GDI 훅킹에 의한 GUI 공유 기법에 관한 연구

김형준*, 박기식*, 조인준**, 정회경**

A Study of GUI-Sharing Mechanism by GDI Hooking

Hyoung-Jun Kim*, Ki-Shik Park*, In-June Jo**, Hoe-Kyung Jung**

Abstract

This paper shows the GUI-sharing mechanism based on GDI hooking that can be useful for group users to share application programs dedicated to specific systems in distributed environments. In order to implement a networked-group working for real-time bases, an application-sharing mechanism should be necessary. This paper presents the implementation issues of application-sharing mechanism and provides a way of group working by implementation of GDI hooking module using GDI library in Windows 95 environment.

요 약

본 논문은 지역적으로 분산된 환경하에서 공동작업 및 데이터 회의 등이 가능하도록 하기 위하여 특정 시스템에 있는 개인 사용자 전용의 어플리케이션 프로그램을 다수의 사용자들이 실시간으로 공유할 수 있도록 하는 어플리케이션 공유 방법을 제안한다. 이를 위해 GUI(Graphical User Interface) 공유 방법에 관한 연구 현황을 살펴보고 GDI(Graphic Device Interface) 훅킹(hooking) 기법에 의한 GUI(Graphical User Interface) 공유 방법을 제시하고 윈도우즈95 시스템 환경에서 GDI 라이브러리를 사용한 GDI 훅킹 모듈을 설계, 구현한다.

I. 서 론

급속한 컴퓨터 기술의 발달에 따른 컴퓨터 환경의 변화로 개방 분산 환경 하에서 여러 사용자가 동시에 작업을 수행하는 공동작업이 CSCW (Computer Supported Cooperative Work)와 그룹웨어 등을 통하여 이루어지고 있다[1]. 이러한 공동작업이 분산된 위치에 관계없이 이루어지기 위해서는

* 한국전자통신연구원
** 배재대학교 컴퓨터공학과 교수
접수일자 : 1998년 10월 7일

각 개인의 어플리케이션을 여러 사용자가 실시간으로 공유할 수 있도록 하는 소위 어플리케이션 공유 기법이 요구된다[2,3,4,5,6]. 이러한 어플리케이션 공유 기법을 이용하면 여러 통신 상대방간의 프로그램 공동 개발 및 데이터나 어플리케이션들을 실시간으로 공유할 수 있게 함으로써 공동작업의 효율성을 높일 수 있는 장점을 갖는다.

어플리케이션 공유를 위한 기법으로는 어플리케이션을 복제하여 이벤트를 공유하는 이벤트 공유(event-sharing)와 GUI를 공유하는 GUI 공유가 있다[7,8,9]. 그러나 이벤트 공유 방법은 어플리케이션들이 동일 환경 속에서 각 시스템에 복제되어 설치되어야 하는 복제 구조를 가지며, 키 입력이나 마우스 클릭과 같은 이벤트 입력 시 공유하는 모든 어플리케이션에 공유되어야 한다. 그러므로 네트워크상의 트래픽 부하는 줄이는 대신 공유하고자 하는 여러 지점 사이의 입력 이벤트가 받아들여지지 않아 동기화 문제가 발생하며, 공동작업이나 그룹 회의에 나중에 참여한 참여자는 공유의 문제를 유발하게 된다.

반면 GUI 공유는 공유되는 어플리케이션에서 수행되는 GUI의 모든 내용들을 각 지점의 시스템에 보내 어플리케이션을 공유하도록 하므로 공동작업을 하는 모든 사용자들은 동일 화면 이미지를 유지할 수 있다. 그러나 GUI 공유는 시스템마다 통일된 사용자 인터페이스를 사용하지 않기 때문에 많은 추가적인 계산이 요구되며, 각 지점에서 동일 화면 이미지를 유지하기 위해서는 화면 이미지에 해당되는 데이터 전송에 의한 트래픽 문제를 야기시키는 단점을 가진다. 이러한 단점을 보완하기 위해 변환된 부분의 이미지만을 처리하는 연구가 많이 진행되고 있다.

본 논문에서는 위의 문제점을 해결하기 위해 분산된 제어 구조하에서 어플리케이션 공유가 가능하도록 GDI 훅킹에 의한 GUI 공유 기법을 제안한다. 이의 설계는 윈도우'95 환경에서 윈도우 시스템 내 GDI 라이브러리를 이용하여 GUI 훅킹 기법을 설계하였으며, 이 GDI 훅킹에 의한 공유 기법을 이용하여 어플리케이션을 공유함으로써 이벤트와 GUI 이미지 공유에서 야기되는 트래픽 부하와 속도 문제를 해결할 수 있다.

이하 본 논문의 구성은 다음과 같다. II장은 어플리케이션 공유 및 관련 연구에 대해 설명하며, III장은 본 논문에서 제안한 GDI 훅킹 모듈에 대해 기술한다. IV장과 V장에서는 GDI 훅킹 모듈의 시뮬레이션 결과와 결론을 기술한다.

Ⅱ. 관련 연구

본 장에서는 본 연구에 관련되는 어플리케이션 공유[6]에 대한 개요 및 윈도우 '95환경 이해에 관련된 개념을 기술한다.

1. 어플리케이션 공유 방법

서론에서 언급한 바와 같이, 어플리케이션 공유 방법은 크게 이벤트 공유와 GUI 공유 방법으로 나뉘어진다. 특히 GUI 공유는 공유되는 어플리케이션에서 수행되는 GUI의 모든 내용들을 각 지점 시스템에 보내 어플리케이션을 공유토록 하므로 공동작업을 하는 모든 사용자들은 동일 화면 이미지를 유지한다. 그러나 GUI 공유는 시스템마다 통일된 사용자 인터페이스를 사용하지 않기 때문에 많은 추가적인 계산이 필요하며, 각 지점에서 동일 화면 이미지를 유지하기 위해 화면 이미지에 해당되는 데이터 전송에 의한 트래픽 문제점을 야기시킨다는 문제를 수반하고 있다.

이에 본 논문에서는 동기화 문제를 해결하고 GUI 공유에서 발생하는 트래픽 문제를 해결하기 위해 GDI 훅킹에 의한 GUI 공유 기법을 제안한다. 이를 위해 어플리케이션 공유의 대상이 되는 프로그램에서 호출하는 GDI32.DLL내의 함수를 기록하여 이 정보로 재 디스플레이 할 수 있는 방법을 개발함으로써 이를 수행할 수 있다. 이 모듈은 지정된 윈도우를 소유한 프로세스내의 모든 모듈(Kernel32.dll, User32.dll, Comctl32.dll) 내에서 디스플레이 내용을 변경시킬 수 있으므로 이 모든 모듈내의 API 함수를 가로챌 수 있어야 한다. 가로채는 함수내에서는 해당 함수가 실행된 뒤의 리턴값을 조사하여 이 함수의 성공 여부에 따라 함수를 호출할 당시의 매개변수를 함께 저장하여 이후 재 디스플레이 시 사용 할 수 있다.

2. 윈도우 95에서 비디오 장치와 어플리케이션 간의 모듈 구성

윈도우 환경은 장치 비의존적인 프로그램 개발 환경을 제공하고 이식성을 용이하기 하기 위해 어플리케이션과 하드웨어 사이에 여러 층이 존재하게 되는데 비디오 장치에의 접근은 위의 모듈들을 통하여 이루어 진다. GDI 함수 훅크 모듈은 그래픽 이미지 변경의 요청을 가로채기 위해 설치될 수 있는 계층으로 그림 1의 A, B등에 설치될 수 있다. B층에의 설치는 많은 수행중인 어플리케이션 중 어떤 어플리케이션이 그래픽 명령을 요구했는 지를 알 수가 없으며, GDI 함수 훅크 모듈이 로컬 시스템상에서 동작중인 모든 프로세스의 이미지 변경을 추적하는 것이 아니므로 적절하지 않다. 그래서 GDI 함수 훅크 모듈이 A 층에 위치하는 것이 어플리케이션의 GDI 함수 호출을 가로채서 GUI를 공유하는 것이 가능하게 된다. GDI 모듈 이외의 모듈인 Direct3D, DirectDraw 모듈에서의 함수 훅크는 GDI 함수 훅크 모듈을 확장함으로 가능해 진다

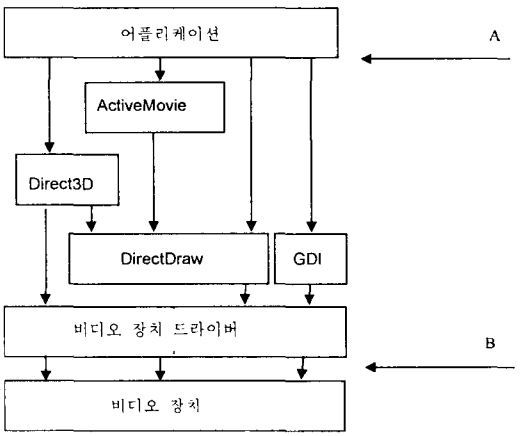


그림 1. Win32그래픽 기술들간의 관계

2.1 디스플레이 관련 GDI 함수

GDI 모듈은 어플리케이션 프로그램에서 GDI 모듈에 보내지는 많은 그래픽 명령을 가지며 API 함수로 구성된 이 명령들은 280개 이상이다. GDI 훅크 모듈에서 중심적으로 다루어야 하는 함수는 실제로 화면상의 윈도우 이미지를 변경시키는 함수와 그 함수들에서 사용되는 환경 값을 관리하는 함수들이다.

디스플레이상의 그리픽 이미지를 변경하는 함수들은 다음과 같은 것들이 있다. [AngleArc, Arc, ArcTo, BitBlt, Chord, Ellipse, ExtFloodFill, ExtTextOutA, ExtTextOutW, FillPath, FillRgn, FloodFill, InvertRgn, LineDDA, LineTo, MaskBlt, MoveToEx, PaintRgn, PatBlt, Pie, PlgBlt, PolyBezier, PolyBezierTo, PolyDraw, PolyPolygon, PolyPolyline, PolyTextOutA, PolyTextOutW, Polygon, Polyline, PolylineTo, Rectangle, RoundRect, SetPixel, SetPixelV, TextOutA, TextOutW]

어플리케이션에서 위의 함수들을 호출하는 경우에는 그래픽 이미지의 변경이 있으므로 이 함수들을 호출할 당시의 매개변수 값들을 함께 저장하여 나중에 디스플레이 할 수 있어야 한다. 위의 함수 외에도 화면 출력을 위한 여러 속성값을 지정하거나 구하는 함수들과 디바이스 컨텍스트(Device Context)를 만들고 삭제하는 함수들로 구성된다.

2.2 Win32 환경의 이해

Win32에서는 Win3.1과는 달리 각각의 프로세스가 독립된 주소 공간에 위치하게 된다. 즉 각각의 프로세스내의 동일한 주소 값은 서로간에 전혀 무관하게 된다. 그림 2에서 처럼 각각의 프로세스는 독립된 공간에 위치하고 여러 프로세스에 의해 공유되는 모듈은 '공유'로 지정된 영역에 설치된다. 모듈이 '공유' 공간에 위치 되기 위해서 DLL로 구성되어야 하며 DLL로 구성된 GDI 함수 훅크 모듈을 지정된 프로세스 공간에 넣기 위한 방법이 필요하다. 각각의 모듈은 그림 2의 아래 그림에서 보는 것과 같은 주소 공간에 위치하게 된다. 이 경우 설치 프로그램과 타켓(Target) 어플리케이션은 서로 다른 주소 공간에 위치하게 된다.

GDI 함수 훅크 모듈은 자신과 자신을 설치한 프로세스와 메시지의 전송은 SendMessage(), PostMessage를 사용하여 통신할 수 있다. 이와 달리 데이터의 전송이 필요한 경우는 서로 다른 주소 공간에 위치하는 프로세스 사이의 통신을 위해 사용되는 WM_COPYDATA 메시지를 사용한다.

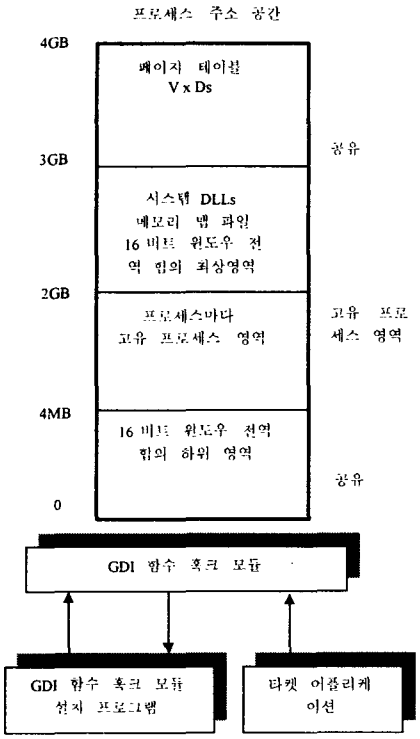


그림 2. 윈도우95 프로세스 주소 공간과 모듈 간의 관계

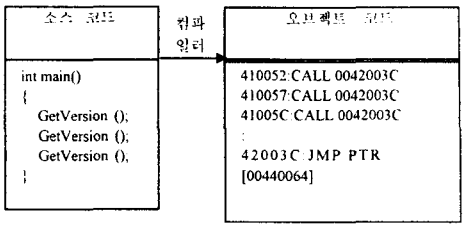


그림 3. 소스 코드와 오브젝트 코드와의 관계

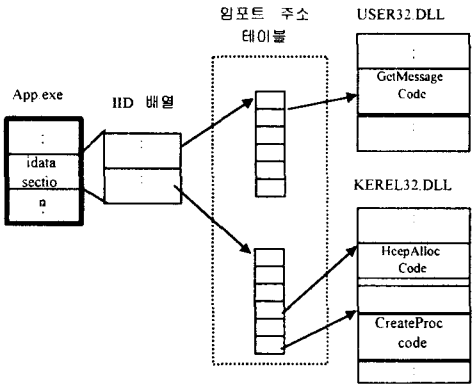


그림 4. 임포트 테이블을 통한 DLL 내 함수 코드와의 연결

Ⅲ. GDI 훅킹 모듈 설계

1. GDI 모듈 설치 근거

GDI 모듈 설치가 가능한 지를 살펴보기 위해 먼저 기존에 존재하는 DLL 내의 함수 호출이 어떻게 이루어지는지를 살펴본다. 그림 3의 왼쪽에 있는 프로그램은 KERNEL32.DLL 내의 함수 GetVersion()을 사용하여 프로그램을 작성하여 컴파일 하였다. 이 경우 컴파일러가 만들어 낸 코드는 오른쪽과 같은 형태로 이루어진다.

즉, 원시 코드내의 GetVersion() 함수 호출이 KERNEL32.DLL 내의 실제 GetVersion() 함수 코드를 직접 부르지 않고 EXE 코드내의 JMP로 진행된다는 점이다. 이것은 프로그램내에서 호출하는 DLL 내의 함수가 임포트(import) 테이블(table)을 참조하여 수행되므로 이 임포트 테이블의 내용을 바꾸게 되면 기존의 존재하는 함수의 수행을 가로챌 수 있음을 의미한다. 이점에 착안하여 함수를 가로채서 이를 공유하고자 하는 지점으로 전송하여 처리함으로써 분산 환경하에서 어플리케이션 공유가 가능하게 된다.

그림 4는 어플리케이션 App.exe에서 DLL인 USER32.DLL의 GetMessage()와 KERNEL32.DLL 내의 HeepAlloc(), CreateProcess() 함수를 사용한 경우 임포트 표를 통한 DLL 내의 함수 코드와의 연결을 보여준다. App.exe에서 USER32.DLL과 KERNEL32.DLL 내의 함수를 호출하게 되면 임포트 주소 테이블을 참조하여 해당되는 DLL내의 함수를 호출하게 된다. 이때 훅크 모듈은 그림에서 점선의 표시된 부분인 임포트 주소 테이블 내의 주소를 훅크 모듈내의 함수로 변경시켜 GDI 함수를 가로 챌 수 있음을 보인다.

분산환경하에서 여러 사용자가 각 사이트의 응용 프로그램을 공유하면서 공동작업을 수행하는 전체

시스템 구성도를 그림 5에 보인다. 시스템은 윈도우 95 환경하에서 GDI 훅크 모듈을 이용하여 어플리케이션을 공유하며, 네트워크 시스템을 이용하여 공동 개발자들이 공동작업을 수행한다.

2. 어플리케이션 공유 모듈 설계

2.1 GDI API 훅크 모듈 구성

GDI API 훅크 모듈의 구성은 다음과 같다.

- 모듈 삽입기: GDI API 훅크 모듈을 지정된 프로세스의 주소 공간에 삽입하는 기능을 담당한다.
- DllEntryPoint - DllMain(): GDI API 훅크 모듈의 시작점 (일반 프로그램에서 main() 역할)으로 이 모듈이 지정된 프로세스에 삽입된 후 그 프로세스내의 API 함수를 훅킹하는 기능을 가진다.
- API 대체기: 훅크의 대상이 되는 기존의 GDI32.DLL내의 함수들의 시작점을 GDI API 훅크 모듈내의 함수로 변경하는 작업을 수행한다.
- 타켓 API 목록: 훅크의 대상이 되는 함수 목록으로 함수의 인수에 대한 정보를 포함한다.
- 공용 함수: GDI32.DLL내의 모든 훅킹된 함수가 호출되기 전에 수행되는 함수이다. 이 함수 내부에서 실제 API 훅크 목록을 처리한다.
- 매개변수 복호기: 훅크된 각 API 함수들이 받아들이는 매개변수들에 대한 처리를 수행한다.

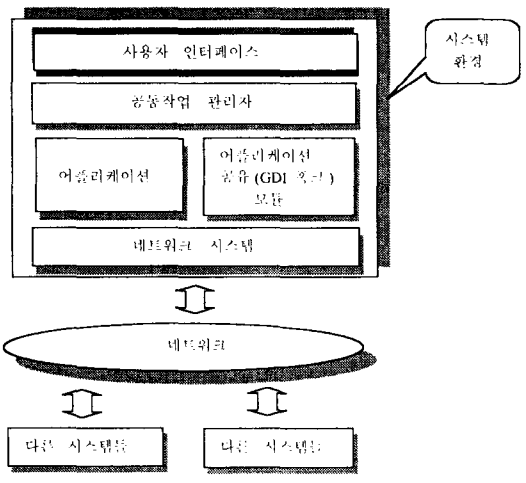


그림 5. 시스템 전체 구성도

- HDC 처리기: 윈도우 그래픽 출력의 기본 정보를 이루는 DC (Device Context)를 일치시키기 위한 처리를 수행한다.
- Log 처리기: GDI32.DLL로 호출된 API 함수의 정보를 파일이나 네트워크상에서 전송하기 위한 처리를 수행한다.
- 매개 변수 구성기: 클라이언트에서 GDI32.DLL 함수를 호출하기 위한 매개변수를 구성하는 기능을 수행한다.

2.2 GDI 훅킹 모듈 처리 흐름

본 절에서는 위에서 설명한 GDI 훅크 모듈의 처리 흐름을 설명한다.

- 훅크 모듈 삽입: Win32 환경하에서 수행되는 각 어플리케이션은 공유한 주소 공간을 가진다. 즉, 한 어플리케이션에서 메모리 주소 0x10000000의 내용은 다른 어플리케이션에서의 메모리 주소 0x10000000의 내용과는 갖지 않을 수도 있다. 즉, 각각의 프로세스가 고유한 영역하에서 수행되도록 운영체제가 관장하고 있다. 이 경우 훅크 모듈은 감시의 대상이 되는 프로세스의 주소 공간 내에 위치해야 한다. 이것을 위해 Win32 API중 CreateProcess() 함수를 이용하여 DEBUG_ONLY_THIS_PROCESS 모드로 새로운 프로세스를 구동시킨다. 새로운 프로세스 이미지가 메모리에 적재된 후 DEBUG 이벤트를 받게 되고, 이때 훅크 모듈을 삽입할 수 있는 처리를 한다. 이 작업은 새로이 생성된 프로세스의 실행 코드를 LoadLibrary() API를 호출하도록 변경하여 새로운 프로세스가 API 훅크 모듈을 적재하도록 한다. 이는 어플리케이션 공유 관리기에서 모듈 삽입기 루틴이 담당하게 된다.
- 훅크 모듈 삽입 이후 모듈 처리: 대상 프로세스내에 훅크 모듈이 삽입되면 훅크 모듈내의 DllMain() 함수가 수행된다. 이 함수는 C언어에서 main() 함수 또는 윈도우 프로그램에서 WinMain()과 같은 역할을 수행하는 DLL 모듈내의 시작점이 된다. 이 함수안에서 임포트 테이블의 변경에 의해 대상 프로그램이 호출하는 API 함수의 호출이 공용 함수로 변경된다. 이후 호출되는 API 함수들은 그 제어가 공용 함

수로 넘어오게 된다. 공용 함수로 넘어온 API 호출은 매개변수 복호기(decoder)에서 호출된 API 함수의 매개변수에 관한 정보를 구성한다. 즉 각각의 매개변수가 함수에서 사용될 실제 데이터일 수도 있지만 데이터의 포인터(Pointer)일 수도 있다. 만약 포인터인 경우는 해당 포인터의 내용을 추출하는 작업을 병행하게 된다.

- HDC 처리기 수행: GDI는 그래픽 출력과 그래픽 이미지 저장을 수행하는 모듈이다. 이중 그래픽 출력에 있어서 Device Context 라는 데이터 구조를 사용하여 그래픽 출력에 사용되는 DC정보를 관리하기 위한 작업을 수행한다. DC는 25가지 그래픽에 관련되는 속성을 가지고 있다. 즉, 호의 방향, 배경색, 배경 모드, 경계 사각형, 붓, 붓의 원점, 자르기 구역, 색상 조정, 현재 위치, 드로잉 모드, 글꼴, 그래픽 모드, 글꼴, 그래픽 모드, 매핑 모드, 꼭지점 한계, 팔레트, 펜, 다각형 채움 모드, StretchBlt 모드, 텍스트 위치 조정, 텍스트 색상, 텍스트 여분 공간, 텍스트 정렬, 뷰포트 범위, 뷰포트 원점, 창의 범위, 창의 원점 등이다. 이미지 재생시 호출하는 함수들은 DC의 핸들(handle)을 "LineTo (HDC hdc, int x, int y)" 와 같이 요구한다. 따라서 이 정보도 함께 저장되어 이 DC 자료를 관리하며 저장하는 작업을 수행한다.
- Log 처리기 수행: Log 처리기는 위에서 추출된 정보를 저장하는 역할을 수행한다. 이 저장된 데이터는 네트워크상에서 지역적으로 분산된 공동작업을 수행하는 참여자가 어플리케이션을 공유 시에 사용한다. GDI 함수에 대한 호출을 기록하기 위한 자료구조는 다음과 같은 형태를 가진다.

```
Process ID : WORD ProcessID
Window Handle : HWND hWnd
Function Name ID : WORD FuncID
Parameters WORD ParSize, char * ParValue
Return Value : void
```

ProcessID는 로컬 시스템내의 어떤 프로세스에서 GDI 함수 호출이 발생했는지를 나타낸다. 이것은 한 시스템 내에서 여러 어플리케이션을 작업 대상으로 지정할 수 있기 때문에 각각의 어플리케이션을 구별할 수 있는 핸들이 필요하기 때문이다.

hWnd는 하나의 프로세스내에서 생성되는 윈도우를 구별하기 위한 것이다. FuncID는 그래픽 이미지 변경에 사용되는 함수가 어떤 것인지를 나타내는 ID이다. 이는 LineTo, Circle 등을 구별하기 위한 ID이다. 비트맵(Bitmap) 데이터 또는 DC 데이터 일 경우에도 ID를 할당해 놓는다. ParValue는 함수 호출에 사용된 매개변수 값들을 모아 놓은 영역을 가리키는 'char' 형 포인터이다. 이는 Log 자료의 저장 시에 함께 저장되는 값이다. 이 구조는 각각의 함수 ID에 따라 서로 다른 형태를 가진다. ParSize는 ParValue의 크기를 나타낸다.

훅크 모듈 설치 이전의 디스플레이 데이터의 저장은 해당 프로세스의 윈도우 비트맵을 구하여 저장하게 된다. 이 경우 위의 자료구조에 저장될 때 특정한 FuncID를 지정하며 ParValue에는 구해진 비트맵 자료를 ParSize에는 이전 자료의 크기를 저장한다. ParValue에 저장되는 비트맵 자료는 비트맵 헤더(Header) 등을 포함한 자료이다. 이 자료가 네트워크 등을 통해 다른 컴퓨터에 전달되는 경우는 HostName을 나타낼 수 있는 항목이 추가된다.

- 재 디스플레이: 클라이언트측인 API Play 윈도우는 매개변수 정보 및 HDC 정보를 이용하여 API 함수 호출에 사용될 매개변수들을 재구성한다. 이렇게 재구성된 정보를 이용하여 GDI API 호출로 그래픽 이미지를 재생한다.
- GDI 훅크 모듈 종료: GDI 훅크 설치 어플리케이션의 종료 요청에 의해 훅크 함수 제거 루틴은 각 모듈의 임포트 표를 저장된 원래의 내용으로 변경시킨 후 훅크 모듈을 그 프로세스의 주소 공간에게 제거하게 된다.

Ⅳ. 구 현

본 논문에서는 분산 환경하에서 공동작업을 하기 위해 요구되는 어플리케이션 공유 방법 중 GDI 훅킹에 의한 공유 기법을 시뮬레이션을 통해 제안하였다. 이에 대한 시스템 개발은 펜티엄-II(300MHz) 컴퓨터의 Windows 95 운영체제하에서 Microsoft Visual C++ 5.0(Developer Studio 97 Enterprise Edition)을 사용하여 구현하였다. 구현 프로그램은

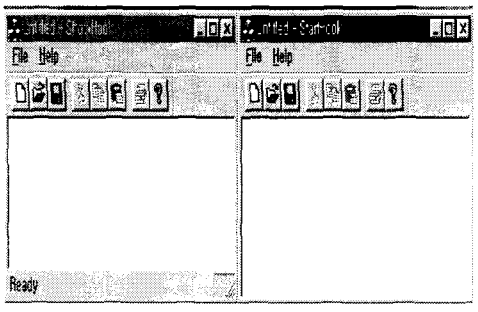


그림 6. StartHook, ShowHook 실행 결과

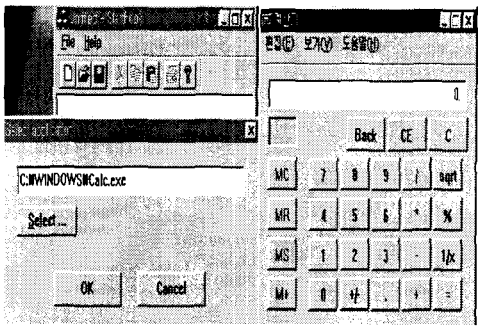


그림 7. 공유할 어플리케이션 선택 결과

다음으로 구성된다.

- StartHook.exe : 어플리케이션 공유 관리기에 해당하며 모듈 삽입 루틴을 가지고 있다. 이 프로그램을 이용하여 어플리케이션 공유의 대상이 되는 프로그램을 실행시키게 된다.
- ApiHook.Dll : API 훅크를 위한 기능을 가진 DLL로 위에서 실행된 프로세스의 API 훅크를 처리하여 Log하는 기능을 담당한다. 이 모듈은 ApiList.txt 파일과 함께 같은 경로에 존재해야 한다.
- ShowApiCall.exe : ApiHook.Dll에서 잡힌 API 호출은 c:\ApiCall.txt화일로 Log된다. 이 log된 정보를 디스플레이 하는 역할을 담당한다.

시뮬레이션을 위해 수행한 결과는 다음과 같다. StartHook.exe와 ShowHook.exe를 실행한 결과는 그림 6과 같다. 다음으로 StartHook내의 메뉴중 'File' 메뉴의 '시작'을 선택한 뒤 'Select' 버튼을 이용하여 프로그램을 선택한 후 OK를 누른다(예로 'calc'를 선택, 그림 7 참조). 이후 프로그램이 실행되면서 호출되는 API 함수들이 c:\ApiCall.txt로 저장되며 이 정보를 이용하여 ShowHook.exe에서 리스트 박스에 이 내용을 출력한다(그림 8 참조). 이 데이터를 공동작업을 수행하는 사용자들에게 GUI를 공유하기 위해 GUI 데이터를 재 디스플레이하는데 사용된다.

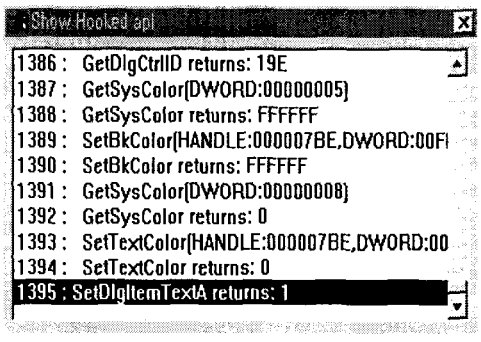


그림 8. 훅킹한 GDI 데이터

V. 결 론

어플리케이션 공유를 이용한 공동작업은 특정 시스템에 있는 개인 사용자 전용 어플리케이션을 지역적으로 분산되어 있는 사용자에게 실시간으로 공유하도록 제공한다. 분산 환경하에서 공동작업을 하기 위해서는 어플리케이션 및 데이터 등을 공유하여야 되는데, 본 논문에서는 어플리케이션 공유를 위한 방법들에 대해 연구하였으며, 윈도우 95 환경에서 윈도우 시스템의 GDI 라이브러리를 사용하여 GDI 훅킹 기법에 의해 GUI를 공유하기 위한 GDI 훅킹 모듈을 설계하고 구현하였다. 이는 이미지에 의한 GUI 공유에서나 이벤트에 의한 공유에서 발생하는 문제점인 트래픽의 과부하 및 속

도 등의 문제가 해결되는 장점을 갖는다. 이는 공동작업을 하는 모든 개발자들이 실시간으로 동일 화면을 공유하며, 네트워크상에서 통신 시스템을 이용하여 화상회의나 정보의 전달 등이 가능하게 하는 많은 응용 분야에 이용되리라 사료된다.

본 논문은 1998년도 한국전자통신연구원 용역과제 및 배재대학교 교내 학술연구비 지원에 의하여 수행되었음.

참고문헌

[1] Walter Renhard, Jean Scweitaer, Gerd Volksen, and Michael Weber, CSCW tools : Concepts and architecture, IEEE Computer, ppp.28-36, May 1994.

[2] K.Strinivas외, Monet : A Multimedia System for Conferencing and Application Sharing in Distributed System. Proc. First Workshop on Enabling Technologies for Concurrent Eng. Enabling Technologies Group, CERC, WestVa, Univ., Morgantown, W.Va., 1992, pp.21-37.

[3] J. C. Lauwer, et al., Replicated Architecture for Sharing Window System : A Critique, Proc. Conf. On Office Information Systems, pp.249-260, April 1990.

[4] S. R.Ahuja, et al., A Comparison on Application Sharing Mechanisms in Real Time Desktop Conferencing System, Proc. Conf. On Office Information Systems, pp.238-248, April 1990.

[5] S. Greenberg, Sharing views and interaction with single-user application Systems, pp. 227-237, April 1990.

[6] 오주병외 3, 어플리케이션 공유를 이용한 소프트웨어 공동개발 플래폼, HCI97 학술대회 발표논문집, pp. 133-136, 1997.

[7] K. Watabe et al., Distributed Multi-party Desktop ACM Press, New York, 1990, pp.27-38.

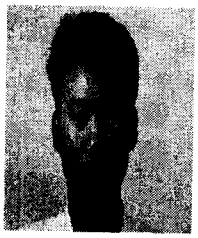
[8] K. Maeno, et al., Distributed Desktop Conferencing System(MERMAID) Based on Group Communication Architecture, Proc. Conf. On ICC91, June 1991.

[9] S Greenberg, Sharing views and interaction with single-user application, Proc. Conf. On Office Information System, pp. 227-237, April 1990.

[10] Andrew Schulman, David Maxey, "Undocumented Windows", Addison wesley, 1994.

[11] Matt Pietrek, "Windows 95 System Programming Secrets", IDG Books, 1995.

[12] 이상엽, "Visual C++ Programming Bible Ver5.x", 영진출판사.



김 형 준(Hyoung-Jun Kim)
1986년 2월 광운대학교 전자계산기공학과 졸업(학사)
1988년 2월 광운대학교 전자계산기공학과 대학원 졸업(석사)
1988년 2월 ~ 1988년 11월 한국전자통신연구원 선임연구원
1988년 11월 ~ 현재: 한국전자통신연구원 정보화 개발팀 팀장
관심분야: 멀티미디어 정보처리 및 미들웨어 프로토콜 설계

박 기 식(Ki-Shik Park)
1981년 2월 서울대학교 영문과 졸업(학사)
1984년 2월 서울대학교 행정대학원 행정학과 졸업(석사)
1995년 2월 충남대학교 대학원 행정학과 졸업(정책학 박사)
1984년 2월 ~ 현재: 한국전자통신연구원 표준연구센터 표준기반연구팀장
1996년 ~ 2000년: 국제전기통신연합(ITU-T) TSAG 부의장
관심분야: 정보통신 표준화 전략 및 지적재산권

조 인 준(In-June Jo)
1982년 2월 전남대학교 전자계산학과 졸업(학사)
1985년 2월 전남대학교 전자계산학과 대학원 졸업(석사)
1997년 10월 - 현재 아주대학교 컴퓨터공학과 박사과정 재학중
1990년 12월 정보처리기술사(전산조직응용)
1983년 9월 - 1994년 2월 한국전자통신연구소 선임연구원
1994년 3월 - 현재 배재대학교 컴퓨터공학과 교수
* 관심분야 : 전산 조직응용 및 정보통신 security

정 희 경(Hoe-Kyung Jung)
1985년 2월 광운대학교 컴퓨터공학과(공학사)
1987년 2월 광운대학교 대학원 컴퓨터공학과 (공학석사)
1993년 2월 광운대학교 대학원 컴퓨터공학과(공학박사)
1994년 3월 - 현재 배재대학교 컴퓨터공학과 교수
* 관심분야 : 하이퍼미디어/멀티미디어 문서정보처리, SGML, XML, HyTime, DSSSL