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

CS3110: Operating Systems Lab

26th September, 2019 **Viva prelims** **4:00pm - 4:30pm**

1. **(Stage 17)** For handling the `Exec` system call, the number of code pages required for the new program is to be computed. Which SPL file has the code for finding the number of code pages? From which OS data structure is this data found and how?
2. **(Stage 17)** In the implementation of `Exec` system call, how is it ensured that the execution of the new program loaded starts with the correct IP value?
3. **(Stage 18)** There are three processes currently active in the system, other than IDLE. Two of these processes wishes to read one disk block into a memory page, and the third process does not make any software interrupts. Draw the state transition diagram indicating how the state of each process changes over time till disk operations of both processes are completed (are back in user mode). In case there are multiple possibilities, explicitly mention the same.
4. **(Stage 18)** What are the different ways in which a process can enter WAIT_DISK state? How does it come out of the WAIT_DISK state in each of these ways, and what does it do in each of these cases when it eventually gets scheduled on the processor?
5. **(Stage 19)** How is it ensured during a page fault exception handling that after loading the required page to memory, the execution in user mode restarts with the same instruction which caused the exception?
6. **(Stage 19)** Show the sequence of function calls, and the corresponding process state when a page-fault occurs in the system, but no free pages are available in the system. What happens in this case?
7. **(Stage 20)** Suppose we want to modify the fork system call implementation such that after a successful fork, the child process is started in the user mode before returning to the parent process. What changes would you need in your kernel code to implementing this?
8. **(Stage 20)** Consider the following scenario:
   - A parent process is not having any HEAP pages allocated to it.
   - The parent process forks a child process.
   - The parent process then `Alloc()`'s a HEAP page.

   Would the pages that were newly `Alloc()`'ed by the parent process be shared with the child process as well? Why or Why not?

[END]