# Operating Systems (CS3510) Quiz 1

SURAJ TELUGU

CS20BTECH11050

**1.sol**

Protection in Dual Mode Operation:

1) The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another
2) We designating some of the machine instructions that may cause harm as privileged instructions which can be only executed in kernel mode. User mode cannot access a privileged instruction
3) The user when gives an instruction if it involves any privileged instructions there is a mode bit provided by the hardware which turns to “kernel” during process execution and later the when process is terminated system ensures that mode bit turns to “user” again.
4) If an attempt is made to execute a privileged instruction directly in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system.
5) This mechanism ensures user not to use any of the privileged instructions and therefore ensures the safety of operating system and system components.

**2.sol**

For each of the following system calls, give a condition that causes it to fail: fork , exec.

Fork (fork()) fails i.e gives an error when:

1) When the number of children in that the process reaches maximum fork may give out an error

Exec (ex: execlp(const char*,*path,*arg)) fails i.e gives an error when:

1) The path is incorrect (i.e instead of “ls” if it is given “l_s”)
2) Insufficient memory

**3.sol**

Virtual machines became very popular because through virtualization a user can use different operating systems simultaneously and can also switch among processes in different operating systems but they do have some disadvantages

Disadvantages of virtual machine:

1) Every machine-level instruction that runs natively on source system (original OS) must be translated to the equivalent function on the target system (Virtual Machine), frequently resulting in several target instructions.
2) If the source and target CPUs have similar performance levels, the emulated code may run much more slowly than the native code. Which degrades the performance of system.
3) Virtual Machines cannot use hardware as efficiently as original OS since the hardware has the architecture corresponding to original OS.

**4.sol**

Output of given program:

CHILD: 0 1

CHILD: 1 0

CHILD: 2 -1

CHILD: 3 -2

CHILD: 4 -3

PARENT: 0 0

PARENT: 1 0

PARENT: 2 2

PARENT: 3 6

PARENT: 4 12

(Note there no '\n' in the command but I have given the output in different lines to understand and differentiate)

Explanation:

Firstly all elements in nums[5] are initialized to 1, fork() system call is used to create child of product. Initially is pid > 0 (i.e unique pid of child) The parent process runs first but there is wait(NULL) command which makes parent process wait (sets its state to waiting in PCB) and now since the child is newly created process ready to execute pid = 0 (by fork()) and child process starts executing

In the child process the loop gives nums[i] = nums[i] – i for i = 0,1,2,3,4 i.e (1-i) is obtained as output at every line once the for loop terminates the child process gets terminated. Now since there is no child the parents process continues its execution after a wait.

In parent process the loop gives nums[i] = nums[i] * i for i = 0,1,2,3,4 since now nums gets modified by child process i.e nums[5] = {1,0,-1,-2,-3} now the above operation is performed on updated nums and we get new nums as nums[5] = {0,0,2,6,12} as parent output

Since the parent and child are terminated, the process ends and above output is obtained.

**5 sol:**

Yes, A process can wait for more than one event when multiple threads of process are running simultaneously. Let the process have thread 1 and thread 2 which execute separately with different program counter then when an interruption occurs and process has to wait. Let the thread 1 is going to stopped by an I/O interrupt and thread 2 is going to be stopped by an child termination interrupt. Then the process would wait in both I/O wait queue and child termination wait queue.

Example

Process (Multi-threaded)

{……

……

scanf() // I/O interrupt

}

{……

……

wait(NULL) // Child termination interrupt

}

The above process would wait in both I/O wait queue and child termination wait queue.