**Score: \_\_\_\_\_**

**OSA3 – Interrupts and System Calls**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2023

Prof. Grant Braught

**Name:**

**Introduction:**

Today’s class built on the notions of processes and multiprogramming from the last class. To support multiprogramming the operating system must be able to share resource among the processes (e.g. memory, devices and the CPU), and protect its data structures and the processes from each other. Having two processor modes, one for running OS code (kernel mode) and one for running user processes (user mode) form the foundation for ensuring that the OS has the authority to enforce sharing and protection. Then device interrupts, system calls and traps provide the mechanisms by which processes and devices can request services that require the privileges only available to the OS when running in kernel mode. In these activities you will explore those ideas further and learn more about the mechanisms by which interrupts and system calls are handled.

**Protection and Sharing:**

🔑 1. In class the college metaphor was used to introduce the ideas of sharing and protecting resources as it relates to operating systems. This question asks you to do the same using your metaphor.

a. In a sentence, reintroduce your metaphor. You can copy this from a previous assignment. I ask for it to help me remember what it was as I read the rest of your answer.

b. Identify the specific elements of your metaphor that you will be using to play the roles of the hardware, the processes and the operating system.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **System Element** | **Metaphor Element** |  |
|  | Hardware |  |  |
|  | Processes |  |  |
|  | Operating System |  |  |
|  |  |  |  |

c. Use the elements of your metaphor from part b to explain the role of the operating system in sharing resources among the processes.

d. Use the elements of your metaphor from part b to explain the role of the operating system in protecting resources and processes from other processes.

e. Identify the mechanisms in your metaphor that give the operating system the authority to carry out sharing and protection and briefly describe how these elements do this. (i.e. like the password or keys to the store room in the college metaphor)

**Processor Modes:**

Processor modes ensure that there are some operations that only the operating system is able to perform. By doing so this guarantees that the OS can enforce sharing between user processes while protecting its own data structures and processes from other processes.

🔑 2. What are the two processor modes?

🔑 3. In discussing user mode, it was said that “the processor must be in user mode anytime a user process is running.” Explain briefly why it is essential that the processor must be in user mode any time a user process is running.

🔑 4. For each of the following operations indicate if it should be allowed to be performed in user mode or if it should only be allowed in kernel mode. For each operation you designate as kernel mode give a brief but specific explanation of why it should only be allowed in kernel mode (i.e. discuss how being able to do it in user mode would compromise the OS’s ability to enforce sharing and protection.)

a. Creating a new user process.

b. A function that sorts an array of values.

c. Setting the duration for the timer device (i.e. how long before it generates an interrupt).

d. Calling a function (i.e. pushing parameters, making the call, getting the return value).

e. Performing a context switch.

f. Reading a file from the disk drive.

🏆 5. In discussing kernel mode, it was said that “the processor must only be in kernel mode when OS code is running.” This implies that the processor does not have to be in kernel mode when running OS code, though clearly sometimes it must be. What advantage might there be to running some OS code in user mode? Hint: You might read a little bit about the “principle of least privilege.”

**Invoking OS Code:**

🔑 6. What is an interrupt service routine?

🔑 7. Identify the three types of events that cause interrupt service routines to be run and briefly describe the purpose of each one in a sentence or two of your own words.

🔑 8. Indicate if each of the following events corresponds to a device interrupt, a system call, a trap or none of the above.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Event** | **Type** |  |
|  | Opening a file to be read |  |  |
|  | Entering data on the keyboard |  |  |
|  | Accessing another process' stack |  |  |
|  | Requesting a web page |  |  |
|  | Receiving a web page |  |  |
|  | A division by 0 |  |  |
|  | Creating a new object |  |  |
|  | Multiplying two values |  |  |
|  | Deleting a file |  |  |
|  |  |  |  |

**Interrupt Handling Mechanisms:**

We now have a good idea of what interrupts are and how they relate to devices, system calls and traps. Now let’s turn to how the appropriate ISR is run when an interrupt occurs.

🔑 9. State what the abbreviation IRQ stands for and then briefly explain in a sentence or two of your own words what an IRQ is.

🔑 10. What piece of hardware generates the IRQs.

🔑 11. Explain in a sentence or two of your own words how the IRQ is used to find and run the ISR for the device that generated the interrupt.

This is not required reading, but if you are interested in more details on how Linux handles interrupts, Stephan Avenwedde gives a really good and accessible overview in his article *How the Linux kernel handles interrupts* on Opensource.com:

* <https://opensource.com/article/20/10/linux-kernel-interrupts>

**System Call Mechanisms:**

If you think about it, programs must be making system calls all of the time. If you think about programs you have written, every time you printed something, read or wrote to a file, or got input from the user your program had to do that via a system call. But it doesn’t seem like it… for example in Python we just use print or input. Or in Java we use System.out.println or the Scanner class. Well, it turns out those are just functions/methods that are wrappers around the code that makes the actual system calls.

🔑 12. All high-level programming languages come with libraries that make it easier for programmers to make system calls. In class it was claimed that these library functions are abstractions. Recall that an abstraction allows the user to focus on the relevant details while ignoring the non-relevant details. Explain how library functions are an abstraction by pointing out what details they require you to focus on, and which they allow you to ignore.

🏆 13. Some system calls request services that take a long time, at least relative to the speed that the processor works. For example, reading or writing the disk or waiting for the user to type input. The system calls that we have been considering and those shown in the video were like this. They move the process (and its PCB) to the waiting state and then call the scheduler to pick a new process to run while the disk processes the request. This makes good sense because other processes will be able to get a lot done while the disk completes its operation.

Other system calls however request operations that are quite fast. For example, displaying information on the screen (e.g. print) or requesting some additional memory (e.g. new). For these types of system calls it does not make sense to move the process to the waiting state. Describe how these system calls might function instead (i.e. sketch out would appear in their boxes on the right in the video.) There are number of reasonable answers here.

**More Details:**

These are not required viewing but might be of interest and might provide a different perspective that clicks for you.

If you would like to have another fairly high-level explanation of how system calls operate in Linux specifically check out *Linux Tutorial: How a Linux System Call Works* from the Linux Foundation:

* <https://www.youtube.com/watch?v=FkIWDAtVIUM> (2:30)

Or if you want a more detailed discussion of how systems calls work at a lower level of abstraction (i.e. real code) watch the very very in depth discussion given in *Syscalls, Kernel vs. User Mode and Linux Kernel Source Code - bin 0x09* by LiveOverflow. If you do watch this, don’t worry if you don’t get every detail. It’s just nice to see how some of this stuff actually plays out in real code, so focus on the parts that connect and the terms that sound familiar.

* <https://www.youtube.com/watch?v=fLS99zJDHOc> (13:23)

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.