CS3510: Operating Systems I
Quiz 1: Autumn 2020

Soumi Chakraborty
ES19BTECH11017

Answer 1

System calls are executed in the kernel-mode (supervisor or privileged mode) by the operating system to satisfy some request on behalf of the user. A system call that gets executed is treated as an interrupt by the hardware. The interrupt passes the control to a service routine in the OS, setting the mode to 1 (kernel mode). The kernel inspects the instructions from the interrupt to decide what kind of system call has been executed and what it is asking for. The **kernel is responsible for verifying that the parameters from the system call are valid and legal**, performing the task and returning control to the appropriate instruction. **Hardware protection detects threats and errors and ensures the safety of the system** when it's executing an instruction in the privileged mode. The errors detected by the hardware is usually handled by the operating system. In the event of an illegal instruction being detected, the hardware traps it to the OS. The trap transfers control to the OS using the interrupt vector, and the failed program is terminated abnormally.
Hence, hardware protection and the kernel ensure the safety of the system.

Answer 2

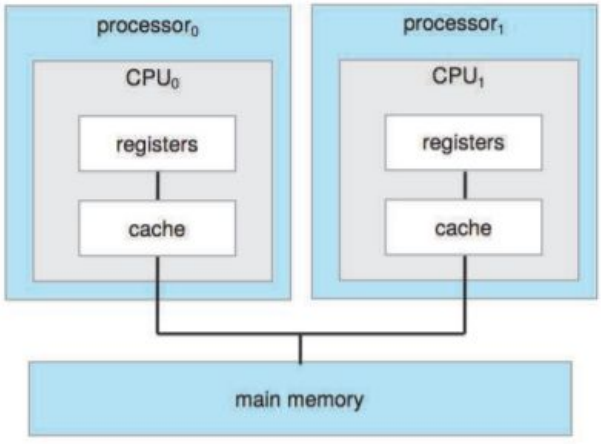

Figure 1.8 Symmetric multiprocessing architecture.

An example to illustrate how data residing in memory could in fact have a different value in each of the local caches:
Let's assume processor 0 reads some data from the main memory to it's local cache. Let's call this data X, with a value of 10. Now, processor 1 too reads the same data into it's local cache. Currently, both copies of X have the same value. But now, let processor 1 multiply X into 5 and change it's value to 50. Since

this update only happened in processor 1's local cache, this will not get reflected back to processor 0. Hence, they both have different values.

Answer 3

If the system call execlp() succeeds, the new program starts running and the control is never returned to the given program, and thus printf("LINE J"); will never be reached. However, if the the system call execlp() fails or encounters an error of some sort, the control is returned to the given program and printf("LINE J"); is executed.
Thus, the line printf("LINE J"); will only be reached if the execlp() call fails in the child process.

Answer 4

Output:
Line X
CHILD: 0 1CHILD: 1 0CHILD: 2 -1CHILD: 3 -2CHILD: 4 -3

Line Y
PARENT: 0 0PARENT: 1 -1PARENT: 2 -2PARENT: 3 -3PARENT: 4 -4

Justification: The child is essentially a copy of the parent, and both the parent in the child have their own copies of the global and local variables. Thus, changes to the nums[] array by the child process doesn't affect the one in the parent one.

Answer 5

a. **Synchronous and Asynchronous communication**
   In synchronous message passing, the sender blocks itself after sending the message (and unblocks after the message is delivered) and the receiver blocks itself after receiving the message. The advantage here is the communication is very methodical and synchronous. However, if a message fails to get delivered for any reason, the sender will remain blocked indefinitely.
   In asynchronous message passing, the sender and receiver do not block themselves or wait and continue working regardless of whether the message delivery was successful or not. The advantage of this method is that process will never get blocked like in the previous case. However, the disadvantage of this method is the fact that the processes won't be synchronized anymore.

b. **Fixed-sized and variable-sized messages**
   Fixed-size messages have a simple and straight-forward implementation at the system-level but make the programming more difficult, because everything like different strings will have to be broken down into smaller units until they match the fixed size. Alternatively, variable-sized messages are a little more complicated to implement at a system level, but the programming is much simpler.