The Reference Monitor: An Idea Whose Time has Come

Terry Rooker
A41
Naval Surface Warfare Center Dahlgren Division
Dahlgren, Va 22448
trooker@xobu.nswc.navy.mil

Abstract

As operating systems were developed, limitations in the hardware and software technologies forced the designers to develop large monolithic core programs called kernels. Over time all major operating system functionality was concentrated into these large unstructured programs. When trusted systems were built from these kernels, the simple idea of a reference validation mechanism was not easily implemented. In place of this simple idea ponderous structures were developed in an effort to break apart and modularize the monolithic kernels. This process has become part of the trusted systems development since it is incorporated into the Trusted Computer System Evaluation Criteria. In the last five years there has been a major change in the way operating systems are built. Investigation of current projects reveals that operating systems are now built along modular lines, and there is a concerted effort to reduce the size of the monolithic kernels. Consequently, implementing a small modular reference validation mechanism is now possible in its original form. In some cases, the operating system designers are incorporating a reference validation mechanism, albeit without assurance, to solve some of their own design problems. So technology has finally caught up with the idea of the reference monitor, and it is now possible to use these implementations. Lest the picture seem too optimistic, there are other trends in operating system design that are less favorable to trusted systems development, and any change in our view of trusted systems must also allow for these other developments.

Historically, as computer operations became more complex, the task of managing the computer resources was increasingly automated. The programs managed these resources came to be called operating systems. Because of limitations in the hardware and software, the operating system core, called a kernel, became increasingly large. Most operating system functionality was included in this core. To improve the performance of this core program, structured programming was not used to avoid the overhead associated with that style of programming. The result was a large, poorly designed program where the only consideration was efficiency.

In the 1970's there was increasing awareness of the risk involved with computer operations, and attempts were made to develop trusted computers. The goal of this work was computers with a specified functionality to support trusted functions, but more importantly assurance requirements were developed to ensure that the desired functionality was implemented correctly. The complex kernels were difficult to understand, much less make assurance statements about. The Anderson Report [2] introduced several concepts in an attempt to provide a framework for developing trusted operating systems. These concepts were fundamental to describing the problem, and have been instrumental since they were proposed [1].

The Anderson Report identified three requirements for a trusted system: 1) an access control mechanism; 2) an authorization mechanism; and 3) controlled execution of operating system services and user programs. The Anderson Report in particular proposed the reference monitor as a means of satisfying these requirements. The reference monitor would validate all references by subjects to ensure both that the access was authorized and also of the correct type (i.e. read, or write). As we will see, these conclusions are still applicable today.

The requirements statement for trusted systems in the US department of Defense is the Trusted Computer System Evaluation Criteria (TCSEC) and its interpretations. The concepts of the Anderson Report were incorporated into the TCSEC. In particular, the TCSEC establishes the reference validation mechanism (RVM) as the implementation of the reference monitor. The Anderson Report listed three design requirements for the RVM that are continued into the TCSEC:

1993 ACM 0-89791-635-2

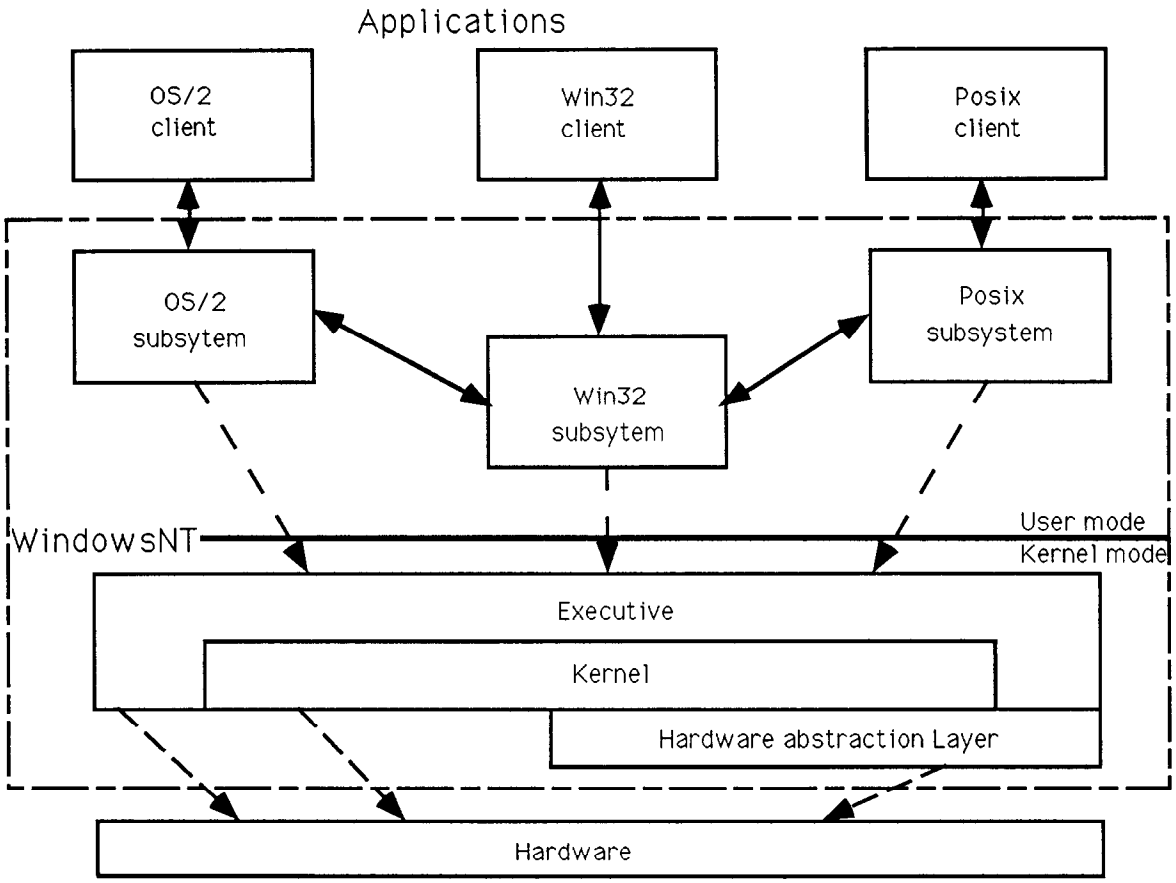

Figure 1: WindowsNT architecture [6]

a. The RVM must be tamperproof.

b. The RVM must always be invoked

c. The RVM must be small enough to be subject to analysis and test, the completeness of which can be assured. [10]

Something New

Even the TCSEC had to face the reality of then current operating system design. In particular the TCSEC had to modify the concept of the reference monitor because of the monolithic nature of the kernels:

> "In order to encourage the widespread commercial availability of trusted computer systems, these evaluation criteria have been designed to address those systems in which a security kernel is specifically implemented as well as those in which a security kernel has not been implemented. The latter case includes those systems in which objective (c) is not fully supported because of *the size or complexity of the reference validation mechanism*. For convenience, these evaluation criteria use the term Trusted Computing Base to refer to the reference validation mechanism, be it a security kernel, front-end security filter, or the entire trusted computer system." [10, p. 67](the emphasis is mine)

Abrams et al fault the TCSEC for not specifying what functions or system software belongs in the TCB. Our interpretation is that the TCSEC simply acknowledged the then state of the art.

Operating system vendors must maintain compatibility with older releases of their product, so there is considerable inertia to only incrementally change their product. Even though better ideas for building operating systems have been around, most major products still rely on some variation of the large monolithic kernel. This trend is starting to change, although most systems in development

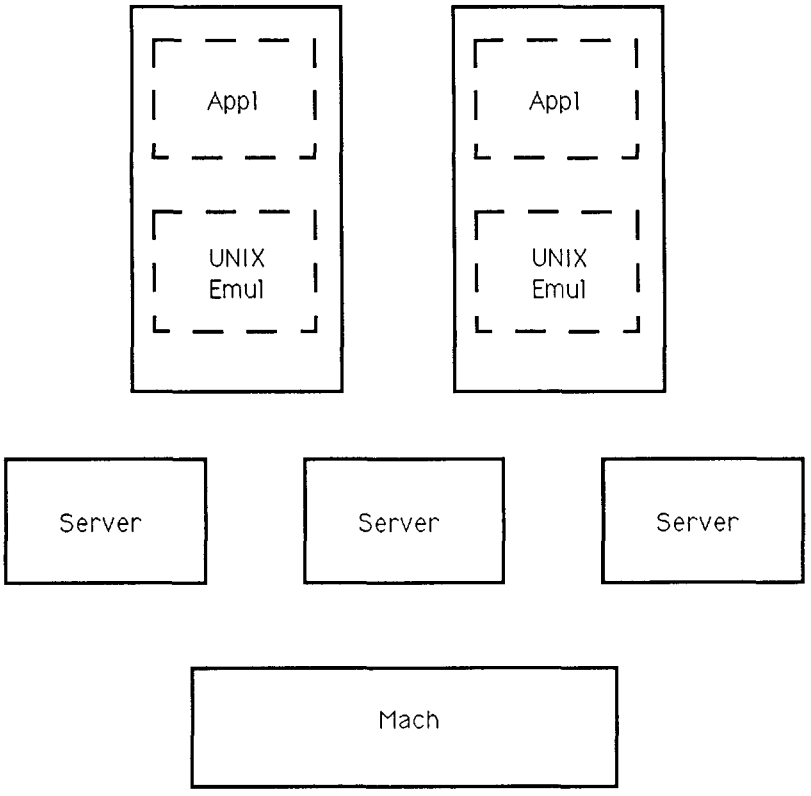

Figure 2: Mach architecture [7]

that use a different structure are new starts, and do not rely on a specific existing product. The new trend is towards client server architectures with backwards compatibility provided by special servers. Ideally, these architectures would be implemented using an object-oriented environment, although none to date have been able to exploit this technology. In particular we will examine three systems currently under development; WindowsNT, Mach, and Hurd.

WindowsNT is an interim product to span the gap between the MSDOS-based Windows 3.1 and the development project called Cairo. Windows, running on top of MS-DOS, was a microcomputer operating system built in the old style. Cairo will be a true object-oriented, client-server system. WindowsNT provides a client-server system [6], although without the advantages of object-orientation. There is a core of operating system functionality that is contained in the kernel, which isolates the applications from the particular hardware. Then there are subsystems that provide services for the specific system calls of an existing operating system, for example there is a OS/2 subsystem, and a Win32s subsystem which allows for compatibility with MSDOS. There is also a part of the kernel that contains all the machine specific code so that porting to different hardware involves only changing code within this section. This design is displayed in figure 1.

Mach is a research project of Carnegie Mellon University[7,9]. The intent of the project was not to provide a stand alone operating system, but to provide a core message passing system that would support client-server operating system developments. The actual client-server micro kernel is Mach 3.0. Earlier releases were more traditional designs with a built-in BSD interface (release 2.5). Mach provides services such as process management and communication (with a port assigned to each thread, the Mach equivalent of a process) that can be used to build specific servers. For example, the BSD interface in Mach 2.5 is just a server in Mach 3.0. This architecture is displayed in figure 2.

The GNU Hurd is an example of a system built (building?) on the Mach 3.0 micro kernel. For those not familiar with the Free Software Foundation (FSF), we should provide some background. The FSF develops a line of products around the name of GNU (a recursive acronym, GNU's Not Unix). These products are distributed under a copyright notice that isknown as the GNU or FSF Copyleft. Copies are distributed for free, and can be re-distributed as long as there is no charge for the FSF software. Among

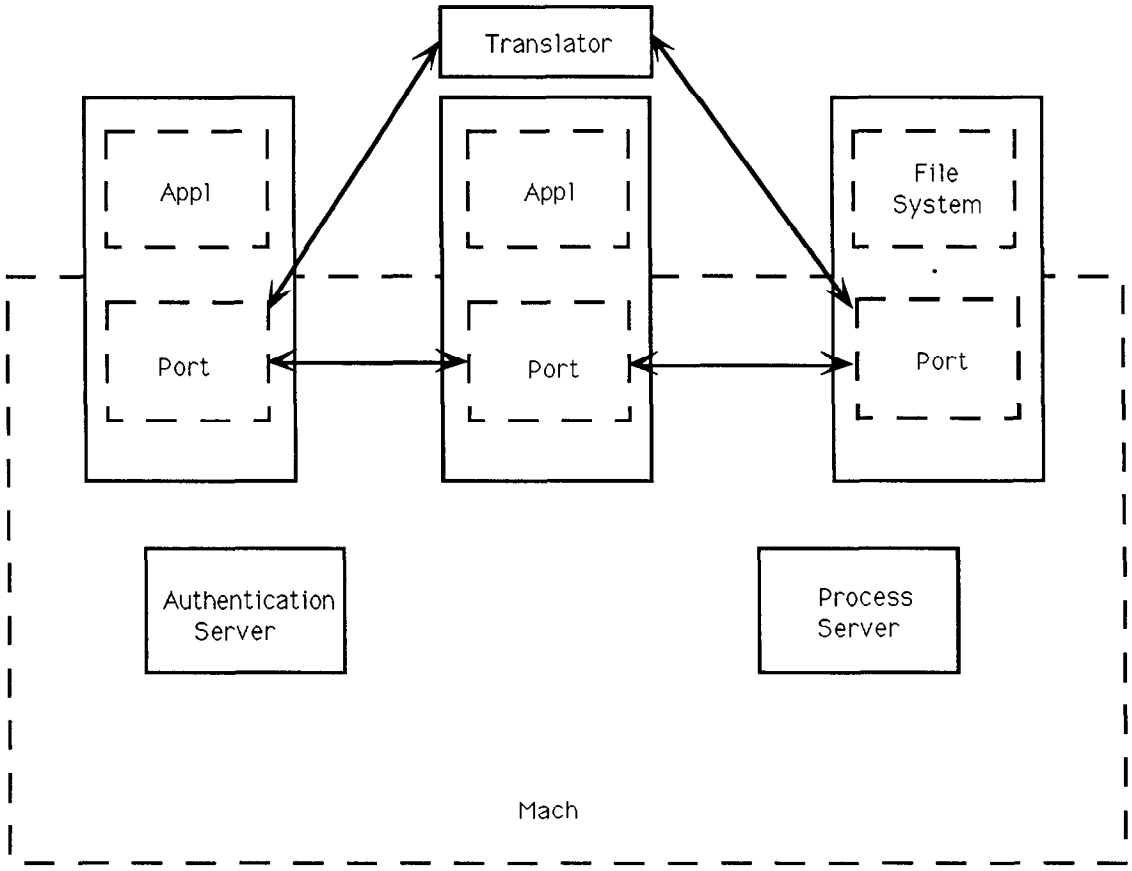

Figure 3: Hurd architecture [4]

their projects is Hurd (also rumored to be a recursive acronym). Hurd is a Unix compatible operating system that would be distributed under the Copyleft. It would be a true client-server system based on the Mach 3.0 micro kernel. Interestingly, in the description of the system [4], Michael Bushnell arrives at the same conclusions as the Anderson Report! On top of the micro kernel there would only be two servers running in kernel mode, the authentication server and the process server. The process server basically provides an interface to the underlying micro kernel for system administration purposes.

The authentication server arbitrates requests for access, as the name implies. The architecture also relies on the Mach abstraction of a port, which is the communications medium. All operating system services would be provided as user mode processes. Other processes would access them through ports. In addition, the owner of a process/application could install a translator that would provide a semantics for the port. For example, the port that accesses the file system would provide a protocol that implements the usual file system semantics. Since Mach ports have a sophisticated access semantics, it is possible that the choice of translator on a given port could be specified for different users. This architecture is displayed in figure 3.

Something Old

With the possible exception of WindowsNT, which is targeted initially at the TCSEC C2 class, these architectures appear to have been developed independently of trusted systems ideas. When we examine the RVM concept there is a surprising parallelism between the RVM and these modern operating systems. Abrams et al discuss some of the architectural implications of the reference monitor, and RVM. The stylized architecture they develop (see figure 4) is very similar to the architectures of the three systems we discussed.

The technology for operating system implementation has finally caught up with theory. The concept of a reference monitor is finally found in some modern operating system designs. This survey of recent development reveals that the

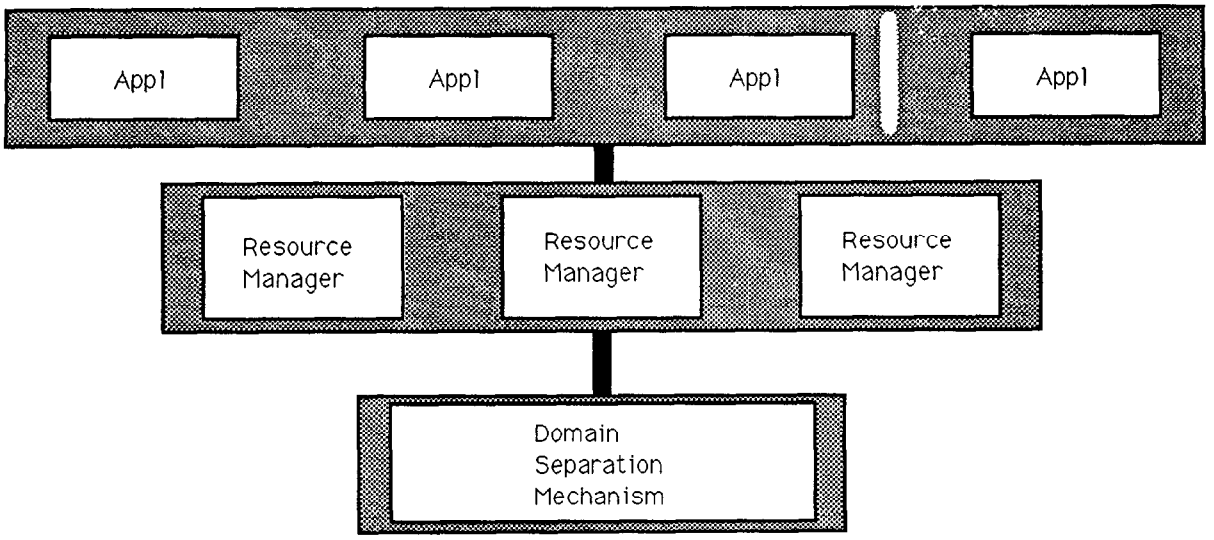

Figure 4: Domain Separation Kernel Concept [1]

reference monitor is embedded in the client-server architectures currently popular for operating system design. Independent of any trust evaluations it is being used to provide for more secure operating systems.

Paradigm Shift

Ironically, the appearance of the reference monitor in modern operating systems will cause a change in how trusted systems are viewed. Since the reference monitor could not be directly implemented in the large monolithic kernels, there have been many techniques developed to overcome this limitation. Since the RVM is now possible in the operating system kernel, these techniques are of little use. This change in view is more apparent at the higher levels (B2+) of TCSEC evaluation. There is an increasing requirement for modularization of the TCB, and the client-server architectures readily support that modularization. With present systems, much effort must be expended to separate the parts of the large kernel, and somehow split the kernel into identifiable modules. It is these techniques that will be of little use in the new architectures.

Abrams et al also discuss the slight change in usage of the terms security kernel and RVM. In view of the fact that monolithic kernels were divided arbitrarily, it is not surprising that there is confusion. With the better defined architectures of the client-server operating systems it is much easier to maintain the distinction. In this case, the RVM and trusted processes map to specific identifiable parts of the system.

In this case, the shift is not away from the old paradigm, but rather that it is now possible to utilize the old paradigm. For twenty years we have known how to build a better system. We just did not have all the right tools. The tools are now getting into place. The problem is that there is an existing body of experience in trying to adapt the old tools to that paradigm [cf. 3]. Slowly as the new tools become available, the experience will change to finally exploit the old paradigm.

While some may argue that this change hardly constitutes a new paradigm, it is important to realize that many complaints with the current paradigm stem from the implementation of that paradigm and not the paradigm itself. The RVM is a theoretical construct that was usually not implemented. Rather a monolithic kernel was arbitrarily divided and a one of the those subdivsions was simply called the RVM. Of the current operating systems described above, all will allow distribution within a single CPU, and across multiple CPUs. Such distribution would allow the 'old' paradigm to help solve 'new' problems such as security in client-server architectures.

New Paradigms?

The problem with change, is that once it is started it is not always easy to stop. The same people that are implementing the client-server architecture which directly provide for a RVM, are also looking at other technologies to exploit. Some of these ideas could be troublesome to the trusted systems community.

Client-server architectures are made possible by the higher performance computers now available. The higher performance comes about from several developments, although we will only discuss a couple of them as examples. The biggest source of improved performance is faster CPU's. One disadvantage of these faster CPU's is

that they can aggravate the covert channel problem. In particular, the increased speeds may allow an increased bandwidth in timing covert channels.

Another source of improved performance are increasing word sizes in modern computers. The current state of the art is 32 bit machines, and 64 bit machines are becoming available. The virtual address space is related to the size of the registers. A 64 bit machine implies a 64 bit virtual address space. There is already at least one research program that is attempting to utilize a 64 bit virtual address space [5]. In this design some artifacts of small address spaces disappear. Specifically the notion of a process with a separate virtual address space is not used. Since virtual process separation by using distinct address spaces is embedded in many notions of trusted systems, discarding these artifacts could have a major impact on our understanding of trusted systems. In this architecture processes are given separate ranges of address space, and inter-process communications are then done by sharing a region of the virtual address space. For high assurance systems, such a system would require a major redefinition of separation.

In the previous discussion of the modern systems, many functions that are currently considered in the kernel would now be run in user mode. In essence, these functions are elevated (demoted?) to the status of trusted applications. Trusted applications are another area where there is insufficient experience for evaluation. Some of the advantages of trusted applications are discussed in a previous paper [11], while some of the disadvantages are also becoming known [8].

What will be shape of this new paradigm? Some of its necessary features are apparent. Covert channels will be at least as much of a concern, possibly aggravated by higher speeds, and shared virtual address spaces. We must come to terms with trusted applications, and methods for their assurance. There are some advantages in the new view. Operating systems will be more modular, and better defined. In addition, the kernel mode portion of the systems will be smaller.

Conclusion

There is an idea that has been around for a long time, that of the reference monitor. For the last twenty years, the large monolithic kernels of operating systems prevented a direct implementation of the RVM. Current work in operating system design exploits some of the features of the RVM, even without the trust considerations. Consequently, we are now in the position where our technology is just now allowing us to adopt a twenty year old concept. This is not a new occurrence, after all tanks, and helicopters were envisioned by Leonardo DaVinci. The problem for system designers is to unlearn the techniques that were used to force-fit the old technology into the new idea. While we are changing these tools, we must also keep an eye open for the new technologies that may invalidate current techniques in a manner not as favorable for trusted system development.

References

[1] M. Abrams, J. Hearney, and M. Joyce, "Mediation and Separation in Contemporary Information Technology Systems". **Proceedings of the 15th National Computer Security Conference**, National Computer Security Center, 1992.

[2] J. Anderson, **Computer Security Technology Planning Study**, ESD-TR-73-51 Vol I, AD-758 206, ESD/AFSC, Hanscom AFB, Bedford, Ma, Oct 72.

[3] J. Arnold, D. Baker, F. Belvin, R. Bottomly, S. Chokani, and D. Downs, "Assessing Modualrity in Trusted Computing Bases". **Proceedings of the 15th National Computer Security Conference**, National Computer Security Center, 1992.

[4] M. Bushnell, "Towards a News Strategy of Operating System Design". Posted to USENET group comp.os.mach, dated Jan 1993.

[5] J. Chase, H. Levy M. Baker-Harvey, and E. Lazoucha, "How to Use a 64-Bit Virtual Address Space". Technical Report 92-03-02, University of Washington, Seattle, Wa, 1992.

[6] H. Custer, **Inside WindowsNT**. Microsoft Press, Redmond, Wa, 1992.

[7] P. Guedes and D. Julin, "Object-Oriented Interfaces in the Mach 3.0 Multi-Server System", Technical Report, Carnegie Mellon University, Pittsburgh, Pa, 1991.

[8] D. Howe, "Information System Security Engineering: Multilevel Distributed Systems Employing Object-Oriented Techniques", submitted to the National Computer Security Conference, 1992.

[9] K. Loepere, **Mach 3 Kernel Principles**. Open Software Foundation, Cambridge, Ma , 1992.

[10] National Computer Security Center, "Trusted Computer System Evaluation Criteria", DOD 5200.28-STD, 1985.

[11] T. Rooker, "Application Level Security Using an Object Oriented Graphical User Interface". To appear in **New Security Paradigms**, 1992.