Project 2: Implement various system call focused on process in xv6 - Updates

Assignment Notes

General Notes

* Please read the assignment instructions carefully, including what files to include/submit.
* Don’t assume any limitations unless they are explicitly stated.
* Your implementations should be robust in the way they handle various argument inputs.
* Developing optimized solutions is good, but I won’t deduct any points if your code is not optimized.
* Please ensure the functionalities requested in these assignments are included; you can add extra but cannot remove any.
* When in doubt regarding what needs to be done, ask. Another plausible option is to test it in the real UNIX or Ubuntu operating system. Unix is your friend; use it wisely. Does your code behave the same way? If not, debug. Remember to use man “command name” for reference.
* Test your solutions and make sure they work. Attach your screenshots showing your steps, your console, and other details.

Part-1 Getting process statistics and modifying system call in xv6- 300 pts

As a part of this assignment, we will go deeper into understanding system calls in xv6. First, we will start by extending the current system proc structure and adding a few fields to understand the process life-cycle better. We will add three main fields: The first is the creation time; whenever a process gets created, we want our kernel code to update the process creation time. Similarly, the second is the end time, which provides information when the process gets terminated or finishes its job, and the final field provides the total time. Call your newly formed system call in user-space (Note: it should be robust enough to call uniq or head separately or together. It should also be able to call with variation given specific flags). To do so, you will create a test.c script, which calls two previously built functions uniq and head function in user-space (This will also be able to call newly created functions as you expand your OS implementation). Your new system call should provide, provide all the requested information (creation time, end time, total time) for these two processes (uniq and head). (Hint: You need to understand how ticks work in xv6 and have to extend the wait system call)

Part-2 Implementing ps on xv6 - 300 pts

Once part 1 is complete, you must create a ps command in xv6, which provides information about the process. In your custom ps, you will display PID number, process status (running, zombie, wait, etc.), start time, total time, and process name.

Your implementation of ps should be able to print the status in multiple manners. I.e. if you want to print the status of a specific process, you should be able to define that as an input. If there is no defined input of a specific process as part of the system call, your command would then print the status of all processes.

Submission Instructions / Rubric

Don’t hard code any solution. Ensure you test your program completely and provide detailed information concerning steps followed/adopted to ensure your code is bug-free. This assignment has a project total of 600 points.

These are the following submission instructions:

Modify the Makefile to compile xv6 with the additional files. Include a Project1-README file with the system environment you used to develop/test the code, and any additional information about your implementation or compilation that might be helpful for us in grading. You will also take screenshots of your outputs for each condition. Submit your whole xv6-public directory as a compressed zip file. See the submission section below for details.

* A zip file of the xv6-public with All of your modified/added code, file, and modified Makefile.
* A detailed Project2-README file that explains your logic (code section), and steps followed to run your code (attach a screenshot and then explain steps). You may also include comments here for the graders regarding your implementation approach. If the README file is missing, then we will deduct 50 points.
* Screenshots showing successful execution of the command ‘make clean.’
* Screenshot showing successful execution of the command ‘make qemu’ or ‘make qemu-nox’.
* Folders called part (task)-1, and part (task)-2 with screenshots showing outputs.
* If there is any discrepancy between your README file and the actual implementation—for example, if you haven’t implemented a certain part but claiming so in your README—we will deduct 75 points.