CS503 HW 1

Haoran Lei

Q1

Operating system integrates hardware and software support and checks the user's privilege before executing any user command.

Hardware:

1) Privileged instructions and non-privileged instructions isolate user from sensitive area of the system. Privileged instructions set restrictions on how the memory address space may be accessed. Users have no direct access to privileged instructions.
2) The system has user mode and kernel mode. Users mode is converted to kernel mode when users make system calls, which invokes a set of privileged instructions that are not being used by users. In user mode, illegal attempt will result in faults/exceptions. The kernel will handle the code of system call as well as interrupts.
3) Trapping mechanism prevents user from directly accessing the kernel. Users communicate with kernel with system calls and we need a way to prevent user to jump into the middle of the current function. Trap is a software initiate interrupt which is used for switch mode and jump to the kernel define location.
4) Segmentation or paging etc. provides memory protection. The physical memory of a system is divided into different areas with different accessibility. Shared memory area like the stack and heap regions are accessible by users while kernel areas are inaccessible. Segment descriptor also contains value of privilege level and flags to be checked to prevent harm on the kernel and the process. Modern OS implement paging and virtual address for memory protection.

Software support:

1) Stack switching is the way of isolation/protection, when the cpu enters kernel mode, the operating system software grants it access to kernel stacks and it does its job there. Hardware switch stack when interrupt handling requires user-kernel mode switch. (when CPL<=DPL of handler's code segment)

Q2

Prototype: int read (int fd, char*buf, int size)

This is a system call and upon calling the cpu switches to kernel mode. All interrupts are disabled and cpu switches to kernel stack, saving all registers. If arguments are illegal they will not be executed, (see below examples), determined by the kernel.

Examples of the situations of protection in read():

If fd is not is not a valid file descriptor or is not open for reading, the function returns EBADF; if buf is outside your accessible address space, it returns EFAULT; if fd is attached to an object which is unsuitable for reading; or the file was opened with the O_DIRECT flag, and either the address specified in buf, the value specified in count, or the current file offset is not suitably aligned, it returns EINVAL; I/O error will happen for example when the process is in a background process group, tries to read from its controlling tty, and either it is ignoring or blocking SIGTTIN or its process group is orphaned. It may also occur when there is a low-level I/O error while reading from a disk or tape and the function returns EIO.

If legal, the function read up to *size* bytes from file descriptor fd and load it into buffer stating at buf. After that the cpu will restore user registers and switch to user stack and return to the user mode.

Q3

Software fault isolation is one of the approach. A software that turns a program into one that satisfies security policies is applied. Modern compliers like GCC or higher-leveled language executers are often used to check and packet the source codes into valid machine languages that can be safely executed on CPU.

Pros:

It does not require a machine to have hardware protection to achieve such a goal. If the security policies related to abstraction that hardware does not know, a program could be add to handle it. Hardware protection takes more time than its software counterpart, thus the latter is more efficient. Protect from plug-ins. Benefit programs having many objects.

Cons :

Denial of service attacks is possible to happen since software is sometimes shared by different users while hardware are always local resources. The hardware protection is more secured.

Q4

The Multics system has a lot of new features that were later inherited by most of the modern OSes such as Linux and Windows.

1) Major modules of the system communicate on an asynchronous basis, which allows the upgrade or modification of a single module to be carried out without affecting other modules;
2) A two-dimensional addressing system allows users to have large virtual memories. This is still the mainstream approach nowadays. Paging and segmentation are used in Multics, as they are used now in memory management. Descriptor bit was introduced

in Multics as the hardware fire-wall which helps isolation/protection. Privileged instructions are not available users. This is still used today;

3) Pure processes are allowed and intersegment binding occurs dynamically as needed during program execution. Each user is provided with a private software "stack" for temporary storage within each subroutine;
4) Decentralized programming and only one type of calling sequence unifies the programming manner and makes it user-friendly to programmers who can modify the system. The system will be open-ended and will be largely created by the users themselves, which is still the best part of Linux.
5) One process can spawn other processes which run asynchronously on several processors. Nowadays processes not only create each other but also become managed with threads and become tasks which helps the user a lot.
6) Shared database. Multiple users sharing one set of data is widely seen in modern systems.
7) The illusion that a user is occupying the whole cpu and file system is given. Also in modern OSes, such illusion is still the case.
8) File privacy is secured and each user's files can be arranged to be completely private to him. Mishaps are possible and files could be destroyed, as they are now in modern systems.

There are also differences between the Multics and modern systems.

1) File backup and recovery is harder in modern OSes as accidental deleted files are no longer easily recoverable. Multics has such a backup system that does the recovery.
2) As a multiple-access system, Multics has less power of handling capacity for simultaneous on-line users. Too many users requesting resources from Multics will probably drain out its power, while Windows or Linux has mature procedures of handling large volume of users at the same time.