OS-344 Lab - 1

Instructions

- This assignment has to be done in a group.
- Group members should ensure that one member submits the completed assignment within the deadline.
- Each group should submit a report, with relevant screenshots/ necessary data and findings related to the assignments.
- We expect a sincere and fair effort from your side. All submissions will be checked for plagiarism through software, and plagiarised submissions will be penalised heavily, irrespective of the source and copy.
- There will be a viva associated with the assignment. Attendance of all group members is mandatory.
- Assignment code, report and viva all will be considered for grading.
- Early start is recommended; any extension to complete the assignment will not be given.
- Before you start coding, read Chapter 1 of the xv6 book

———————————————————————

Task 1.1: Prime PID Assignment (Easy)

In this assignment, you will implement a simple feature for xv6: **all newly created processes must be assigned process IDs (PIDs) that are prime numbers.** This means whenever a new process is created, its PID should be set to the next available prime number.
Your implementation should make sure that every new process gets a unique prime PID, starting from the first prime (2) and continuing sequentially for subsequent processes.

Deliverables:
——————

- Ensure your changes are functional and xv6 boots and runs as expected.
- Test your implementation by creating multiple processes (you can use existing user programs like ls, cat, etc., or write a simple test).

Hints:
———

- Look into the **kernel folder** of xv6, particularly where processes are created, to figure out where to adjust PID allocation logic.
- You will likely need to implement a helper function to find the next prime number.
- Make sure your changes don't break other parts of the kernel.
- Add your test program (if you create one) to UPROGS in the Makefile, so it gets compiled when you run make qemu.
- The xv6 source code is small; reading through kernel/ folder will help.
- You can find examples of prime-checking functions online or write a simple one yourself (efficiency is not critical for this assignment).

Notes:
———

- This is an **easy assignment**, meant to help you get familiar with navigating and modifying xv6.
- Focus on understanding where PIDs are assigned and how to integrate your changes cleanly.
- Don't forget to recompile xv6 after making your changes:

  make clean
  make qemu

Task 1.2: Implement top Command (Moderate/Hard)

In this assignment, you will implement a simple version of the top command for xv6. Your top program should print the following information for all currently running processes:

- Process ID (PID)
- Process State (e.g., RUNNING, SLEEPING, ZOMBIE, etc.)
- Number of ticks (CPU time consumed by the process)
- Process Name

Your implementation should display this list in a readable tabular format, similar to the standard top command (but simplified).

Deliverables:

———————

- Write your top program in user/top.c.
- The program must call into the kernel to retrieve the list of processes and their information.
- Display the output neatly (one process per line).
- Ensure that your program works correctly by running top from the xv6 shell.

Hints:

———

- Look into the kernel folder to understand where process information is maintained (e.g., process table structures).
- You will likely need to make kernel changes to expose process information to user space via a **new system call**.
- You may also need to modify header files in the user folder so your program can access this new system call.
- Use existing examples (like how ps or similar tools are implemented in xv6) as a reference for structure.
- Add top to UPROGS in the Makefile so it gets compiled when you run make qemu.
- Start by printing raw process information; once it works, format it cleanly.
- To get process state names (RUNNING, SLEEPING, etc.), check how these states are represented in the kernel (proc.h).

Notes:

———

- This assignment is slightly more involved than the previous one because it requires kernel and user-level modifications.
- Focus on understanding how the process table works in xv6.
- Don’t forget to recompile xv6 after your changes.

    make clean
    make qemu