CSE 221: Homework 1

Hao-en Sung [A53204772] (wrangle1005@gmail.com)
Department of Computer Science
University of California, San Diego
San Diego, CA 92092

1 A fundamental aspect of protection in operating systems is rights amplification. Rights amplification enables a more privileged protection domain to perform an operation on behalf of a less privileged protection domain in a controlled fashion without violating protection in the system. For each of the following operating systems, state (a) the protection domain that they support, (b) the mechanism for crossing protection domains, (c) how rights are represented, (d) how rights are amplified crossing domains, and (e) how the OS determines whether to allow the domain crossing.

1.1 Hydra

1.1.1 (a)

A protection domain in Hydra is the *local name space (LNS)*,

- "At any instant, the execution environment (domain) of a program is defined by an LNS object associated with it...the rights lists in each capability define the permissible access rights of this program at this instant." (Hydra p. 341).

1.1.2 (b)

Kernel provides *CALL* and *RETURN* functions.

- "The execution domain is, at any instant, defined by the current LNS, and an LNS is uniquely associated with an invocation of a procedure; thus, execution domains change precisely when a procedure is entered or exited. The kernel provides two primitive functions, CALL and RETURN which allow a procedure to invoke a new procedure or to return to the current procedure's caller." (Hydra p.341)

1.1.3 (c)

The rights are presented as a *capability*.

- "A capability consists of a reference to an object together with a collection of "access rights" to that object." (Hydra p. 339)

1.1.4 (d)

The callee's right will be merged into caller's right.

- "If the caller's rights are adequate, a capability is constructed in the (new) LNS which references the object passed by the caller and which contains rights formed by merging

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

the caller's rights with the rights ... This implies that a callee may have greater freedom to operate on an object than the caller who passed it as a parameter, but the caller can in no way obtain that freedom for himself." (Hydra p. 339)

1.1.5 (e)

OS will check *type* and *right* in serial. If check is passed, a new LNS is constructed.

- "The template contains a type attribute, which specifies the required type of the corresponding actual parameter... If a type mismatch occurs, the call is not permitted and an error code is returned to the caller. If the types agree, the rights are then checked... The rights contained in the actual parameter capability must include the rights specified in the check-rights field of the template; otherwise, a protection failure occurs and the call is not permitted. If the caller's rights are adequate, a capability is constructed in the (new) LNS which references the object passed by the caller..." (Hydra p. 341)

1.2 Multics

1.2.1 (a)

A protection domain in Hydra is the *protected subsystem*.

- "The method used in Multics is to permit any user to construct a protected subsystem, which is a collection of programs and data with the property that the data may be accessed only by programs in the subsystem, and the programs may be entered only at designated entry points." (Multics p.389)

1.2.2 (b)

When one refers to a protected subsystem, only the designated entry points can be called. Descriptor will be checked then.

- "... the data may be accessed only by programs in the subsystem, and the programs may be entered only at designated entry points." (Multics P.389)
- "In Multics, the function of the descriptor is extended to include modes of access (read, write, and execute) and to provide for protected subsystems which share object names with their users." (Multics p.395-396)

1.2.3 (c)

Multics represent rights through *access control list* (first level) and *descriptor* (second level).

- We may consider the access control list to be the first level of mechanism providing protection for stored information. Most of the burden of keeping users' pro grams from interfering with one another, with protected subsystems, and with the supervisor is actually carried by a second level of mechanism, which is descriptor-based. (Multics p.395)

1.2.4 (d)

Right is amplified through the *protection ring*.

- If a user is concerned about a borrowed program examining residual virtual memory contents in his own process, he may choose to run the borrowed program in a protection ring of lower privilege. (Multics p.398)

1.2.5 (e)

Each subsystem is assigned a number, and hardware permits a subsystem to use descriptors with greater or equal to its own number.

- To simplify its support of protected subsystems, Multics imposes a nesting constraint on all subsystems which operate within a single process: each subsystem is assigned a number, between 0 and 7, and the hardware permits a subsystem to use all of those descriptors containing protected subsystem numbers greater than or equal to its own. (Multics p.396)

1.3 Unix

1.3.1 (a)

Each user has different but unique id; while every process has different address domain in Unix for protection.

- Each user of the system is assigned a unique user identification number... Also given for new files is a set of seven protection bits. Six of these specify independently read, write, and execute permission for the owner of the file and for all other users. (Unix p.367)
- The user-core part of an image is divided into three logical segments... During execution, this segment is write-protected and a single copy of it is shared among all processes executing the same program. (Unix p.370)

1.3.2 (b)

Unix uses set-user-ID feature and system call to cross protection domains.

- The set-user-ID feature provides for privileged pro- grams which may use files inaccessible to other users. (Unix p.367)
- When fork is executed by a process, it splits into two independently executing processes. The two processes have independent copies of the original core image, and share any open files. (Unix p.370)

1.3.3 (c)

Rights in Unix are presented through *access control scheme* and *set-user-ID feature*.

- Although the access control scheme in UNIXis quite simple, it has some unusual features. Each user of the system is assigned a unique user identification number. (Unix p.367)
- The set-user-ID feature provides for privileged pro- grams which may use files inaccessible to other users. (Unix p.367)

1.3.4 (d)

Rights in Unix can be amplified if *set-user-ID* bit is on.

- If the seventh bit is on, the system will temporarily change the user identification of the current user to that of the creator of the file whenever the file is executed as a program. (Unix p.367)

1.3.5 (e)

Access control scheme and *set-user-ID* bit will be checked, similar to (c).

1.4 Pilot

1.4.1 (a)

No protection domain is used in Pilot.

- "All computations on the machine (including Pilot itself) run in the same address space..." (Pilot p.83)

1.4.2 (b)

No crossing domain, since there is only one domain. Same as (a).

1.4.3 (c)

The rights are presented as a "capability".

- "...they are mechanically similar to capabilities used in other systems for absolute protection, but are not designed to withstand determined attack by a malicious programmer." (Pilot P.82)

1.4.4 (d)

No right amplification is used, since there is only one user.

1.4.5 (e)

No crossing domain, since there is only one domain. Same as (a).

2 Some of the systems we have read about and discussed use specialized hardware to facilitate their implementation. Choose one such instance, describe the hardware that was used, and what advantage it gave the system implementors and designers. What is one drawback of relying upon specialized hardware? Do we still use hardware of this form today?

2.1 Hardware

According to the description in Unix paper, Unix uses processor, PDP-11, to detect program faults.

- "The PDP-11 hardware detects a number of program faults, such as references to nonexistent memory, unimplemented instructions, and odd addresses used where an even address is required." (Unix p.373)

2.2 Advantages

Since the hardware can detect those errors and trigger corresponding *interrupt* signal to specific program, that program can easily use a signal handler to deal with it. Otherwise, another program needs to keep busy *supervising* any potential error, which is very inefficient in real application.

2.3 Drawbacks

In the view that those error detections are supported by hardware, the portability of OS is highly limited; unless, it gradually becomes a standard requirement of hardware.

2.4 Today Usage

We still use the same way today: hardware will help detect errors and trigger signals, and then processes can deal with those signals with default or self-defined signal handlers.

3 Pilot made a strong and persuasive argument for tailoring the design and implementation of operating systems to personal computers. We have also seen commercial operating systems like MS-DOS, Windows before NT, and "classic" MacOS tailored towards personal computers as well. Why do you think we still run multi-user timesharing systems like Unix on our PCs? (Consider, for example, the requirements we have of the systems that we use today.)

The assumption "there is only one user in a personal computer" provided by Pilot is too strong today. In modern personal computer, it is easy to have more than one user in one computer. Thus, the protection for each user, for example, file protection, process protection, etc., becomes a crucial issue to the OS.

I believe multi-user timesharing system is a much more flexible design, and probably that is the reason why this kind of design gradually becomes a trend.