OS homework 1

Zhenming Wang Net ID: zw1806 Sextion 002

February 2021

1 problem 1

One difference I can tell is that OS running on server would support better hardware. For example, I learned that Windows 10 has a limit for RAM of 2TB, but Windows Server can support up to 24TB!
From my perspective, the reason for this is that a server is designed to handle heavier tasks, for example, a server would have to deal with a lot of requests for resources at any second.

2 problem 2

What exactly happen when CPU deals with this call: First, the wrapper function ***fscanf*** takes care of copying arguments from user's invokation to specific registers. Next, In the wrapper function *fscanf*, there will be a trap execution. Thus, when the wrapper function *fscaf* is invoked, it will executes the **trap** execution. Now, the CPU will switch from **user mode** to **kernel mode**. Now, in response to the trap, the kernel invokes the system call routine, which is also called system call handler. The handler saves register values onto kernel stack. Next, the handler is going to use a table of system call number and corresponding system call to invoke appropriate system call service routine. The routine is going to perform required actions. After all these actions, the status of execution is going to return to the system call routine. Next, the handler is returned to wrapper function, and the CPU will switch back to user mode.

3 problem 3

The two modes are user mode and kernel mode.
To answer the reason of why: Some machine instructions could cause harm, which are called privileged instructions. If we give every program the authority to run those instructions, the operating system would go very wrong. Therefore, We need to protect the operating system from errant users.
To answer the question of what they do: When the system is in user

mode, the computer system is executing on behalf of a user application. In the kernel mode. the computer system executing on behalf of the operating system, for example. those privileged instructions that are only allowed to be executed by the OS.

4 problem 4

multiprogramming: There are multiple jobs loaded in main memory at the same time.
why useful: A single program cannot keep wither the CPU or the I/O device busy at all times. With multiprogramming, we are able to increase CPU utilization.
difference from time-sharing: Multiprogramming provide an environment in which the various system resource are utilized effectively, but it does not provide for user interaction with the computer system. Time-sharing makes the CPU executes multiple jobs by switching among them frequently, so that the users can interact with each program while it is running.

5 problem 5

Timers are useful in the sense that they help prevent a user program from running too long. For example, a user program can get stuck in an infinite loop. Timers help ensure that the operating system maintains control over the CPU.
how are they implemented: A timer is generally implemented by a fixed-rate clock and a counter. The operating system sets the counter. When the clock ticks, the counter get decremented. When the counter reaches 0, there will be an interrupt and the CPU will enter into kernel mode and hand in full control to the operating system

6 problem 6

It is a system call.
System programs provide a convenient environment for program development and execution. System programs can consist a bunch of system calls, or they can be simply user interfaces to system calls. However, the code in the question only consists one **read** system call
difference between nread and nb2read: **nb2read** indicates the amount of bytes we'd like the kernel to read for us, while **nread** does not always equal to **nb2read**, **nread** tells the actual number of bytes get read.
Useful or not: This is a useful feature. If **nread** is less than **nb2read**, then we may be closed to the end of file.

7 problem 7

a,c,d,f
Explanation:

1. **a:** Setting the value of timer should be privileged since timers are used to prevent user programs from running forever because of erroneous code. If setting the value of a timer can be done in user mode, then the operating system lose the control to CPU.

2. **c:** Obviously, clearing memory cannot be a non-privileged instruction. If a user program can clear the memory, there will be disastrous consequences: all the works in main memory can get erased.

3. **d:** Obviosuly, turnning off interrupts need to be a privileged instruction, since interrupts are used to send signal to the CPU, for example, hardware interrupt or system call trap. If user program can turn off interrupts, then the operating system cannot function at all

4. **f:** Access an IO device need to be a privileged instruction. Let's say there may be malicious program that reads password while users are typing on keyboard, or some process can arbitrarily write to disk that can ruin other important information.

8 problem 8

difference between system program and system call:
System calls provide interfaces to services made available by an operating system. Those services are generally 'privileged instructions', and only operating system can execute those instructions.
System programs provide a convenient environment for program development and execution. System programs could be a collection of system calls, or a user interface to system calls. The view of the operating system seen by most users is defined by applications and system programs, rather than by the actual system calls.
System calls can only be executed in kernel mode.
System programs does not necessarily run in kernel mode.
an example of system calls manifest in system programs
When we open up our text editor, we want to first open up a file. This require a system call of access file in the disk.
When we are working on a text editor, as we are typing on the keyboard, the system call of accessing our keyboard input is triggered.
When our inputs have been collected by the kernel through relevant system calls, our text editor need to output those texts we inputted. This need another system call to output on the screen connected to our system.
When we are finished with typing, we click save to save our changes. This require a system call to write to disks.

Finally, when we quit the text editor, the process terminate normally. This termination is another system call.