Subject: Re: Kqueue/epoll based interface for TcpSocket and WebSocket
Posted by shutalker on Mon, 07 May 2018 07:54:53 GMT
View Forum Message <> Reply to Message

Quote:
But not at the same time, right? That would be a big problem IMO...

Yes.

Quote:
IMO, there will be a class instance that will provide the communication with single client and you will want to let it read the data. You will probably have some container of these instances. In the end, you need to map incomming event to a class instance somehow. If all ID you have is GetSocket, you will need map socket->instance.

There is no need in mapping socket->instance. Let me clear things up with how SocketEventQueue::Wait(...) works. First of all, sockets that need to be tracked for events are attached to the kernel event queue via SocketEventQueue::SubscribeSocket****(...). Then Wait(...) is invoked to obtain pending events from kernel event queue:

```
   // kqueue-based 'wait for events' system call
   int avail = kevent(queueFD, nullptr, 0, rawEvents, TRIGGERED_SET_SIZE,  tvalp);

   if(avail < 0)
   {
      // error handling
   }

   VectorMap<SocketHandler, SocketEvent<T_SOCKET>> triggered;

   for(int iEvent = 0; iEvent < avail; ++iEvent)
   {
      SocketHandler  handler = rawEvents[iEvent].ident;  // kqueue based implementation to obtain socket descriptor

      if(handler == ancillaryReadFD)
      {
         // listening socket close handling (pipe-trick)

                  ...

         // sockhdlr is a listening socket descriptor that was closed
         // socket is a pointer to correnponding socket class instance
         SocketEvent<T_SOCKET>  &event  = triggered.GetAdd(sockhdlr, SocketEvent<T_SOCKET>(socket));
         event.SetTriggeredException();

         continue;
```

```
        }

        T_SOCKET *socket  = socketHandler.Get(handler, nullptr);  //socketHandler is
VectorMap<SOCKET, *SOCKET_T>

        // if socket was removed after ancillaryReadFD data was delivered
        // pipe-trick implementation to track listening socket closing
        if(!socket)
            continue;

        SocketEvent<T_SOCKET>  &event  = triggered.GetAdd(handler,
SocketEvent<T_SOCKET>(socket));

        if(rawEvents[iEvent].flags & (EV_EOF | EV_ERROR))
            event.SetTriggeredException();

        if(rawEvents[iEvent].filter == EVFILT_READ)
            event.SetTriggeredRead();

        if(rawEvents[iEvent].filter == EVFILT_WRITE)
            event.SetTriggeredWrite();
    }

    return pick(triggered.GetValues());
```

I use VectorMap containers instead of Vector as SocketEventQueue member, therefore searching of triggered sockets may be a little bit more effective than iterating over the Vector.
All events instances are generated every time Wait(...) was invoked, so the only necessity for user is to keep socket instance pointer valid (by placing sockets in Array, for example).

I really like your idea to expand existing SocketWaitEvent interface and work with indices, not helper class instances. And I think it's possible to use VectorMap<SOCKET, int> with VectorMap::Find(...) returning index instead of Vector<Tuple<SOCKET, dword>>. What do you think about this?