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

CS3110: Operating Systems Lab

14th November, 2019 **End-sem exam** **2:00 - 6:00 pm**

Instructions

1. The exam has two parts: (i) design (1 hour), and (ii) implementation (3 hours).
2. You can start the implementation section only if your design document has been approved.
3. The design document should clearly mention how you will go about implementing the suggested changes.
4. **SUBMISSION INSTRUCTIONS**:
   - Rename the SPL files so that the given `in.txt` will work without any errors on the xfs-interface.
   - Put the SPL files into a folder named `spl_progs/`, and any EXPL files you wish to submit into a folder named `expl_progs/`. Zip the folder(s) into a file named `ROLL_NUMBER.zip`. Upload `ROLL_NUMBER.zip` to Moodle.
   - DO NOT INCLUDE `.xsm` FILES IN YOUR SUBMISSION.

1. **(Stage 20-22) A Modified Semaphore in ExpOS:**
   In the current implementation of Semaphores in ExpOS, when the process that has currently locked a semaphore calls the "Release Semaphore" function to unlock the semaphore, the "Release Semaphore" function wakes up **all** the processes waiting for this particular semaphore.

   In this question, we want you to modify the current implementation of "Release Semaphore" so that it will wake up only **one** process instead of all processes waiting for that Semaphore. To do this, the "Release Semaphore" would set the state to READY of only the process whose PID is next in the round-robin order of the process that called the "Release Semaphore" function.

   Note that you have to log the system activity by putting print statements that indicate the following:

   - The first kind of log message is:
     `PID`
     `ACQUIRES/LOCKS/UNLOCKS/RELEASES`
     `SEMTABLEINDEX/SEMID`
     This will be printed as a process acquires/locks/unlocks/releases a semaphore respectively.
   - The second kind of log message is:
     `PID`
     `READY`
     This will be printed when the state of a process is changed to `READY` from `WAIT_SEMAPHORE`.

2. **(Stage 23-26) Link-unlink in ExpOS:**
   A "Link" is a file of type LINK (a new type) that points to another file in the file system called the "Target file". Therefore, one can access the target file using the link just like he/she would directly access the target file. The "Open"/"Exec" works on the target file through the link in the same way as it would have worked on the target file directly. However, deleting a link will only delete the link file, and not the target file itself.

   The above can be implemented using the system call "Test1" provided in the ExpOS already. Look at the Disk and Memory Organization, and System Call interface given in the following links to see how you can implement "Test1" (Interrupt Routine 18) system call (number 97).

Listing 1 is the specification of "Test1" system call (to create a link).

Listing 1: Test1 system call

```
Function code : "Test1"
        Arg1: "link_name"
        Arg2: "target_file"
        Arg3: Unspecified
Return Value  :
         0 Success or link_name already exists and points to the target_file
        -1 if link_name already exists and points to a different target_file
        -2 if the file "target_file" does not exist
        -3 if the target_file is not DATA or EXEC
        -4 if no free inode table entry exists
```

In this question, you are required to implement the Link-Unlink mechanism into ExpOS that meets the requirements listed below.

Requirements:

- The system call "Test1" should create a new file with name "link_name" of type LINK, that is pointing the target file "target_file" and return values as specified above.
- You should be able to call the "Open" system call with link_name as the first argument.
- If the target file linked to link_name is not a DATA file, then "Open" system call should return a failure code -1. Otherwise, the call should function so that the file descriptor returned is actually referring to a new open instance of the target file.
- The actions performed when the "Read"/"Write"/"Close" system call that uses the file descriptor returned when a file 'link_name' is "Opened" as mentioned above, should be as if the call was made on an open file instance of the target file.
- Deleteing the target file should not be permitted, till all open instances of it (either directly or through a link) are closed.
- The call to "Delete" system call with link_name as the first argument should only delete the link and not the target file.
- The return values of "Delete" system call should remain unchanged.
- If "Delete" is called on the target file, any links pointing to the target file should also be deleted.
- It should be possible to call "Exec" using link_name for a link to a target file that is executable.
- Permission bit in inode table can be set to "Exclusive" for links so that only the owner of the file and root will be able to delete the link.

[END]