**1. 블로킹 소켓**

1) 서버 소켓

|  |
| --- |
| # create an INET, STREAMing socket  serversocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # bind the socket to a public host, and a well-known port  serversocket.bind((socket.gethostname(), 80))  # become a server socket  serversocket.listen(5)  while True:  # accept connections from outside  (clientsocket, address) = serversocket.accept()  # now do something with the clientsocket  # in this case, we'll pretend this is a threaded server  ct = client\_thread(clientsocket)  ct.run() |

서버 소켓은 clientsocket을 처리하기 위해

a) 스레드로 보내거나,

b)clientsocket를 처리할 새 프로세스를 만들거나,

c) 비 블로킹 소켓을 사용하도록 이 응용 프로그램을 재구성하고,select를 사용하여 《서버》 소켓과 활성clientsocket들 간에 다중화(multiplexing)합니다.

어떤 데이터도 보내지 않고, 어떤 데이터도 수신하지 않습니다. 단지 클라이언트 소켓을 생성할 뿐입니다. 각clientsocket은 우리가 바인드 한 호스트와 포트로connect()를 수행하는*다른*《클라이언트》 소켓(외부에서 접속하는) 에 대한 응답으로 만들어집니다.clientsocket를 만들자마자, 더 많은 연결을 기다리는 것으로 돌아갑니다. 두 개의 《클라이언트》 는 자유롭게 대화를 나눌 수 있습니다 - 그들은 대화를 끝낼 때 재활용되는 어떤 동적으로 할당된 포트를 사용합니다.

위의 소스는 서버소켓이 50000 포트에 바인드 했고 새로운 커넥션이 들어오는지 기다리고 있습니다. 새로운 커넥션은 accept()로 이 함수는 새로운 클라이언트가 접속할때까지 멈춰있습니다. 새로운 커넥션이 들어오면 클라이언트의 주소를 리턴하고

recv()를 이용하여 1024 바이트의 데이터를 받아서 sendall 메소드를 이용하여 클라이언트 전체에 데이터를 전송한다.

**2) 클라이언트 소켓**

클라이언트 소켓은 일반적으로 하나의 교환(또는 일련의 작은 교환 집합)에서만 사용됩니다

|  |
| --- |
| import socket  s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  s.bind(('localhost', 50000))  s.listen(1)  conn, addr = s.accept()  while 1:  data = conn.recv(1024)  if not data:  break  conn.sendall(data)  conn.close() |

위의 소스는 클라이언트소켓이 서버소켓으로 접속하여 데이터를 전송하고 1024바이트의 데이터를 받습니다. (recv함수)

**3) 소켓 닫힘**

아마도 블로킹 소켓 사용에 관한 최악의 경우는 상대방이 (close를 수행하지 않고) 갑자기 다운되었을 때 일어나는 일입니다. 소켓이 멈출 수 있습니다. TCP는 신뢰성 있는 프로토콜이며, 연결을 포기하기 전에 아주 오랜 시간 동안 기다립니다. 스레드를 사용하고 있다면, 스레드 전체가 실질적으로 죽습니다. 블로킹 읽기를 수행하는 동안 락을 잡는 것과 같은 어리석은 짓을 하지 않는 한, 스레드는 자원을 많이 소비하지 않습니다. 스레드를 죽이려고 하지*마십시오*- 스레드가 프로세스보다 효율적인 부분적인 이유는 자원 재활용과 관련된 오버헤드를 회피한다는 것입니다. 즉, 스레드를 죽이면 전체 프로세스가 엉망이 될 가능성이 있습니다.

HTTP와 같은 프로토콜은 하나의 전송에만 소켓을 사용합니다. 클라이언트는 요청을 보낸 다음 응답을 읽습니다. 그게 전부입니다. 소켓은 버려집니다. 이는 클라이언트가 0바이트를 수신하여 응답의 끝을 감지할 수 있음을 뜻합니다.

그러나 추가 전송을 위해 소켓을 재사용할 계획이라면,*소켓에는*EOT*가 없다*는 것을 알아야 합니다. 반복합니다: 소켓send또는recv가 0바이트를 처리한 후 반환되면 연결이 끊어진 것입니다. 연결이 끊어진 것이*아니라면*, 소켓은 (당분간) 읽을 것이 아무것도 없다는 것을 알려주지*않을*것이므로,recv에서 영원히 기다릴 수 있습니다. 이것에 대해 조금 더 생각해보면, 소켓의 근본적인 진실을 깨닫게 될 것입니다:*메시지는 고정 길이거나*(억),*구분자로 표시되거나*(어깨를 으쓱),*얼마나 긴지 표시하거나*(훨씬 낫다),*연결을 닫아서 끝내야 합니다*. 선택은 전적으로 여러분의 것입니다, (하지만 어떤 방법이 다른 것보다 올바릅니다).

**2. 넌 블록킹 소켓**

**위의 소켓들은 블록킹 형태입니다. 예를 들어 하나의 소켓이 읽고 쓰면 다른 소켓은 아무것도 할수가 없습니다. 이에 대한 해결책은 스레드를 생성하는 것입니다.**

**그러나 스레드를 생성하면 컨텍스트 스위칭이 일어나므로 부하가 걸릴 수 있습니다. 이에 대해서 비동기 방식의 소켓 전송이라는 개념이 있습니다. 이것의 주요 개념은 소켓의 상태를 os(operating system)에서 관리하도록 하고 프로그램에 읽고 쓸 일이 생겼음을 소켓이 준비되었을때 알려주는 것입니다.**

**1) poll(linux)**

**2) kqueue, kevent(BSD)**

3) select (crossplatform)