대 한 조 선 학 회 논 문 집
제 40 권 제 6 호 2003 년 12 월

Journal of the Society of Naval Architects of Korea
Vol. 40, No. 6, pp. 69-79, December 2003

인터넷 기반 프로펠러 설계 시스템 개발

이왕수†*, 박범진**, 이창섭*

충남대학교 공과대학 선박해양공학과*, 충남대학교 공과대학 항공우주공학과**

Development of Internet-Based Propeller Design System

Wang-Soo Lee*, Bum-Jin Park** and Chang-Sup Lee*

Dept. of Naval Architecture and Ocean Eng., Chungnam National Univ. *
Dept. of Aerospace Eng., Chungnam National Univ. **

Abstract

Existing large-scale complex programs usually reside in a single computer, and the user has to be physically in contact with the computer. With the wide spread use of the internet, the need to carry out the design and analysis tasks geographically away from the main computer is increasing. In this paper existing Windows-based propeller design and analysis package is separated into the server-client modules and the protocol program is developed to implement the communication between multi-client computers and a single server computer. A new protocol packet is designed to use the Windows socket and the server/client programs control the receive/send operations using the information transmitted in the packet. Test runs show that the remote user, connected to the server computer through the internet only, can perform the required tasks.

※Keywords : client-server computing(클라이언트-서버간 계산), protocol(프로토콜), internet(인터넷), propeller design(프로펠러 디자인)

1. 서론

기존의 프로펠러 설계는 개인용 컴퓨터급 이상의 컴퓨터에서 수행되고 있다. 대부분의 경우 기존 설계의 각 단계를 전산화한 프로그램이 가장 널리 쓰이고 있으며, 일부는 Microsoft 의 Windows 환경에서 설계를 수행하도록 한 종합설계 패키지(Propeller Design and Analysis System, ProDAS)가 개발되어 있다.(정인숙 1996, 정인숙 1997, Lee 2001)

그러나 지금까지 개발된 소프트웨어의 특징은

접수일: 2003 년 5 월 28 일, 승인일:2003 년 10 월 24 일
†주저자, E-mail : amacav@msn.com
Tel : 042-821-7762

모든 계산이 설계자가 직접 손으로 접촉할 수 있는 컴퓨터에서 수행되었다는 것이다. 일반 컴퓨터 사용자와 서버가 분리되어 운영되는 예로서는 이경호/이동곤(1999)에서 볼 수 있는 것과 같은 선박의 구난지원 시스템으로 사고 지점과 문제 해결 방법을 찾는 위치가 다른 경우에 당연히 인터넷과 같은 통신 방법을 필요로 하는 경우가 있으며, 이상욱/이규열(1999)에서 보는 바와 같이 서로 다른 시스템 사이의 통합을 위하여 에이전트 통신 언어를 사용하는 예를 들 수 있다.

인터넷이 널리 보급된 지금 설계를 반드시 모든 소프트웨어가 들어 있는 컴퓨터에서 설계를 수행할 필요는 없다. 특히 설계 프로그램이 제한적인 컴퓨터에서만 설치되는 경우에는 온라인 상에서 프로그램을 직접 접속 사용하여야 한다.

이와 같은 프로그램 설치 위치에 따른 제한을 극복하기 위해서는 프로그램은 서버에 두고, 인터넷을 통하여 자료를 주고 받으며 설계를 수행할 수 있는 능력이 필요하게 되었다. 즉 서버-클라이언트로 공간적으로 떨어진 상태에서 인터넷을 통하여 설계를 수행하는 프로그램의 개발이 필요해졌다. 본 개발의 내용은 다음 세가지로 요약할 수 있다.

1. 기존의 개인용 PC 에서 작동하던 프로펠러 설계 프로그램 ProDAS 를 네트워크상에서 실행 가능하도록 한다. 이를 위하여 우선 프로그램 모듈을 보관하고 있고, 원격 사용자가 접속하여 계산을 수행할 수 있도록, 서버 프로그램과 클라이언트 프로그램으로 나누어 원하는 프로그램을 수행할 수 있도록 한다.
2. 서버 프로그램은 ProDAS 의 모든 모듈 프로그램을 보관하고 있으며, 클라이언트(사용자, 설계자)의 접속을 관리하고, 사용자의 요청을 받아 원하는 프로그램을 사용하여 계산을 수행하며, 그 계산 결과를 클라이언트에게 보내주는 역할을 수행한다.
3. 클라이언트 프로그램은 사용자가 네트웍을 통하여 서버 컴퓨터에 접속하고, 사용자가 준비한 자료를 서버에게 보내어, 서버가 처리한 계산 결과를 받을 수 있도록 한다.

네트웍을 통하여 계산을 수행하기 위하여는 네트워크 프로그래밍에 사용되는 윈속(WinSock)을 이해할 필요가 있다.(Jones 2003) 윈속은 하나 이상의 응용프로그램들(혹은 프로세스들)이 같은 장치 내에서 또는 네트웍을 통하여 통신하기 위한 프로그래밍 인터페이스(API)이며 주로 네트웍을 통하여 데이터를 통신하기 위한 목적으로 설계되었다. 윈속은 TCP/IP 와 같은 네트웍 프로토콜에 대한 프로그래밍 인터페이스를 제공한다. 우리는 우선 하나의 장치로부터 네트웍상의 다른 장치와 연결을 시도하고, 연결을 받아들이고, 데이터를 주고받는지를 보이고자 한다. 윈속 응용프로그램은 기본적으로 윈속의 생성, 연결, 연결 수락, 송신, 수신 등을 수행한다.

Fig. 1 은 윈속 서버와 클라이언트가 어떤 기능을 갖고 있나를 보여준다. 서버나 클라이언트에서 socket() 함수 또는 WSASocket() 함수는 소켓을 생성하는 일을 수행한다. 서버는 bind()함수를 이용하여 소켓에 클라이언트가 연결할 수 있는 인터넷 주소와 포트번호를 할당하고, listen()으로 클라이언트가 접속하는 것을 기다리는 대기상태(listening mode)로 만들고 클라이언트가 connect()로 연결을 시도하면, accept()로 연결을 수락(accepting connections)한다.

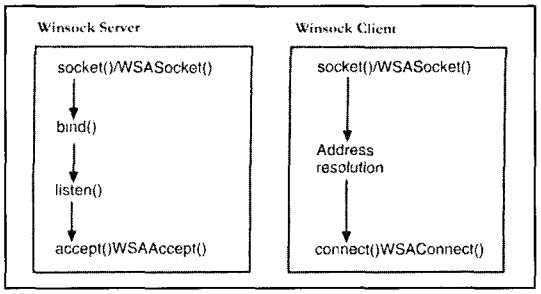

Fig. 1 Winsock server and client

응용프로그램이 TCP 를 통하여 통신하면, 두 개의 지점간 가상 연결(virtual connection)이 이루어지며 양방향으로 스트림(stream) 방식의 통신이 가능해 진다. 원격지에 위치한 클라이언트는 자료를 서버에 보내어 원하는 계산을 수행하게 되며,

서버는 그 결과를 다시 클라이언트에 보내게 된다. 이 방법을 이용하여 설계용 계산 프로그램(서버)에서 멀리 떨어져 있는 설계자(클라이언트)가 원격지에서 설계를 수행하고 그 결과를 즉시 확인할 수 있게 되는 것이다.

2. 스트리밍 프로토콜 설계

데이터 송수신은 서버와 클라이언트가 연결된 상태에서 데이터의 크기 정보를 포함하는 패킷 형태로 이루어진다. 메시지의 크기가 가변적인 경우도 다룰 수 있도록 스트리밍 프로토콜(streaming protocol, 문자열 통신규약)은 다음과 같은 통신 패킷 형식으로 정의한다. 한 패킷은 Table 1 및 2에서 보는 바와 같이 헤더(4 바이트)와 바디로 구성되며, 헤더는 바디부분 전체의 사이즈(2 바이트)와 패킷타입(2 바이트)로 정의된다.

Table 1 Design of protocol packet

통신 패킷(packet)		
헤더(header)		바디(body)
바디사이즈 (2 바이트)	패킷타입(2 바이트)	데이터

패킷의 바디 부분에 문자열을 넣을 때에는 앞에 반드시 문자열의 사이즈 정보와 문자열을 이어서 넣어야 한다. 바디 부분은 이러한 문자열 쌍이 여러 번 반복될 수 있다.

Table 2 Formation of multiple datasets in a packet

바디			
문자열 A 의 사이즈 (2 바이트)	문자열 A	문자열 B 의 사이즈 (2 바이트)	문자열 B

다음에는 패킷 타입에 따라 호출되는 함수를 각각 선언한다(함수 베이스 방식).

예를 들어, 패킷 타입이 REQUEST_LOGIN 이면 OnRequestLogin() 함수를 부르게 된다. Table 3은 통신 패킷을 생성하거나 해석할 때는 packet.h 헤더 파일에서 선언된 CPakcet 클래스를 이용하는 예를 보여준다.

Table 3 Sample code showing usage of CPacket class object

```
#define REQUEST_LOGIN 0 // 프로토콜에서 사용될 헤더 정보 (protocol.h 에 enum 선언)

char buf[128];  // 문자열 정보를 담을 버퍼 선언
char data[256]; // 패킷으로 사용할 버퍼 선언
CPacket packet; // 패킷 클래스 객체 선언

strcpy(buf, " 전송하려는 문자열" ); // buf 에 " … …" 에 있는 문자열 정보 복사

packet.SetBuf(data);              // 객체에서 사용할 버퍼 설정
packet.PutShort(strlen(buf));     // 패킷의 바디 부분에 문자열 사이즈 기록
packet.PutText(buf, strlen(buf)); // 패킷의 바디 부분에 문자열 기록
packet.SetHeader(REQUEST_LOGIN); // 패킷의 헤더에 바디크기 및 패킷타입 기록
```

3. 서버 프로그램 파일 구성

통신을 위하여는 약속을 하고 지켜야 할 것이 있다. 그 중에서 송신, 수신하는 패킷에는 정보의 크기, 종류, 내용 등을 기록하는 것이 가장 기본이다. 정보의 종류는 Table 4 에서 보는 바와 같이 protocol.h 헤더 파일에 열거형으로 정의된다.

Table 4 Declaration of the type of packet in a protocol header file

```
// protocol.h
enum {
  REQUEST_LOGIN = 0,
```

```
        ... ... ...
    REQUEST_INPUT_KPDL,
    ANSWER_OUTPUT_KPDL,
    INVALID_HEADER
  };
```

송수신의 한쪽 주체인 서버 정보는 Table 5 에서 보는 바와 같은 구조체로 선언된다. 전역변수로 SERVERCONTEXT Server 객체가 만들어진다.

Table 5 The structure containing server and socket information.

```
  typedef struct {
    SOCKET sockListener;
    HANDLE hIocpWorkTcp,  hIocpAcpt;
    SOCKETCONTEXT *sc;
    OBJECTNODE *pn;
    CProcess *ps;
    CDataBase *db;
    int       iMaxUserNum,      iCurUserNum,
iInWorkerTNum, iDataBaseTNum,
          iPortNum, iMaxProcess;
  } SERVERCONTEXT, *LPSERVERCONTEXT;

  extern SERVERCONTEXT Server;
```

다음 Table 6 은 패킷 생성 및 해석을 위한 클래스 선언의 일부를 보여준다. 이를 사용하는 방법은 Table 3 에 보여준 바 있다. Table 3 과 6 으로부터 통신 패킷을 만들고 사용하는 방법을 이해할 수 있다.

Table 6 Definition of CPacket class

```
  class CPacket {
  public:
    CPacket();            // Ctor
    virtual ~CPacket();   // Dtor
  private:
    char *cpPacketBegin, *cpPacketEnd;
  public:
    void SetBuf( char *cpBuf );
    int SetHeader( short sType );
    void GetHeader( short *sBodySize, short
*sType );
    ... ... ...
    void PutText( char *cpData, int iSize );
    void GetText( char *cpData, int iSize );
  };
```

Table 5 에서 서버와 소켓을 구조체로 정의한 것과 유사하게, 사용자를 정의하기 위한 구조체 선언, 데이터 송수신을 위한 버퍼 관리 함수 선언, 기본적인 데이터 송수신 함수 선언, 패킷 해석을 위한 함수 포인터 타입 선언, 클라이언트 접속허가 스레드(thread) 선언, 클라이언트와의 응답을 처리하는 작업 스레드 선언 등 I/O 와 관련된 함수를 선언하고 정의해 줄 필요가 있으나 내용이 방대하고 전문가의 협력으로 대체할 수 있는 부분이므로 여기서는 생략하기로 한다.

서버에서는 정기적으로 큐를 검사하여 데이터를 처리하여야 한다. Table7 은 이를 처리하기 위한 함수를 호출하는 스레드를 선언하고, 실제 패킷을 처리하는 함수(여기서는 kpdl() 함수로부터의 입력을 처리하는 OnRequestInputKpdl() 함수)를 친구 함수로 선언하는 CProcess 클래스의 일부를 보인다.

Table 7 Typical portion of CProcess class and friend member functions

```
  #define GAMERINGBUFSIZE          8192

  class CProcess {
  public:
    CProcess();           // Ctor
    virtual ~CProcess();  // Dtor
  private:
```

```
    int iIndex;
    CRITICAL_SECTION cs;
    LPSOCKETCONTEXT
iGameRingBuf[GAMERINGBUFSIZE];
    int iBegin, iEnd, iRestCount;
    HANDLE hGameTEvent;
  public:
    void InitProcess( int index );
    void
GameBufEnqueue(          LPSOCKETCONTEXT
lpSockContext );
    void
GameBufDequeue(          LPSOCKETCONTEXT
*lpSockContext );
    friend     unsigned     int     __stdcall
GameTProc( void *pParam );
... ... ...
    // processing function for input data from
client
    friend                                int
OnRequestInputKpdl(      LPSOCKETCONTEXT
lpSockContext,
          char *cpPacket );
  };
```

프로세스 루틴에 있는 InitProcessLayer() 함수는 서버의 여러 개의 프로세스를 초기화하고, 함수포인터 배열에서 호출될 함수를 등록한다. Table 8 은 Process.cpp 에 정의된 InitProcessLayer() 함수의 일부를 보여 준다. 또한 Table 8 의 후반부에는 클라이언트로부터 kpdl() 프로그램을 계산하기 위한 입력 자료를 패킷으로 받아 프로젝트 디렉토리에 저장하고, DLL 프로그램으로 컴파일된 kpdl() 프로그램을 기동시킨다. kpdl() 프로그램이 정상적으로 수행되면, 계산 결과가 프로젝트 디렉토리에 기록되게 되므로, 이를 읽어서 다시 패킷으로 클라이언트에게 보내는 OnRequestInputKpdl()를 보여준다.

Table 8 A portion of process routine in server

```
// declare dll extern function
extern __declspec(dllimport) void kpdl();

int InitProcessLayer() {
  int iN;
  Server.ps              =              new
CProcess[Server.iMaxProcess];

  if ( Server.ps == NULL )  return 0;

for  ( iN=0;  iN<Server.iMaxProcess;  iN++ )
    {
Server.ps[iN].InitProcess( iN );
}
... ... ...
OnTransFunc[REQUEST_INPUT_KPDL].proc
= OnRequestInputKpdl;
return 1;
  }
  int
OnRequestInputKpdl(     LPSOCKETCONTEXT
lpSockContext, char *cpPacket ) {
// lpSockContext : client socket context,
cpPacket : packet data
printf( "OnRequestInputKpdl…₩n" );

CPacket packet;
int iSize;
char data[1024];
short size;
memset(data, 0, sizeof(data));

// analyze received packet
packet.SetBuf(cpPacket); // set buf to read
packet.GetShort(&size); // get string size
packet.GetText(data, size); // get string using
size
printf("Received Data:₩n%s₩n", data);
```

```
// write input file into the project directory
FILE *fp;
fp = fopen("kpdl.dat", "w");
fprintf(fp, "%s", data);
fclose(fp);

// call function to work in DLL;
kpdl();

// read output file if "kpdl.out" was made from
DLL
char output[1028] = {0,};
int count = 0;
fp = fopen("kpdl.out", "r");

while( !feof( fp ) ) output[count++] =
fgetc(fp);
fclose(fp);

// make the packet to send to client
packet.SetBuf(data);        // set buffer
packet.PutShort(count-1); // put string length
packet.PutText(output, count-1); // put string
iSize                                    =
packet.SetHeader(ANSWER_OUTPUT_KPDL);
// set header type
PostTcpSend(1, (int *)&lpSockContext, data,
iSize); // send packet to client
return 1;
  }
```

4.클라이언트 프로그램 파일 구성

클라이언트 프로그램은 사용자가 사용하기 편리하게 윈도우(Windows) 프로그램으로 작성하기로 한다. 윈도우(Windows) 프로그램은 마이크로소프트(Microsoft)사의 MFC(Microsoft Foundation Class)(Microsoft Visual C++ 1994)를 사용하여 쉽게 코딩 작업을 수행할 수 있다. Windows 프로그램을 작성하여 프로그램을 개발한 예는 정인숙등(정인숙 1996, 정인숙 1997)에서 찾아볼 수 있어 여기에서는 상세한 설명을 생략한다.

Table 9 는 CClientApp 클래스의 유일무이한 객체 theApp 를 생성하고, 클라이언트 창을 초기화하고, AfxSocketInit() 함수를 이용하여 통신을 위한 소켓을 작동시키며, CClientDlg 클래스의 객체 생성을 통하여 사용자와의 대화를 가능하게 할 대화상자를 만드는 과정의 핵심과정을 보여주고 있다.

Table 9 Program segments showing client initialization and opening dialog box

```
  // CClientApp
... ... ...
  // The one and only CClientApp object
  CClientApp theApp;

  // CClientApp initialization
  BOOL CClientApp::InitInstance() {
if (!AfxSocketInit()) {
AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
return FALSE;
}
... ... ...
CClientDlg dlg;
m_pMainWnd = &dlg;
int nResponse = dlg.DoModal();  // create
dialog box
... ... ...
return FALSE;
  }
```

Fig. 2 는 클라이언트 프로그램을 작동하였을 때 최초로 뜨는 대화상자 화면을 보여준다. 본 논문에서는 자세한 설명을 생략하였지만, 프로젝트의 이름을 관리하는 기능이 우선 있어, 모든 자료를 한 프로젝트에 관리하도록 하고 있음을 관찰할 수 있다.

여러 개의 단추는 각 단추마다 설계에 필요한 모듈화된 프로그램을 구동하도록 되어 있다. 예를 들어 KPDL 단추를 클릭하면, 프로젝트 디렉토리

에 준비되어 있는 입력자료를 서버에 보내고, 서버가 계산을 수행한 후에 보내준 결과를 프로젝트 디렉토리에 저장한다. 이 과정 중에 서버-클라이언트 사이의 통신이 인터넷을 통하여 이루어 지는 것이다.

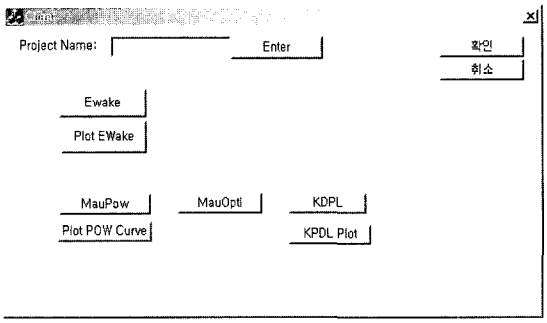

Fig. 2 Initial windows created by the client program

Windows 프로그램은 메시지 전달에 의해 구동된다. 따라서 대화상자를 만들면서 우선적으로 메시지 지도를 작성하여야 한다. Table 10 은 사용자가 마우스로 Fig. 2 의 KPDL 단추를 클릭한 경우에 연결이 될 함수를 메시지 지도에서 찾는 함수가 OnButtonKpdl()일 것을 보여준다. Fig. 2 는 대화상자가 필요로 하는 소켓의 객체를 전용변수로 포함하고 있음도 보여준다.

Table 10 Program segments showing CClientDlg class declaration

```
#include "SockObject.h"

class CClientDlg : public CDialog {
... ... ...
// Implementation
protected:
  HICON m_hIcon;
  // Generated message map functions
  //{{AFX_MSG(CClientDlg)
  virtual BOOL OnInitDialog();
  afx_msg void OnSysCommand(UINT nID,
LPARAM lParam);
  afx_msg void OnPaint();
... ... ...
  afx_msg void OnButtonKpdl(); // Function
called when Kpdl button pressed
  ... ... ...
  //}}AFX_MSG
  afx_msg LONG OnReceiveData(UINT
wParam, LPARAM lParam);
  DECLARE_MESSAGE_MAP()
private:
  CSockObject *m_pSock;
};
```

단추의 선택으로 필요한 함수를 호출하는 것은 인식표(ID)를 사용하여 이루어 진다. 이들 인식표로 Windows 메시지의 요구를 알 수 있도록 되어 있다.

Table 11 Typical ID tags used to identify the message tag

```
// Microsoft Developer Studio generated
include file used by Client.rc
... ... ...
#define                IDD_CLIENT_DIALOG
102
#define          IDP_SOCKETS_INIT_FAILED
103
#define                IDR_MAINFRAME
128
#define                IDC_BUTTON_KPDL
1001// KPDL button 을 부를 때 쓰는 ID
... ... ...
```

클라이언트가 대화형식으로 서버에 접속하고, 입력자료를 송부하며, 서버에서의 계산 결과를 받는 과정을 수행하는 대화 프로그램은 Table 12 의 CClientDlg 클래스에 정의되어 있다. Table 12 는 우선 CClientDlg 클래스를 초기화할 때, 소켓을 새로 만들고, 연결할 서버의 IP 주소, 포트번호 정보를 제공함을 보여준다. 예로 보여준 IP 주소는

자기 자신 컴퓨터를 서버로 사용하는 경우의 특별한 주소를 보여주고 있다.
Table 12 는 또한 메시지 지도에서 사용자가 마우스로 Fig. 2 의 KPDL 단추를 클릭한 경우에 IDC_BUTTON_KPDL 이라는 인식표를 통하여 연결될 함수가 OnButtonKpdl()임도 보여준다.
클라이언트가 서버로부터 계산 결과를 패킷으로 받으면 일단 버퍼에 저장하고 헤더로부터 데이터의 타입(sType)을 판단하는 것이 중요하다. OnReceiveData() 함수는 이 판단을 근거로 적절한 파일 이름으로 저장한다. 또한 Table 10 에 선언만 되어 있는, 서버에 작업을 의뢰하는 OnButtonKpdl() 함수의 실체를 구현하는 내용을 포함하고 있다.

Table 12 Part of CClientDlg class showing message map and implementation of member functions

```
  CClientDlg::CClientDlg(CWnd*          pParent
/*=NULL*/)  // CClientDlg dialog
: CDialog(CClientDlg::IDD, pParent) {
... ... ...
// Note that LoadIcon does not require a
subsequent DestroyIcon in Win32
m_hIcon          =          AfxGetApp()-
>LoadIcon(IDR_MAINFRAME);
  }

  BEGIN_MESSAGE_MAP(CClientDlg, CDialog)
//{{AFX_MSG_MAP(CClientDlg)
ON_WM_SYSCOMMAND()
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
ON_BN_CLICKED(IDC_BUTTON_KPDL,
OnButtonKpdl)
    ... ... ...
//}}AFX_MSG_MAP
ON_MESSAGE(WM_RECEIVED_DATA,
OnReceiveData)
  END_MESSAGE_MAP()
```

```
  ////////////////////////////////////////////////
////////////////////////
  // CClientDlg message handlers
  void CClientDlg::Initialize()
  {
m_pSock = new CSockObject(this);
m_pSock->Create();
m_pSock->Connect("127.0.0.1",   10004);   //
basic IP & port number for server
  }

  LONG        CClientDlg::OnReceiveData(UINT
wParam, LPARAM lParam) {
CPacket packet;
short sBodySize, sType;

char buf[1024];
m_pSock->Receive(buf,  1024);  //  서버로부터
데이터 수신, buffer 에 저장

packet.SetBuf(buf);
packet.GetHeader(&sBodySize, &sType);

if ( sType == 2 )  { // ANSWER_LOGIN
char data;
packet.GetChar(&data);
if (data == 0)
MessageBox("Success login");
else
MessageBox("login failure");
}

if ( sType == 7 )  { // ANSWER_OUTPUT_KPDL
short size;
char text[1024];

memset(text, 0, sizeof(text));

packet.GetShort(&size);
packet.GetText(text, size);
```

```
FILE *fp;  // 서버로부터의 데이터를 저장위한 file

fp = fopen("kpdl.out", "w");
fprintf(fp, "%s", text);
fclose(fp);

MessageBox(text);  // Dialogbox 에 display
}
return 0L;
  }

  void CClientDlg::OnButtonKpdl() {
// TODO: Add your control notification handler code here
FILE *fp;                    // 클라이언트가 준비한 데이터를 저장할 파일
short count = 0;
char input[1024] = {0,};

fp = fopen("pow.dat", "r");  // 서버로 보낼 입력 자료를 읽을 파일 open

while( !feof( fp ) )         // 파일 끝까지 읽고
input[count++] = fgetc(fp);  // 글자수까지 count
fclose(fp);

char data[1024] = {0,};
int size;

CPacket packet;              // 서버로 보낼 패킷 생성
packet.SetBuf(data);
packet.PutShort(count-1);    // size of input string except null character
packet.PutText(input, count-1);
size = packet.SetHeader(6);  // REQUEST_INPUT_KPDL 을 헤더에 기록

m_pSock->Send(data, size);   // 서버로 데이터 전송
  }
```

5. 서버 프로그램 스레드 구성

스레드(Thread)는 프로세스의 스케줄링 부담을 줄여 성능을 향상시키기 위한 프로세스의 다른 표현 방식이다. Fig. 3 은 서버가 여러 개의 스레드를 갖도록 설계되어 있음을 보여준다. 예를 들면, 클라이언트로부터의 새로운 연결을 접속 처리하여 초기화하는 일을 수행하는 AcceptTProc() 함수, AcceptTProc()이 처리하는 새로운 연결 처리를 제외한, 클라이언트로부터의 모든 통신 처리 작업을 수행하는 스레드로 접수된 패킷을 분석하여 GameTProc() 스레드에 넘기는 작업을 수행하는 InWorkerTcpTProc() 함수가 있다.

또한, GameTProc() 스레드는 자체의 버퍼를 사용하여, 주기적으로 연결된 패킷의 내용을 해석한 후 관련함수를 호출한다.

예를 들어, 패킷타입 정보가 REQUEST_LOGIN 인 경우는 OnRequestLogin()을 호출하고, 패킷타입이 REQUEST_INPUT_KPDL 인 경우는 OnRequestInputKpdl()을 호출한다.

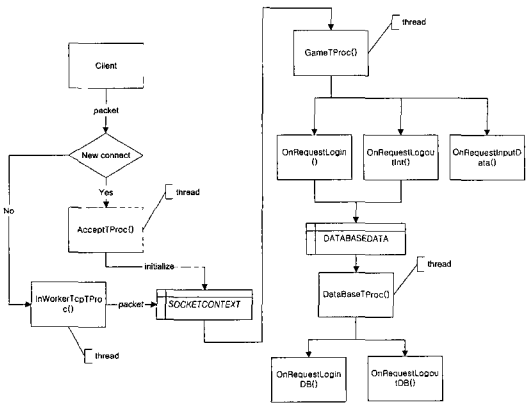

Fig. 3 Server program showing mulitple threads structure

6. 프로그램 운용 예

서버 프로그램은 도스 모드(DOS mode)에서 작동하는 프로그램으로 작성하였으며,클라이언트는 Windows 환경에서 작동되는 프로그램으로 작성하였다.

서버 컴퓨터에서 서버 프로그램(server.exe)을 작동시키면 즉시 Fig.4 와 같은 화면이 생성되며 클라이언트로부터의 데이터를 기다리고 있다는 메시지를 화면에 보인다. 그림으로부터 접속허가를 위한 스레드 AcceptTProc(), 작업을 접수하기 위한 스레드 InWorkerTcpTProc() 2 개, 통신 내용을 분석하기 위한 스레드 GameTProc() 2 개, 데이터베이스를 작동하기 위한 스레드 DatabaseTProc() 등 모두 6 개의 스레드가 작동하고 있음을 알 수 있다. 또한 서버가 제공하는 포트번호가 10004 임도 표시해 주고 있다.

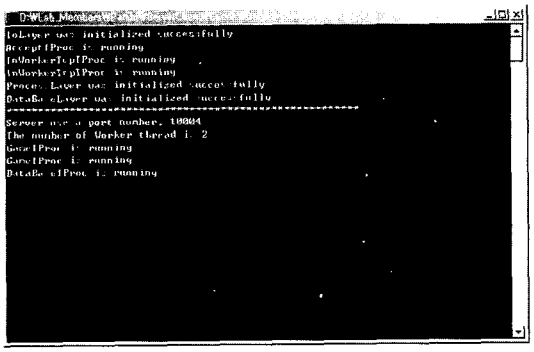

Fig. 4 Initial console message showing the server running

클라이언트 프로그램(Client.exe)을 작동시키기 전에 우선, Fig. 5 에서 보는 바와 같이 MauPow(MAU 계열을 이용한 프로펠러 초기 설계 프로그램) 프로그램의 입력데이터 파일 powinput.dat 를 준비하여 클라이언트 프로그램이 있는 디렉토리에 복사한다. Client.exe 를 구동시키면, Fig. 2 에서 보는 것과 같은 대화상자가 뜨며 프로젝트 이름을 입력하고, 우선 성공적으로 로그인이 수행되어 서버에 연결되었음을 확인한 후에, MauPow 단추를 클릭하면, 즉시, 서버에 계산을 요청하고 Fig. 6 에서 보는 바와 같은 계산 결과가 즉시 화면에 나타난다.

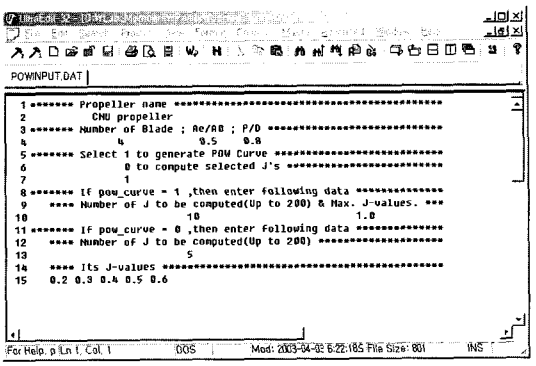

Fig. 5 View showing powinput.dat

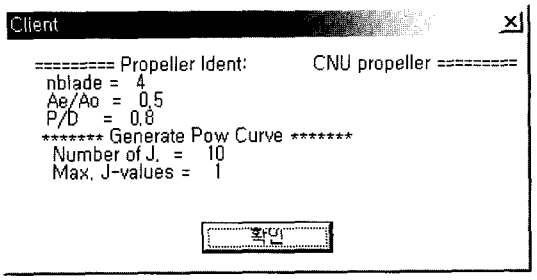

Fig. 6 Message window showing the result of computation

7. 결 언

기존의 단일 컴퓨터에서 작동하던 프로펠러 설계 패키지를 서버-클라이언트 환경에서 작동하도록 프로그램을 개조하는 방안을 제시하였다. 서버 프로그램은 도스 환경에서 작동을 하고, 클라이언트는 Windows 환경에서 작동하도록 프로그램을 구성하여 보여 주었다. 원격지에 떨어져 있는 설계자가 인터넷으로 서버와 연결하여 설계를 수행할 수 있는 기본적인 통신 방법과 서버-클라이언트 작업 환경 구성에 필요한 대화상자 등 제반 프로그램을 작성하는 기본적인 방법을 실제 작동하도록 코드화한 프로그램을 통하여 보였다. 인터넷의 급속한 보급을 최대한 이용하여, 향후 대규모 기능을 필요로 하는 복합적인 프로그램을 계산 기능

과 사용자 환경으로 분리하여 수행할 수 있는 기반을 완성하였다.

후 기

본 연구는 첨단조선공학연구센터(ASERC)의 산업체 과제의 일부로 수행되었습니다. 지원에 감사를 드립니다.

참 고 문 헌

- 이경호, 이동곤, 1999, " 선박 안전성 평가 및 구난지원 시스템," 대한조선학회 논문집 제 36 권 3 호, pp. 115-121
- 이상욱, 이규열, 1999, " 에이전트 기반의 시스템 통합을 위한 에이전트 기본 아키텍쳐에 관한 연구," 대한조선학회 논문집 제 36 권 3 호, pp. 93-106
- 정인숙, 이창섭, 임효관, 1996, " GUI 를 이용한 프로펠러 설계법," 대한조선학회, 춘계학술대회 논문집, pp. 220-223
- 정인숙, 이창섭, 조충호, 1997, " 데이터베이스와 응용프로그램 사이의 인터페이스 설계 및 구현," 대한조선학회 논문집, 제 34 권 제 4 호, pp. 139-147
- Jones, A. and Ohlund, J(김남식역), 2003, Network Programming for Microsoft Windows, 648p, 정보문화사, 서울
- Lee, C.-S. and Cho, C.-H., 2001, " Propeller design and analysis system using object-oriented database in Windows environment," Intl Symp on Practical Design of Ships and Other Floating Structures, Shanghai.
- Microsoft Visual C++, 1994, Programming with MFC and Win32, Vol 2, Six-Volume Documentation Collection for Microsoft Visual C++ Version 2.0 for Win32, Microsoft Press, Redmond, Washington.

< 이 왕 수 >

< 이 창 섭>