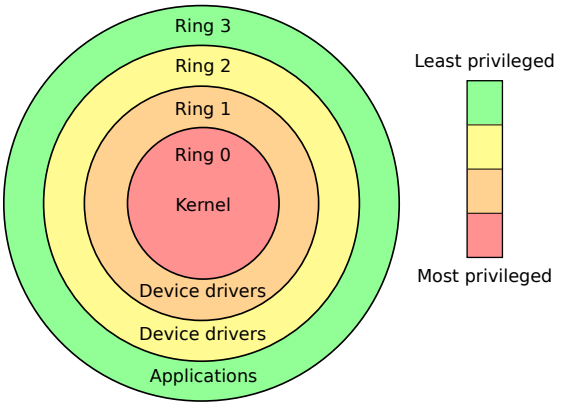
Protection ring

For other uses, see Ring.
"Ring 0" redirects here. For the Japanese horror prequel, see Ring 0: Birthday.
"Ring 3" redirects here. For the road in Finland, see Ring III. For the road in Norway, see Ring 3 (Oslo).

In computer science, **hierarchical protection domains**,[1][2] often called **protection rings**, are mechanisms to protect data and functionality from faults (by improving fault tolerance) and malicious behaviour (by providing computer security). This approach is diametrically opposite to that of capability-based security.



Privilege rings for the x86 available in protected mode

Computer operating systems provide different levels of access to resources. A protection ring is one of two or more hierarchical *levels* or *layers* of privilege within the architecture of a computer system. This is generally hardware-enforced by some CPU architectures that provide different CPU modes at the hardware or microcode level. Rings are arranged in a hierarchy from most privileged (most trusted, usually numbered zero) to least privileged (least trusted, usually with the highest ring number). On most operating systems, Ring 0 is the level with the most privileges and interacts most directly with the physical hardware such as the CPU and memory.

Special gates between rings are provided to allow an outer ring to access an inner ring's resources in a predefined manner, as opposed to allowing arbitrary usage. Correctly gating access between rings can improve security by preventing programs from one ring or privilege level from misusing resources intended for programs in another. For example, spyware running as a user program in Ring 3 should be prevented from turning on a web camera without informing the user, since hardware access should be a Ring 1 function reserved for device drivers. Programs such as web browsers running in higher numbered rings must request access to the network, a resource restricted to a lower numbered ring.

1 Implementations

Multiple rings of protection were among the most revolutionary concepts introduced by the Multics operating system, a highly secure predecessor of today's Unix family of operating systems. The GE 645 did not support rings in hardware, so Multics supported them by trapping ring transitions in software;[3] its successor, the Honeywell 6180, implemented them in hardware, with support for eight rings.[4] However, most general-purpose systems use only two rings, even if the hardware they run on provides more CPU modes than that. For example, Windows 7 and Windows Server 2008 (and their predecessors) use only two rings, with ring 0 corresponding to kernel mode and ring 3 to user mode,[5] because earlier versions of Windows ran on processors that supported only two protection levels.[6]

Many modern CPU architectures (including the popular Intel x86 architecture) include some form of ring protection, although the Windows NT operating system, like Unix, does not fully utilize this feature. OS/2 did to some extent, as it used three rings:[7] ring 0 for kernel code and device drivers, ring 2 for privileged code (user programs with I/O access permissions), and ring 3 for unprivileged code (nearly all user programs). Under DOS, the kernel, drivers and applications typically run on ring 3 (however, this is exclusive to the case where protected-mode drivers and/or DOS extenders are used; as a real-mode OS, the system runs with effectively no protection), whereas 386 memory managers such as EMM386 run at ring 0. In addition to this, DR-DOS' EMM386 3.xx can optionally run some modules (such as DPMS) on ring 1 instead. OpenVMS uses four modes called (in order of decreasing privileges) Kernel, Executive, Supervisor and User.

A renewed interest in this design structure came with the proliferation of the Xen VMM software, ongoing discussion on monolithic vs. micro-kernels (particularly in Usenet newsgroups and Web forums), Microsoft's *Ring-1* design structure as part of their NGSCB initiative, and hypervisors embedded in firmware such as Intel VT-x (formerly Vanderpool).

The original Multics system had eight rings, but many modern systems have fewer. The hardware remains aware

of the current ring of the executing instruction thread at all times, with the help of a special machine register. In some systems, areas of virtual memory are instead assigned ring numbers in hardware. One example is the Data General Eclipse MV/8000, in which the top three bits of the program counter (PC) served as the ring register. Thus code executing with the virtual PC set to 0xE200000, for example, would automatically be in ring 7, and calling a subroutine in a different section of memory would automatically cause a ring transfer.

The hardware severely restricts the ways in which control can be passed from one ring to another, and also enforces restrictions on the types of memory access that can be performed across rings. Using x86 as an example, there is a special *gate* structure which is referenced by the *call* instruction that transfers control in a secure way towards predefined entry points in lower-level (more trusted) rings; this functions as a supervisor call in many operating systems that use the ring architecture. The hardware restrictions are designed to limit opportunities for accidental or malicious breaches of security. In addition, the most privileged ring may be given special capabilities, (such as real memory addressing that bypasses the virtual memory hardware).

Ring protection can be combined with processor modes (master/kernel/privileged/supervisor mode versus slave/unprivileged/user mode) in some systems. Operating systems running on hardware supporting both may use both forms of protection or only one.

Effective use of ring architecture requires close cooperation between hardware and the operating system. Operating systems designed to work on multiple hardware platforms may make only limited use of rings if they are not present on every supported platform. Often the security model is simplified to “kernel” and “user” even if hardware provides finer granularity through rings.

2 Supervisor mode

In computer terms, *supervisor mode* is a hardware-mediated flag which can be changed by code running in system-level software. System-level tasks or threads will have this flag set while they are running, whereas userspace applications will not. This flag determines whether it would be possible to execute machine code operations such as modifying registers for various descriptor tables, or performing operations such as disabling interrupts. The idea of having two different modes to operate in comes from "with more control comes more responsibility" – a program in supervisor mode is trusted never to fail, since a failure may cause the whole computer system to crash.

Supervisor mode is “an execution mode on some processors which enables execution of all instructions, including privileged instructions. It may also give access to a different address space, to memory management hardware and to other peripherals. This is the mode in which the operating system usually runs.” [8]

In a monolithic kernel, the operating system runs in supervisor mode and the applications run in user mode. Other types of operating systems, like those with an exokernel or microkernel, do not necessarily share this behavior.

Some examples from the PC world:

- Linux and Windows are two operating systems that use supervisor/user mode. To perform specialized functions, user mode code must perform a system call into supervisor mode or even to the kernel space where trusted code of the operating system will perform the needed task and return the execution back to the userspace.
- DOS (for as long as no 386 memory manager such as EMM386 is loaded), as well as other simple operating systems, and many embedded devices run in supervisor mode permanently, meaning that drivers can be written directly as user programs.

Most processors have at least two different modes. The x86-processors have four different modes divided into four different rings. Programs that run in Ring 0 can do *anything* with the system, and code that runs in Ring 3 should be able to fail at any time without impact to the rest of the computer system. Ring 1 and Ring 2 are rarely used, but could be configured with different levels of access.

In most existing systems, switching from user mode to kernel mode has an associated high cost in performance. It has been measured, on the basic request getpid, to cost 1000-1500 cycles on most machines. Of these just around 100 are for the actual switch (70 from user to kernel space, and 40 back), the rest is “kernel overhead”.[9][10] In the L3 microkernel the minimization of this overhead reduced the overall cost to around 150 cycles.[9]

Maurice Wilkes wrote:[11]

> ... it eventually became clear that the hierarchical protection that rings provided did not closely match the requirements of the system programmer and gave little or no improvement on the simple system of having two modes only. Rings of protection lent themselves to efficient implementation in hardware, but there was little else to be said for them. [...] The attractiveness of fine-grained protection remained, even after it was seen that rings of protection did not provide the answer... This again proved a blind alley...

To gain performance and determinism, some systems place functions that would likely be viewed as application

logic, rather than as device drivers, in kernel mode; security applications (access control, firewalls, etc.) and operating system monitors are cited as examples. At least one embedded database management system, *eXtreme*DB Kernel Mode, has been developed specifically for kernel mode deployment, to provide a local database for kernel-based application functions, and to eliminate the context switches that would otherwise occur when kernel functions interact with a database system running in user mode.[12]

Functions are also sometimes moved across rings in the other direction. The Linux kernel, for instance, injects a vDSO section in processes which contains functions that would normally require a system call, i.e. a ring transition. But instead of doing a syscall, these functions use static data provided by the kernel which prevents the need for a ring transition which is more lightweight than a syscall. The function gettimeofday can be provided this way.

3 Hypervisor mode

Recent CPUs from Intel and AMD offer x86 virtualization instructions for a hypervisor to control Ring 0 hardware access. Although they are mutually incompatible, both Intel VT-x (codenamed "Vanderpool") and AMD-V (codenamed "Pacifica") create a new "Ring −1" so that a guest operating system can run Ring 0 operations natively without affecting other guests or the host OS.

> "To assist virtualization, VT and Pacifica insert a new privilege level beneath Ring 0. Both add nine new machine code instructions that only work at "Ring −1," intended to be used by the hypervisor. "[13]

4 Hardware features usage

Many CPU hardware architectures provide far more flexibility than is exploited by the operating systems that they normally run. Proper use of complex CPU modes requires very close cooperation between the operating system and the CPU, and thus tends to tie the OS to the CPU architecture. When the OS and the CPU are specifically designed for each other, this is not a problem (although some hardware features may still be left unexploited), but when the OS is designed to be compatible with multiple, different CPU architectures, a large part of the CPU mode features may be ignored by the OS. For example, the reason Windows uses only two levels (ring 0 and ring 3) is that some hardware architectures that were supported in the past (such as PowerPC or MIPS) implemented only two privilege levels.[5]

Multics was an operating system designed specifically for a special CPU architecture (which in turn was designed specifically for Multics), and it took full advantage of the CPU modes available to it. However, it was an exception to the rule. Today, this high degree of interoperation between the OS and the hardware is not often cost-effective, despite the potential advantages for security and stability.

Ultimately, the purpose of distinct operating modes for the CPU is to provide hardware protection against accidental or deliberate corruption of the system environment (and corresponding breaches of system security) by software. Only "trusted" portions of system software are allowed to execute in the unrestricted environment of kernel mode, and then, in paradigmatic designs, only when absolutely necessary. All other software executes in one or more user modes. If a processor generates a fault or exception condition in a user mode, in most cases system stability is unaffected; if a processor generates a fault or exception condition in kernel mode, most operating systems will halt the system with an unrecoverable error. When a hierarchy of modes exists (ring-based security), faults and exceptions at one privilege level may destabilize only the higher-numbered privilege levels. Thus, a fault in Ring 0 (the kernel mode with the highest privilege) will crash the entire system, but a fault in Ring 2 will only affect rings 3 and beyond and Ring 2 itself, at most.

Transitions between modes are at the discretion of the executing thread when the transition is from a level of high privilege to one of low privilege (as from kernel to user modes), but transitions from lower to higher levels of privilege can take place only through secure, hardware-controlled "gates" that are traversed by executing special instructions or when external interrupts are received.

Microkernel operating systems attempt to minimize the amount of code running in privileged mode, for purposes of security and elegance, but ultimately sacrificing performance.

5 See also

- System call
- Protected mode (x86-compatible CPUs of the 80286 series or later)
- I/O Privilege Level (IOPL on x86 CPUs)
- Current Privilege Level (CPL0, CPL1, CPL2, CPL3 on x86 CPUs)
- System Management Mode (SMM) — sometimes called "ring −2"

6 References

[1] Paul A. Karger, Andrew J. Herbert, *An Augmented Capability Architecture to Support Lattice Security and Trace-*

ability of Access, sp, p. 2, 1984 IEEE Symposium on Security and Privacy, 1984

[2] Walter Binder, *Design and Implementation of the J-SEAL2 Mobile Agent Kernel*, saint, p. 35, 2001 Symposium on Applications and the Internet (SAINT'01), 2001

[3] "A Hardware Architecture for Implementing Protection Rings". Retrieved 27 September 2012.

[4] "Multics Glossary - ring". Retrieved 27 September 2012.

[5] Russinovich, Mark E.; David A. Solomon (2005). *Microsoft Windows Internals* (4 ed.). Microsoft Press. p. 16. ISBN 978-0-7356-1917-3.

[6] Russinovich, Mark (2012). *Windows Internals Part 1. 6th Ed*. Redmond, Washington: Microsoft Press. p. 17. ISBN 978-0-7356-4873-9. The reason Windows uses only two levels is that some hardware architectures that were supported in the past (such as Compaq Alpha and Silicon Graphics MIPS) implemented only two privilege levels.

[7] Presentation Device Driver Reference for OS/2 - 5. Introduction to OS/2 Presentation Drivers

[8] FOLDOC supervisor mode

[9] Jochen Liedtke. *On µ-Kernel Construction*, *Proc. 15th ACM Symposium on Operating System Principles (SOSP)*, December 1995

[10] Ousterhout, J. K. 1990. *Why aren't operating systems getting faster as fast as hardware?* In Usenix Summer Conference, Anaheim, CA, pp. 247-256.

[11] Maurice Wilkes *Operating systems in a changing world* ACM SIGOPS Operating Systems Review. Volume 28 , Issue 2 (April 1994). pp. 9 - 21 ISSN 0163-5980 quote from.

[12] Gorine, Andrei and Krivolapov, Alexander. "Kernel Mode Databases: A DBMS Technology For High-Performance Applications", *Dr. Dobb's Journal*, May 2008.

[13] Dornan, Andy (1 November 2005). "Intel VT vs. AMD Pacifica". CMP. Archived from the original on 2013-05-30. Retrieved 11 November 2012.

7 Further reading

- David T. Rogers: A FRAMEWORK FOR DYNAMIC SUBVERSION Thesis, June 2003 (pdf)
- Glossary of Multics acronyms and terms: Ring
- William J. Caelli: Relearning "Trusted Systems" in an Age of NIIP: Lessons from the Past for the Future. 2002 (pdf)
- Haruna R. Isa, William R. Shockley, Cynthia E. Irvine: A Multi-threading Architecture for Multilevel Secure Transaction Processing 1999 (pdf)
- Ivan Kelly: Report Porting MINIX to Xen 2006
- Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield: Xen and the Art of Virtualization 2003 (pdf)
- Marcus Peinado, Yuqun Chen, Paul England, and John Manferdelli: NGSCB: A Trusted Open System (pdf)
- Michael D. Schroeder, Jerome H. Saltzer: A Hardware Architecture for Implementing Protection Rings 1972
- Intel Architecture Software Developer's Manual Volume 3: System Programming (Order Number 243192) Chapter 4 Protection; section 4.5 Privilege levels. (pdf)
- Tzi-cker Chiueh, Ganesh Venkitachalam, Prashant Pradhan: Integrating segmentation and paging protection for safe, efficient and transparent software extensions 1999 Chapter 3: Protection hardware features in Intel X86 architecture; section 3.1 Protection checks. (pdf)
- Takahiro Shinagawa, Kenji Kono, Takashi Masuda: Exploiting Segmentation Mechanism for Protecting Against Malicious Mobile Code 2000 chapter 3 Implementation; section 3.2.1 Ring Protection (pdf)
- Boebert, William Earl and R. Kain. *A Practical Alternative to Hierarchical Integrity Policies*. Proceedings of the 8th National Computer Security Conference, 1985.
- Gorine, Andrei and Krivolapov, Alexander. Kernel Mode Databases: A DBMS technology for high-performance applications, *Dr. Dobb's Journal*, May 2008.

8 Text and image sources, contributors, and licenses

8.1 Text

- **Protection ring** *Source:* https://en.wikipedia.org/wiki/Protection_ring?oldid=702001921 *Contributors:* Taw, Dmd3e, Furrykef, Ldo, Scott McNay, DavidCary, Levin, Uzume, Yayay, Bumm13, Sladen, Bender235, Jnestorius, StYxXx, Guy Harris, Bestchai, Njahnke, Johnwcowan, Frankie1969, Qwertyus, Rjwilmsi, Pmc, Nihiltres, GünniX, BMF81, Hydrargyrum, Tilex~enwiki, Rushyo, SmackBot, Fractal3, Bluebot, Agateller, AndrewBuck, Patriarch, Kostmo, David Morón, Frap, JonHarder, Kaste, Euchiasmus, Errantminion, Kvng, Dl2000, CmdrObot, Hertzsprung, Sanspeur, Flying Saucer, EvaK, Ferenczy, Frozenport, Druiloor, Widefox, HAH, JAnDbot, .anacondabot, OzJuggler, Objectivesea, Scostas, Gwern, Mcgiwer, JonathonReinhart, Ultra two, Red Thrush, Squids and Chips, AlleborgoBot, Android Mouse, Lightmouse, MarkMLl, Accessory, ClueBot, Hus787, Nanobear~enwiki, Socrates2008, Andy16666, Cmcqueen1975, Dsimic, Addbot, Bluebusy, Luckas-bot, Yobot, Golftheman, AnomieBOT, Drachmae, Rubinbot, 1exec1, Absemindprof, Abelon, Ale And Quail, RobertMfromLI, GoingBatty, ZéroBot, ClueBot NG, Matthiaspaul, Helpful Pixie Bot, HolgerBlasum, Jamesx12345, Monkbot, Wandering Logic, Andrybak and Anonymous: 81

8.2 Images

- **File:Computer-aj_aj_ashton_01.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d7/Desktop_computer_clipart_-_Yellow_theme.svg *License:* CC0 *Contributors:* https://openclipart.org/detail/105871/computeraj-aj-ashton-01 *Original artist:* AJ from openclipart.org
- **File:Internet_map_1024.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg *License:* CC BY 2.5 *Contributors:* Originally from the English Wikipedia; description page is/was here. *Original artist:* The Opte Project
- **File:Priv_rings.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/2f/Priv_rings.svg *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia to Commons. *Original artist:* Hertzsprung at English Wikipedia
- **File:Text_document_with_red_question_mark.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/a4/Text_document_with_red_question_mark.svg *License:* Public domain *Contributors:* Created by bdesham with Inkscape; based upon Text-x-generic.svg from the Tango project. *Original artist:* Benjamin D. Esham (bdesham)

8.3 Content license

- Creative Commons Attribution-Share Alike 3.0