OS Quiz 1
S.V.Harshith - EE19BTECH11018

Q1)

As the textbook says -

A system call is executed as a software interrupt by the hardware. Control passes through the interrupt vector to a service routine in the operating system, and the mode bit is set to zero(kernel mode).The system call service routine is part of operating system. The kernel examines the interrupting instruction to determine what system call has occurred; a parameter indicates what type of service the user program is requesting. Additional information needed for the request may be passed in registers, on the stack, or in memory (with pointers to the memory locations passed in registers). The kernel verifies that the parameters are correct and legal, executes the request, and returns control to the instruction following the system call. Once hardware protection is in place, it detects errors that violate modes. These errors are normally handled by the operating system. If a user program executes an illegal instruction, then the hardware traps to the operating system.

Q2)

The fork system call fails if there are no more slots left in process table, and there is no more memory left.

The exec system call fails if the program file does not exist or it doesn’t represent any valid executable file.

Q3)

The concept of virtualisation is to run software on top of host operating system. This means that the virtual machines have to request access to the hardware form the host and cannot access the hardware directly. This will slow down the usage of virtual machine and hence make it less efficient than real machines. Since it uses the resources of host, for a fast virtual machine we need a very powerful host. Also if multiple virtual machines run simultaneously on a single host they may affect each other due to the workload and maybe sowed down.

Q4)

The output will be as follows –

CHILD: 0 1

CHILD: 1 0

CHILD: 2 -1

CHILD: 3 -2

CHILD: 4 -3

PARENT: 0 0

PARENT: 1 -1

PARENT: 2 -2

PARENT: 3 -3

PARENT: 4 -4

Explanation for output at Line X –

When the child process is created after the fork() call it returns the pid of child process to the parent and the child process gets a pid of 0.

Now in the child process the “if(pid==0)” is satisfied and the for loop is executed and we get the following output –

CHILD: 0 1 (Since i = 0 and nums[0] = 1-0 = 1)

CHILD: 1 0 (Since i = 1 and nums[1] = 1-1 = 0)

CHILD: 2 -1 (Since i = 2 and nums[2] = 1-2 = -1)

CHILD: 3 -2 (Since i = 3 and nums[3] = 1-3 = -2)

CHILD: 4 -3 (Since i = 4 and nums[4] = 1-4 = -3)

Explanation for output at Line Y–

The parent process satisfies the “else if(pid>0)” , and wait(NULL) is executed so the parent waits for the child process to finish and after that the for loop is executed. Also, the nums array changed in the child process is a copy of nums and the nums array in parent is unchanged. So, we get the following output –

PARENT: 0 0 (Since i = 0 and nums[0] = 1*0 = 0)

PARENT: 1 -1 (Since i = 1 and nums[1] = 1*-1 = -1)

PARENT: 2 -2 (Since i = 2 and nums[2] = 1*-2 = -2)

PARENT: 3 -3 (Since i = 3 and nums[3] = 1*-3 = -3)

PARENT: 4 -4 (Since i = 4 and nums[4] = 1*-4 = -4)

Q5)

Yes , it is possible that we would want to allow a process to wait on more than one event at the same time.

For example –

If the process requests input from two or more devices, then it will request from all input devices at once and will go into the wait queue until all of the inputs are available for usage.

Similarly, If the process requests from two or more devices like if it wants to transfer data from one device to the other, it will request from both the devices at once and will go into the wait queue until both of them are available for usage.