KaOS

kaOS waits for a card to be inserted and a reset button to be pressed, at which point a program is loaded from the card and executed.  At any time, a new card with a new program can be inserted and run.  Executing a new program doesn't require reprogramming the Atmel processor.

# High Level Design

The design of kaOS was broken up into two major components:

* The operating system itself
* The card reader and program loader.
  + The card reader is accessed via the Atmel's SPI interface by the program loader, which places the program into flash memory. **Once kaOS loads a program, it creates a thread for it and jumps to its main() method.**
  + kaOS also supports messaging between threads as a means of inter-thread communication

# KaOS Specs:

* Main target Atmel Mega32 microcontroller
* Real-time
* Multithreaded
  + Create Thread, Sleep Thread, Suspend Thread, Terminate Thread, Resume Thread, Find Thread
  + Maximum THREAD\_COUNT 8
* Preemptive operating system for the Atmel Mega32 microcontroller, which loads and executes programs from a Secure Digital or MMC card.
* Supports creation of up to 8 threads
* Supports priorities. Threads with the same priority are alternately preempted to give both equal processing time.
* Supports messaging between threads as a means of inter-thread communication.
* No file system is used
* Each thread has :
  + An ID
  + Priority
    - Threads with the same priority are alternately preempted to give both equal processing time.
    - Higher priority threads always run unless they sleep.
    - Equal-priority threads are swapped back and forth by the scheduler.
    - Priorities between 0 and 255
  + State
  + Sleep duration (if any),
  + Message box to facilitate inter-thread communication
    - Maximum MESSAGE\_COUNT 16
  + Its own stack, the stack size is defined by the user program and each thread can be created with a custom-sized stack.
    - Maximum STACK\_SIZE 256
* Doesn’t support hardware interrupts
* For synchronization, Semaphores are used. kaOS supports creation, waiting and signaling of semaphores. It’s implemented as a list of active semaphores and a list of free structs. Each semaphore holds a pointer to the threads waiting on it. When a semaphore is signaled, a waiting thread is removed from this list and reactivated.
  + SEMAPHORE\_COUNT 8
* The scheduler is called every 10ms and schedule a new thread if necessary. The scheduler keeps all active threads in a queue. Each time the scheduler runs, it finds the highest priority active thread, choosing the least-recently scheduled in case of a tie. It removes it from the queue, schedules it, and places it at the end of the line.
* States:
  + VOID
  + RUNNING
  + BLOCKED
  + SUSPENDED