```
//Starting point in Win_main.c

int WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
   Sys_CreateConsole
   Sys_Milliseconds
   Sys_InitStreamThread

   Com_Init
   {
      Com_InitPushEvent
      Com_InitSmallZoneMemory()
      Cvar_Init ();

      Com_ParseCommandLine( commandLine )

      Cbuf_Init ();


      Com_InitZoneMemory();
      Cmd_Init ();

      Com_StartupVariable( NULL );   // override anything from the config files with command line args
      Com_StartupVariable( "developer" ); // get the developer cvar set as early as possible


      CL_InitKeyCommands(); // done early so bind command exists

      FS_InitFilesystem ();

      Com_InitJournaling();

      Cbuf_AddText ("exec default.cfg\n");
      Cbuf_AddText ("exec autoexec.cfg\n");
      Cbuf_Execute ();

      Com_StartupVariable( NULL ); // override anything from the config files with command line args


      Com_InitHunkMemory(); // allocate the stack based hunk allocator
   }

   _getcwd                    //Get working directoty

   NET_Init
   {
      WSAStartup
      NET_GetCvars
      NET_GetCvars
   }

   while( 1 )
   {
      if ( g_wv.isMinimized || ( com_dedicated && com_dedicated->integer ) ) {
         Sleep( 5 );

      IN_Frame                    // mouse and joystick  : WFT Why not take keyboard input here as well ?
      {
         IN_JoyMove
             Sys_QueEvent         // Write to sysEvent_t        eventQue[MAX_QUED_EVENTS];
         IN_ActivateMouse
             Sys_QueEvent         // Write to sysEvent_t        eventQue[MAX_QUED_EVENTS];
         IN_MouseMove
             Sys_QueEvent         // Write to sysEvent_t        eventQue[MAX_QUED_EVENTS];
      }

      Com_Frame                   // run the game
      {
         key = 0x87243987;

         Com_WriteConfiguration   // write config file if anything changed

         Com_ModifyMsec


         // Pump the sysEvent_t queue and also pump UDP incoming queue
         Com_EventLoop
         {
              Com_GetEvent
               Com_GetRealEvent     // Journaling of event injection from journal is done here.
                 Sys_GetEvent       //pump network packets, console command and win32 messages: everything goes in the central eventQue
                 {
                   PeekMessage
                      GetMessage
                   Sys_ConsoleInput
                   Sys_GetPacket
                      recvfrom
                   return eventQue
                 }

               //Pump every single event from the queue eventQue

               Com_RunAndTimeServerPacket
               {
                   SV_PacketEvent
                   {
                     for (i=0, cl=svs.clients ; i < sv_maxclients->integer ; i++,cl++)
                     {
                         SV_Netchan_Process       //This is where the client pumps messages from the server.
                            SV_Netchan_Decode
                         SV_ExecuteClientMessage
                            SV_UserMove
                              SV_ClientThink
                                 VM_Call( gvm, GAME_CLIENT_THINK, cl - svs.clients );   //WTF: Why do we have the server VM called here ?
```

```
                }
             }
          }
          or
                      CL_PacketEvent
                        CL_Netchan_Process(&clc.netchan, msg)
                            CL_ParseServerMessage
                            {
                                CL_ParseCommandString
                                  or
                                CL_ParseGamestate
                                      or
                                    CL_ParseSnapshot
                                  or
                                CL_ParseDownload
                            }

    }
    Cbuf_Execute (); // After the event queue has been pumped, execute any command in the buffer
       Cmd_ExecuteString


  SV_Frame(msec)
  {
     if ( SV_CheckPaused() )
        return;

     if (!com_dedicated->integer)
        SV_BotFrame( svs.time + sv.timeResidual );   //Botlib.lib

     if ( com_dedicated->integer && sv.timeResidual < frameMsec )
     NET_Sleep(frameMsec - sv.timeResidual)  // No point looping since we would return over and over again:
     {
         //Q: WTF ?!??!!  Why is NET_Sleep empty on Windows? It is doing the job on Unix (unix_net.c)
         //A: http://icculus.org/pipermail/quake3/2007-August/001910.html
         //   Maybe noone cared if the dedicated server burns cpu cycles on Windows.

     }
     SV_CalcPings

     // run the game simulation in chunks at a FIXED FREQUENCY (10Hz, every 100ms)
     while ( sv.timeResidual >= frameMsec )
     {
        VM_Call( gvm, GAME_RUN_FRAME, svs.time )  //Calling the game VM
        {

           if ( gvm->entryPoint )
               gvm->entryPoint(callnum,argvs)      // if we have a dll loaded, call it directly
                  G_RunFrame                       //Advances the non-player objects in the world
           else
           if ( gvm->compiled )
              VM_CallCompiled( gvm, &callnum );
                  G_RunFrame                       //Advances the non-player objects in the world
           else
              VM_CallInterpreted( gvm, &callnum );
                  G_RunFrame                       //Advances the non-player objects in the world

        }
     }

     SV_CheckTimeouts
     SV_SendClientMessages
     {
         for (i=0, c = svs.clients ; i < sv_maxclients->integer ; i++, c++) {
         {
             if ( svs.time < c->nextSnapshotTime )
                              continue;          // not time yet

                         // send additional message fragments if the last message
                                 // was too large to send at once
                                 if ( c->netchan.unsentFragments ) {
                                         c->nextSnapshotTime = svs.time + SV_RateMsec( c, c->netchan.unsentLength - c->netchan.unsentFragmentStart );
                                         SV_Netchan_TransmitNextFragment( c );
                                         continue;
                                 }

             SV_SendClientSnapshot( c );
             {
                SV_BuildClientSnapshot     //Decides which entities are going to be visible to the client, and copies off the playerstate and areabits.
                MSG_Init (&msg, msg_buf, sizeof(msg_buf));
                MSG_WriteLong
                SV_UpdateServerCommandsToClient
                SV_WriteSnapshotToClient
                    //Decide if we can delta encode from the last frame the client had or if a full snapshot is required.
                    //The server keeps a list of the 32 last frames from each clients.
                    SV_EmitPacketEntities // // delta encode the entities
                SV_WriteDownloadToClient
                SV_SendMessageToClient
             }
         }
     }
     SV_MasterHeartbeat                //Send a message to the masters every few minutes
  }

  if ( !com_dedicated->integer )
  {

     Com_EventLoop();
             Cbuf_Execute ();
                  Cmd_ExecuteString

     CL_Frame
     {
         CL_CheckUserinfo     // see if we need to update any userinfo
         CL_CheckTimeout      // if we haven't gotten a packet in a long time drop the connection
```

```
CL_SendCmd           // send intentions now
 {
                     CL_CreateNewCommands
                        CL_CreateCmd is called each frame and ask each input to fill a usercmd_t structure
                        {
                           Com_Memset( &cmd, 0, sizeof( cmd ) );
                           CL_CmdButtons( &cmd );
                           CL_KeyMove( &cmd );
                           CL_MouseMove( &cmd );
                           CL_JoystickMove( &cmd );
                           CL_FinishMove( &cmd );
                        }
                     CL_WritePacket
                     {
                            ===================
                                CL_WritePacket

                                Create and send the command packet to the server
                                Including both the reliable commands and the usercmds

                                During normal gameplay, a client packet will contain something like:

                                4       sequence number
                                2       qport
                                4       serverid
                                4       acknowledged sequence number
                                4       clc.serverCommandSequence
                                <optional reliable commands>
                                1       clc_move or clc_moveNoDelta
                                1       command count
                                <count * usercmds>

                                ===================

                                CL_Netchan_Transmit    //Send ONE(1) command to the server
                                   Netchan_Transmit
                                       NET_SendPacket
                                       {
                                          NET_SendLoopPacket
                                                 or
                                          Sys_SendPacket
                                       }
                         }
                }


CL_CheckForResend(); // resend a connection request if necessary
CL_SetCGameTime();   // decide on the serverTime to render

SCR_UpdateScreen();  // update the screen
{
   SCR_DrawScreenField( STEREO_CENTER );
   {
        re.BeginFrame( stereoFrame );        // Renderer.lib

        switch ( cls.state):                  // Depending on the gamestate will either send a message to ui VM or cg VM
        {
            CA_CINEMATIC     SCR_DrawCinematic
            CA_DISCONNECTED  VM_Call( uivm, UI_SET_ACTIVE_MENU, UIMENU_MAIN );
            CA_CONNECTED     VM_Call( uivm, UI_REFRESH, cls.realtime );
                             VM_Call( uivm, UI_DRAW_CONNECT_SCREEN, qfalse );
            CA_PRIMED        CL_CGameRendering( stereoFrame );
                             {
                               VM_Call( cgvm, CG_DRAW_ACTIVE_FRAME, cl.serverTime, stereo, clc.demoplaying );
                                    CG_DrawActiveFrame
                             }
                             VM_Call( uivm, UI_REFRESH, cls.realtime );
                             VM_Call( uivm, UI_DRAW_CONNECT_SCREEN, qtrue );
            CA_ACTIVE        CL_CGameRendering( stereoFrame );
                             {
                               VM_Call( cgvm, CG_DRAW_ACTIVE_FRAME, cl.serverTime, stereo, clc.demoplaying );  // Here we jump in the VM
                                 CG_DrawActiveFrame
                                    CG_PredictPlayerState

                                    CG_AddPacketEntities
                                    CG_AddMarks
                                    CG_AddParticles
                                    CG_AddLocalEntities

                                    CG_PlayBufferedSounds
                                    CG_PlayBufferedVoiceChats



                                    CG_DrawActive
                                    {
                                         trap_R_RenderScene
                                           syscall( CG_R_RENDERSCENE, fd )          // Here we jump back to Quake3.exe via a system call
                                             RE_RenderScene
                                                 R_RenderView
                                                 {
                                                    R_RotateForViewer
                                                    R_SetupFrustum
                                                    R_GenerateDrawSurfs
                                                    {
                                                      R_AddWorldSurfaces
                                                      {
                                                          R_MarkLeaves
                                                          {
                                                             R_PointInLeaf
                                                             R_ClusterPVS
                                                          }
                                                          ClearBounds
                                                          R_RecursiveWorldNode
```

```
                                                                       }
                                                                       R_AddPolygonSurfaces
                                                                       R_SetupProjection
                                                                       R_AddEntitySurfaces
                                                                     }
                                                                     R_SortDrawSurfs
                                                                     R_DebugGraphics
                                                                  }
                                                      }
                                             }
                                             SCR_DrawDemoRecording();
                        }
                        Con_DrawConsole
                   }
                   re.EndFrame( &time_frontend, &time_backend );
                       R_IssueRenderCommands           //Frontend: if running in SMP this method will block until the backend is done.
                           RB_ExecuteRenderCommands    //Backend : if runninf in SMP with will be skipped since RB_ExecuteRenderCommands is in an infinite
loop in its own thread
                }

                S_Update();          // update audio
                SCR_RunCinematic();  // advance local effects for next frame

                Con_RunConsole();
                cls.framecount++;
            }

         }

         key = lastTime * 0x87243987;
         com_frameNumber++;
      }
   }
}
```

Question for John Carmack:

- Why is NET_Sleep doing nothing on Windows stations ?

- The renderer seems to have a frontend and backend just like Doom3 via r_smp
   backEnd.smpFrame = 0 | 1

```
   ResetEvent           Sets the specified event object to the nonsignaled state.
   SetEvent             Sets the specified event object to the signaled state.
   WaitForSingleObject  Waits until the specified object is in the signaled state or the time-out interval elapses.

   renderCommandsEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
   renderCompletedEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
   renderActiveEvent = CreateEvent( NULL, TRUE, FALSE, NULL );

   Frontend :
   ==========
    R_IssueRenderCommands
    {
      GLimp_FrontEndSleep
         WaitForSingleObject( renderCompletedEvent, INFINITE );

      GLimp_WakeRenderer
      {
        // after this, the renderer can continue through GLimp_RendererSleep
            SetEvent( renderCommandsEvent );

            *********************************************************
            ***** WaitForSingleObject( renderActiveEvent, INFINITE );
            *********************************************************

      }
    }

   Backend :
   =========

    GLimp_RenderThreadWrapper
    {
         glimpRenderThread
         {
             RB_RenderThread
             {
                 const void     *data;
                 while ( 1 )
                 {
                     data = GLimp_RendererSleep();
                     {
                         *********************************************************
                         ***** ResetEvent( renderActiveEvent );
                         *********************************************************

                         SetEvent( renderCompletedEvent );
                         WaitForSingleObject( renderCommandsEvent, INFINITE );

                         ResetEvent( renderCompletedEvent );
                         ResetEvent( renderCommandsEvent );

                         data = smpData;
```

```
                        ********************************************************
                        ***** SetEvent( renderActiveEvent );
                        ********************************************************

                        return data;
                    }
                    renderThreadActive = qtrue;
                    RB_ExecuteRenderCommands( data );
                    renderThreadActive = qfalse;
                }
            }
        }
        qwglMakeCurrent
    }
```

    WTF: 3 locks ? Why god why ? The SMP code seems to be really messy...why not a single event object with producer/consumer JAVA like model ?!

    The frontend will block until the backend is done flipping rendercommand buffer.
    I wonder how the synchronization method worked considering WindowsNT 10ms granularity. Win98 was way better and had something like 1ms

    Also it seems renderThreadActive is marked as volatile but this is a misusage: volatile does not guaranty synchronization.
    This is not too bad since renderThreadActive seems to be used for statistics only.

    Two locks are used to protect the transfert of smpData to data: renderActiveEvent and renderCommandsEvent

    The VSD alternatively issue backEndData_t to backEndData[0] or backEndData[1] in a double buffering mecanism.

- Why not have included the SMP code in Doom3 if it was there is idTech3 ? I cannot even find relica of SMP synchronization between the frontend/backend in Doom III, where was it ?

```
SV_ClientCommand
   SV_ExecuteClientCommand
       VM_Call( gvm, GAME_CLIENT_COMMAND, cl - svs.clients );
```