

Chapter Eighteen

CONCURRENT ANIMATION

Overview

This chapter describes concurrent animation, in which a stream of trace commands is passed directly from a running program to functions contained in a Proof library, without use of a trace file. Proof libraries are provided in the forms of Windows DLLs (Dynamic Link Libraries). Two Proof DLLs are available: `p4.dll` (the commercial library), and `sp4.dll` (the Student Proof library). Both DLLs are installed in `Program Files\Wolverine\P4`. The two libraries are functionally identical, except that the commercial library requires a security key (dongle), and the student library imposes the limits placed on Student Proof (See page 2-6).

Functions in a Proof library are called from your program. Throughout this chapter, we'll refer such programs as *driving applications*.

Who's in Control?

Concurrent animation requires carefully designed cooperation between a driving application and Proof. Often, both Proof and the driving application would "like" to be in control. Proof's implementation as a DirectDraw application imposes a number of constraints. First and foremost, Proof runs only in full-screen (exclusive) mode; it cannot share the screen with another application. Microsoft's DirectDraw is extremely sensitive to actions taken by software other than the currently running DirectDraw application. With DirectDraw, it's all or nothing-at-all. For example, if an application other than the current full-screen DirectDraw application tries to write information to the screen, DirectDraw will summarily suspend the DirectDraw application.


If your application needs to share the screen with Proof, you'll have to carefully suspend and resume Proof execution before and after periods when you need to control the screen.

In order to deal with the control issues described above, the library versions of Run Mode and Debug Mode differ slightly from their post-processed counterparts.

Alterations to Run and Debug Modes for Concurrent Animation

When you run a library version of Proof, Proof's Run Mode and Debug toolbars are expanded to include "yield sign." The extended Run Mode toolbar is shown below:



Clicking on the  yield sign causes Proof to yield control back to the driving application. Details of how Proof communicates this information back to your application are given later on.

The interpretations of the standard Windows buttons in the upper right hand corner of the Proof window are altered from their normal Windows usage. The "minimize" button not only minimizes the Proof window, but also yields control back to the driving application. The "maximize" and "kill" buttons are disabled (grayed out). To quit a concurrent Proof application, you must yield to the driving application and quit under its control. (You can't jerk the rug out from under a driving application.)

Driving Applications

Functions contained in Windows DLLs can be called by a wide variety of software; however, in most cases, Proof DLLs are called from C/C++ programs. This chapter explains only C/C++ use.

Declarations supporting C/C++ use are contained in the file `proofdll.h`, whose contents are partially reproduced in Figure 18-1.

```

//*****
//      proofdll.h - include file for C, C++ use of Proof DLL
//*****

#ifndef PROOF_DLL_INCLUDED
#define PROOF_DLL_INCLUDED    ONCE_ONLY

//*****
//      DLL request Return Codes
//*****

#define DLLRC_OK                1          // normal return
#define DLLRC_FAILURE           2          // error return
#define DLLRC_TERMINATED        3          // currently unused (you cannot
                                           // terminate the DLL)
#define DLLRC_YIELDED           4          // the user clicked the yield sign

//*****
//      DLL Status Flags (Returned by ProofStatus)
//*****

#define PSF_IDLE                0x01       // 1 => Proof is idle
#define PSF_INPUT_NEEDED        0x02       // 1 => Proof is waiting for a trace line
#define PSF_INPUT_AVAILABLE    0x04       // 1 => An unprocessed trace line is
                                           // ready
#define PSF_END                 0x08       // 1 => End command has been processed
#define PSF_RUNNING             0x10       // 1 => Proof is executing trace commands
#define PSF_SUSPENDED           0x20       // 1 => Proof is suspended
#define PSF_MINIMIZED           0x40       // 1 => Proof Window is minimized
#define PSF_LAYOUT_CHANGED      0x80       // 1 => Proof layout has been changed

```

Figure 18-1. Proofdll.h

Functions Contained in the Proof Libraries

Table 18-1 summarizes the functions contained in the Proof libraries:

Function	Argument	Purpose	Returned Value
ProofDLL	int on_off	A non-zero argument causes library initialization. A zero argument shuts down Proof.	Non-zero => success. Zero => failure.
ProofOpenLayout	char *filename	Opens the specified layout.	DLLRC_OK or DLLRC_FAILURE
ProofSendTraceLine	char *command	Transmits a trace stream command to Proof.	DLLRC_OK DLLRC_FAILURE DLLRC_YIELDED
ProofSuspend	void	Suspends Proof.	void
ProofResume	void	Resumes Proof.	void
ProofStatus	void	Returns the status of Proof.	PSF_... flags shown in Figure 18-1

Table 18-1. Proof Library Functions

To call the functions shown above, a driving application must first call the Windows LoadLibrary API function to retrieve pointers to functions in the Proof DLL.

Typical Operation of a Driving Application

A driving application usually operates as follows:

- It calls ProofDLL with a non-zero argument to initialize the Proof library.
- It calls ProofOpenLayout, specifying the name of the layout file to be used.
- It repeatedly calls ProofSendTraceLine, passing trace stream commands to Proof.
- It may optionally call ProofSuspend and ProofResume, to suspend and resume Proof execution.
- It may optionally call ProofStatus to ascertain the current status of Proof.

ProofSendTraceLine Details

The dominant activity in a concurrent animation typically very large number of calls of ProofSendTraceLine. This function is not only the most frequently called; it's also the most complex. Other functions are really quite simple.

When the Proof library is initialized, Proof boosts its priority above that of the driving application; however, each time Proof has completed its internal updating of the screen, it yields time that would otherwise go unused to the driving application. To compensate for variation in the amount of time the driving application requires to generate trace stream commands, Proof maintains a buffer of up to a dozen commands that have been transmitted to it. During periods of light load in the driving application, commands are quickly generated, and the buffer fills up. During periods of heavy load in the driving application, Proof can remove queued commands from the buffer. This lessens the possibility of Proof's being unable to proceed due to a lack of input (so-called input starvation).

When a driving application calls ProofSendTraceLine, one of three possible values is returned. A return value of DLLRC_OK indicates that the transmitted command line has been accepted into Proof's command buffer. A return value of DLLRC_FAILURE indicates that a serious error has occurred, and the likelihood of being able to continue is low. A return value of DLLRC_YIELDED indicates that the person viewing the animation has clicked on the **Yield** button or standard Windows "Minimize" button.

In the event ProofSendTraceLine returns a value of DLLRC_YIELDED, the command the driving application attempted to send has been rejected. Under such circumstances, it is the driving application's responsibility to reissue the rejected command, should it decide to resume the animation.

Sample application

A sample C++ driving application is shown in Figure 18-2. The application illustrates all of the Proof library functions.

The driving application periodically creates Proof Objects and places them on a circular Path, so that as time progresses, more and more Objects circulate around the Path.

Every 10th Object, the application suspends Proof, displays a dialog on the screen, and if the person viewing the application clicks **OK**, resumes the animation.

At the conclusion of execution, the application sends an **end** command to Proof. The application then repeatedly calls ProofStatus, and when Proof indicates that it has processed the end command, the application shuts down the Proof library.

Finally, to illustrate Proof's restartability, the application restarts Proof and immediately shuts it down.

```
//-----  
//      dlltest.cpp - Example of ProofDLL Usage  
//-----  
  
#define WIN32_LEAN_AND_MEAN      // Faster compilation  
  
#ifdef __WATCOMC__  
    #define CDECL      __cdecl  
    #define EXPORT      __declspec(dllexport)  
#else  
    #define CDECL  
    #define EXPORT      _declspec(dllexport)  
#endif  
  
#define FOREVER while(1)  
  
#include <windows.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <process.h>
```

```

#include "proofdll.h"    // Wolverine-provided header file

//-----
//      Local Function Prototypes
//-----

int      GetEntryPoint      (char *name);
int      SendCommand        (char *command);

//-----
//      Local Statics
//-----

int      (CDECL *ProofDLL)      (int on_off);
int      (CDECL *ProofOpenLayout) (char *filename);
int      (CDECL *ProofSendTraceLine) (char *image);
void     (CDECL *ProofSuspend)    (void);
void     (CDECL *ProofResume)     (void);
int      (CDECL *ProofStatus)     (void);

static HINSTANCE      hProofLib;    // Proof DLL instance

//-----
//      main()
//-----

main()
{
    int      iterations, rc;

    char      command[256];    // command buffer

//---> if ((hProofLib = LoadLibrary("p4.dll")) == NULL)    // Commercial DLL
if ((hProofLib = LoadLibrary("sp4.dll")) == NULL)    // Student DLL
    {
        printf("DLL load failed!\n");
        exit(0);
    }

    printf("ProofDLL loaded\n");

    *((int *) &ProofDLL)      = GetEntryPoint("ProofDLL_");
    *((int *) &ProofOpenLayout) = GetEntryPoint("ProofOpenLayout_");
    *((int *) &ProofResume)     = GetEntryPoint("ProofResume_");
    *((int *) &ProofSendTraceLine) = GetEntryPoint(
        "ProofSendTraceLine_");
    *((int *) &ProofSuspend)    = GetEntryPoint("ProofSuspend_");
    *((int *) &ProofStatus)     = GetEntryPoint("ProofStatus_");

```

```
rc = (*ProofDLL) (999);          // argument currently unused

if (rc == 0)
{
    printf("Unable to initiate Proof");
    exit(0);
}

printf("Opening blttestx.lay\n");

rc = (*ProofOpenLayout) ("blttestx.lay");

if (rc == DLLRC_OK)
    printf("blttestx.lay loaded OK\n");
else
{
    printf("Unable to load blttestx.lay\n");
    exit(0);
}

for (iterations = 1; iterations <= 40; iterations++)
{
    if (iterations % 10 == 0)
    {
        (*ProofSuspend) ();          // from our side

        printf("We have issued a ProofSuspend() call\n");

        rc = MessageBox(
            GetDesktopWindow(),
            "Suspend/Resume Test",
            "Press OK to resume executing.",
            MB_OKCANCEL | MB_SETFOREGROUND);

        if (rc != IDOK)
            exit(0);

        printf("We'll resume in one second\n");
        Sleep(1000);

        (*ProofResume) ();
    }

    sprintf(command, "Create xact %i", iterations);

    if (SendCommand(command) < 0)
        break;

    sprintf(command, "Place %i on loop", iterations);
```



```
        if (SendCommand(command) < 0)
            break;

        sprintf(command, "Write count %i", iterations);

        if (SendCommand(command) < 0)
            break;

        sprintf(command, "Time %i", iterations * 3);

        if (SendCommand(command) < 0)
            break;
    }

    SendCommand("End");

    printf("Last trace command sent\n");

    FOREVER
    {
        rc = (*ProofStatus) ();           // current status

        if (rc & PSF_END)                 // end command processed
            break;

        printf("State = %2X\n", rc);

        Sleep(1000);                     // wait one second
    }

    Sleep(1000);

    printf("Shutting down the Proof DLL\n");

    (*ProofDLL) (FALSE);

    Sleep(1000);
    printf("Restarting the Proof DLL\n");
    Sleep(1000);

    (*ProofDLL) (999);

    Sleep(1000);
    printf("Shutting down the Proof DLL again\n");
    Sleep(1000);

    (*ProofDLL) (FALSE);
```

```
        printf("Trace streaming test complete!\n");

        sleep(2000);
        exit(0);

    }          // end of main()

//-----
//      GetEntryPoint
//-----

int GetEntryPoint(char *name)
{
    void      *address;

    address = GetProcAddress(hProofLib, name);

    if (address == NULL)
        fprintf(stderr,

"Unable to find \"%s\" in the Proof Library (proofdll.dll)\n", name);

    return (int) address;
}

//-----
//      SendCommand
//-----

int SendCommand(char *command)
{
    int      rc,          // DLL return code
            status1,      // test ProofStatus()
            status2;

    FOREVER
    {
        status1 = (*ProofStatus) ();          // current status

        rc = (*ProofSendTraceLine) (command); // trace stream
command

        switch (rc)
        {
case DLLRC_YIELDED:    status2 = (*ProofStatus) ();    // current status

                        printf("Proof has yielded back to us (%2X/%2X)\n",
status1, status2);
```

```

        Sleep(1000);

        rc = MessageBox(
            GetDesktopWindow(),
            "Yield/Continue Test",
            "Press OK to resume executing.",
            MB_OKCANCEL | MB_SETFOREGROUND);

        if (rc != IDOK)
            exit(0);

        printf("We'll continue in one second\n");
        Sleep(1000);

        (*ProofResume) ();          // re-enable Proof
        continue;                  // retry

case DLLRC_OK:                    // "OK" return
        return 1;

default:
        printf("Trace stream error\n");

        return -1;                // error return
    }
} // end of SendCommand()

```

Figure 18-2. A Sample Driving Application

