Chapter XII
Device Driver Based Computer in Broadband Age

Yoshiyasu Takefuji
Keio University, Japan

Koichiro Shoji
SciencePark Corporation, Japan

Takashi Nozaki
SciencePark Corporation, Japan

ABSTRACT

In this chapter, we present a device-driver-based computer that realizes the reduction of mode (domain or vertical) switching overheads between user and kernel mode with innovative attributes, including shared keyboards and mice, access-controlled files, and timed files. Experimented results show that old personal computers can revive again with the proposed Driverware technology. The proposed Driverware can improve the CPU resource utilization by three times.

BACKGROUND

On April 19, 1965, Moore predicted that the number of transistors per integrated circuit would double every 18 months (Moore, 2001). Ruley showed that the speed of CPU has been double every 18 months (Ruley, 2001). Although the recent progress of semiconductor technology has been providing us over 2 GHz (gigahertz) CPU devices, you may not be able to obtain such dramatic improvements with your personal computer because you do not sense the high-speed CPU devices. In existing popular operating systems, including Microsoft Windows, Macintosh, Linux, and FreeBSD, your user software programs do not maximize the performance of such a high-speed CPU and other resources. In this chapter, a device driver-based computer is proposed, where the “Driverware” software program plays a key role. “Driverware” allows you to dramatically improve user software programs in your personal computer without replacing the current operating system. The technical aspects of the root of all evil that waste your CPU resources in user software programs are caused by device driver programs. A device driver is a software program to control a hardware component or peripheral device of a computer, such as a hard disk, a CD/DVD player, a network device, a mouse, keyboard, a video card, and so on. Driverware provides the current operating system a thin layer closest to the hardware layer, where communications and controls among device drivers can be established. Overheads involved in device drivers are significantly reduced by the driver-to-driver communications and controls. When playing a CD/DVD audio/video player or a network video stream on your GHz CPU-based personal computer in the broadband network, you may still face frame drop or abrupt audio skips if you do Web browsing simultaneously. Four device drivers, including a network device driver, a video device driver, audio device driver, and a display device driver, are involved in this network video stream processing. If the priority of network device driver is lower than that of the other drivers, you may face the frame drop. If four device drivers are properly balanced and their priorities are taken care of, then you will be satisfied with the current network video stream without using a special hardware such as a network video stream receiver. Driverware allows device drivers to communicate each other, and to efficiently control computer resources in order to reduce overheads involved in device drivers. Frame drops and abrupt audio skips are minimized by the proposed Driverware. Until the advent of Driverware, there has been no general-purpose device-driver-based computer. The key element of Driverware is based on distributed delegation of authority, while conventional user programs are based on the centralized control. In user programs, overheads are caused by unnecessary thread* switching in a process, context (horizontal) switching between processes, and mode (domain or vertical) switching between kernel and user mode. The proposed Driverware contributes to reduce the mode-switching overheads. We have measured the mode-switching overheads involved in device drivers for the first time in the world. Based on the measured result, your old personal computers in the garage can revive again.

The framework for minimizing overheads of vertical and horizontal switching was proposed in 1994 (Inohara & Masuda, 1994). Horizontal switching problems in user mode were discussed in Borg (2001). Ousterhout has measured the mode and context-switching overheads in RISC/CISC machines under the Unix operating system in 1990 (Ousterhout, 1990). In 1996, Lai and his colleagues measured the mode and context-switching overheads in Intel x86-based machines under Linux/FreeBSD/Solaris operating systems, respectively (Lai & Baker, 1996). Based on the existing results, reducing the context-switching overheads plays a key role in overall system performance, where the context switching overhead is larger than the mode-switching overhead by 10 times. Ultimately optimized context-switching operating system, RT-Linux has been introduced (Ramamritham & Stankovic, 1994; Zhou & Petrov, 2006). Based on the context-switching overhead reduction, TUX□ Threaded Linux Web server (Bar, 2000)), and other Web servers have shown the high performance, respectively (Quynh & Takefuji, 2006).

In the real world, the majority of users use Microsoft operating systems for personal use (Sun, Lin, & Wu, 2006). Based on our study, the mode-switching overhead reduction is more important than the context-switching overhead reduction under Microsoft operating systems. Figure 1 shows the table of the mode-switching and context-switching overheads, where more than 2 months were used for switching overhead measurement. In Windows 2000 operating system (Microsoft, 2006), 7.4 microseconds are required per the mode-switching overhead with a Celeron

800MHz CPU. For mode-switching measurement, 10 million integer additive operations were experimented under Windows 2000, Linux, and FreeBSD respectively.

The relationship between waste resource rate vs. ftp-network transfer speed, as shown in Figure 2, indicates how much your CPU resources are not used for your applications. Figure 2 shows that,

Figure 1. Mode-switching and context-switching overheads

Switching Time: unit: microsecond

	Mode Switching		Context Switching	
Windows 2000 (Professional)	7.4	Celeron 800MHz	7.10	*7 Pentium III 500MHz
Linux (RedHat 7.2)	0.25	Celeron 800MHz	1.56	* Celeron 800MHz
FreeBSD (4.5R)	0.48	Celeron 800MHz	2.40	* Celeron 800MHz

*7 John D. Regehr, * measured by lmbench 2.0

Figure 2. Waste resource rate vs. transfer speed

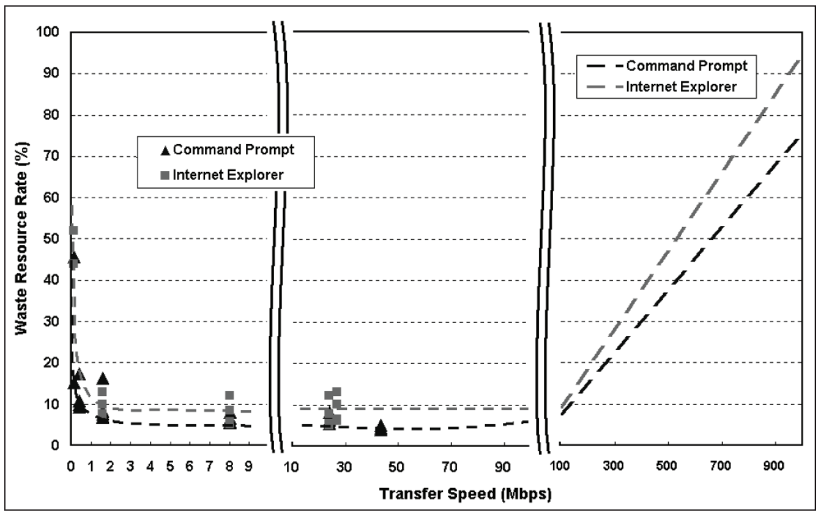


Figure 3. File transfer speed comparison

Load	Program	ftp processor time (%)	Transfer speed (MB/sec)	Estimated transfer speed with cpu full processing (MB/sec)
on	DriverWare	21.9	5.629	25.650
	ftp.exe	37.0	3.173	8.566
off	DriverWare	21.3	5.916	27.797
	ftp.exe	89.5	8.261	9.230

**A thread is one part of a larger program that can be executed independent of the whole (ComputerUser, 2006).*

from 0 Mbps to 1 Mbps, the waste resource rate increases along with less transfer speed. From 1 Mbps to 100 Mbps it is almost flat and from 100 Mbps to 1 Gbps it increases with transfer speed. For example, you will waste 92 % of CPU resources in 1 gigabit-per-second transfer speed using the Internet Explorer. In other words, only 8% can be used for your applications. We assume that 100% CPU resources are used only for ftp process in our measurement. Figure 3 shows file transfer speed comparison between the normal ftp program and the Driverware ftp program. It indicates that CPU utilization is improved by about three times. In this chapter, the mode-switching overhead reduction is detailed.

Figure 4. Processing time flow between user and kernel mode

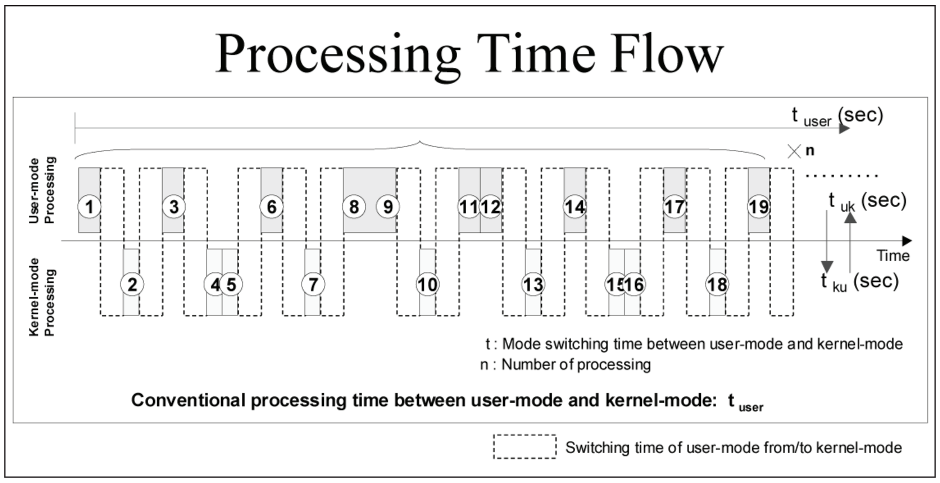
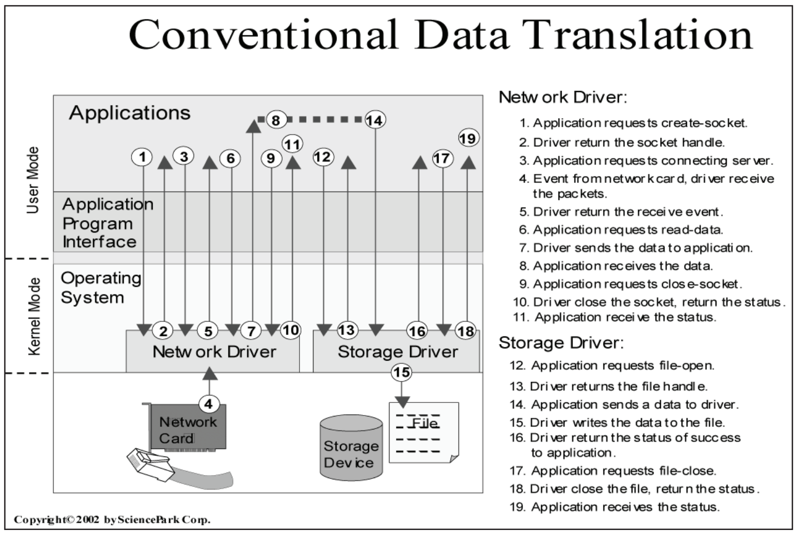

Processing Time Flow
t user (sec)
× n
t uk (sec)
t ku (sec)
Time
User-mode Processing
Kernel-mode Processing
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
t : Mode switching time between user-mode and kernel-mode
n : Number of processing
Conventional processing time between user-mode and kernel-mode: t user
Switching time of user-mode from/to kernel-mode

Figure 5. Sequence flow between user mode and kernel mode



Network Driver:

1. Application requests create-socket.
2. Driver return the socket handle.
3. Application requests connecting server.
4. Event from network card, driver receive the packets.
5. Driver return the receive event.
6. Application requests read-data.
7. Driver sends the data to application.
8. Application receives the data.
9. Application requests close-socket.
10. Driver close the socket, return the status.
11. Application receive the status.

Storage Driver:

12. Application requests file-open.
13. Driver returns the file handle.
14. Application sends a data to driver.
15. Driver writes the data to the file.
16. Driver return the status of success to application.
17. Application requests file-close.
18. Driver close the file, return the status.
19. Application receives the status.

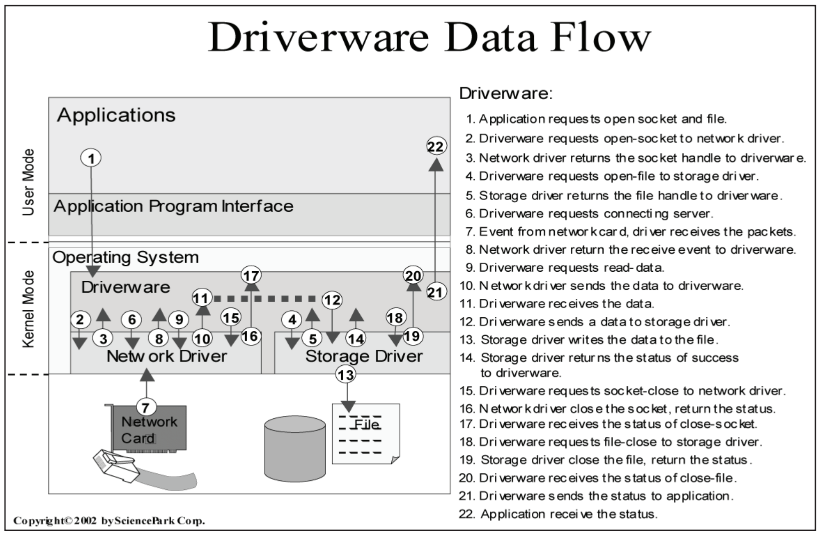
What is Going on in your Centralized Control User Programs?

In your personal computer, the system is composed of hardware (keyboard, mouse, hard disk, CPU board, network card, LCD display, and so on), operating system, and user application programs. Unless you buy a special hardware device, conventional user application programs based on the centralized control generate mode-switching overheads between user and kernel mode. For example, in the network device driver, when the user application program requests open socket, the mode switching from user to kernel mode is needed to complete the task. Centralized-control user programs inherently have a disadvantage of mode-switching overheads between user and kernel mode. All operations involved in network and storage device driver are shown in Figure 4 and Figure 5. Figure 4 shows that the total mode-switching overheads, surrounded by broken lines, are equivalent to the total waste of CPU resources. Eleven operations are involved in network device driver and eight operations in storage device driver. Fourteen operations (O1, O2, O3, O5, O6, O7, O9, O10, O12, O13, O14, O16, O17, O18) out of nineteen operations must cross the user-kernel boundary, which causes fundamental mode (domain or vertical) switching overheads. When you do browsing over the Internet, you would like to download some files. The operation sequence flow is as follows:

When clicking the downloadable file using a Web browser (O1), the browser requests data acquisition to the network device driver after setting Web server address and port number (O1 and O2). Web server establishes the connection and sends the requested file. Network card returns the data-receive event to the network driver (O4). The network driver returns the receive event to the Web browser (O5). Web browser receives the data (O6, O7, and O8). Web browser displays the dialog window to specify the directory for saving the downloadable file and user must input the file name (O12 and O13). Download progress bar is displayed and the file downloading will be completed (O14-O19).

As shown in Figure 4 and Figure 5, a sequence flow is described where 14 operations must transit from user mode to kernel mode or vice versa. The

Figure 6. Driverware solution to operations as shown in Figure 5



Driverware:

1. Application requests open socket and file.
2. Driverware requests open-socket to network driver.
3. Network driver returns the socket handle to driverware.
4. Driverware requests open-file to storage driver.
5. Storage driver returns the file handle to driverware.
6. Driverware requests connecting server.
7. Event from networkcard, driver receives the packets.
8. Network driver return the receive event to driverware.
9. Driverware requests read-data.
10. Networkdriver sends the data to driverware.
11. Driverware receives the data.
12. Driverware sends a data to storage driver.
13. Storage driver writes the data to the file.
14. Storage driver returns the status of success to driverware.
15. Driverware requests socket-close to network driver.
16. Networkdriver close the socket, return the status.
17. Driverware receives the status of close-socket.
18. Driverware requests file-close to storage driver.
19. Storage driver close the file, return the status.
20. Driverware receives the status of close-file.
21. Driverware sends the status to application.
22. Application receive the status.

mode-switching time tuk from user mode to kernel mode and tku from kernel mode to user mode are consumed as waste of CPU resources. The total computation time of the typical user application program user is composed of user processing time, kernel processing time, accumulated switching time from user mode to kernel mode, and accumulated switching time from kernel mode to user mode.

Device Driver-Based Computing

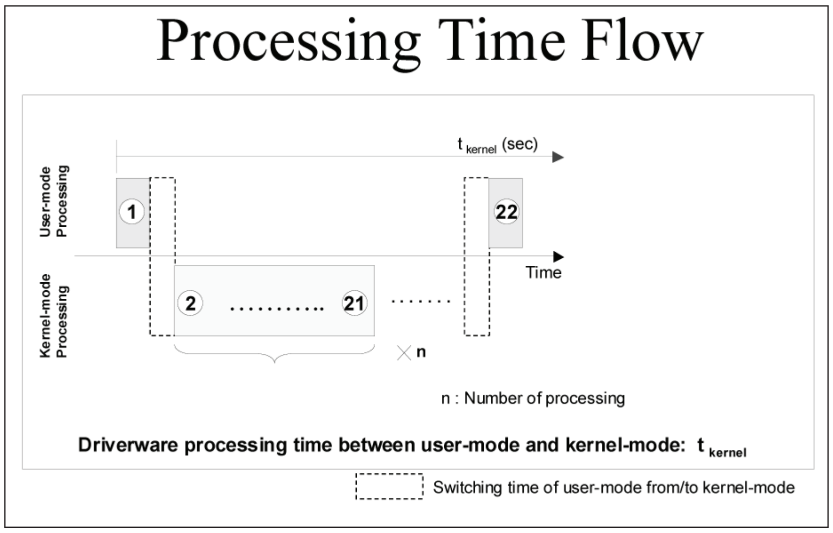
Device driver-based computing is based on Driverware where the operations of driver-driver communications and controls are achieved. Figure 6 shows the device-driver-based computing where most mode-switching overheads are reduced. Fourteen operations that cross the user-kernel boundary are reduced into two operations in Driverware. The overhead reduction is achieved by enabling driver-driver communications and controls in the kernel mode. Until now, there has been no general purpose distributed-control solution by reducing mode, vertical, or domain-switching overheads between user and kernel mode. Vertical or mode-switching overheads can be reduced by the distributed delegation of authority.

Between several device drivers, security controls can be established within Driverware where several new attributes are generated as a by-product. A file in Driverware is called access-controlled file. For example, as soon as a user without access permission touches or reads the access controlled file, immediately the user's keyboard will be locked. Another new attributed file is called timed file in Driverware, like a tape in the "mission impossible" movie where it will disappear within a certain specified period of time without user's intention. Driverware allows every device driver to have a state sense operation and an action operation against any computing resources.

CONCLUSION

Device driver-based computer is proposed in this chapter, where mode (domain or vertical)

Figure 7. A processing time flow by Driverware solution


Processing Time Flow
t kernel (sec)
User-mode Processing
Kernel-mode Processing
1
22
Time
2
21
× n
n : Number of processing
Driverware processing time between user-mode and kernel-mode: t kernel
Switching time of user-mode from/to kernel-mode

switching overheads between user and kernel mode are dramatically reduced, and every computer resource will have new attributes, including shared keyboards and mice, access controlled files, and timed files like mission impossible files tape. Experimented results show that old personal computers can revive again with the proposed Driverware technology. The proposed Driverware can improve the CPU resource utilization by three times.

ACKNOWLEDGMENT

We thank Tomoyuki Kawade, Hideaki Miura, and Jianping Wei for their many helpful comments and experiments of device driver benchmarking. Part of this research is supported by Department of the Air Force through a special contract (Takefuji, 2003; Takefuji, 2005).

REFERENCES

Bar, M. (2000). Kernel korner. *The Linux Process Model. Linux Journal, 71*,(24).

Borg, A. (2001). *Avoiding blocking system calls in a user-level thread scheduler for shared memory multiprocessors*. Dissertation of Univ. Malta, June 2001.

ComputerUser. (2006). Thread. *ComputerUser high-tech dictionary.* Retrieved December 30, 2006, from http://www.computeruser.com/resources/dictionary/definition.html?lookup=8392

Inohara, S., & Masuda, T. (1994). A framework for minimizing thread management overhead based on asynchronous cooperation between user and kernel schedulers. *TR94-02, University of Tokyo*.

Lai, K., & Baker, M. (1996). A performance comparison of UNIX operating systems on the Pentium. *Proceedings of the USENIX 1996 Annual Technical Conference.*

Microsoft. (2006). Windows 2000. *Microsoft Windows 2000.* Retrieved December 30, 2006, from http://www.microsoft.com/windows2000/default.mspx

Moore, G. E. (2001). Cramming more components onto integrated circuits. *Electronics, 38*(8).

Ousterhout, J. K. (1990). Why aren't operation systems getting faster as fast as hardware? *USENIX Summer Conference, June 11-15, 1990.*

Ramamritham, K., & Stankovic, J. (1994). Scheduling algorithms and operating systems support for real-time systems. *Proceedings of IEEE, 8*(21), 55-67.

Regehr, J. D. (2001). *Using hierarchical scheduling to support soft real-time applications in general-purpose operating systems*. Dissertation of Univ. of Virginia, 2001.

Ruley, J. D. (2001). The future of Moore's law, Part 1. Retrieved June 25, 2001, from http://www.byte.com

Sun, H. M., Lin, Y. H., & Wu, M. F. (2006). API monitoring system for defeating worms and exploits in MS-Windows system. *Proceedings of Information Security and Privacy, Lecture Notes in Computer Science, 4058*, 159-170.

Takefuji, Y. (2003). Technical report of DRIVERWARE IMMUNE. *US Air Force Office of Scientific Research with Grant Number AOARD 03-4049*

Takefuji, Y. (2005). Nullification of unknown malicious code execution with buffer overflows. Driverware IMMUNE – Final Report. Technical report of DRIVERWARE IMMUNE, *US Air Force Office of Scientific Research with Grant Number AOARD 03-4049.*

Quynh, A. N., & Takefuji, Y. (2006). Towards an invisible honeypot monitoring system. *Proceed-*

ings of Information Security and Privacy, Lecture Notes in Computer Science, 4058, 111-122.

Zhou, X., & Petrov, P. (20 control applications. In *Proceedings of the 43rd annual conference on Design automation* (pp.352-257).