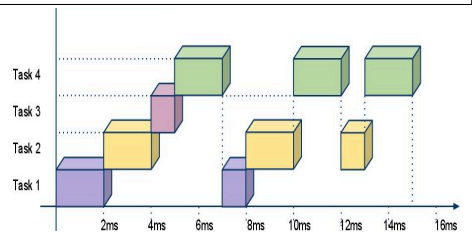
Operating Systems for Wireless Sensor Networks

Researcher: Peter Tueller Mentor: Dr. Cody Youngbull

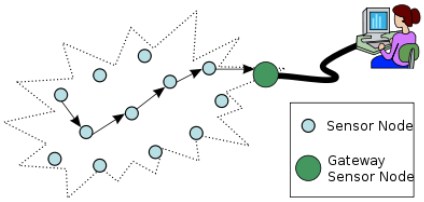
School of Earth and Space Exploration

Abstract: Wireless Sensor Networks (WSNs) pose unique challenges from a software standpoint due to their fundamental reliance on hardware and embedded systems and also due to the relative autonomy of each node in the network; unpredictability can be injected at every level of the system. This project sets out to design an embedded operating system with two primary purposes: to robustly maintain data sampling rates and to rapidly service complex block transfers of data to and from other portions of the network.

Introduction: Much embedded software runs without an operating system (OS), i.e. “bare metal”, which can be easier to debug and design, but is not suited for extension to complex systems. Operating systems can organize multiple pieces of software, or tasks, to run in parallel, but the overhead can be a burden.



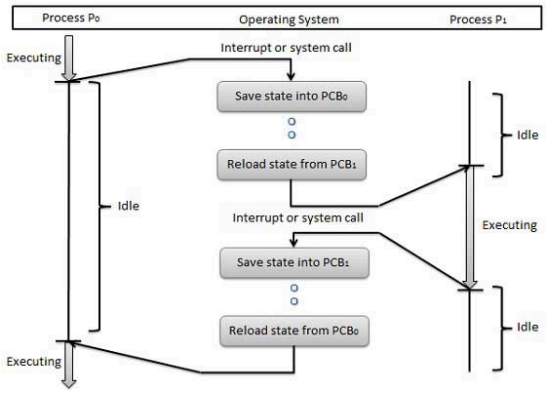
Right: Round-Robin scheduling with interrupts



Left: An example of a wireless sensor network

Complexity: A significant downside of bare-metal code is preemption: i.e. it is difficult to switch from one task to another before the first task has finished. As data rates increase and complexity increases, the tasks become longer and longer, and the likelihood that we will miss a deadline increases.

Environment: Underwater WSNs are unique in that the relative positions of each node is dynamic; ocean currents and wildlife are constantly shifting the geography of the environment. At a moment’s notice, a node may need to be ready to receive data.



Left: Context switching in a uniprocessor

Progress: The slimmed-down OS is still under development. It must have a certain level of completeness before it can be benchmarked or compared against other solutions. The context switching operation has been implemented and tested, which allows two processes to be interchanged. This is done by saving system registers onto the stack of one process and loading the stack of another process, and is achieved using a blend of C and ARM Assembly.

Conclusion: There is still work that needs to be done to determine the effectiveness and viability of this solution, but a code analysis indicates that this OS switches with less overhead than conventional solutions. The true test will come in long-term field applications that are planned for Summer 2016.

ASU/NASA Space Grant Program