ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

응용 맞춤형 그래픽 분할 실행 라이브러리에 기반한 효율적인 온라인 소프트웨어 서비스☆

An Efficient On-line Software Service based on Application Customized Graphic Offloading Library

최 원 혁[1*] 김 원 영[1]
WonHyuk Choi Won-Young Kim

요 약

본 논문에서는 응용 맞춤형 그래픽 분할 실행 라이브러리에 기반한 효율적인 온라인 소프트웨어 서비스에 대하여 소개한다. 그래픽 분할 실행을 이용한 소프트웨어 서비스는 클라이언트 렌더링을 통하여 3D 그래픽 소프트웨어와 같은 고사양의 소프트웨어를 서버 기반의 온라인 소프트웨어 서비스로 제공할 수 있다. 그래픽 분할 실행은 서버에서 소프트웨어가 실행될 때, 그래픽 관련된 작업은 클라이언트의 GPU를 이용하여 처리하고, 데이터 관련 작업은 서버의 CPU를 이용하여 처리하는 방식이다. 그래픽 분할 실행 소프트웨어 서비스의 성능을 향상시키기 위하여, 비동기 전송 채널을 추가하고, 최적화 된 소프트웨어 공통 모듈과 소프트웨어 맞춤형 모듈을 기존의 그래픽 분할 실행 엔진에 추가한다. 이를 위하여, 본 논문에서는 그래픽 관련 API와 메시지들을 분석하여 소프트웨어 맞춤형 모듈을 구현하고, 서버 사이드 캐싱 방법을 통하여 최적화된 소프트웨어 공통 모듈을 구현하는 방법에 대하여 기술한다. 마지막으로, 성능 비교 실험을 통하여 개선된 분할 실행 엔진이 더 나은 성능을 가짐을 보여준다.

☞ 주제어 : 그래픽 분할 실행, 응용 맞춤형 라이브러리, 소프트웨어 서비스

ABSTRACT

In this Paper, we introduce an efficient on-line software service using an application customized graphic offloading library. The software service based on graphic offloading provides high-end software, like a 3D graphic design tool, as an on-line software service through using a client graphic rendering. When software is executed on server, its graphic works are handled by a client's GPU, while its data works are handled by a server's CPU. To improve the performance, we apply an asynchronous transmission channel scheme to our developed basic graphic offloading engine. Also, we add optimized common module and application specific module to our engine. To do that, we introduce how to implement the application specific module using analyzing patterns of graphic related APIs and messages that are generated by an executed software process. Also, we propose how to design the optimized common module using server side information caching. Finally, through the performance comparison experiment, we show that improved offloading engine has the better performance than old basic offloading engine.

☞ keyword : Graphic Offloading, Application Customized library, Software service

1. Introduction

Recently, the wide spread of cloud-client computing has changed a personal computing environment to a server-based on-line computing based on virtualization technologies. Server-based desktop virtualization technology provides end-users with software service using virtual machine on server. It is a solution for the problems, such as a data security and a personal computer purchase and management cost occurred in a personal computer based computing because of managing user data and environments on the server, it can improve end-user's work efficiency to provide consistent access regardless of the type of devices and the location and time of end-users [1].

Technologies for supporting a server-based computing

[1] Electronics and Telecommunications Research Institute, 218, Gajeong-ro, Yuseong-gu, Daejeon, 34129, Korea
* Corresponding author (whchoi@etri.re.kr)
[Received 15 April 2015, Reviewed 22 April 2015, Accepted 10 August 2015]
☆ This work was supported by the Technology Innovation Program [10035185, Development of a Separated Software Execution Technology for Software Service] funded by the Ministry of Knowledge Economy (MKE, Korea).
☆ A preliminary version of this paper was presented at ICONI 2014 and was selected as an outstanding paper.

include VMware Horizon[2], Citrix XenDesktop[3], Microsoft Remote Desktop Service[4] and so on. These technologies adopted the thin client technology[5] that a server executes software and a client only serves as a terminal[6]. These lead to problems of huge cost for buying and maintaining servers. Also, the thin client technology is hard to serve high-end 3D graphic software. For serving 3D graphic software, Xendesktop supports to dedicate a graphic card to a virtual machine, Microsoft Remote Desktop Service servers to share a high cost GPU and VMware Horizon also needs high cost GPUs for dedicating service or sharing service. These solutions have problems to buy and manage GPUs and servers for expanding service. Also, these cause to degrade the quality of service according to the constraints of network bandwidth because that compressed result images performed on the server transmit to a client terminal [1].

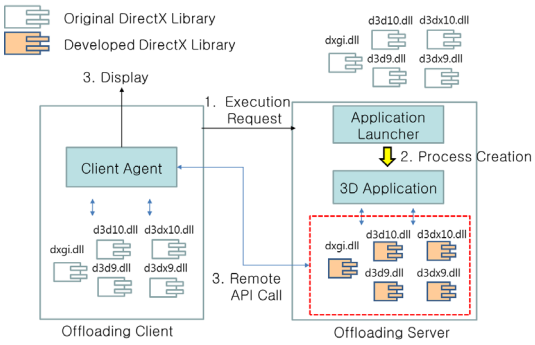
However, with the recent development of manufacturing technology of hardware, cheap high performance personal computer is rapidly becoming widespread. Using this, researches about client remote rendering have been actively attempted. Client remote rendering is that client's GPU processes graphic works generated by executed software process while 3D software is executed on the server. It can take advantages of the thin client technology because of executing and managing software on the server. It can handle by splitting server's huge loads to client terminal. Also, it can provide a service with a relatively small bandwidth because of processing and transferring graphic command and data.

To do that, we introduce a client-based remote 3D rendering concept using graphic offloading[7] for server-based on-line software service. We propose a window message virtualization to provide a virtualized desktop window that the 2D window created by offloading software is processed independently in the server's window management system. Finally, to improve the performance of graphic offloading service, we suggest two efficient schemes. We apply an asynchronous transmission channel scheme to the graphic offloading server/client engine. Alos, we add optimized common module and application specific module to the graphic offloading server/client engine. To do that, we show how to implement the application specific module using analyzing patterns of graphic related APIs and messages that are generated by executed software. Also, we show how to design the optimized common module using a server side caching technique, such as window Z-order caching, window information caching and device context caching. Finally, through the performance analysis, our proposed optimized engine proves more efficient than the basic offloading engine.

2. Remote Client Rendering

2.1 3D client rendering

Figure 1 describes the concept of a proposed 3D graphic client rendering based on API remoting. When client requests to execute DirectX-based 3D software, the application launcher of the offloading server executes by binding the requested application to the developed DirectX wrapper library instead of the original DirectX library in the Windows OS. When the executed process calls DirectX API, the developed wrapper library is called and transmits the called API information and parameters to the client via communication channel. The client agent received API information calls the corresponding DirectX API in the offloading client and then the rendering result is displayed on the offloading client display device [8, 9].

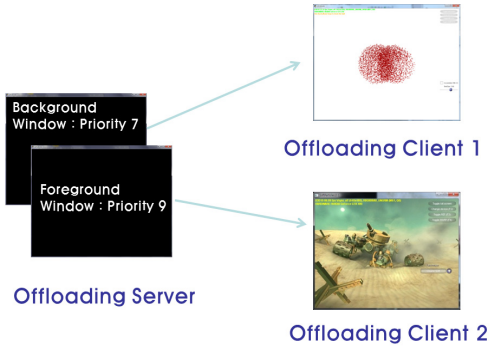


(Figure 1) The concept of 3D graphic client rendering using remote API call

2.2 2D client rendering based on virtualizing desktop window

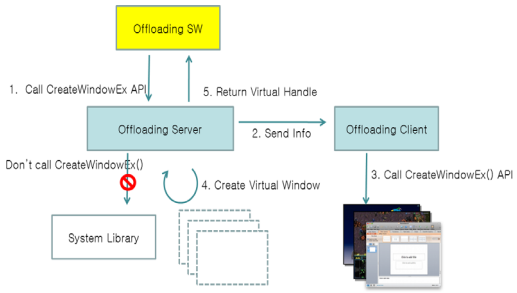
Figure 2 shows the interference effects between client windows on the offloading server when requested applications are executed by their own window on the same server. Our proposed solution should be able to offload not only 3D graphic

API but also window and 2D GDI API on the offloading client's GPU. A window is the unit of application process that has the message processing loop in the Windows OS. In case of creating the corresponding windows on the offloading client and server to serve window procedure when an offloading application is executed on the offloading server, the performance of an executed application, like a FPS, can be affected because of the priority of an application window created on the server system. Also, it can affect to the an another application process behavior that connect to the same offloading server due to window changing messages, such as Focus Gain/Lost, Hide/Restore, Draw/Redrawing, Minimize/Maximize and so on, occurred by created windows.



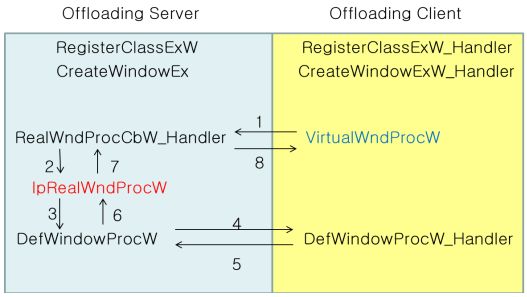
(Figure 2) The interference effects of client windows on server

To resolve this problem, we propose a desktop window virtualization that virtualizes a window created by the server on the client system. To provide a desktop window virtualization, we need two elements-an API remoting of window/GDI library and a remote processing of window message.



(Figure 3) The concept of processing window creation API

Figure 3 is the concept of remote executing for 2D GDI and window related API call. Like the processing concept of 3D API, if the offloading software executed by server calls CreateWindowEx API to create a window, then the server hooks API call[10] and transfers API data to the client instead of calling directly the own system library. Then, the client calls CreateWindowEx API in the client system. At this time, when the offloading server creates virtually a window, the generated virtual handle is mapped to the real handle returned by the client. Using this handle, GDI and DirectX API could be rendered remotely[1].



(Figure 4) The sequence of handling window message

Figure 4 is the representative sequence of handling window message based on virtual window when offloading client receives user inputs. The created window on the server only has window procedure routine to process messages. The virtual window created by the client also has a virtual window procedure corresponding to that of the server. Using this virtual window procedure, the offloading system processes messages generated by user inputs[1].

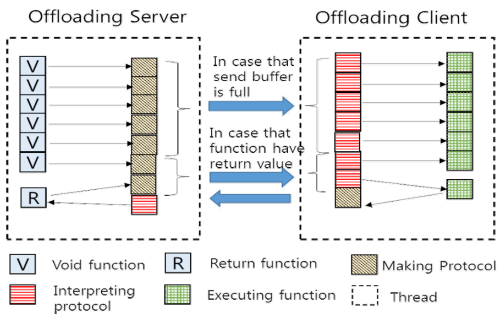
3. Asynchronous Transmission Channel

A commercial software occurs over 10,000 times graphics processing function calls per frame. These frequent function calls cause a performance degradation problem. For solving this problem, we apply an asynchronous transmission channel to the basic offloading engine middleware. Data transmission channels of the basic offloading middleware is implemented as independent send/receive channels per single thread. As

Figure 5, when the server-side send buffer of offloading middleware is full or function which has return value is executed, these command data are transmitted to offloading client and are handled by offloading client.

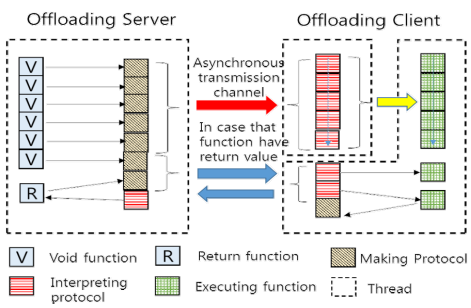
In this paper, we add two concepts to data transmission channel as follows.

Until the server-side send buffer is full, the offloading client waits to receive data. It is one of important factors to degrade the performance.Even if the server-side send buffer is not full, the offloading server's thread transfers buffer's data to the client. So, we can improve the performance of offloading service because the offloading client can't wait long in the listening state.



(Figure 5) The structure of data transmission in basic offloading system

Also, the offloading client executes the separately protocol intepretation step and the function execution step on each independent thread. By the way, one thread processes interpretation jobs of received protocol messages, and the other thread only executes a proper function corresponding to interpreted result from interpretation thread.



(Figure 6) The structure of asynchronous transmission channel

To do that, we implement an asynchronous transmission channel and thread as Figure 6. A bi-way function command, which has return value, is processed according to the original step because it needs to reliable processing. But, because one-way functions doesn't need return value, these could be frequently transmitted via an asynchronous channel. After the offloading client receives and interprets these one-way functions, it puts interpreted results in its own receive buffer. Next, the command execution thread takes over those result and calls proper functions[11].

4. Application Customized Graphic Offloading Library

For all that we apply an asynchronous transmission channel to offloading engine, the modified graphic offloading engine includes following representative performance degradation factors due to process remote API call as usual.

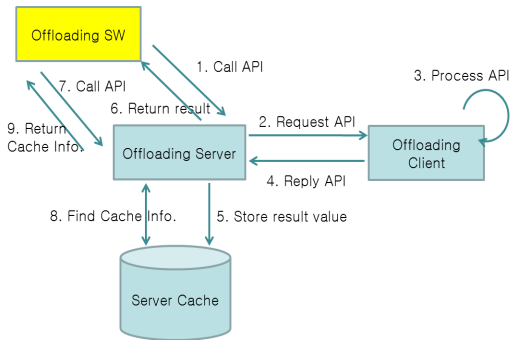
- The delay of message transfer between the offloading server and client
- The delay of memory value transfer such as a bitmap image processing
- Huge amount of message due to virtualize desktop window(mouse moving, window ordering, etc.)

To solve these problems, we suggest the aplication customized graphic offloading library.

4.1 The improving method of rendering performance

In this section, we introduce three performance improvement schemes through analyzing call patterns of APIs and messages that generated by an offloading application process.

First, the offloading engine processes bi-way function as one-way function. A bi-way function incurs execution delay because it waits for the offloading client's reply value that corresponds to request by server's graphic API call. To solve this problem, when a bi-way function calls, the offloading client engine responds an assumed value by analyzing sequences of offloading application's API call. The assumed response value mustn't incur side effects that cause a malfunction.



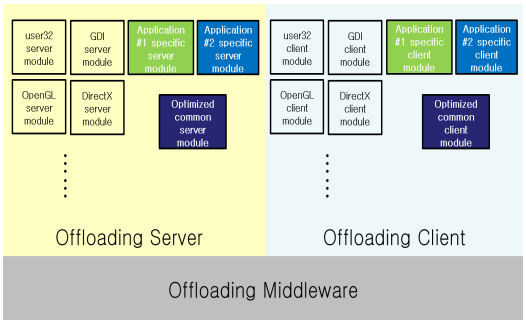
(Figure 7) The sequence of processing remote function call with server cache

Second, the offloading server ignores API and message calls that don't affect application execution to the offloading client. The GUI library, such as MFC, happens a lot of function and message calls. Although some functions and messages, such as mouse moving message, may be ignored, these don't cause any effect on the entire application execution.

Finally, the offloading server doesn't send some frequent query functions to the client and can handle those functions using the server's cache values that are already stored in the server's cache. Figure 7 is the sequence of processing remote function call with server cache.

4.2 The construction of application customized graphic offloading library

As Figure 8, the application customized graphic offloading engine consists of basic offloading modules, optimized common module and application specific modules.



(Figure 8) The construction of application customized graphic offloading library module

The optimized common module is a set of handling routine for common APIs and messages to improve the performance of various applications. These APIs and messages are selected from offloading modules, such as user32, GDI32, OpenGL and DirectX library. The optimized common module is developed by applying three performance improvement schemes mentioned in section 4.1.

The application specific module, that could provide the better performance to specific application, is designed by analyzing API and message call patterns used in that application. Three performance improvement schemes are also applied to the application specific module. The application specific module may be added in case that the rendering performance of the application is important because it is not always essential to serve the graphic offloading service.

4.3 The optimized common module

The optimized common module focuses features to improve performance of rendering 2D graphic and window. To do that, we apply server side caching scheme for three information to the optimized common module. Three information are window Z-order information, window attribute information and device context attributes.

A window Z-order is an ordering of overlapping two-dimensional windows on the Windows OS GUI. Creating, moving, resizing and terminating window and changing window's focus happens many API calls related to the window Z-order for rendering windows on the display. Depending on the application, the number of API call associated with querying the Z-order information may take up to 70 to 80% of the entire number of API call generated by application process. It may cause to decrease rendering performance when running a graphic offloading application. To solve this problem, the optimized common module utilize a portion of the total offloading client's Z-order information previously stored in the offloading server-side cache to handle API calls associated with querying Z-order information. This server-side cache value is updated in case of exceeding a predetermined time or calling an API to change Z-order information.

Window attribute information, that includes the status, size and position of a window control, could be cached on the offloading server because it is frequently queried and used

when an individual window is rendered.

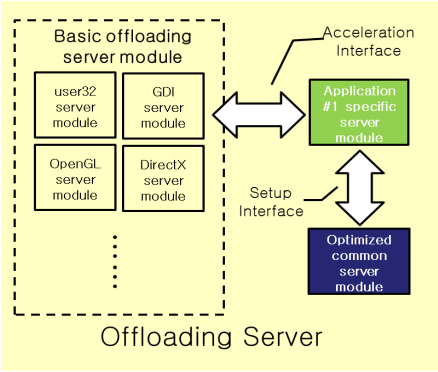
Also, various attributes of device context, that include background color, foreground color, text alignment, text color, map mode, background mode, the current position of drawing line and so on for rendering 2D graphic, could be cached on the offloading server.

4.4 Application specific module

As Figure 9, the application specific module has the interface of basic offloading modules and the optimized common module.

The interface with basic offloading modules is Acceleration interface which includes performance improvement features of APIs and messages selected on basic offloading modules. According to analyze API call patterns generated by a specific offloading application, the selected API may be treated differently than basic offloading API handling routine in order to improve the performance of that application. To do that, server-side application specific modules provide below interface functions which are used by offloading server.

- *typedef BOOL (*LPATTACHCB)(int id, int iCommand, PVOID*, PVOID, LPCSTR);*
- *void RegisterAccelerators(LPATTACHCB lpAttachCb)*



(Figure 9) Two Interface of application specific module

The RegisterAccelerator function initializes server-side application specific module and hooks functions to handle in application specific module. The LPATTACHCB callback function provides feature to substitute function of basic offloading module with it of application specific module. Functions, that are implemented in application specific module, consist of functions applying to three features decribed in section 4.1.

Client-side application specific modules provide below function for interfacing with client-side basic offloading modules.

- *void RegisterAccelerators()*

Likewise, the RegisterAccelerator function initializes client-side application specific module and registers handler functions which are handle API call requested by the application specific server module.

Application specific module provides below function to handle Windows OS messages for improving performance. Likewise, basic offloading modules access these functions through Acceleration interface.

- *DoBeforeAddMessage(…):* If this function returns true value, message don't be transmitted to offloading client module and is processed by using predicted value or stored value on server cache.
- *DoAfterAddMessage(…):* It transmits additional information to client module or processes additional work.
- *DoBeforeResolveMessage(…):* It analyzes information added by DoBeforeAddMessage function.
- *DoBeforeResolveMessage(…):* It analyzes information added by DoAfterAddMessage function. If it needs, it can invalidate value on the server cache.

The interface with optimized common module is Setup interface which includes preference settings of performance improvement features used by the optimized common module.

One feature is that selects applicability of performance improvement features - Z-order caching, window information caching and device context caching - on the optimized common module of that application. The applicability of those features is stored in the INI file. When offloading server engine is started, it loads information in the INI file and then it will apply selected features to offloading application service. Figure 10 is a INI file example for Setup interface.

```
...
[accelerator]
zordercache=Y
wininfocache=N
dccache=Y
...
```

(Figure 10) INI file example for Setup interface

The other feature is the setting of size values related on performance improvement features - Z-order caching size, window information caching size and device context caching size - on the optimized common module of that application. Likewise, those values are stored in the INI file. Experimentally, if window information caching size and device context caching size is sufficiently set, it causes to improve the performance of that application offloading service.

5. Performance Analysis

In this paper, to improve the performance of graphic offloading engine, we applied an asynchronous transmission channel scheme to the graphic offloading engine. Also, we added optimized common module and application specific module to the basic graphic offloading server and client modules. We performed comparison tests divided two steps in order to demonstrate the performance improvement effect of our proposed features. First, we measured FPS(Frame Per Second) of our offloading software service before and after applying an asynchronous transmission channel to the offloading engine. Next, we also measured FPS after applying all performance improvement features.

Our test environment consists of two personal computers based on Windows 7. The one is used as an offloading server and the other is used as an offloading client. The server consists of Intel XeonE5-2440 2.4 GHz CPU, 32G RAM. The client consists of Intel(R) Core(TM) i3-3220 3.3GHz CPU and NVIDIA GeForce GTX650 graphic card. The network environment is a gigabit LAN. Because the graphic rendering job of our solution is processed by an offloading client, the graphic card's specification of an offloading server is not important. To measure the performance, we use two 3D graphic softwares - Archispace LT and GPUBoids. Archispace LT is a commercial architectural design tool. GPUBoids is a DirectX 10 sample program. The estimated result is follow Table 1.

(Table 1) Performance comparison test result

	Archispace LT	GPUBoids
Basic offloading engine	6~7	30~33
Applying ATC[1)]	11~12	46~48
Applying ATC[1)] and ACL[2)]	14~15	50~53

1) Asynchronous Transmission Channel
2) Application Customized Library

The effect of an aynchronous channel is that it reduces wait time of an offloading client. Likewise, when an offloading client processes bi-way function, an offloading server can transfer other data through an asynchronous transmission channel without a reponse delay. Also, the effect of applying an application customized library is that it reduces an entire amount of transferred data between server and client. As shown in Table 1, after applying all improvement features to offloading engine, it led to the performance improvement of approximately 65~120% for that application's FPS.

6. Conclusions

In this Paper, we introduced an efficient on-line software service using an application customized graphic offloading module library. Also, we showed a client-based remote 3D rendering concept using graphic offloading for server-based on-line software service and window message virtualization to provide a virtualized desktop window which the 2D window created by offloading application software is processed independently in the server's window management system. To improve the rendering performance of graphic offloading software service, we added an asynchronous transmission channel and thread to the offloading server/client engine. Also, we added optimized common module and application specific module to the graphic offloading engine. To do that, we introduced how to implement the application specific module

using analyzing patterns of graphic related APIs and messages that are generated by executed software. Also, we showed how to design the optimized common module using server side information caching, such as window Z-order caching, window information caching and device context caching.

By applying our proposed performance improvrment features, we measured a rendering performance of the offloading software service. It led to the performance improvement of approximately 65~120% for that application's FPS. The performance improvement analysis through applying and testing various applications and the compression transmission of data between server and client will be left for future work.

Reference

[1] Won Hyuk Choi, Su Min Jang, Ji Hoon Choi, Won Young Kim, "A Design of SW Service Based on Graphic Offloading Computing Using Client's Desktop Window Virtualization", ICONI 2012, pp.221- 224, 2012.

[2] http://www.citrix.com/products/xendesktop/overview.html

[3] http://www.citrix.com/xendesktop

[4] http://technet.microsoft.com/en-us/library/hh831447.aspx

[5] Aymen Abdullah Alsaffar, Song Biao, Mohammad Mehedi Hassan, Eui-Nam Huh, "A Framework of N-Screen Session Manager based N-Screen Service using Cloud Computing in Thin-Client Environment", Journal of Internet Computing and Services, v.13, no.2, pp.21-32, Apr. 2012.
http://dx.doi.org/10.7472/jksii.2012.13.2.21

[6] Vishakha Gupta, Ada Gavrilovska, Karsten Schwan, Harshvardhan Kharche, Niraj Tolia, Vanish Talwar, Parthasarathy Ranganathan, "GViM: GPU-accelerated virtual machines", HPCVirt 09, pp.17-24, 2009.
http://dx.doi.org/10.1145/1519138.1519141

[7] Huerta-Canepa G., Dongman Lee, "An Adaptable Application Offloading Scheme Based on Application Behavior", Advanced Information Networking and Applications Workshops, pp.387-392, 2008.
http://dx.doi.org/10.1109/WAINA.2008.148

[8] Su Min Jang, Won Hyuk Choi, Won Young Kim, "Client Rendering Method for Desktop Virtualization Services", ETRI Journal, v.35, no. 2, pp. 348-351, 2013.
http://dx.doi.org/10.4218/etrij.13.0212.0213

[9] Won Hyuk Choi, Won Young Kim, "An Implementation of Graphic Offloading Computing using GPU Virtualization based on API Remoting on a Server-based Software Service", *Journal of Internet Computing and Services*, v.12, no.6, pp.53-62, Dec. 2011.
http://www.jksii.or.kr/upload/1/889_1.pdf

[10] Dusung Back, Kihyun Pyun, "A Protection Technique for Kernel Functions under the Windows Operating System", *Journal of Internet Computing and Services*, v.15, no.5, pp.133-139, Dec. 2014.
http://dx.doi.org/ 10.7472/jksii.2014.15.5.133

[11] Moonyoung Chung, Jihoon Choi, Won-Hyuk Choi, Won-Young Kim, "An Efficient Thread Management for the Client-side Graphic Rendering on a Server-based Software Service", *Proceedings of the Korea Information Processing Society Conference*, v.19, no.2, pp.209-211, Nov. 2011

◐ 저 자 소 개 ◑

최 원 혁 (WonHyuk Choi)
1999년 경북대학교 컴퓨터공학과 졸업(학사)
2001년 경북대학교 컴퓨터공학과 졸업(석사)
2001년~현재 한국전자통신연구원 재직
관심분야: SW 가상화, SW 서비스

김 원 영 (Won-Young Kim)
1989년 이화여자대학교 전산학과 졸업(학사)
1991년 KAIST 전산학과 졸업(석사)
1998년 KAIST 전산학과 졸업(박사)
1999년 ~ 현재 한국전자통신연구원 고성능컴퓨팅SW연구실 실장
관심 분야: SW 서비스