ENEE 447 Project 6 Writeup

Grant Hoover

With the existing solution, several bugs can be seen when the multicore kernel is used. Some of the problems I experienced were the PS command creating an infinite loop, dumping the information for the shell thread infinitely; the LED command not actually running the LED program, even after calling HANG; and the kernel calling PS randomly without user input directing it to do so. The kernel also did not behave the same way every time I booted it. I believe the cause of the incorrect behavior is that the cores are not communicating back to the kernel correctly and so shared resources like the IO devices and stack are being accessed by multiple cores at the same time or data is being trampled by another process.

The way I would suggest to fix this issue is to have a single core manage the entire kernel and all other cores. In addition, each core would have its own stack space. The main core would dictate to the others which threads they are to be running, handle all IO requests, and perform all timing and context switching tasks. IO requests from one of the three secondary cores could be run through the main core through software-generated interrupts (SGIs, interrupt IDs 0-15 according to the ARM documentation). ARM specifically states that SGIs are the primary means by which intercore communication should be handled. On each of the three secondary cores, the user-level code would run the same as it does on only one core. User-level programs would not need to be changed to be run on any particular core and IO would appear to user programs the same way it does on the main core.

The IO process can be explained through the following hypothetical process:

- Core 3 needs IO access
- User code calls function that interrupts Core 1 and requests IO
- Core 1 returns result
- Core 3 behaves just like existing solution - waits if available, puts on queue if not
- When ready, Core 1 uses IO and returns result through interrupt to Core 3
- Core 3 user code keeps running as usual

I believe this would be an effective solution because it should solve the problems of the current implementation and would be relatively simple. Additionally, instead of rewriting the kernel from the start to be run on multiple cores at once, this solution should allow for most of the existing structures and code to remain, with new code put on top of the old. Although I doubt this solution would be the most efficient or fastest possibility, I believe its simplicity makes it an attractive option for our OS.