MMORPG '대륙전쟁' 서버에 사용된 Lock-free Queue

서버 프로그래머, 류형진

개 요

포트폴리오로 제출한 TestCode-LockFreeQueue 은 제가 파트장으로 참여한 프로젝트, MMORPG '대륙전쟁' 의 게임 서버에 적용된 무잠금 큐의 샘플 코드입니다.

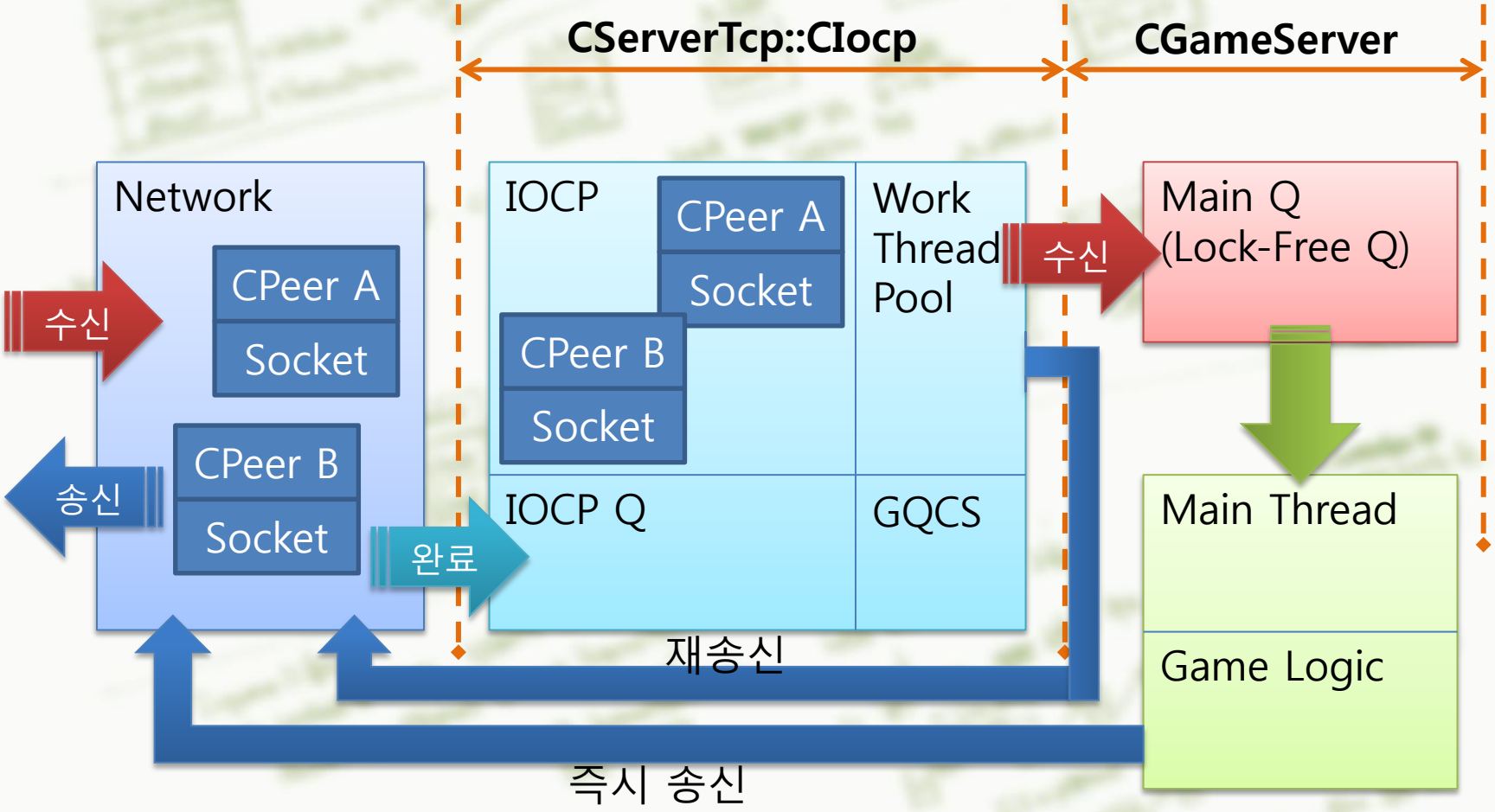
게임 서버에 무잠금 큐를 적용하기 전에 안정성 검사를 위해 이 샘플 코드를 먼저 구현했고 며칠간 테스트를 통해서 적용해도 문제가 없다는 판단을 내린 뒤 게임 서버에 적용한 것입니다.

이후에도 클로즈 베타 테스트를 거치면서 전혀 문제 없이 동작했기에 안정성은 검증되었다고 할 수 있습니다.

이 PPT에서는 무잠금 큐가 대륙전쟁 게임 서버에 어떻게 적용되었는지 설명하며 샘플 코드 사용법에 대해 안내하고 있습니다. 이 샘플 코드를 통해 저의 코딩 스타일도 파악하실 수 있으리라 생각됩니다.

목 차

'대륙전쟁' 게임 서버의 내부 구조



통신 상대 1개 객체인 CPeer

```
class CNetwork
{
public:
    CNetwork( __in INT iNetworkKind = NETWORK_KIND_DEFAULT );
    virtual ~CNetwork( VOID ) {};

    BOOL CreateNBOLSocket( VOID );  //'N'on-'B'locking 'O'ver'L'apped
    bool CreateConnection( __in LPSTR lpszAddr, __in USHORT unsPort );

    BOOL Listen( USHORT port, INT backLog );
    BOOL Accept( SOCKET listenSocket );  // TODO: NPC 서버의 패킷 처리 부분 적용이 완료도

protected:
    TCriticalSection m_CS;
    TCriticalSection m_CSForReceiving;
    TCriticalSection m_CSForSending;

    INT Receiving( VOID );
    INT Sending( __in BYTE* pbyData, __in DWORD dwDataLength );

    VOID UpdateResultOfOverlappedRecv( VOID );
    INT QueuingReceivedData( __in BYTE* pbyEmptySection, __in INT nRemainingLength );

private:
    INT m_iNetworkKind;
    SOCKET m_uni64NSocket;

    BYTE m_byReceiveBuffer[PACKET_BUFFER_SIZE];

    OVERLAPPED_EX m_stAcceptOverlappedEX;
    OVERLAPPED_EX m_stReceiveOverlappedEX;
    OVERLAPPED_EX m_stSendOverlappedEX;
};
```

CNetwork 에는 송수신을 위한 기본 함수와 소켓, 버퍼, OVERLAPPED 구조체를 멤버로 가지고 있으며,

CPeer 은 이런 CNetwork 를 부모로 가지고 네트워크 모델별 수신 함수와 상대방으로부터 수신하거나 송신해야 하는 미해결 데이터를 관리하는 자료구조를 멤버로 가지고 있습니다.

즉, **CPeer 는 통신 상대 1개 객체를 의미**합니다.

```
class CPeer : public CNetwork
{
public:
    CPeer( VOID ) {};
    CPeer( __in INT iNetworkKind ):CNetwork( iNetworkKind ) {};
    virtual ~CPeer( VOID ) {};

    virtual bool ThrowToMainQ( DWORD dwReceivedData ) { return true; }; // TODO: WSAEventSelect 모델을 사용하는 곳에서는 Main Q

    BOOL StartP( VOID );
    BOOL EndP( VOID );

    bool ReadPacketForIOCP( __in DWORD dwReceiveSize = 0, __out BYTE* pbyReceivedPacket = NULL, __in DWORD dwProcessData = 0 );
    bool ReadPacketForWSAES( __out BYTE* pbyReceivedPacket = NULL );  //for 'WSAE'vent'S'elect
    INT WritePacket( __in WORD wSendSize = 0, __in BYTE* pbySendData = NULL );

private:
    BYTE m_byReceivedDataBuffer[PACKET_BUFFER_SIZE * 3];
    INT m_nRemainingLength;  //m_byReceivedDataBuffer에 빈 공간 크기

    CIOPacketDepository m_ReadPD;
    CIOPacketDepository m_WritePD;

    bool IntegratedLoadForRawData( VOID );
    INT LoadUpIntoQueue( VOID );
};
```

게임 서버의 IOCP(1)

```
4  class CIocp
5  {
6  public:
7      CIocp(VOID);
8      virtual ~CIocp(VOID);
9
10 private:
11     HANDLE m_hIOCP;
12     DWORD          mWorkerThreadCount;
13
14     std::vector<HANDLE> mWorkerThreadVector;
15     HANDLE         mStartupEventHandle;
16
17 protected:
18     virtual VOID   OnIoRead(VOID *object, U32 dataLength) = 0;
19     virtual VOID   OnIoWrote(VOID *object) = 0;
20     virtual VOID   OnIoConnected(VOID *object)          = 0;
21     virtual VOID   OnIoDisconnected(VOID *object)       = 0;
22
23 public:
24     BOOL           Begin(VOID);
25     BOOL           End(VOID);
26
27     BOOL           RegisterSocketToIocp(SOCKET socket, ULONG_PTR completionKey);
28     VOID           WorkerThreadCallback(VOID);
29
30     virtual VOID   OnIoForceKill(VOID *pObject)        = 0;
31 };
```

IOCP의 작업 스레드 개수와 작업 스레드 핸들을 관리하는 자료구조

완료된 IO에 대한 핸들러. 자식 클래스 CServerTcp 에서 정의

IOCP에 관찰하려는 소켓 등록하는 함수

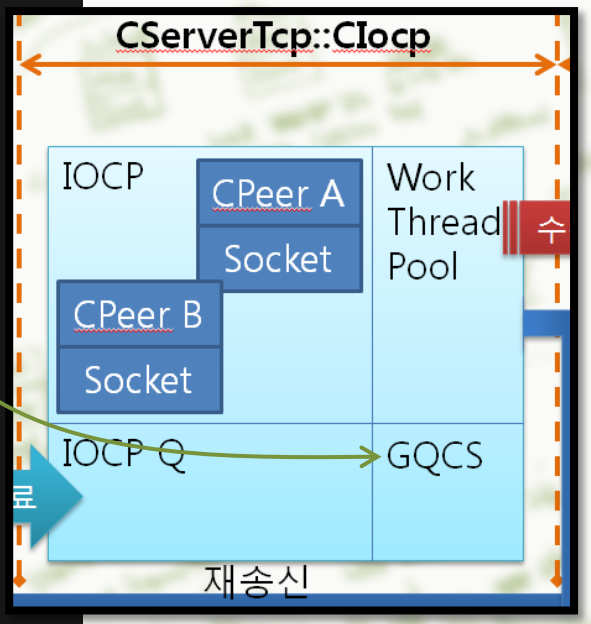
GQCS가 있는 작업 스레드 콜백 함수

CIocp를 부모로 두고 있는 자식 클래스 CServerTcp에서는 가상 함수를 정의하고 접속해오는 상대방들(CPeer)의 현황을 자료구조로 가지고 있습니다.

게임 서버의 IOCP(2)

```
105 VOID CIocp::WorkerThreadCallback(VOID)
106 {
107     //...(중략)...
108
109     while( TRUE ) {
110         bResult = GetQueuedCompletionStatus( m_hIOCP, &dwNumberOfBytes, ( PULONG_PTR )&pCompletionKey, &pstOverlapped, dwMilliseconds );
111         if( bResult == FALSE ) {
112             dwErrNo = GetLastError();
113
114             //...(중략)...
115         }
116         else {
117             if( pstOverlapped == NULL )
118                 return; // CIocp::End()의 PostQueuedCompletionStatus( m_hIOCP, 0, 0, NULL );에 의한 작업 스레드 종료
119
120             pstOverlappedEx = ( OVERLAPPED_EX* )pstOverlapped;
121             pObject = pstOverlappedEx->Object;
122             if( pObject == NULL )
123             {
124                 //...(중략)...
125             }
126
127             switch( pstOverlappedEx->IoType )
128             {
129             case IO_ACCEPT:
130                 OnIoConnected( pObject );
131                 break;
132
133             case IO_READ:
134                 if( dwNumberOfBytes == 0 ) // 클라이언트 연결 종료 요청
135                 {
136                     OnIoDisconnected( pObject );
137                     break;
138                 }
139
140                 OnIoRead( pObject, dwNumberOfBytes );
141                 break;
142
143             case IO_WRITE:
144                 OnIoWrote( pObject );
145                 break;
146
147             default:
148                 TLOG( LOG_DEBUG, _T( "[ERROR] Unknown value(%d)." ), ( pstOverlappedEx->IoType ) );
149                 break;
150             }
151         }
152     }
153 }
```

OnIoRead()에서 상대방(CPeer)으로부터 수신한 데이터를 읽는 함수(ReadPacketForIOCP) 호출



특정 상대방에 대해 송신이 완료되면 이 상대방에 해당하는 CPeer에서 또 송신해야 할 데이터가 있는지 확인하여 **재송신**

수신한 '날 데이터(Raw data)' 읽기

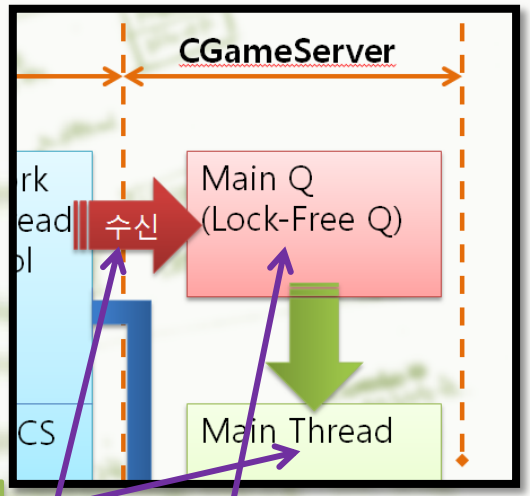
```
34  bool CPeer::ReadPacketForIOCP( __in DWORD dwReceiveSize/*=0*/, __out BYTE* pbyReceivedPacket/*=NULL*/, __in DWORD dwProcessData/*=0*/ )
35  {
36      INT iResult = 0;
37
38      //주의! 수신 데이터에 대한 1차 적재 처리와 2차 적재 처리에 비율
39      //      수신 데이터를 CNetwork::m_byReceiveBuffer에서 CPeer::m_byReceivedDataBuffer로 옮기는 것을
40      //      1차 적재라고 한다면 CPeer::m_byReceivedDataBuffer에서 수신용 Queue로 옮기는 것을 2차 적재라고 하자.
41      //      1차 적재 처리와 2차 적재 처리에 비율은 Packet을 수신하는 과정에서 Hang이 발생하는 것을 줄이는데 도움이 될 것 같다.
42      //      여기서 고려된 비율은 '1차 적재 처리 : 2차 적재 처리 = 2 : 3'이다.
43      //      즉, 1차 적재 처리가 2번 이뤄지면 2차 적재 처리는 3번 이뤄지도록 구현하여 1차 적재에 대비하여
44      //      CPeer::m_byReceivedDataBuffer가 항상 비워진 상태가 될 수 있도록 유도하는 것이다.
45      if( pbyReceivedPacket == NULL ) // 수신 후, 수신용 Queue에 삽입
46      {
47          if( dwReceiveSize )
48          {
49              CNetwork::m_dwNumberOfBytesRecvd = dwReceiveSize;
50
51              if( IntegratedLoadForRawData() == false )
52              {
53                  TLOG( LOG_DEBUG, _T("[ERROR] Failed to receive packet.") );
54                  return false;
55              }
56          }
57
58          // 수신 요청
59          iResult = CNetwork::Receiving();
60          if( iResult == NETWORK_IOC_STATE_FAILURE )
61          {
62              TLOG( LOG_DEBUG, _T("[ERROR] Failed to
63              return false;
64          }
```

수신한 데이터를 한 단위 패킷으로 쪼개고 버퍼를 정리합니다. 한 단위 패킷으로 쪼개진 데이터는 다음 페이지에서 설명하는 CGameServer의 멤버 함수 InsertIntoMainQ()에 의해 무잠금 큐(m_GameSMainQ)에 삽입됩니다.

```
273 VOID CGameServer::InsertIntoMainQ( TSession* pHostOfReceivedData, DWORD dwReceivedData )
274 {
275     UnitNode stUnitNode;
276     stUnitNode.pHostOfReceivedData = pHostOfReceivedData;
277     stUnitNode.dwReceivedData = dwReceivedData;
278
279     m_GameSMainQ.Enqueue( stUnitNode );
280 }
```

수신 패킷을 처리하는 CGameServer는?

```
7  class CGameServer : public TSingleton<CGameServer>
8  {
9      struct UnitNode
10     {
11         TSession* pHostOfReceivedData;
12         DWORD dwReceivedData;
13
14         UnitNode() : pHostOfReceivedData( NULL ), dwReceivedData( 0 ) {}
15     };
16
17 public:
18     CGameServer( VOID );
19     ~CGameServer( VOID ) {}
20
21     bool InitData( VOID );
22     bool Begin( VOID );
23     UINT GameSMainThread( VOID );
24     VOID InsertIntoMainQ( TSession* pHostOfReceivedData, DWORD dwReceivedData );
25     bool End( VOID );
26
27 private:
28     CLockFreeQueue<UnitNode> m_GameSMainQ;
29     TAuto_Ptr<CServerTcp> m_pTcpServer;
30 };
```



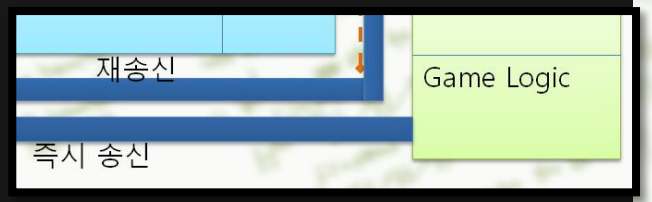
게임 로직을 처리하는 메인 스레드

IOCP 작업 스레드에 의해 호출되어 메인 스레드에게 넘길 작업 단위를 메인 큐에 삽입하는 함수

메인 스레드가 처리할 작업 단위가 담긴 Lock-Free Queue의 메인 큐

무잠금 큐에서 한 단위 작업 추출

```
195 UINT CGameServer::GameSMainThread(VOID)
196 {
197     //TODO: 일감 #157 관련, 서버 종료 처리 작업시 이 부분도 같이 종료시킬 수 있도록 한다.
198     DWORD dwCurtTick = 0, dwUpdateObjTick = timeGetTime(), dwChkTableTick = dwUpdateObjTick;
199
200 #ifdef TICK_RESOLUTION 비활성 전처리기 블록
202 #endif TICK_RESOLUTION
203
204     g_PerformanceCheck.resize(10);
205
206     m_bMainThreadRunning = true;
207
208     while(TRUE)
209     {
210         UnitNode stUnitNode;
211
212         //TODO: 임시 쓰레드 종료 처리
213         if(!m_bMainThreadRunning)
214         {
215             //현재까지 받은 모든 패킷 처리
216             while(m_GameSMainQ.Dequeue(stUnitNode))
217             {
218                 if(stUnitNode.pHostOfReceivedData)
219                     stUnitNode.pHostOfReceivedData->PacketProcess(stUnitNode.dwReceivedData);
220             }
221
222             break;
223         }
224
225         //주의! m_GameSMainQ.Size()의 부정확성 경고
226         //    if(m_GameSMainQ.Dequeue(stUnitNode)) 대신에 if(0<m_GameSMainQ.Size()) 으로
227         //    블럭 진입 여부 판단시, m_GameSMainQ.Size()의 부정확성을 반드시 개선해야 한다.
228         if(m_GameSMainQ.Dequeue(stUnitNode))
229         {
230             if(stUnitNode.pHostOfReceivedData)
231                 stUnitNode.pHostOfReceivedData->PacketProcess(stUnitNode.dwReceivedData);
232         }
```

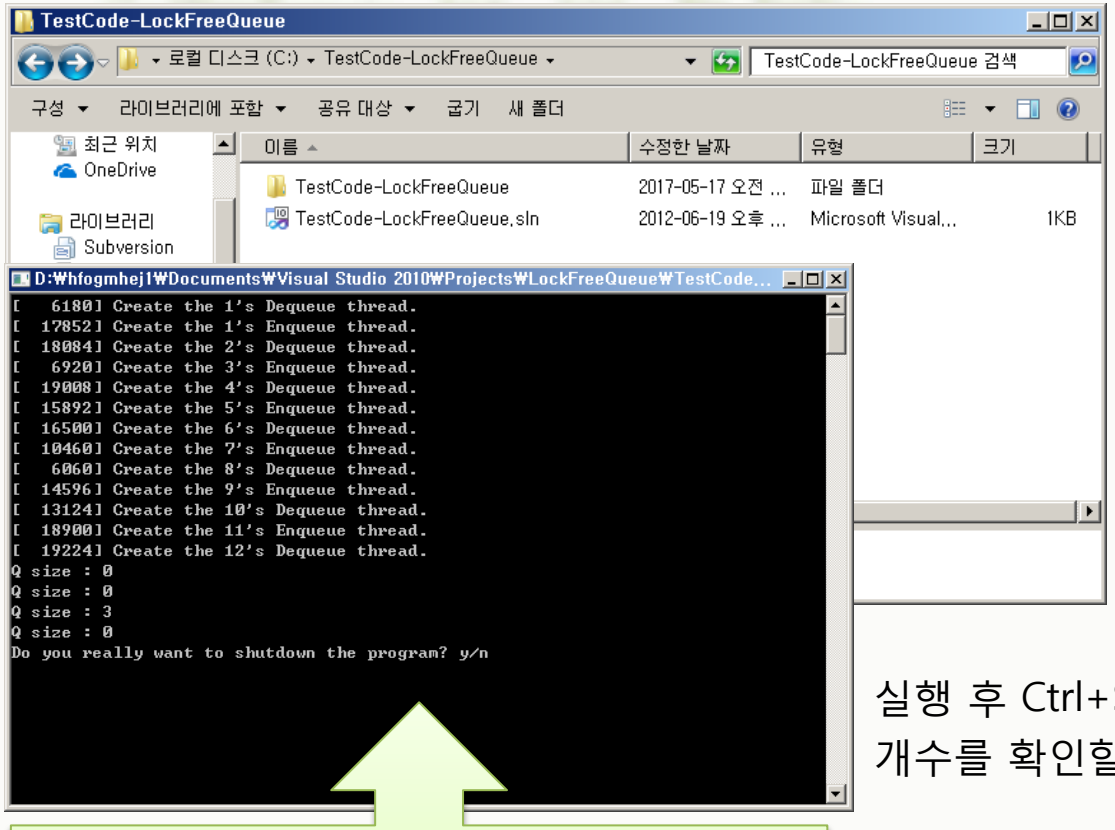


메인 스레드가 처리해야 할 작업 단위를 메인 큐에서 추출

수신된 패킷 프로토콜별 핸들러 호출

메인 스레드에 의해 메인 큐로부터 추출된 작업 단위를 처리하는 과정에서 게임 로직 처리가 이뤄지고 이 과정에서 필요할 경우 상대(CPeer)에게 송신(**즉시 송신**)하기도 합니다.

TestCode-LockFreeQueue는?



삽입(Enqueue) 스레드 6개, 추출(Dequeue) 스레드 7개가 서로 경쟁하고 있으며 임의의 순간 큐에 3개 작업이 쌓였던 적이 있는 상황을 볼 수 있습니다.

포트폴리오로 제출한 샘플 코드 TestCode-LockFreeQueue는 앞에서 설명한 '대륙전쟁' 게임서버의 메인 큐(Lock-free Queue)만 추출한 코드입니다.

컴파일 후 실행하게 되면 무잠금 큐를 중심으로 몇 개의 스레드가 서로 경쟁하며 데이터를 주고 받게 됩니다.

실행 후 Ctrl+S 를 누르면 무잠금 큐에 쌓인 작업 개수를 확인할 수 있습니다.

Ctrl+X 를 누르면 샘플 프로그램을 종료합니다.