Principle of Capturing Word from Screen and Its Implement Methods

Xiaoyu Ji, Tingyi Zhao and Chaoxiang Liang
Department of Wuzhou University, the Computer Science, No. 82,
Fumin 3rd Road Wuzhou, Guangxi, Postal Code 543002, China

Abstract: Capturing word from screen refers to the software can recognize the characters at the position pointed by mouse along with its movements. The aim of this study is to assist in browsing online and online reading of articles in foreign language. After introducing the principles for capturing word from screen under Windows 2000/XP context, this study proposes a method which obtains the character string at the mouse position in terms of code interception, mouse HOOK and screen refresh and more, then gives the implementation on Delphi. The application demonstrates that this method possesses the favorable commonality and flexibility. In the end, it makes analysis on the limitations of this method under some particular situations.

Key words: Capturing word from screen, Hook, code interception

INTRODUCTION

The key of capturing words from screen is how to capture the string at the position of mouse, for which the dynamic link and message response mechanism of Windows provide the realization ways (Chen, 2006). Although this technique seems simple, in fact, the way in windows system is extremely complex and there are mainly two implement modes:

- By intercepting part of the gdi call to api, such as textout, textouta and so on
- By coping every device context (dc) and tracking all the operations that modify the context (dc)

The second method is more powerful, but its compatibility is not ideal; the powerfulness of interception of the Windows API call used in the first method may be far beyond your imagination and it is no exaggeration to say that you can change the entire operating system by using Windows API intercept technology. As for the interception of the Windows API call in the first method, specifically speaking, there are also two methods:

- By directly rewriting the images of Windows API in memory and embedding assembly code, which is led into an specified address when called to realize the interception
- By rewriting IAT (import address table) and redirecting the Windows API function call to realize the interception of the Windows API

In this study, the second interception method will be introduced, for this method runs stably and has a relatively compatibility in 32-bit Windows system. Generally speaking, the capturing of the string at the position of mouse on screen is realized mainly through the steps below:

- **Code interception**: Windows provides system services in the way of DLL, which is convenient for getting the address of the Windows character output API, modifying the entry code and intercepting the application calls to them
- **Mouse HOOK**: The global mouse HOOK process such as the type of WH-MOUSE is installed to monitor the movement of mouse on the entire screen
- **Screen refresh**: An area around the mouse is made invalid, which can force the window of the mouse position to refresh the screen output. Window procedure responds to the WM-PAINT message and the call of character output API like ExtTextOut refreshes the string in invalid region. These calls are intercepted by us and parameters of the character API, such as string address, length, output coordinates, HDC, cutting area and other information passed by Window procedure from the stack are obtained

Since the relatively basic knowledge such as Windows virtual memory management, breaking of the wall of process boundaries, injection of code into process space of applications, PE (portable executable) file format,
IAT (import address table) and so on is required, at first, the general introduction to the related knowledge will be made in this study and then the key code of interception part will be provided.

**PRINCIPLE ANALYSIS**

Each process in 32-bit windows system can get an allocation of 4GB address space and the system reserves an address space from 2GB to 4GB prohibited to process access. Actually, this area of the virtual address space is shared by all Win32 process and it is visible for each process. Besides, this area of the address space is loaded with win32.dll, memory-mapped files, vxd, memory manager and file system. The windows system has reserved address space from 0 to 4MB for 16-bit operating system and from 4MB to 2GB is the private address space only for win32 process. Since the address space of each process is relatively independent, which means if the program wants to intercept API function call in other process, the boundary wall of the process must be broken up and the code of intercepted API function call should be injected in its process. This work will be completed by the hook function (setWindowsHookEX) and all the function from the remote-hook should be stored in dynamic libraries. In this case, when the process calls function from a dynamic library implicitly or explicitly, this dynamic library will be mapped to the virtual address space of the process by the system, which makes dll to become a part of the process executed as part of it by using the stack of this process, namely, the code in dynamic linking library has been injected with other GUI process’s address space by the hook function.

When the DLL containing the hook is injected into other processes, the base addresses of various modules (EXE and DLL) mapped to the virtual memory of this process can be acquired and the mapped position of EXE and DLL in virtual memory space is decided by their base addresses. The base address is determined by the linker upon linkage. When you create a new win 32 project, the linker of compiler will use the default base address 0x00400000. The base address of the module can be changed through the Base option of linker. Usually, with exe mapped to 0x00400000 in virtual memory, the base address of DLL will be accordingly different and mapped into the same virtual address space of different process in most cases. EXE and DLL are mapped into the virtual memory space by system and their structures in memory and static file structures on disk are the same, namely, PE (Portable Executable) file format. After obtaining the base address of process module, all the image-import-descriptor arrays of this module can be exhausted to identify whether the dynamic link library with the functions that need to be intercepted is introduced into process space. For example, if the function “TextOutA” needs to be intercepted, a check must be done if the dynamic link library “gdi32.dll” has been introduced. In fact, all process calls to the given API function are realized by a place in PE files, that is, IAT import address table from the “Idata” segment in the module (can be EXE and DLL). All the other DLL function names and addresses called by this module can be found there. The call to its DLL function actually just jumps to the import address table and then jumps to the real DLL function entry from the import address table.

In the Windows2000/XP systems, most of the characters on screen is displayed though several functions (John, 2004) in gdi32.dll and user 32.dll: TextOutA, TextOutW, ExtTextOutA, ExtTextOutW, DrawText and DrawTextA. The call relation is as shown in Fig. 1.

From Fig. 1, it is clear that no matter the function is outputted through whichever character, application program will output the character to the screen eventually through ExtTextOutA and ExtTextOutW from GDI 32.dll. Thereinto, ExtTextOutA output of character is in the ANSI format and ExtTextOutW output of character is in the UNICODE format. Therefore, as long as functions of ExtTextOutA and ExtTextOutW are intercepted, all string outputted from the application will be intercepted.

**IMPLEMENT STEPS**

In this study, Delphi will be used to capture words from screen under Windows2000/XP system and following are the Implement steps and key pseudo codes.
Interception of character output through API

Basic thinking of code interception: In order to achieve the interception of API such as ExtTextOutA/ExtTextOutW, a dynamically generated "JMP<replacement function>" instruction should be placed at the function entry and the JMP operand is the address of an intercepted replacement function which is provided (Richter, 2000). When the API is called, the JMP instruction will be executed firstly, jumping to the replacement function. Replacement functions are responsible for works like getting parameter from stack, calculating coordinate of the string, siphoning off words at the mouse position. After the execution is completed, intercepted function will be called again by replacement function to finish normal character outputting and then turn back. Figure 2 is the normal calling process of TextOut by the application and Fig. 3 is the process after TextOut system being intercepted.

Construction of the intercepted replacement function:
Interception of replacement function which will insert into the process of intercepted API needs same parameters and prototype of returned value. Take the interception of TextOutExtA as an example (Gu and Diao, 2004) and the Delphi key codes of replacement function myExtTextOutExtA are written as follows:

```delphi
Boolean myExtTextOut(HDC hdc, integer x,y, LPSTR lpszstr,integer cbstr)
{ DoSpy(hdc,x,y,lpszstr,cbstr);
  //Save the parameters and do all the work to grasp the words.
  RestoreCode();
  //Recover the original instruction of entry to TextOut
  TextOut(hdc,x,y,lpszstr,cbstr);
  //Call the original API
  SpyCode();
  //Put in the JMP instruction again
  return TRUE;
}
```

Firstly, function calls DoSpy () to grasp the words, then function Restore Code () recovers the intercepted codes in front of the function entry, after which instructionText-Out () will be called to execute normal character output and then SpyCode () will put the JMP instruction in front of the intercepted function entry again and return to the calling process finally.

Obtaining of the dynamic link address of the intercepted API: The address of ExtTextOutW/ExtText-OutA can be obtained by GetProcAddress and following is the Delphi pseudo code taking myExtTextOut as an example:

```delphi
Var SysFunc: Pointer;
//The address of the system function
SysFunc:= GetProcAddress(GetModuleHandle('gdi.dll'), 'ExtTextOutA');
```

Dynamically generation of JMP instruction: Dynamically generated JMP instruction occupies five bytes and is stored within TCode5 recorded. Taking JMP instruction of replacement function myExtTextOut
which will generate and jump to myExtTextOut for example, following is the critical code of Delphi:

```pascal
Var
  pThunkFunc: Pointer;
  // Address of the replacement function
  codeBak: TCodes;
  // The first five bytes of the old function
  codeThunk: TCodes;
  // The five bytes of the new function
  pThunkFunc := GetProcAddress(hInstance, 'myExtTextOutA');
  codeBak.sizemp := FByte(pSysFunc);
  codeBak.dwAddr := DWORD(DWORD(pSysFunc - Func) + 1);
  codeThunk.sizemp := ShortInt(SE9);
  codeThunk.dwAddr := DWORD(pThunkFunc) - DWORD(pSysFunc) - 5;

Modification of entry of the intercepted functions: Modification of the entry of ExtTextOutW/ExtTextOutA is capable of redirecting to its own code of myExtTextOutW/myExtTextOutA upon system call of the two functions. Modifications can be made through WriteProcessMemory because its API can copy codes with specified length into the address space of appointed process (John, 2004). The key codes are written as:

- WriteProcessMemory(hProc, pSysFunc, @codeThunk, 5)

Statements above mean coping the first five codes in codeThunk (namely the entry address of replacement function myExtTextOut) to the space of pSysFunc process (that is the entry address of system function ExtTextOut), which means the completion of interception and modification of API such as ExtTextOut and so on.

Application of HOOK to monitor mouse movements: With the installation of a global HOOK of WH-MOUSE type, all mouse messages can then be intercepted (Xiong, 2002). Once in case of mouse event, HOOK process is called to send messages of mouse positions to main window after judging out the mouse movement. In the global HOOK process, DLL is required to be added in and only a system hook of WH-MOUSE type is enough, whose key codes are as follows:

```pascal
SetWindowsHookEx(WH-MOUSE, HOOKPROC(MouseProc, GetModuleHandle("hookdll.dll")), HOOKPROC(MouseProc);
// My dynamic database
/Indicate the system hook, which intercepts all messages
In callback function:
End:
```

Refreshing of screen output: A rectangular region is formed in accordance with the mouse position (more exactly, the position requiring to capture words) and then this region is made to be invalidated by calling InvalidateRect to compel the refreshing of screen output of the target window. In response to the information in WM-PAINT system, the call of target window to the character output API (ExtTextOutW/ExtTextOutA) is intercepted and dealt with, in order to complete one process of capturing words. Key codes are written as:

```pascal
Var rect: TRECT;
// Invalidated rectangular region
ScreenToClient(hWnd, &MousePoint); // Converted coordinate
rect.left := MousePoint.x;
rect.right := MousePoint.x + 1;
rect.top := MousePoint.y;
rect.bottom := MousePoint.y + 1;
InvalidateRect(hWnd, @rect, FALSE);
// Invalidate
```

**PROGRAM IMPLEMENT**

In accordance with the analysis above, the program of the following word capturing examples will be realized under the context of Delphi 7.0 and the file lists are shown in Fig. 4, of which the FreeDict.exe is the main program.

![Diagram](image_url)  
**Fig. 4:** Source files list of capturing word software developed by Delphi 7.0, of which the FreeDict.exe is the main program.
Fig. 5: Demo of capturing the word “EclipseUML” of folder by intercepting the function ExtTextOutW in windows system.

As the main program interface shown in Fig. 5, when mouse is at the position of file “EclipseUML”, the main window gets the text “EclipseUML” in the mouse position and displays it in the main process window by intercepting the ExtTextOutW method in windows system.

CONCLUSION

This study makes analyses on screen word-capturing theories and various mainstream word capturing techniques in the 32-bit windows system. The method of application of hook to capture words is applied in this study, which is proved to be feasible. Such kind of technique is based on the principle of character output theory from calling Windows system ExtTextOut by the application program; while hook mode then cannot intercept the characters in buffer areas if the application program possesses its own character output system instead of ExtTextOut output.

Due to the above limitations, the technique OCR can be considered to be applied in capturing words from screen, namely directly get one block of graphics from screen and then recognize the characters, but the problem of its efficiency can still not be solved rightly (Ji, 2008).

ACKNOWLEDGMENTS

This study is supported by the Science Research and Technology Development Project of Guangxi Zhuang Autonomous Region of China under Grant No. 11107007-8, University Talents Program of Guangxi Zhuang Autonomous Region of China under Grant No. GHR-2012-41 and Department of Education of Guangxi Zhuang Autonomous Region of China under Grant No. 201106LX872.

REFERENCES