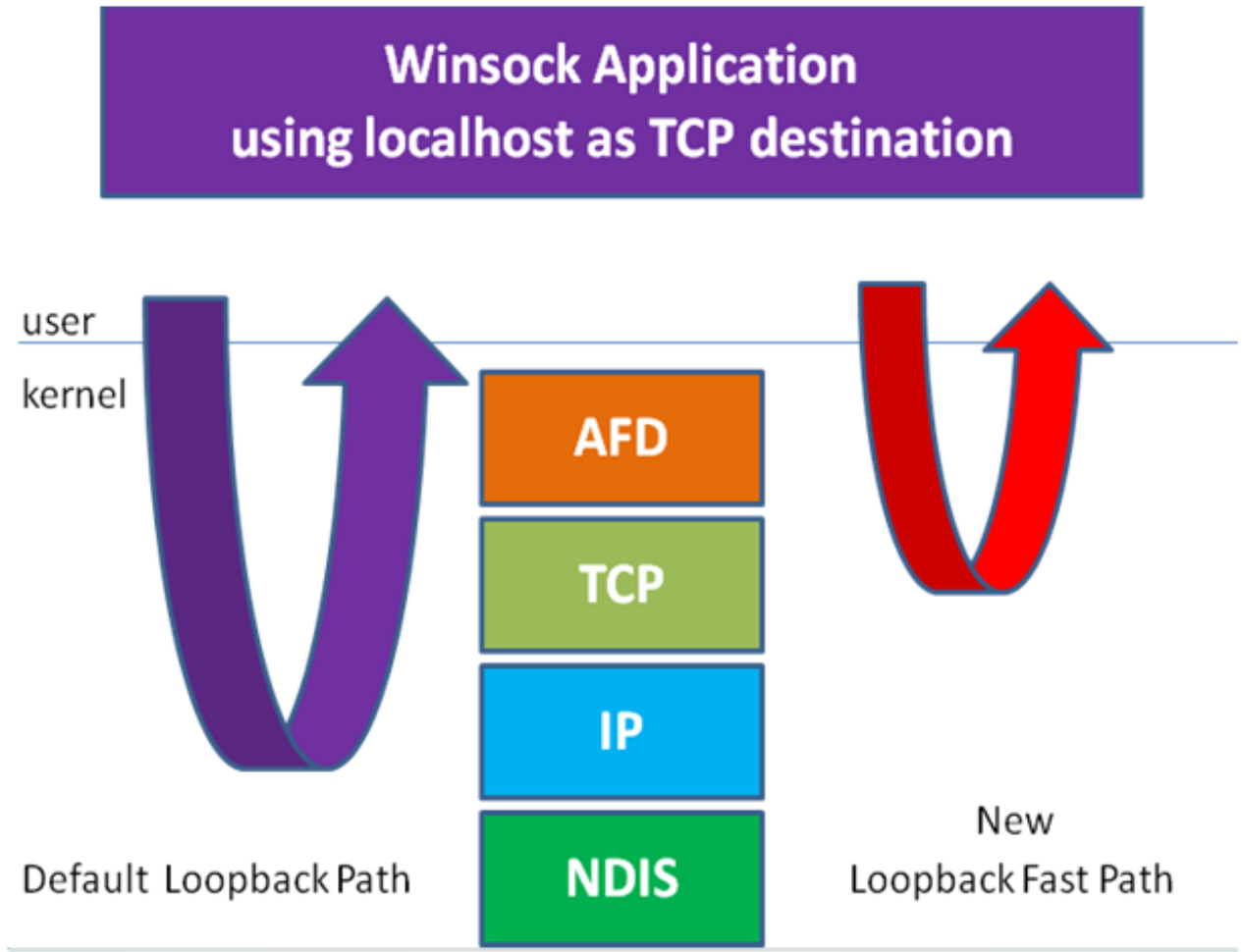
http://gpgstudy.com/forum/viewforum.php?f=18

Online Game Server만 네트워크를 사용하는 것은 아니다.

동접 1000명 가능한 서버 만들기 보다 동접 1000명이 접속하는 재미있는 게임 만들기가 훨씬 더 어려운 듯...

소규모 게임에도 (실시간)온라인 게임이 많았으면...

Fast TCP Loopback Performance and Low Latency with Windows Server 2012 TCP Loopback Fast Path



UDS (Unix Domain Socket)

프로세스 끼리의 통신 UDS

내부 프로세스들 끼리의 통신을 위한 IPC 방법 중에 UDS 를 이용하여 프로그램을 작성하는 방법에 대해 알아 봅니다.

UDS

UDS(Unix Domain Socket)은 내부 프로세스를 끼리 TCP 또는 UDP 프로토콜을 이용하여 통신할 수 있도록 도와주는 소켓입니다. 이 소켓을 이용하시면 기존의 TCP/IP, UDP/IP에서 사용하던 함수로 같은 방법을 사용하여 프로세스들끼리 통신할 수 있어 매우 편리하며, 이점이 장점입니다.
말씀 드린 바와 같이 TCP/IP, UDP/IP와 같은 함수를 사용하고 다만 필요한 변수값만 바꾸어 주면 됩니다. 예를 들어 소켓을 생성해 보겠습니다.
TCP/IP와 UDP/IP에서 처럼 socket() 함수를 이용합니다. 우선 #include <sys/socket.h> 대신에 #include <sys/un.h>를 사용합니다.
자, socket() 함수를 호출해 보겠습니다.

```
int sock;
sock = socket( PF_FILE, SOCK_DGRAM, 0);
```

지금 UDP 프로토콜을 이용하기 위한 UDS 소켓 하나를 만들었습니다. 기존의 UDP/IP와는 달리 PF_INET 대신에 PF_FILE을 사용했습니다. UDS는 데이터를 전송하기 위해 파일을 이용하기 때문입니다. UDS의 사용은 이와 같습니다. 기존의 TCP/IP, UDP/IP 통신에서 사용하는 함수를 그대로 사용하면서 다만 필요한 정보를 몇 가지 바꾸어 주기만 하면 됩니다. socket()에 대한 더 자세한 말씀은 "Unix C Reference의 11장 7절 소켓 열고 닫기"를 참고하십시오.
이번에는 bind() 함수를 이용하여 소켓에 통신 케이블을 연결해 보겠습니다. bind() 함수를 이용하기 위해서는 주소정보가 필요합니다. 이를 위해서는 struct sockaddr_in 대신에 struct sockaddr_un을 사용합니다.

출처: http://www.dreamy.pe.kr/zbxe/CodeClip/119393

유닉스 도메인 소켓(Unix Domain Socket)

유닉스 도메인 소켓(Unix Domain Socket)은 프로세스간의 데이터 교환을 위한 기술 중 하나로, 파일 시스템을 통해 소켓통신 방식으로 내부 프로세스간의 통신을 하는 구조로 이뤄져있다. 약칭으로 UDS라고 표기하며, 간단하게 '유닉스 소켓(Unix socket)'이라고 부르기도 한다. IPC(Inter Process Communication)의 일부로서 분류할 때는 'IPC소켓'이라고 부르기도 한다. POSIX의 표준 운영체제 구성요소 중 하나로 포함되어있다.

UDS의 가장 큰 특징은 소켓통신 방식을 써서 만든 프로세스에 사용이 가능하기 때문에 소켓프로그래밍 구조를 유지한 채로 로컬 프로세스와의 효율적 통신을 가능케 한다는 점이다. TCP, 혹은 UDP형식 데이터를 파일 시스템을 이용해서 통신하는 구조로, 파일 시스템을 통해 파일 주소 및 inode로 각 프로세스에서 참조되며, 통신은 운영체제의 커널상에서 이뤄지기 때문에 inet소켓을 이용해서 네트워크단을 이용해 전달하는 것보다 빠르며 부하가 적게 걸린다.(기본적으로 소켓통신 방식이 TCP/IP의 4계층을 거쳐 전달되기 때문에 지연이 발생하는데 반해서 유닉스 소켓은 어플리케이션 계층에서 TCP계층으로 내려가 바로 데이터를 전달하고, 수신측도 TCP계층에서 수신해 어플리케이션 계층으로 올라간다)

이러한 특성 때문에 네트워크 관련 프로그램에서 널리 사용되며, X윈도우 등의 프로젝트에서도 활용되고 있다.

C언어를 비롯해 php, 자바, 펄 등 다양한 언어에서 지원을 위한 라이브러리를 기본 제공하고 있다.

Fast TCP Loopback !!!

마이크로소프트의 윈도우 시리즈에서는 사용이 불가능하기 때문에 윈도우로 포팅될 땐 보통 localhost(128.0.0.1)의 임의 포트를 이용하도록 변경된다. NGINX(엔진엑스) 등의 서버 어플리케이션이 윈도우 상에서 효율이 나쁜 이유 중 하나이기도 하다.

추가로 프로세스는 sendmsg()와 recvmsg() 시스템콜을 이용해서 유닉스 도메인 소켓에 접속해 파일 기술자(file descriptors)를 주고받을 수도 있다.

출처: http://wiki.nex32.net/%EC%9A%A9%EC%96%B4/%EC%9C%A0%EB%8B%89%EC%8A%A4_%EB%8F%84%EB%A9%94%EC%9D%B8_%EC%86%8C%EC%BC%93

Fast TCP Loopback

- Windows 8 / Windows Server 2012 에서 추가된 Winsock 기능.
- **SIO_LOOPBACK_FAST_PATH IOCTL은 지연 시간을 줄이고, TCP over 루프백의 성능을 향상시킨다**.
  Device Input and Output Control (IOCTL)
- Windows에서 IPv4 또는 IPv6 루프백(localhost, 127.0.0.1) 주소를 사용하는 패킷은 네트워크 계층을 사용한다. 이 IOCTL은 루프백을 통한 TCP 패킷이 **전송 계층을 대신 사용하도록 동작을 변경**한다.
- 이렇게 하면 루프백 패킷이 OS에서 통과 해야하는 **코드 경로가 단축되므로 대기 시간이 단축되고 성능이 향상**된다.

- 이 향상된 기능을 사용하려면 **루프백(클라이언트와 서버)을 사용하는 소켓 모두가 IOCTL을 설정해야한다**.

- 클라이언트측 소켓은 연결 호출을 작성하기 전에 이 IOCTL을 설정 해야 하나다. 그리고 서버측 소켓은 들어오는 소켓을 수락하기 전에 이 IOCTL을 적용 해야한다.

- 이 향상된 기능의 제한 사항은 Windows 필터링 플랫폼(WFP) 필터와 함께 사용할 수 없다는 것이다. 또 다른 사소한 제한은 이 IOCTL이 설정 될 때 소켓 옵션의 제한된 수만 사용될 수 있다는 것이다.

WFP: 소켓이 OS 내에서 처리되기 전에 필터 하거나 변경할 수 있다. 즉 방화벽이나 안티 바이러스, 안티 스파이웨어 대응 소프트웨어 등이 어떤 데이터를 시스템에서 수신할지를 보다 효율적으로 컨트롤 할 수 있다.

사용 예

```
#include <WinSock2.h>
#include <mstcpip.h>
```

```
TcpServerSocket::TcpServerSocket(std::string const &localAddress,
                                 unsigned short localPort,
                                 int queueLen)
    : Socket(SocketAddress::isIPv4Address(localAddress) ? AF_INET : AF_INET6, SOCK_STREAM, IPPROTO_TCP) {

    if (IsWindows8OrGreater()) {
        if (!ioctl(SIO_LOOPBACK_FAST_PATH)) {
            throw SocketException("Set SIO_LOOPBACK_FAST_PATH failed (WSAIoctl())");
        }
    }
```

출처:
https://github.com/billlin0904/librapid/blob/95864eed190bee0752502f88c57c7dcef6965050/source/details/socket.cpp

```cpp
OsTcpSocket::OsTcpSocket()
        : handle_(::WSASocketW(AF_INET,
                SOCK_STREAM,
                IPPROTO_TCP,
                nullptr, 0,
                WSA_FLAG_OVERLAPPED))
{
        panic_if(handle_ == INVALID_SOCKET, "WsaSocketW");

        int OptionValue = 1;
        DWORD NumberOfBytesReturned = 0;

        int status = 0;

        status = WSAIoctl(
                handle_,
                SIO_LOOPBACK_FAST_PATH,
                &OptionValue,
                sizeof(OptionValue),
                NULL,
                0,
                &NumberOfBytesReturned,
                0,
                0);

        panic_if(SOCKET_ERROR == status, "SetFlags");
```

출처: https://github.com/GorNishanov/await/blob/c23d240a7b9a3f49df3fd6f9b44e2ae78c45899d/2015_CppCon/SuperLean/PAL.cpp

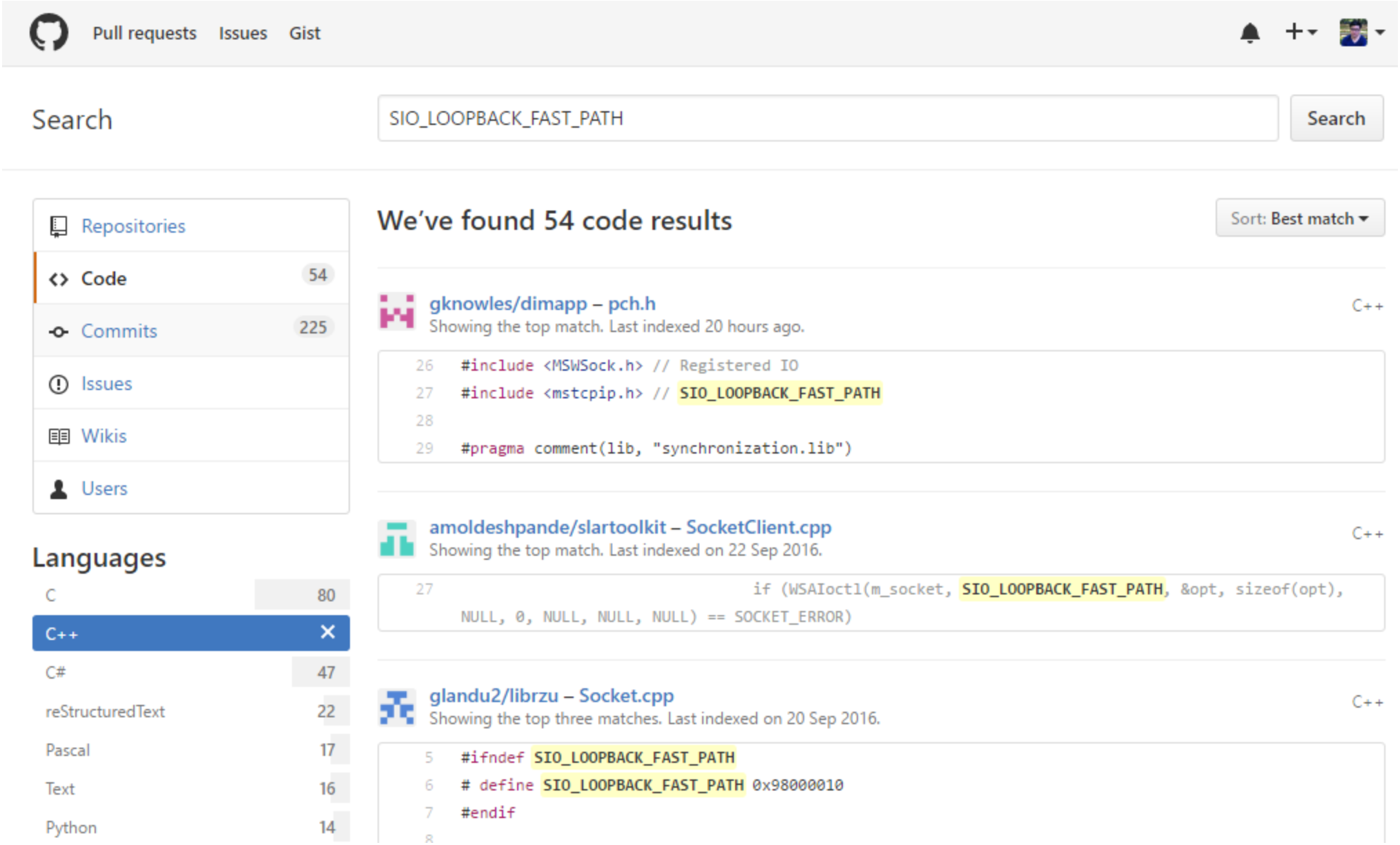
```
// enable faster operations on the loop-back if requested
if (settings->UseFastLoopback == true)
{
        UInt32 optionValue = 1;

        DWORD dwBytes;

        int enableFastLoopbackResult = ::WSAIoctl(listenSocket, SIO_LOOPBACK_FAST_PATH, &optionValue, sizeof(UInt32), nullptr, 0, &dwByte

        // check if attempt has succeed
        if (enableFastLoopbackResult == SOCKET_ERROR)
        {
                return false;
        }
}
```

출처: https://github.com/stas-sultanov/SXN.Net/blob/d7bbba5f56900813b4f88d38b54bc12a560ded13/src/TcpServerCli/TcpWorker.h

C#

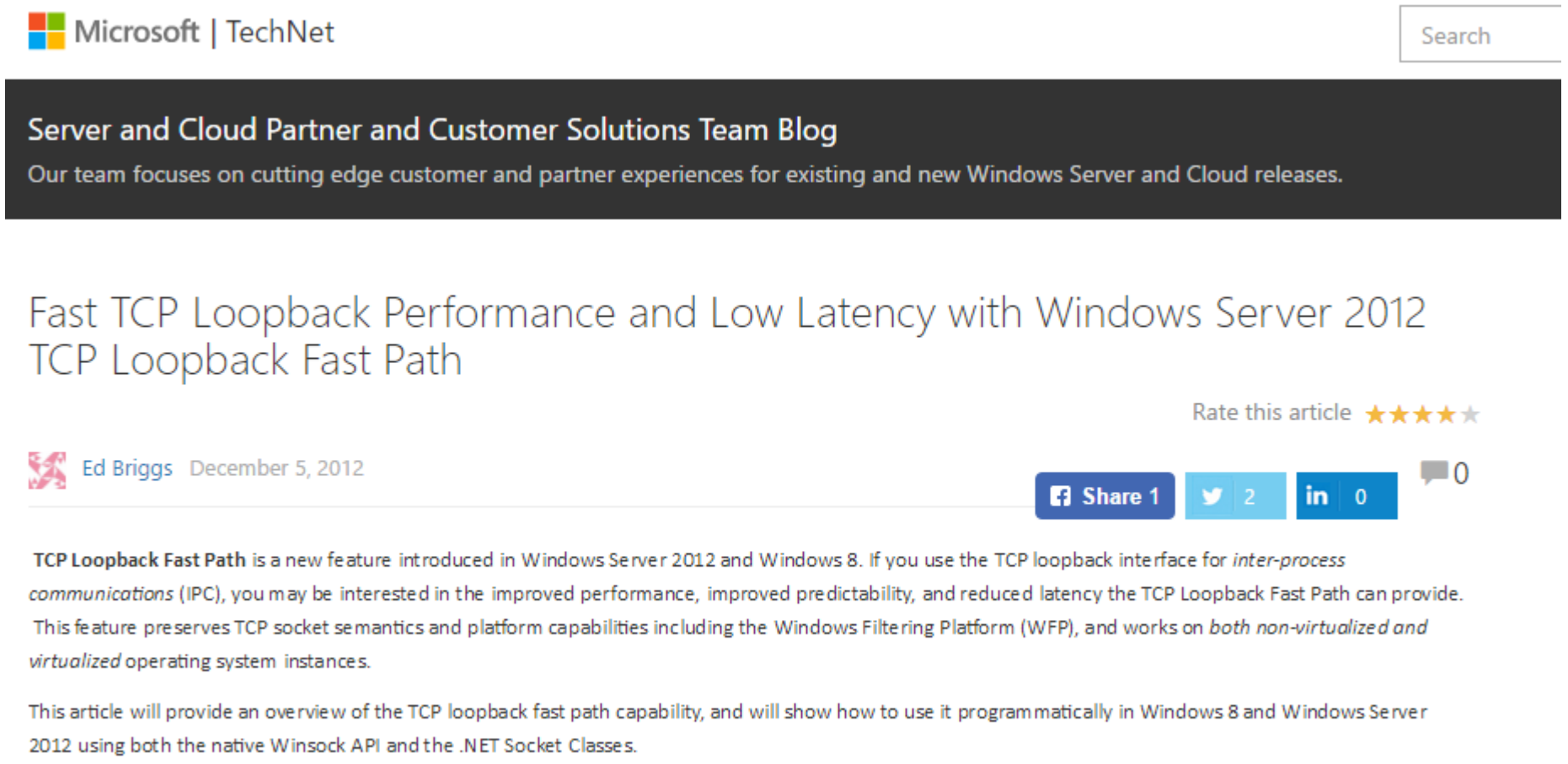
```
public static Socket FastSocket(Socket socket)
{
    CheckDebug.NotNull(socket);

    socket.NoDelay = true;

    if (IsWindows)
    {
        unchecked
        {
            try
            {
                // defined in `mstcpip.h`
                const int SIO_LOOPBACK_FAST_PATH = (int)0x98000010;
                socket.IOControl(SIO_LOOPBACK_FAST_PATH, optionInValue: Yes, optionOutValue: null);
            }
            catch (Exception e)
            {
                Kon.DebugException(e);
            }
        }
    }

    return socket;
}
```

출처: https://github.com/imiuka/rtmp-sharp/blob/834f7079bc1b32676dabbd6806b9b746b15f5717/src/_Sky/Hina/Net/SocketEx.cs



https://github.com/search?l=C%2B%2B&q=SIO_LOOPBACK_FAST_PATH&type=Code&utf8=%E2%9C%93

Fast TCP Loopback Performance and Low Latency with Windows Server 2012 TCP Loopback Fast Path



Microsoft | TechNet

Search

Server and Cloud Partner and Customer Solutions Team Blog

Our team focuses on cutting edge customer and partner experiences for existing and new Windows Server and Cloud releases.

Fast TCP Loopback Performance and Low Latency with Windows Server 2012 TCP Loopback Fast Path

Rate this article ★★★★★

Ed Briggs December 5, 2012

Share 1 2 0 0

TCP Loopback Fast Path is a new feature introduced in Windows Server 2012 and Windows 8. If you use the TCP loopback interface for *inter-process communications* (IPC), you may be interested in the improved performance, improved predictability, and reduced latency the TCP Loopback Fast Path can provide. This feature preserves TCP socket semantics and platform capabilities including the Windows Filtering Platform (WFP), and works on *both non-virtualized and virtualized* operating system instances.

This article will provide an overview of the TCP loopback fast path capability, and will show how to use it programmatically in Windows 8 and Windows Server 2012 using both the native Winsock API and the .NET Socket Classes.

https://blogs.technet.microsoft.com/wincat/2012/12/05/fast-tcp-loopback-performance-and-low-latency-with-windows-server-2012-tcp-loopback-fast-path/

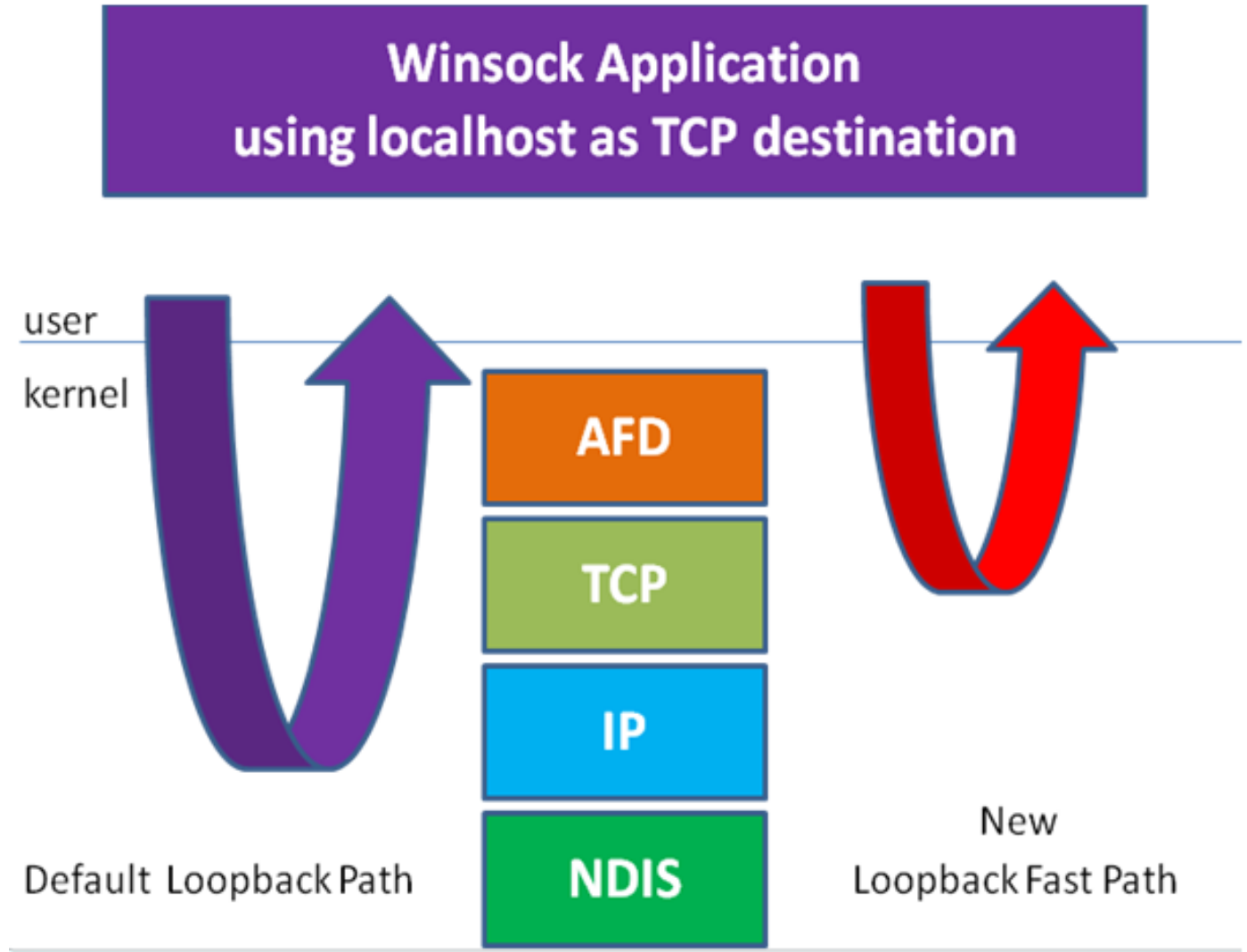
Performance

The performance of the default TCP loopback interface is quite good, and adequate for most applications. However, applications with more demanding throughput requirements, or applications in which achieving low latency IPC is important may benefit from the use of this newly introduced capability.

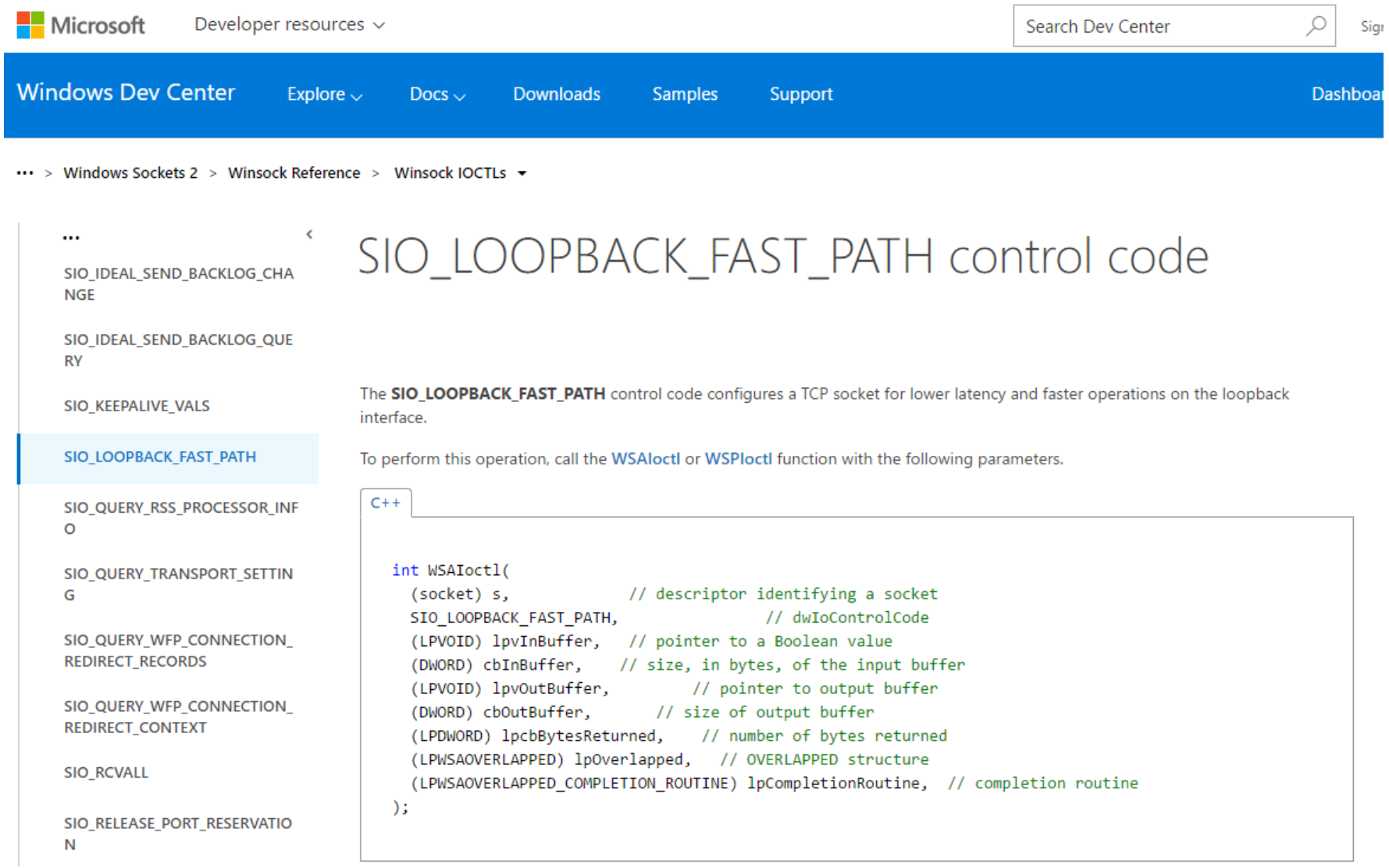
The following table provides a comparison of TCP loopback latency, message rates, and jitter for Windows Server 2008R2 and Windows Server 2012 using IO completion ports and RIO sockets in conjunction with the loopback fast path capability.

The test uses two single-threaded processes: one sending TCP messages, and the other echoing the messages back to the sender. Each outbound message contains a payload of 64 bytes, and includes a 64 bit origination timestamp from **QueryPerformanceCounter.** When the sender receives the message back from the echo process, it again calls **QueryPerformanceCounter** , and calculates the round-trip delay using the timestamp in arriving message. Each round-trip time measurement is stored in a table which is used to generate a statistical summary at the end of the test run. On this particular machine, a desktop class 6-core AMD 3.2 GHz processor**,** the time stamp returned from **QueryPerformanceCounter** has a resolution of 319 nanoseconds and is derived from the CPU timestamp counter (TSC) register. The message send rate is constrained by the round-trip-time in this test, and is therefore equal to the reciprocal of the round trip time because the sender must wait for a response before sending the next outbound message. Because there is very little variation in the round-trip times in a particular test run, the arrival rate is uniform. The TCP Nagle algorithm is disabled (**TCP_NODELAY** socket option) on both the sender and echo process sockets.

Windows Server Version	Socket Completion Style	Loopback Fast Path Enabled	Mean Latency in Microseconds (Round Trip)	Round Trip Messages Per Second	Relative Latency Reduction	Relative Jitter Reduction
WS 2008R2	IOCP wait	No	27	37,000	0%	0%
WS 2008R2	IOCP poll	No	26	38,600	4%	12%
WS 2012	IOCP wait	No	17	59,000	37%	45%
WS 2012	IOCP wait	**Yes**	12	83,000	55%	45%
WS 2012	RIO poll	No	9	111,000	67%	76%
WS 2012	RIO poll	**Yes**	3	333,000	89%	90%


Winsock Application
using localhost as TCP destination
user
kernel
AFD
TCP
IP
NDIS
Default Loopback Path
New
Loopback Fast Path

MSDN



https://msdn.microsoft.com/en-us/library/windows/desktop/jj841212(v=vs.85).aspx

Python 3.6

socket.**ioctl**(*control*, *option*)

Platform:	Windows

The `ioctl()` method is a limited interface to the WSAIoctl system interface. Please refer to the Win32 documentation for more information.

On other platforms, the generic `fcntl.fcntl()` and `fcntl.ioctl()` functions may be used; they accept a socket object as their first argument.

Currently only the following control codes are supported: `SIO_RCVALL`, `SIO_KEEPALIVE_VALS`, and `SIO_LOOPBACK_FAST_PATH`.

Changed in version 3.6: `SIO_LOOPBACK_FAST_PATH` was added.

https://docs.python.org/3/library/socket.html

ioctl() 함수에서 SIO_LOOPBACK_FAST_PATH를 사용할 수 있게 되었다.
Windows 환경(Win8 이상)에서 TCP loopback 성능을 향상 시킬 수 있게 되었다.