CSC 714 Project Report 2

Nachiket S Deshpande
nsdeshpa@ncsu.edu

"Study of memory leakage on an IBM PowerPC 405LP Embedded Processor and ways to reduce energy consumption, by combining sleep and low-power modes."

Project URL: http://www4.ncsu.edu/~nsdeshpa/project.html

Solved Issues

- Code was refined by removing unnecessary printf's from the signal handler, clock() was replaced by gettimeofday(), and sigaction() was successfully tested and integrated in the code.
- A problem was found regarding the timer granularity associated with application-level timers provided in Linux. The smallest granularity that can be handled is 10ms. Any value lower than this reports results similar to that of 10 ms. The same was observed for 20 ms, 30 ms, etc.
- Two approaches to tackle this problem were brought out:
    a) Use the on-board PIT to refine the granularity to 1ms and even lower.
    b) Patching the existing Linux distribution with KURT Linux (a project undertaken by Kansas University for overcoming the 10ms granularity problem associated with Linux kernel timers, using functions like nanosleep, etc.)
- Successfully combined two separate programs (temp.c which writes into pm_alarm) and signal.c (which simulates the timers) into one single code file, which puts the system to sleep and on waking up, triggers the timers and measures the overhead). Small bugs in it are being worked out right now.
- Currently, an attempt is being made to synchronise the two timers (the internal one which wakes up the system and the setitimer programmed by the user – as it is being seen that the overheads reported are different when one varies the sec and the usec fields of it_value, (which controls the time when the first interrupt is generated), in the alarm calling function. This is being worked upon right now.
- Simulated alarm generating function both under "fully-awake" (before going to sleep) and during the asleep to waking up transition. It was noted that alarm interrupts arrive precisely at or around 10ms in the fully awake status, but they do not arrive on time when the system is in transition from asleep to fully awake. It is seen that the first interrupt and

the second one are mismatched, indicating that the system is fully waking up between the first interrupt and the second (1st 10ms interrupt). So, wakeup overhead (worst-case = 10ms currently). If we compare the differences between the times of the interrupts for the fully awake and the transition phase, we see a difference, which is the wakeup overhead. Thus, the total wakeup overhead will be = overhead associated with 1st interrupt plus overhead associated with the 1st 10ms interrupt.

- The next step would be to refine this 10ms granularity to 1ms or lower, using either the PIT or a software-based solution (on the lines of KURT linux).

Open Issues

- Refining the granularity to 1ms and lower by going on to the hardware level and using the PIT on the board and then finding the new, refined overhead.
- Exploring the KURT Linux angle and trying out the overhead measurements with this new functionality.
- Putting this functionality into a system call and seeing which approach would be better (programming the /proc interface VS doing it with a system call)
- Integration with DVS and further experimentation

Next Steps

- Getting a 1ms granularity (by Nov 18)
- System call implementation (by Nov 22)
- Further experimentation and final sleep overhead reporting (by Nov 28)