Department of Computer Science and Engineering, Indian Institute of Technology Palakkad

CS3110: Operating Systems Lab

24th October, 2019 **Viva prelims** **4:00pm - 4:30pm**

1. **(Stage 20)** When a `fork()` system call is processed by the kernel, what is the maximum and minimum number of pages that would change from `FREE` to `ALLOCATED`? Why?
2. **(Stage 20)** What are the states of the child process before it gets scheduled for the first time. When does each state transition to the next state?
3. **(Stage 21)** How would you modify the implementation to ensure that the calling processes will not wait for a processes that is already waiting for the calling process.
4. **(Stage 21)** Which are the data structures used in the implementation of the `wait()` system call to ensure that the calling process does not wait for itself, or any terminated process?
5. **(Stage 22)** We know that the `fork()` system call will share the semaphores acquired by the parent process with the child process. Consider the scenario where the parent has locked a semaphore and calls a `fork()` from within the critical section. What will happen in this case? Would both the parent and the child process execute inside the critical section or would mutual exclusion be preserved? Why?
6. **(Stage 22)** What is the maximum number of semaphores that a process can simultaneously hold in eXpOS? Why? Consider the following scenario. A process with pid = 5 calls `semget()`, and gets a return value of 15. During this activity, how are the entries in the semaphore table modified?
7. **(Stage 23)** What will go wrong if `delete()` system call does not clear the dirty bit in the `buffer table` corresponding to the disk blocks it is using?
8. **(Stage 23)** In the implementation of the `delete()` system call, it is required to check if the current user owns the file to be deleted. How is this check implemented?

[END]