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

CS3110: Operating Systems Lab

12th September, 2019 **Viva prelims** **2:00pm - 2:30pm**

1. **(Stage 15)** After a process makes a `write` system call for performing a console write, what are the possible process states the process (which made the system call) could go through before changing to `Running` state again? Is it possible that the process does not change its state at all? Why?

2. **(Stage 15)** Consider the situation when a process is making a `write` system call to write some data to the terminal.

   - While servicing this request, in the device manager module, there is a call to `AcquireTerminal` function. Suppose you put a breakpoint just before this call. At this breakpoint, what is/are the possible process state(s) of the process which made the `write` system call? Why?
   - In the `AcquireTerminal` function, consider the situation after checking the `STATUS` field in the terminal status table and seeing that it is busy and before changing the process state to `WAIT_TERMINAL`, what is/are the possible process state(s) of the current process? When was the process state set to this value?

3. **(Stages 15-16)** The `STATUS` field in the terminal status table is sufficient to keep track of whether the terminal is busy or is free. Then, what are the reasons to have a `PID` field in the terminal status table? What will happen if this field is not set?

4. **(Stage 16)** Suppose you have implemented upto stage 16.

   - Imagine a situation where the terminal status was free and two user processes made read system calls to read from the terminal. Is it guaranteed that they enter user mode back in the same order as they made the read system call? Why?
   - `TerminalWrite` function calls both the `AcquireTerminal` and `ReleaseTerminal` functions whereas the `TerminalRead` function calls only `AcquireTerminal`. Then, how is the terminal status again made free after receving the data in the input port?

5. **(Stage 17)** Suppose we include the `ReleaseBlock` function's code as part of the `ReleasePage` function and perform the actions related to `ReleaseBlock` whenever `ReleasePage` is called and altogether remove the `ReleaseBlock` function. What will go wrong while handling the `exec` sytem call?

6. **(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?

7. **(Stage 18)** Suppose the disk interrupt handler is modified, so that it makes only one of the processes in `WAIT_DISK` state (specifically, the process whose pid is in the Disk Status Table) to ready state. Then, what will go wrong?

8. **(Stage 19)** Suppose you have implemented upto stage 19. At this stage, demand paging is implemented only one code page of a process is loaded into memory initially. Consider a user program which has two code pages and no heap pages. When an exception due to page fault occur for this process for the first time, what would be the values of `EC` and `EPN` registers? At this time, is it possible to predict the value of `EIP` register precisely? Why or why not?

[END]