



# MMORPG「Rodinia War」サーバー に使われているLock-free Queue

サーバープログラマー, リュウ ヒョンジン

MMOs

# 概要

ポートフォリオで提出したTestCode-LockFreeQueueは私がメイン（シニア）サーバープログラマーとして参加したプロジェクト、MMORPG「Rodinia War」のゲームサーバーに適用したロックフリーキューサンプルコードです。

ゲームサーバーにロックフリーキューを適用する前に安定性検査のためにこのサンプルコードを先に具現して何日間テストをしました。結果、実装しても問題ないと判断してゲームサーバーに適用したものです。

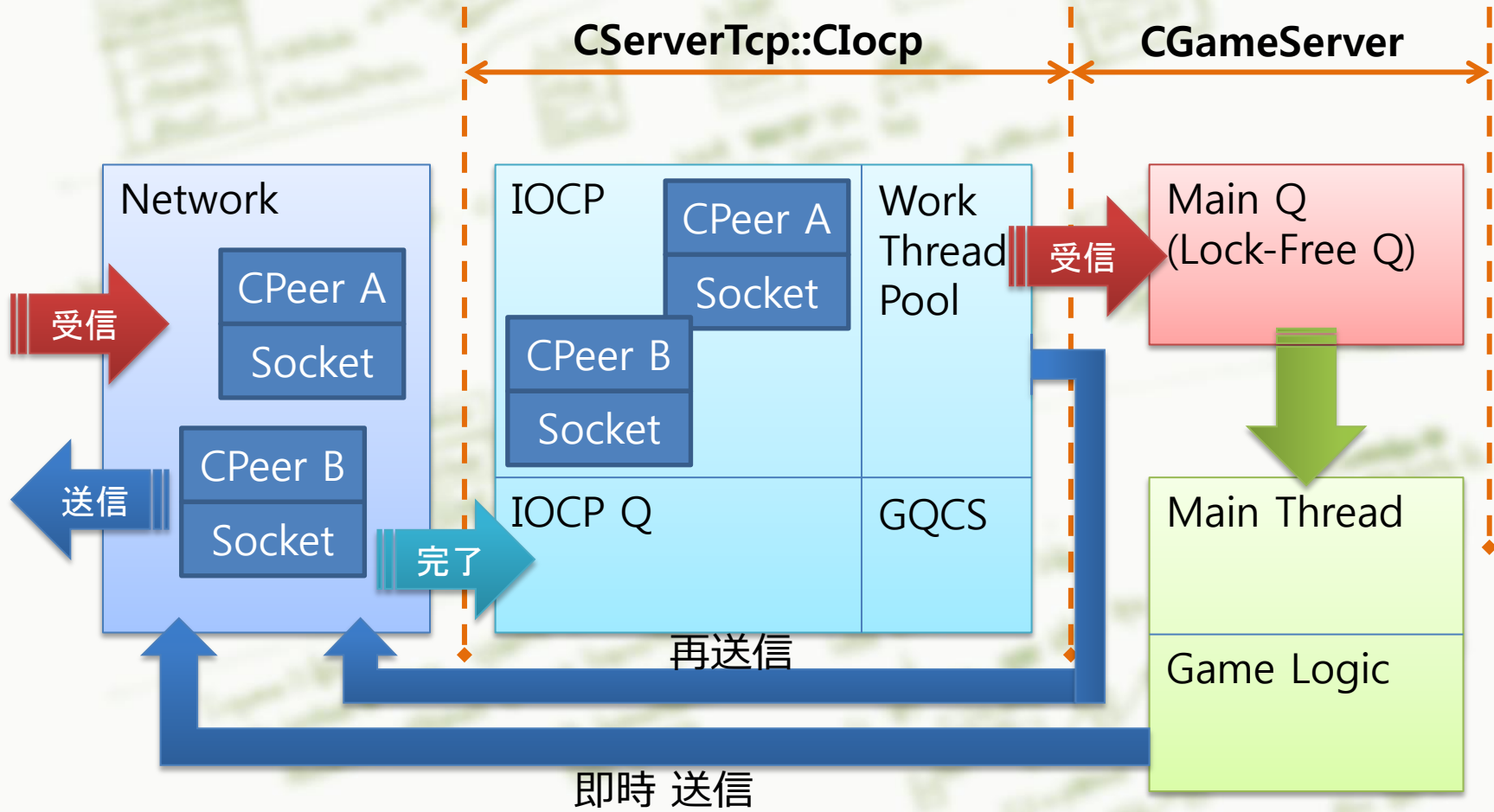
適用した後クローズベータテストを通して全然問題なく機能したので安定性は検証されたと言えます。

この資料ではロックフリーキューが「Rodinia War」ゲームサーバーにどうやって適用されているか説明しながらサンプルコードの使い方について案内します。このサンプルコードで私のコーディングスタイルも把握できると思います。

# 目次

- 「Rodinia War」 ゲームサーバーの内部構造 [4page](#)
- 通信相手 1 個固体のCPeer [5page](#)
- ゲームサーバーのIOCP(1) [6page](#)
- ゲームサーバーのIOCP(2) [7page](#)
- 受信した「ローデータ(Raw data)」読み [8page](#)
- 受信パケットを処理するCGameServerとは? [9page](#)
- ロックフリーキューで一個単位の値を取り出し [10page](#)
- TestCode-LockFreeQueueの使い方は? [11page](#)

# 「Rodinia War」 ゲームサーバーの内部構造





# 通信相手 1 個固体のCPeer

```
class CNetwork
{
public:
20 → CNetwork( __in INT iNetworkKind = NETWORK_KIND_DEFAULT );
21 → virtual ~CNetwork( VOID );
22
23 → BOOL CreateNBOLSocket( VOID ); // 'N' on 'B' locking 'O' ven 'L' apped
24 → bool CreateConnection( __in LPSTR lpszAddr, __in USHORT unsPort );
25
26 → BOOL Listen( USHORT port, INT backlog );
27 → BOOL Accept( SOCKET listenSocket ); // TODO: NPC 서버의 패킷 처리 부분 적용이 완료
28
29 protected:
30 → TCriticalSection m_CS;
31 → TCriticalSection m_CSForReceiving;
32 → TCriticalSection m_CSForSending;
33
34 → INT Receiving( VOID );
35 → INT Sending( __in BYTE* pbyData, __in DWORD dwDataLength );
36
37 → VOID UpdateResultOfOverlappedRecv( VOID );
38 → INT QueuingReceivedData( __in BYTE* pbyEmptySection, __in INT nRemainingLength );
39
40 private:
41 → INT m_iNetworkKind;
42 → SOCKET m_uni64NSocket;
43
44 → BYTE m_byReceiveBuffer[PACKET_BUFFER_SIZE];
45
46 → OVERLAPPED_EX m_stAcceptOverlappedEX;
47 → OVERLAPPED_EX m_stReceiveOverlappedEX;
48 → OVERLAPPED_EX m_stSendOverlappedEX;
49 };
```

CNetworkは送受信するための基本関数とソケット、バッファ、OVERLAPPED構造体をメンバーとして持っています。

CPeerはこのCNetworkを親にしてネットワークモデル別受信関数と相手から受信したり送信しないといけない未解決データを管理する資料構造をメンバーとして持っています。

つまり、**CPeerは通信相手一個固体を意味**します。

```
class CPeer : public CNetwork
{
public:
8
9 → CPeer( VOID );
10 → CPeer( __in INT iNetworkKind ): CNetwork( iNetworkKind );
11 → virtual ~CPeer( VOID );
12
13 → virtual bool ThrowToMainQ( DWORD dwReceivedData ) { return true; }; // TODO: WSAEventSelect 모델을 사용하는 곳에서는 Main.Q
14
15 → BOOL StartP( VOID );
16 → BOOL EndP( VOID );
17
18 → bool ReadPacketForIOCP( __in DWORD dwReceiveSize = 0, __out BYTE* pbyReceivedPacket = NULL, __in DWORD dwProcessData = 0 );
19 → bool ReadPacketForWSAES( __out BYTE* pbyReceivedPacket = NULL ); // for 'WSAE'vent'Select
20 → INT WritePacket( __in WORD wSendSize = 0, __in BYTE* pbySendData = NULL );
21
22 private:
23 → BYTE m_byReceivedDataBuffer[PACKET_BUFFER_SIZE * 3];
24 → INT m_nRemainingLength; // m_byReceivedDataBuffer에 빈 공간 크기
25
26 → CIOPacketDepository m_ReadPD;
27 → CIOPacketDepository m_WritePD;
28
29 → bool IntegratedLoadForRawData( VOID );
30 → INT LoadUpIntoQueue( VOID );
31 };
```

# ゲームサーバーの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     std::vector<HANDLE> mWorkerThreadVector;
14     HANDLE mStartupEventHandle;
15
16 protected:
17     virtual VOID OnIoRead(VOID *object, 'U32' dataLength) = 0;
18     virtual VOID OnIoWrote(VOID *object) = 0;
19     virtual VOID OnIoConnected(VOID *object) = 0;
20     virtual VOID OnIoDisconnected(VOID *object) = 0;
21
22 public:
23     BOOL Begin(VOID);
24     BOOL End(VOID);
25
26     BOOL RegisterSocketToIocp(SOCKET socket, 'ULONG_PTR' completionKey);
27     VOID WorkerThreadCallback(VOID);
28
29     virtual VOID OnIoForceKill(VOID *pObject) = 0;
30 };
31
```

IOCPのワークスレッド個数とワークスレッドハンドルを管理する資料構造

CIocpを親にする子クラスCServerTcpでは  
仮想関数を定義し、接続してくる相手たち  
(CPeer)の現況を資料構造で持っています。

完了したIOに対するハンドラー。  
子クラスCServerTcpで定義

IOCPで観察しようとする  
ソケットを登録する関数

GQCSがあるワーク  
スレッドコールバック関数

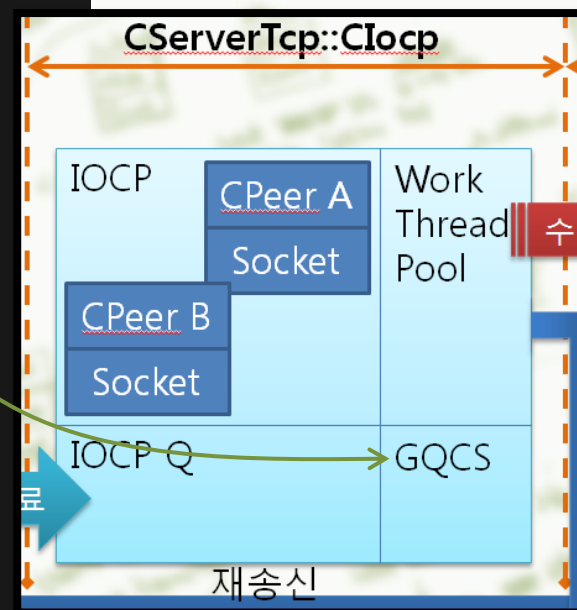
# ゲームサーバーのIOCP(2)

```

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) ), (pstOverlappedEx->IoType) );
149                     break;
150             }
151         }
152     }
153 }

```

OnIoRead()で相手(CPeer)から  
受信したデータを読む関数  
(ReadPacketForIOCP)を呼び出す



재송신

特定の相手に対して送信が完了したらこの相手に該当するCpeerで再び送信しなきゃいけないデータがあるかどうか確認して**再送信**

# 受信した「ローデータ(Raw data)」読み

```
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         }
65     }
```

受信したデータを一つの単位パケットに分けてバッファ를整理します。一つの単位パケットに分けたデータは次のページで説明するCGameServerのメンバー関数InsertIntoMainQ()によってロックフリーキュー(m\_GameSMainQ)に入れられます。

```
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 }
```



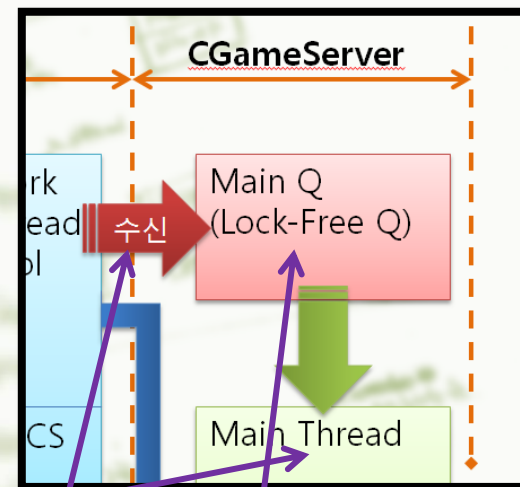
# 受信パケットを処理するCGameServerとは?

```
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 };
```

ゲームロジックを処理する  
メインスレッド

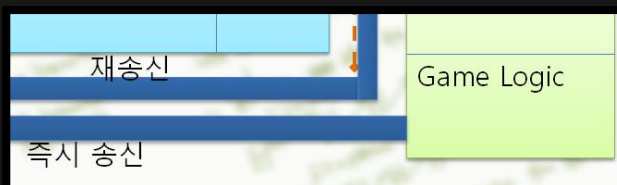
IOCPワークスレッドから呼ばれてメインスレッドに  
渡す作業単位をメインキューに入れる関数

メインスレッドが処理する作業単位が入っ  
ているLock-Free Queueのメインキュー



# ロックフリーキューで一個単位の値を取り出し

```
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
201 #endif TICK_RESOLUTION
202
203     g_PerformanceCheck.resize(10);
204
205     m_bMainThreadRunning = true;
206
207     while(TRUE)
208     {
209         UnitNode stUnitNode;
210
211         //TODO: 임시 쓰레드 종료 처리
212         if(!m_bMainThreadRunning)
213         {
214             //현재까지 받은 모든 패킷 처리
215             while(m_GameSMainQ.Dequeue(stUnitNode))
216             {
217                 if(stUnitNode.pHostOfReceivedData)
218                     stUnitNode.pHostOfReceivedData->PacketProcess(stUnitNode.dwReceivedData);
219             }
220             break;
221         }
222
223         //주의! m_GameSMainQ.Size()의 부정확성 경고
224         //if(m_GameSMainQ.Dequeue(stUnitNode)) 대신에 if(0 < m_GameSMainQ.Size())으로
225         //블럭 진입 여부 판단시, m_GameSMainQ.Size()의 부정확성을 반드시 개선해야 한다.
226         if(m_GameSMainQ.Dequeue(stUnitNode))
227         {
228             if(stUnitNode.pHostOfReceivedData)
229                 stUnitNode.pHostOfReceivedData->PacketProcess(stUnitNode.dwReceivedData);
230         }
231     }
232 }
```

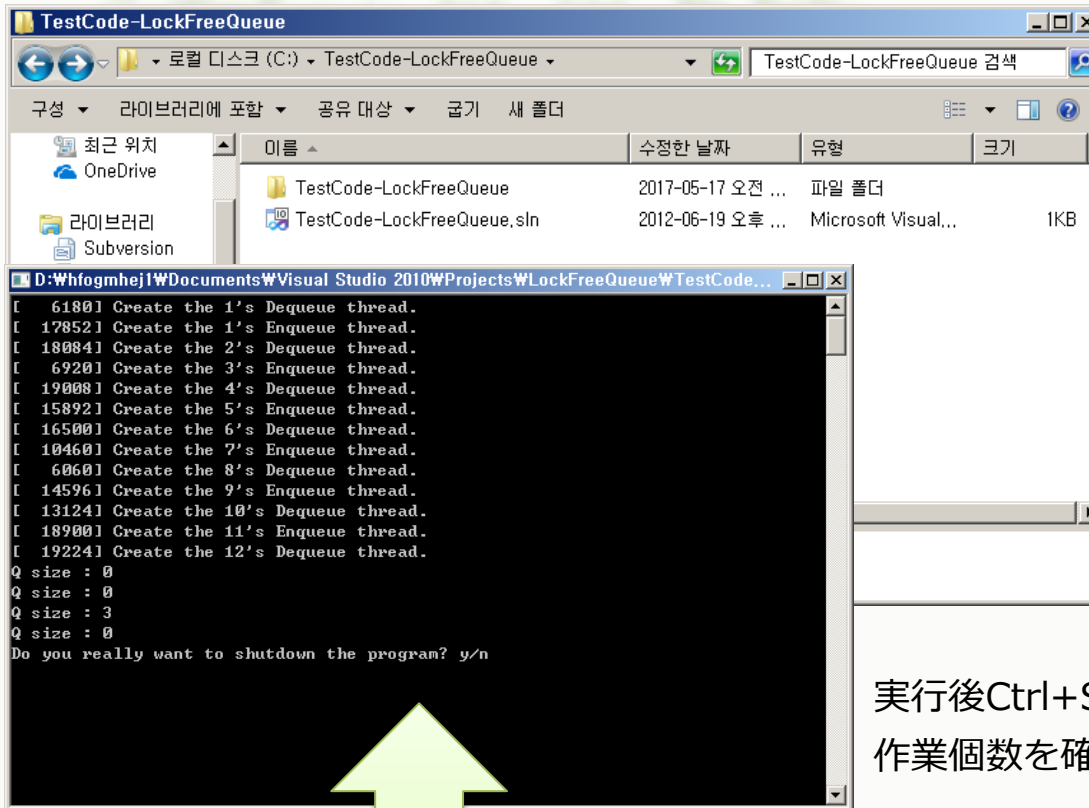


메인스레드가処理すべき作業單位をメイン큐어から抽出

메인스레드によって  
메인큐어에서 추출된  
작업單位를 처리하는  
중  
게임 로직 처리가  
이때, 필요에 따라  
상대방  
(CPeer)에 전송(**即時送信**)  
하는 경우도 있습니다.

受信したパケットプロトコ  
ル別ハンドラー呼び出し

# TestCode-LockFreeQueueの使い方は?



The screenshot shows the Visual Studio 2010 interface. The top window displays the 'TestCode-LockFreeQueue' project in the Solution Explorer. Below it, the Output window shows the execution log of the program. The log contains several lines of thread creation messages, followed by queue size reports, and a prompt to shutdown the program.

```
[ 6180] Create the 1's Dequeue thread.  
[ 17852] Create the 1's Enqueue thread.  
[ 18084] Create the 2's Dequeue thread.  
[ 6920] Create the 3's Enqueue thread.  
[ 19008] Create the 4's Dequeue thread.  
[ 15892] Create the 5's Enqueue thread.  
[ 16500] Create the 6's Dequeue thread.  
[ 10460] Create the 7's Enqueue thread.  
[ 6060] Create the 8's Dequeue thread.  
[ 14596] Create the 9's Enqueue thread.  
[ 13124] Create the 10's Dequeue thread.  
[ 18900] Create the 11's Enqueue thread.  
[ 19224] Create the 12's Dequeue thread.  
Q size : 0  
Q size : 0  
Q size : 3  
Q size : 0  
Do you really want to shutdown the program? y/n
```

挿入(Enqueue)スレッド 6 個, 抽出(Dequeue) スレッド 7 個がお互い競争している間任意の瞬間に 3 個の作業が積もったことがある状況が見られます。

ポートフォリオとして提出したサンプルコードTestCode-LockFreeQueueは今まで説明した「Rodinia War」ゲームサーバーのメインキュー(Lock-free Queue)だけ抽出したコードです。

コンパイル後実行するとロックフリーキューを中心として何個のスレッドがお互い競争しながらデータを取り交わします。

実行後Ctrl+Sを押すとロックフリーキューに積もった作業個数を確認できます。

Ctrl+Xを押すとサンプルプログラムを終了します。