COMP 4550
Tasks and Threads

Autonomous Agents Lab, University of Manitoba
jacky@cs.umanitoba.ca
http://www.cs.umanitoba.ca/~jacky
http://aalab.cs.umanitoba.ca
RTOS Design

- As seen in the example of the line tracking robot, it is almost always better to use separate threads in robotics, because
- Robotics applications must react to asynchronous events
- Code is easier
  - develop
  - debug
  - maintain
- Separation of concerns
RTOS Design

- Design a small RTOS which can be used for robotics applications
- Requirements
  - Manage threads
  - Thread synchronization
  - Memory
- Constraints
  - Small memory footprint (4KB static RAM)
- Simple to develop
RTOS Design: Multi-tasking

- OS simulates parallel execution on a single CPU by switching very quickly between various threads
- Time-multiplexing
- Cooperative
  - Threads *yield*, i.e., return control back to the OS
  - A badly programmed thread will bring the OS down
- Pre-emptive
  - A timer interrupt (system tick) interrupts a running thread, saves the thread state and starts the next thread
RTOS Requirements: Thread Creation

- Manage threads/processes
- Create threads/processes
  - Statically – fixed number of threads
  - Dynamically – variable nr. of threads
- Stop/kill dynamic threads
  - Adds a lot of complexity since the thread needs to release all resources, otherwise may lead to deadlock
- Deadlock occurs if no process can proceed because it is waiting for a resource owned by a different thread.
  - Dining philosophers problem
RTOS Requirements: Priorities

- Thread priorities
- Often have threads that are more/less important than others
- Number of priority levels (4)
- Static/dynamic priorities
RTOS Pitfalls: Priority Inversion

- Priority inversion is a phenomenon which can arise in a concurrent programming environment where a high priority task (H) is blocked by a low priority task (L), e.g. because L has locked some resource needed by H, and L then has to wait for a medium priority task (M). The net result is that H ends up waiting for M instead of the other way round - the priorities become inverted.
- This can be a problem if, for example, M takes a long time, causing H to miss a deadline.
- A possible cure is to have tasks inherit the maximum priority of any task that is waiting for them. In that case L temporarily becomes high priority until H can proceede, thus preventing M from running in place of H.
RTOS Requirements: Synchronization

- Counting semaphore primitives
  - P (wait), V (increment, signal)
  - P try-to-decrement must be atomic
- Disable context switching
- Disable interrupts
- Others
  - Must work with avr-gcc and avr-libc
- Use a timer to generate a 50Hz interrupt to drive the thread scheduler
- Idle thread
- Stack management
- Reduce memory usage
- Register saves
- Stack memory