An Application Recorder and Player

Chee-Wen Shiah, Jyh-Ching Cheng, and Wen-Chin Chen

Communication and Multimedia Lab.
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C

Abstract: This paper will discuss how to record the execution progress of an application and reproduce it. The recording and reproducing of an application execution progress, called *application recorder* and *application player* correspondingly, can be applied to the application-sharing function of a groupware system or the instruction mechanism of a software package.

The application recorder is responsible for recording the execution progress of an application and storing them into a recorded file, such that the application player can reproduce the execution progress. An application execution typically invoke the requests of system calls, and processing of messages. The system calls and messages, therefore, are the basic elements to be recorded.

The recording process consists of many steps, includes: interception, analysis, packet and storing; while the reproducing process includes: retrieve, analysis, packet and invocation. There are also many implementation details should be considered, such as: resource processing, choice of interception point, time simulation and reproducing direction. In the future, the recorded content could be further developed toward a script-like language to provide the editing capability.

1 Introduction

1.1 Background

The application sharing capability had been considered as a very important feature of a CSCW system. There are many shared application systems developed, such as: XpleXer (STC German), SharedX (HP), X/TeleScreen (NIS Inc.), Xwedge(ET Zurich), XTV (UNC, ODU), Joint/X(SieTec), ShowMe ShareApp(Sun), ProShare (Intel), NetMeeting (Microsoft), XMX(BU). Most of them are X-based shared window system, except the ProShare and NetMeeting, they are MS-Windows-based. Those systems are based on the interception and further processing of the communication between shared applications and the display interfaces. Two basic components are: the interceptor and the reproducor, we call them application recorder and application player correspondingly. The application recorder and player can be applied to other situation such as: instruction, product presentation, annotation and so on. We choose the MS Windows environment as our developing platform, due to the already existed rich popular application packages and the demand of our experimental virtual classroom.

1.2 Basic Concept

There are several recording and playing tools developed in the X Windows. Before introducing our system architecture, we will review the basic concept of the X-based application sharing system. Most of well known issues can be found from the related research. Then a brief description of how to implementing a similar system on the MS Windows operating system will be provided.

1.2.1 X-based window sharing

The execution model in X Windows system is a client/server model. An application in X Windows is called an X client, while the display server called X server. When an application started, the X client will establish a connection to an X server. The X client sends requests to X server, and X server responds replies, events or errors. The requests, replies, events and errors form the X protocol. By means of intercepting the X protocols(i.e. keep listening on certain port) and processing (i.e. recording) them, we can monitor and reproduce the whole execution progress of an application. An application sharing system could consists of three parts: a recorder , a player and the record file. The recorder intercepts the X protocols while the execution progressing and stores them in the record file. If we want to reproduce (or playback) the execution progress, we can utilize the player. In a typical player tool, there could be two components: a pseudo X client and a control panel. The pseudo X client reads the data from record file, transfers it through some translation process to the control panel, and sends it to X server. Beside that, the control panel can send control instructions to the pseudo X client to control the playback direction(reverse, pause, stop, or random access).

1.2.2 MS Windows system concept

When an application starts, the MS Windows operating system can be roughly divided into five portion from the application’s viewpoint: the application itself, the window subsystem, some message queues, the base operation system and several hardware devices. The figure 1-1 illustrates this point of view. While application executing, it will invoke many system calls for system services. A system call is an API in MS Windows system. The services are supported by the window subsystem which is mainly composed by 3 DLL: USER, KERNEL, and GDI. Besides the system API calls, an application also sends or fetches messages to/from the message queue to achieve its task. Therefore, the API calls and messages are the basic elements which we will intercept, analyze and record in our application recording system. However, to intercept the API calls and messages within the MS Windows system are more difficult and complicated than to intercept the X protocol within X-Window system, due to the former is not a native client/server platform. An Win32 API interception mechanism for the Windows 95/NT had been proposed by Matt Peitrek in [3].

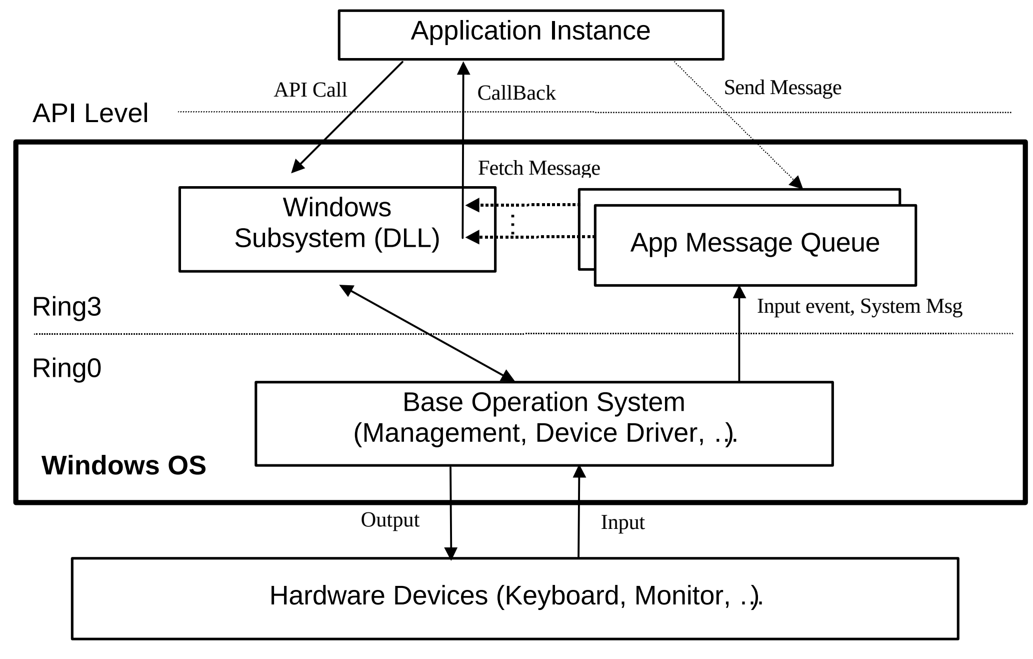


Fig.1-1 Overview of MS Windows system from applications

Besides API call and message, the resource handling is another important issue in the MS Windows system. There are many kinds of resources such as bitmap, menu or dialog box, by which an application can achieve their content presentation. All resources have their own life cycle and a resource identifier (i.e. resource handle). Any reference of a resource is achieved by the resource handle, and should be carried out within the life cycle of that resource. The handles of all resources are allocated, destroyed, and maintained by the base operating system.

2. System Architecture

According to the features and MS window environment, we consider the system architecture of recorder and player. The system can be divided into three components : a recorder, a player and the recorded files. In the conferencing system or virtual classroom , there could be more than one applications executed concurrently, so the recorder must have the ability to record multiple applications simultaneously. Just as playing a video tape, the player should support the speed and direction control. Besides, recorded files size and efficiency of recording must be took into consideration. If the recorded files size is too large, it will occupied the hard disk space. The efficiency of recording is important by reason of it will slow down the speed of the application execution if the recording overhead is large. There are many components in the recorder and player.

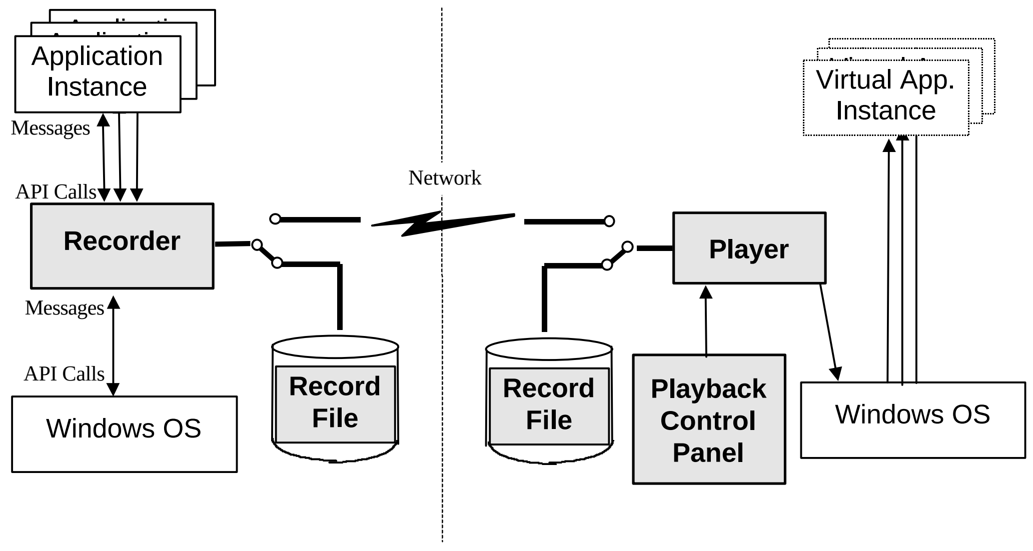


Fig.2-1 Architecture of Application Recording System

2.1 Recorder

The Recorder consists of API interceptor, message interceptor, API filter, message filter, main recorder module and recorded data. The figure 2-1 is the architecture of the recorder. The application will invoke API call, send or fetch message while execution. The API and message interceptor first intercept the API call and message. After that, the interceptors transfer them to API and message filters. The filters are responsible for analysis, packing and forward them to main recorder module for recording.

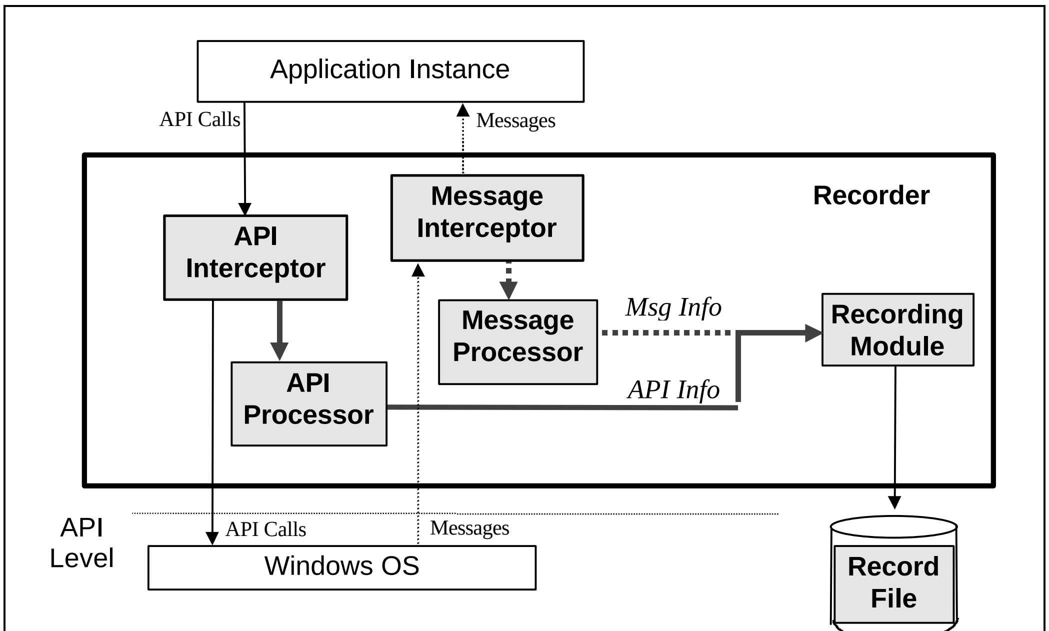


Fig.2-2 Components of Recorder

2.1.1 Main Recorder Module

There are two actions of the main recorder module.

The main recorder module is responsible to injecting the interceptors and filters modules into the application process. This is because the Win32 environment. The method about injection is discussed in another article[4].

Beside that, it also receives the data from filters and record them into the files while application execution.

2.1.2 Interceptor Module

The interceptor is a bridge between application instance and API(Message) processor. In the original situation, the application invoke an API call and execution path enter the API entry of Windows OS. If the Interceptor is injected into the application space, an API call will be intercepted and transmitted its associated data to API processor. After that, the interceptor will invoke the original API call. The message interceptor is similar to API interceptor except that message is pass to application for process. Windows has built in interceptors of message. One can invoke some API to make use of the interceptor.

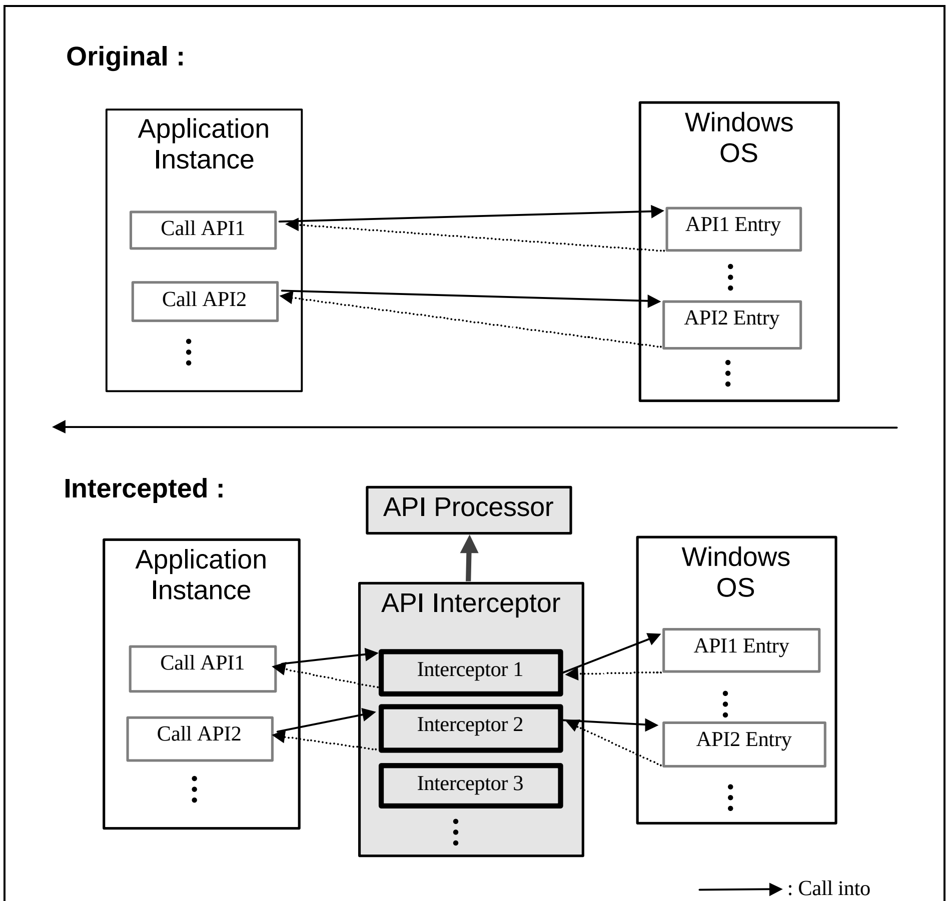


Fig.2-3 API Interceptor concept

2.1.3 Processor Module

The Processor consists of two modules : API processor and Message processor. Both processors have the responsibility to analyze, extract and pack the information of an API call or a message. The most consumed time in recording process is the stage of this process because there are various kind of API calls and messages. The process must distinguish the type and process it by different kind of methods. Both will take much time to do. The efficiency of the recorder largely depend on processor module.

The process flow of API(Msg) processor is shown in figure 2-3. For time simulation, the time point of one API call should be recorded. After that, an API call or message will be processed. Both API call and messages have many different types. These different types should be processed by different methods. After special type process, the common process will do for any API calls and messages. The common process includes packing system time, types, name, parameters and return values. Finally the packed data will be sent to main recorder module for real recording.

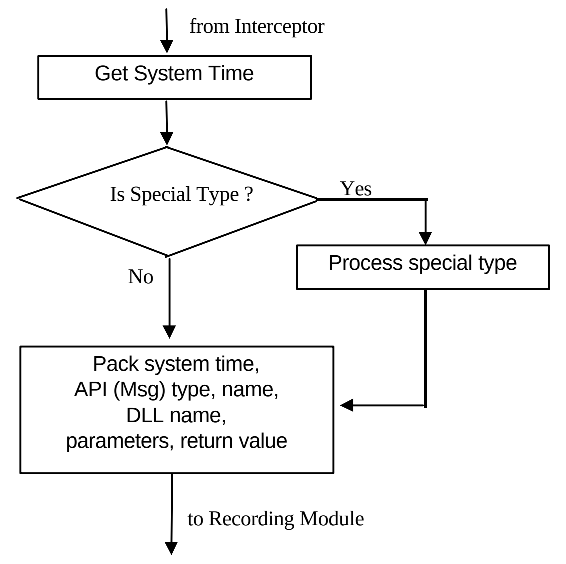


Fig.2-4 Control diagram of API (Msg) Processor

2.2 Player

The player is composed of four components : reproducing process, player control panel, recorded data and Windows OS. The reproducing process is responsible for the main reproducing task. The recorded data stored in recorded files will be retrieved, analyzed, unpacked, translated and reproduced into an API call or message. The reproducing process will invoke the API call or route the message. The series of the reproducing of the reproducing process will make the progress of application execution play again. The visual application instance is the one shown in the output device. The player control panel is an interface for user to control the reproducing process. It will send control instructions to reproducing process such as speed or direction control.

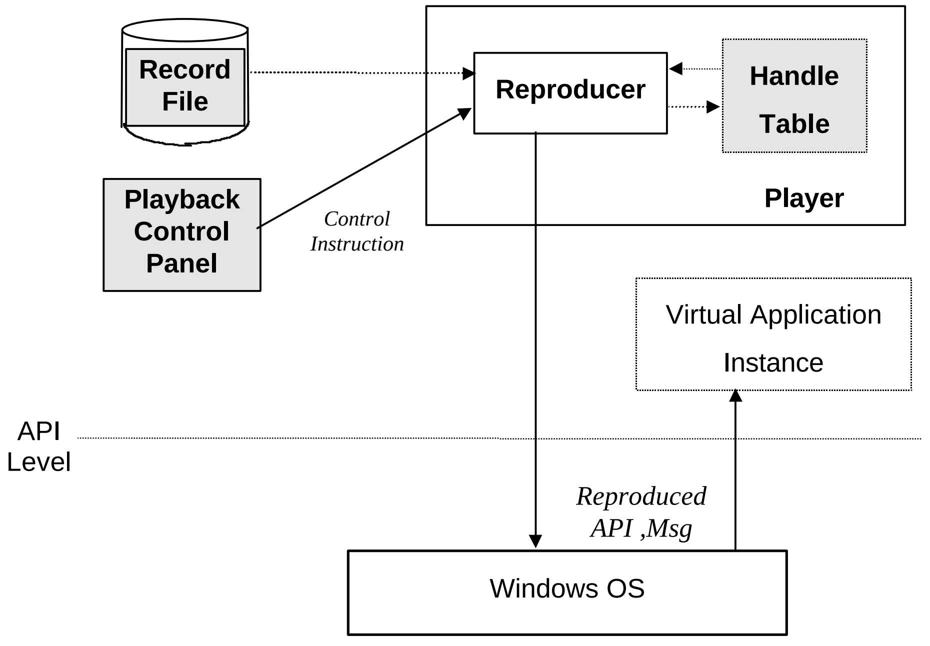


Fig.2-5 Components of Player

2.2.1 Reproducing Process

The reproducing is responsible for reproducing the original API call and message. The process of reproducing an API call or message is shown in figure 2-5. An recorded API call or message is one packet in the recorded files. The reproducing process will read these packets and process them. The reproducing process consists of five components : dispatcher, API reproducing module, message reproducing module, resource reconstructor and handle table. After one packet is read from the recorded files, the dispatcher first distinguishes the type of the packet and dispatches the packet to its API or message reproducing module. If the packet contain resource, the API or message reproducing modules will invoke the resource reconstructor to reconstruct the resource. All of the API, message and resource reconstructor will access the handle table. The API module will reproduce the original API call and invoke it. It is similar to message reproducing process, except that the message is routed.

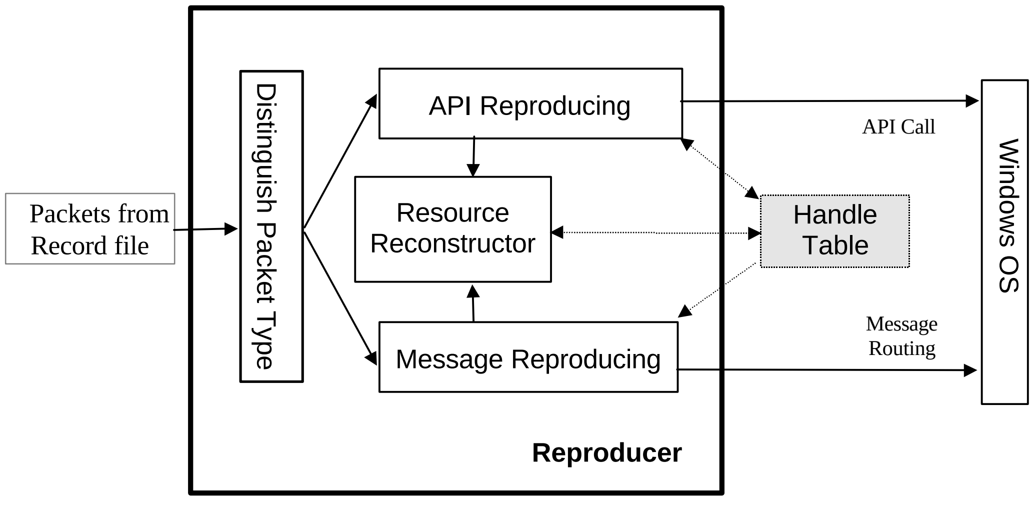


Fig.2-6 Architecture of Reproducer

2.2.1.1 API (Message) Reproducing Modules

The API or message reproducing modules are responsible for reproducing an API call or message. The figure 2-6 is the steps of API or message reproducing including API (Msg) type checking, parameters preparation, resource reconstruction and identifier management, time simulation. Different type of API and message have different special process. The resource reconstruction step is processed by resource reconstructor.

2.2.1.2 Resource Reconstruction and Management Module

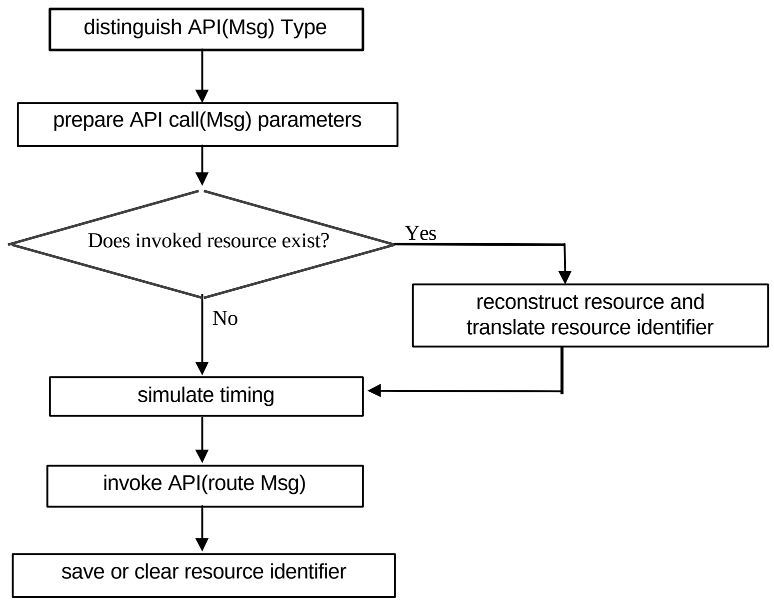

distinguish API(Msg) Type
prepare API call(Msg) parameters
Does invoked resource exist?
Yes
No
reconstruct resource and translate resource identifier
simulate timing
invoke API(route Msg)
save or clear resource identifier

Fig.2-7 Control diagram of API (Message) Reproducing

The resource must be constructed before it can be used and destroyed after it is no longer be used. There maybe some resource references between creation and destruction. Resources are constructed by API calls. If the packet is an API call creating a resource, the API reproducing module will invoke the resource reconstructor to reconstruct the resource. Because the resources are referenced by identifiers allocated dynamic by Windows OS, there must be a management module to manage the resource mapping mechanism. The responsibility of the management module includes saving or clear identifier mapping entries, identifier translation.

2.2.2 Play Control panel

The speed and direction controls are completion by the cooperation of play control panel and reproducing process. The play control panel provides an interface for user to control the reproducing speed or direction. The figure 2-8 is shown to us the steps. The play control panel setup the initial parameters for play such as speed, play style, etc. Then it starts the reproducing process. The reproducing process will enter a loop to reproduce each API call or message. While the reproducing process completes an API call or message reproduction, it will check the play status changed or modified by play control panel. The play status includes speed factor, direction factor and play style.

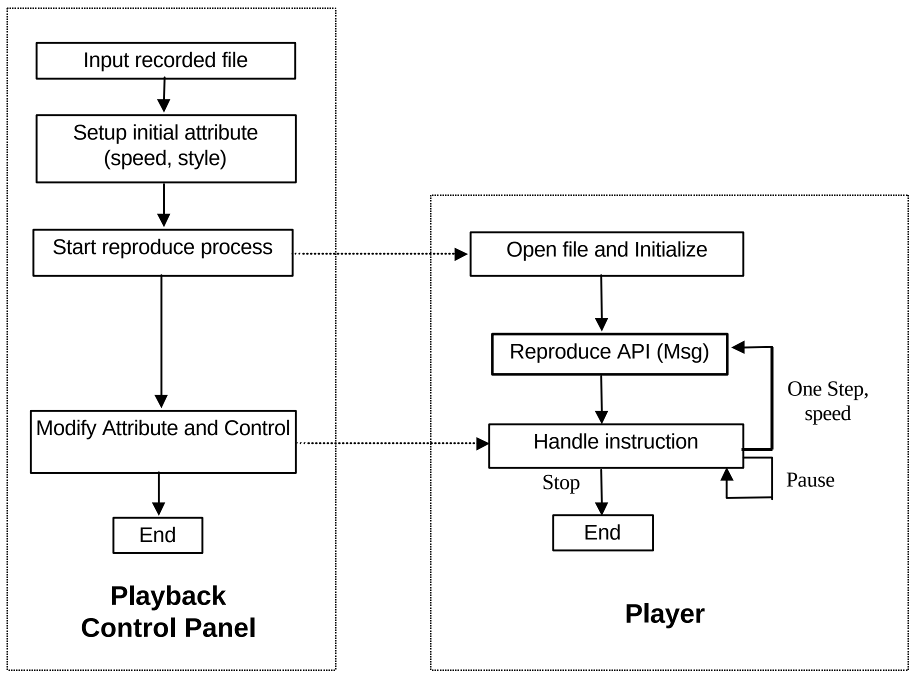


Fig.2-8 Interaction between Playback Control Panel and Player

2.3 Recorded File

The recorded file consists of packets. Figure 2-9 is the packet format. There are six fields in one packet. Packet Type is the field to decide if it is an API packet or message packet. The API call and messages may be classified into several types. Each type will be processed differently. These types are recorded in Sub Type field. The Time field is the time point of an API call invocation or message routing. The field is used for time simulation. The Data Size field is the size of the Packet Associated Data field. Because the Packet Associated Data Field contains information about an API call or message such as names, parameters and a return value, its size is variable. We must record its size by another field. The last Packet Size field is used for reversibly playing.

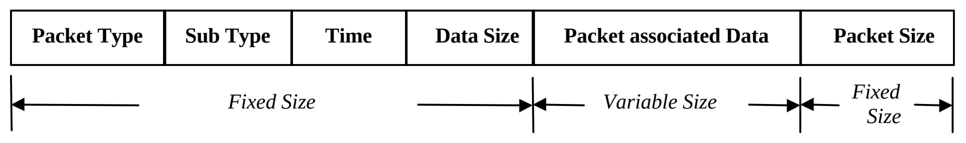


Fig.2-9 The format of recorded data

3 Issues and Discussion

3.1 Resource Retrieve, Reconstruction and Management

The resource has its own life cycle: from creation, modification, use to destruction. Each action has corresponding process for recording and playing sides. If the packet is a type of resource creation, the resource data should be extract and recorded. The resource identifier must be also recorded in the files at recording side. At playing side, the player will reconstruct the resource according to the resource data and save the recorded identifier and newly created one for later references. If the packet is a type of

Type	Example	Process of recording side	Process of playing side
Creation	`LoadMenu()`	Extract and record resource data and identifier	Reconstruct resource and its identifier, save resource identifier into table
Modification	`AppendMenu()`	Record the resource identifier	Translate corresponding resource identifier
Use	`SetMenu()`	Record the resource identifier	Translate corresponding resource identifier
Destruction	`DestroyMenu()`	Record the resource identifier	Delete corresponding resource identifier

Fig.3-1 Resource Process

modification or use, the recorder should record the resource identifier being referenced while the player should translate the resource identifier to its corresponding one. At last, when the resource is destroyed, the recorder do the same thing as modify or record while the player should first translate the resource identifier to its corresponding one and remove the pair. No matter what the resource type is, they can be managed by those methods in the figure 3-1.

3.2 Choice of Interception Point

Because the one process may contain execution file and other non system DLLs, there exists some system API call invocations in the non system DLLs. Besides, the system DLLs may invoke API calls in other system DLLs, so the points being intercepted are not simple. The figure 3-2 illustrates the various paths of API calls. The path 1 is not the choice of interception because the API calls are not ones of system DLLs. Not every machine has this DLLs, so we don't choose to intercept. The path 2,3,5 must be intercepted, but the path will cause some problem. Because the DLLs has only code instance in the system, it will interfere each other if not only one application instance use the DLL concurrently. We must distinguish the recorded one and normal execution one. The path 4 sometimes should be care and sometimes not according to its regularity of invocation times. If the invocation times are fix, it can be not care about.

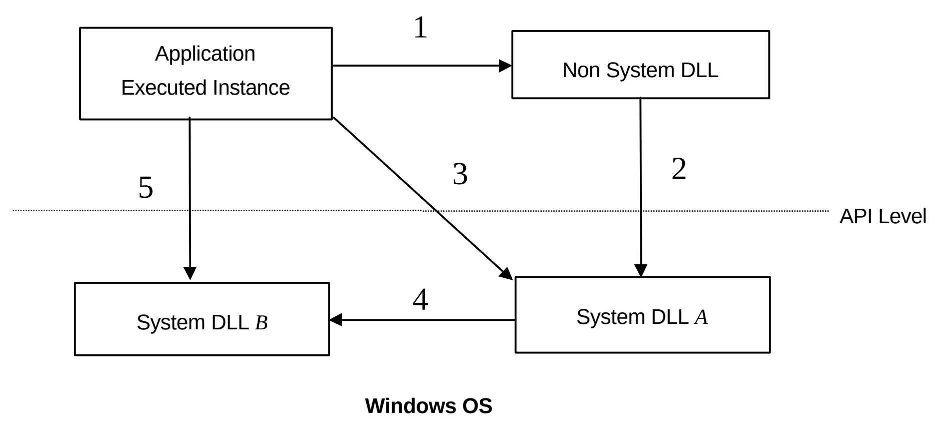


Fig.3-2 The point of Interception

3.3 Time Simulation

The concept of time simulation is simple. If the time slice between two API calls or messages routing in recorder side is the same as the one in play side, the time simulation is done. The time spent in recorder side includes recording steps, API calls or message routing, application execution. The time spent in player side includes reproducing preparing, API calls or message routing and the time delay to simulate.

3.4 Reproducing direction and Jumping Reproducing

The function of reproducing direction and jumping reproducing will meet two main problems. The first one is output problem. Simply recording the API calls and messages is not enough to solve the problem because the output will disturb each other. If we want to back to some time point previously, we must restore the output status in that time point. But we cannot know the output status in that time point.

Another problem is resource management. The resource has many steps from its creation to destruction. If we jump some steps, there must be some problem occurs such as skipping the resource creation or destruction step. The figure 3-3 is an example of execution progress. If the reproducing process has been reproduced the first three API calls and the user wants to jump to API 8, there are two situations will encounter because one resource is destructive and another is creative. The resource 1 has been destroyed at the time point of API 8, so API 7 should be called to destroy the resource 1 and the resource identifier entry corresponding to resource 1 should be removed. Second, the resource 2 is already created at time point of API 4 and modified at time point of API 6. So we must first reproduce API 4 and 6, let the resource 2 correctly in its state. After all, before reproducing API 8, the API 4, 6 and 7 should be reproduced first and the jump will succeed from API 8.

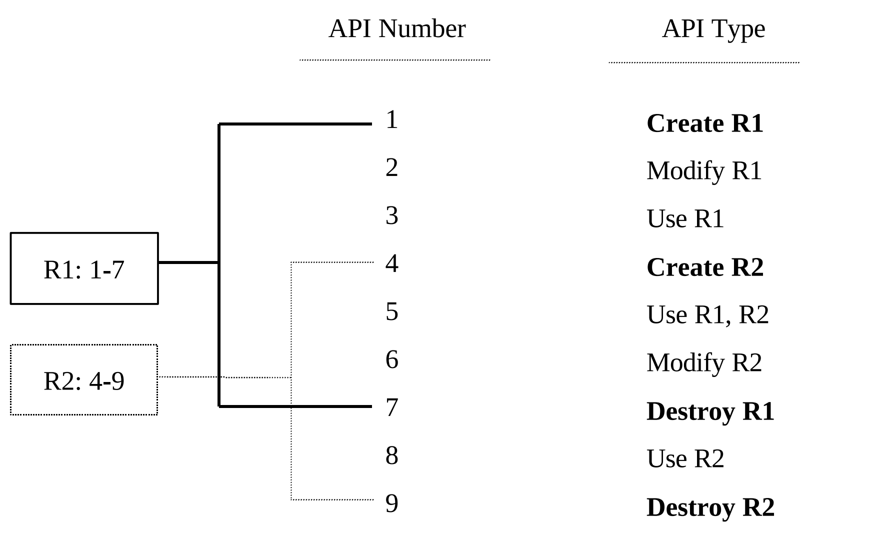


Fig.3-3 An Example of Resource Usage

3.5 System Performance Analysis

The recorded file size and packet number are illustrated in figure 3-4 and 3-5. As time goes, the quantum increased. There are three points should be taken care from the curve of figure 3-4. In the initial stage, the curve is a slope. This is because there are many resource being constructed in this stage. The resource usually has raw data largely. After that, the execution enters the interactive stage, the small pulses are resulted from user operation. The sharp slope at 23 sec is an action opening a file. The file contents are fully recorded in the recorded files. After 24 sec, the recorded size is about 30k bytes totally. The figure 3-5 has one point to notice. The message packets increase more than API ones. This is because the mouse or keyboard inputs are frequent in the interactive stage while the execution progress is in initial stage, the message packets is close to zero.

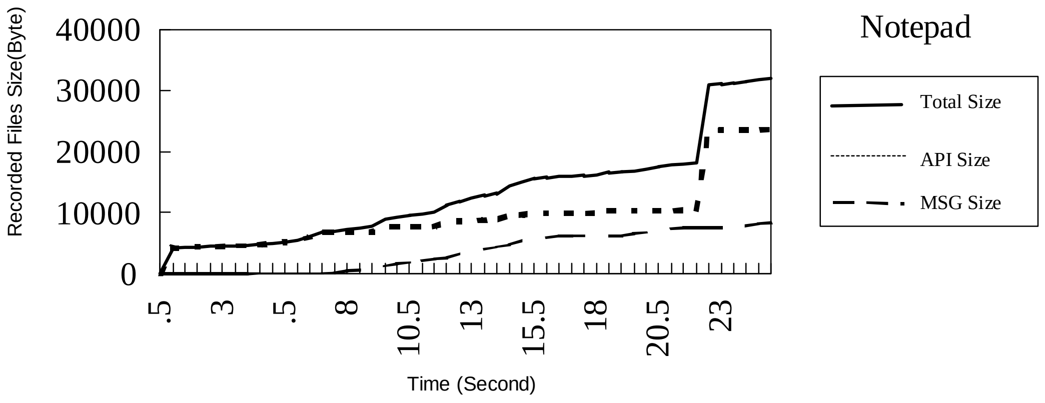


Fig.3-4 Recorded File size of Notepad application

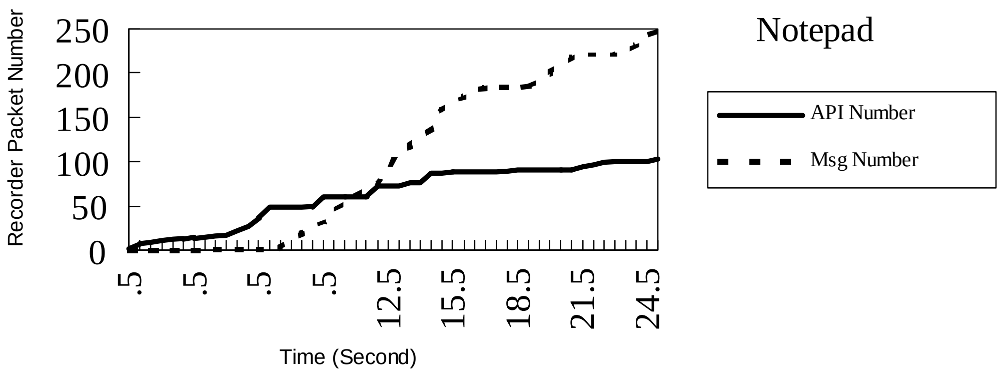


Fig.3-5 Recorded packets number of Notepad application

4 Future Works

4.1 Recorded File Size Minimization

In the recorded files, the data of resource take a large portion. The resource data often contains binary data such image raw data or text. If we can compress the resource data, the recorded files size will be small.

4.2 Reversibly Play

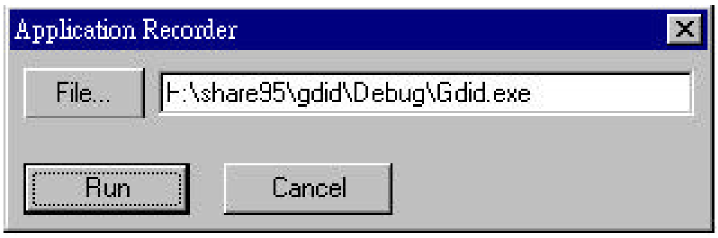
The ability to reversibly play is useful. Just like video play, the reproducing of execution progress make a user to review the application execution.

4.3 Edit

One can modify the recorded data to edit the execution progress such as adding some portion of the progress into another or erasing some portion. Moreover, the recorded files format can be developed to script language for more editing.

5 Conclusion

The figure 5-1 is the interface of application recorder. One can press the file button to select a file from dialog or type directly in the edit box. After that, press run button to start and record the application or cancel button to escape.



F:\share95\gdid\Debug\Gdid.exe

Fig.5-1 Interface of Application Recorder

The figure 5-2 is the interface of application player. One can select a recorded file by the open button and play the progress by the play button. There are three status will be shown in the interface : no(no application running), running and pause. There are two play styles : normal and step. One can choose continuously play or step by step trace. The Time check box is an option to simulate time factor of the execution progress.

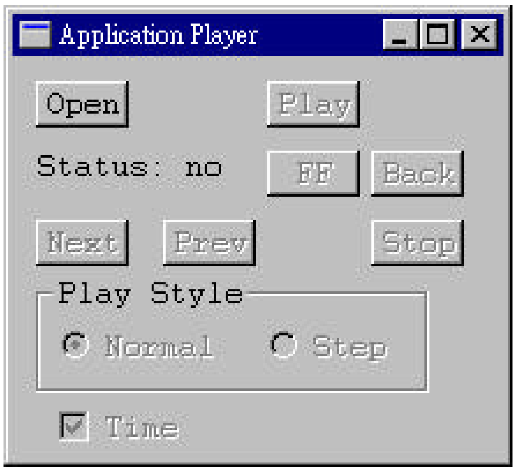


Fig.5-2 Interface of Application Player

6 Reference

[1] "A Practical Point-to-point Filter-based Shared Window System", Chee-Wen Shiah, Jye-Ching Cheng, and Wen-Chin Chen,1995

[2] "Recording and Playing of X Protocol Messages in XTV", Hussein Abdel-Wahab, September 20 ,1995

[3] Matt Pietrek "Intercept Win32 API", PC MANAZINE Chinese version APRIL 10, 1995 , p 246-255

[4] “Load Your 32-bit DLL into Another Process‘s Address Space Using INJLIB”,MJS:1994#5(May),Microsoft Development Library

[5] “Learn System-Level Win32 Coding Techniques by Writing an API Spy Program”,MJS:1994#12(Dec) Microsoft Development Library

[6] Abdel-Wahab,Hussein M.andFeit,Mark A.,“XTV: A Framework for Sharing X Windows Clients in Remote Synchronous Collaboration”, Communications for distributed Applications & Systems, Chapel Hill, North Carolina, pp. 159-167, April 1991

[7] Chung,Goopeel, “Accommodating latecomers in a system for synchronous collaboration”, MS Thesis, Department of computer Science, University of North Carolina at Chapel Hill, 1991

[8] Issues, Problems and Solutions in Sharing X Clients on Multiple Displays“, Internetworking: Research and Experience, Vol.5,pp 1-15,1994

[9] Jeffrey Richter, “Simulating Keyboard Input Between Programs Requires a (key)Stroke of Genius”, MSJ: 1992#8. Microsoft Development Library

[10] James Finnegan, “Hook and Monitor Any 16-bit Windows Function with Our ProcHook DLL.” Microsoft Journal, January 1994, Vol.9, No.1.