CS50300: Operating Systems

HW1 Answers

Zichen Wang
wang4113@purdue.edu

September 11, 2018

1

In modern operating systems, isolation and protection are typically achieved by both hardware and software support. The hardware support includes (1) non-privileged and privileged instruction set, (2) kernel mode and user mode, (3) trap instruction, (4) memory protection. The software support is stacking switching when an user application makes a system call.

Every modern CPU typically has two types of instructions. One is privileged instructions, which can only be executed on kernel mode. The other is non-privileged instructions, which can be executed on both user and kernel mode. The privileged instructions are related to some crucial operations, such as input and output, enable or disable interrupt, directly access to hardware address, so they cannot be executed by user apps without permission, otherwise the whole system will be dead by malicious apps or crashed process by executing privileged instructions improperly. Thus, user processes cannot gain direct access to system-related resources, like I/O devices, clock, and so on. If a user process would like to do privileged things, it must execute the trap instruction, i.e. make system call, which can switch from user mode to kernel mode. The main purpose of trap instruction is to provide the user process with a interface to jump to the kernel code. The kernel code is always right and would never cause crash on the whole system. Memory protection is usually achieved by hardware in modern computer architectures. The purpose of memory protection is to prevent a process from accessing the memory of kernel and another process that has not been shared to it. An unauthorized access of memory will result in hardware fault, such as segmentation fault and terminating the process. Older X86 has 4 segment registers and a global descriptor table in order to give each process a memory segment, which is to protect kernel's memory from user process. Modern computer architectures which support paging usually use virtual address and pages to protect one process's memory from another process. With segmentation and paging, neither innocuous nor malicious apps can gain direct access to other normal process's or kernel's memory.

Software or OS also helps to achieve protection. When a process makes a system call, the stack will switch from user stack to kernel stack for that process, so that user process would not be able

to access this kernel stack which includes some special return addresses, old register values and local variables that belong to kernel.

2

Before an app invokes read() system call, it must prepare arguments for read() at its user stack. Then, the app typically executes the *int* or *sysenter* instruction to tell the kernel that it would like to read data from a file. Trap instruction is a set of non-privileged instruction that an app can use to trap itself into kernel mode, because the app itself isn't allowed to execute privileged instructions on user mode. The CPU will switch from user mode to kernel mode after executing this trap instruction, so that it can execute some privileged instructions related to reading data from a file.

After switching to kernel mode, the stack will also switch from user stack to kernel stack for this app to save the return address, old register values and some other important local variables used in read(). This stack cannot be accessed on user mode in order to make sure that innocuous or malicious apps have no right to read or write the content in it. Next, the CPU will execute privileged reading instructions defined in read() on kernel mode. The OS will read data from a file to a memory buffer that can be accessed by the app.

When reading finished, the system call will return. The final return address to the app has been saved in the kernel stack. Therefore, the CPU will load the address into *CS* and *IP* register, and return to the user app. It will also switch from kernel mode to user mode on returning time.

3

The main aspect of isolation/protection is the isolation between processes and the protection of kernel. First, to protect memory, the OS must become a virtual machine. The compiler and linker would no longer compile the source code to the machine code, which means that the machine code cannot run directly under the OS. They can only compile the source code to the 'OS byte-code', like Java Virtual Machine. Then, the OS translates the byte-code into the machine code.

Like the modern hardware support OS, pure software support OS should also held a page table to manage each process's memory. The virtual address space of an app does not need to change, and the OS is responsible for translation.

With the OS byte-code, one process cannot access another process's memory. The bottom CPU and its registers are not transparent to the top apps.

The advantage of achieving isolation and protection purely in software is that the hardware would become very simple to design, because it is not necessary to distinguish running modes, privileged and non-privileged instructions or design some special registers. The CPU may focus more on calculation instead of protection. Moreover, software has more flexibility. It may provide a lot of optimization on common process running scenarios, such as process communication, shared memory and I/O operations, but hardware is always dead code and not easy to modify.

However, the disadvantage of that is easy to find. First, software support could be inefficient.

The OS may use extra memory to save status and kernel related variables, which will increase the execute time of an app. Moreover, the translation of a virtual address should be finished by the OS. Second, nobody can guarantee that the software or OS doesn't have bugs. The more code, the more bugs. So, the stability of the whole system would decrease.

4

Similarities:

1. The design purpose of both the Multics system and modern desktop operating systems is to support many users simultaneously on a single computer. Each user can initial jobs asynchronously and determine when to finish the job. Thus, time-sharing becomes the core idea to support this feature. In the Multics system, each user's job is divided into a few tasks which are placed in an appropriate queue. Moreover, the supervisor program in the Multics system would decide which task should be treated by the CPU. Similarly, in today's desktop operating systems, the supervisor program is a part of the kernel, call scheduling program, and a job is a process. The kernel typically assigns each process a priority and a time-slice. When a process has been running out of its time-slice, the kernel will schedule another process to run at the clock interruption. Like the Multics system, the kernel is not a process, either.

2. In the Multics system, the memory management is adopt two-dimensional virtual memory system, i.e. segmentation and paging, which is similar to today's operating system. First, every address given in a program is a logical address. The Multics system translates this logical address into a linear address by specifying a staring point of a segment. Then, the mechanism of paging can translate a linear address into the corresponding physical address. Moreover, paging allows an incompletely loaded program, and avoids the cost of moving a segment in the primary memory. These features are the same with modern operating systems. The hardware of memory management is called MMU in modern OS. Both the Multics system and today's OS implement memory protection base and segmentation and paging. There are some bits in the memory segment and paging that indicate which program can access. Only the supervisor program, i.e. the kernel today, have the ability to change these bits.

3. Inter-segment binding occurs dynamically during program execution in the Multics system, which is similar to dynamic linking of today's operating systems, and it is totally automatic. In dynamic linking, a running process could request other segments to add to its address space. Procedures could be compile alone in order to share among users. In addition, the 'stack' provided for each user in the Multics system is used for temporary storage within each subroutine, which is similar to a process's running stack in modern OS.

4. The similarity of the file system in both the Multics system and modern OS is that all files are symbolic names, not physical addresses. The system allows users to change a file by referring its name and invoking system calls (letting supervisor program do in the Multics system). Moreover, each file has a privilege that indicates which user can read, write or execute it. The file system of the Multics system and modern OS also allows multiple users to read a shared file simultaneously, but uses lock to prevent a file from writing at the same time.

5. For I/O devices, both the Multics system and today's operating systems would use primary memory as a inter-medium between I/O controls and the CPU. Interruption is the core mechanism to achieve this feature.

Differences:

1. There is something different when making memory segments for a program. In the Multics system, any segment can grow (or shrink) during execution, without planning ahead, while in modern operating systems, segment information is usually determined after compilation and linking. When loading a program, segments are fixed and saved into the global descriptor table.

2. The scheduling strategy is pretty simple in the Multics system. All tasks are equal in the queue, while in modern OS, the scheduling is much more complicated. Time sharing is only a basic idea, from which a lot of more reasonable scheduling strategies, such as time-sharing with priority, multiple priority queues, have been developed in today's operating systems. In addition, different scenario will use different strategy. For example, mobile OS must provide the app running front a higher priority.