MIT 16.07 Lecture 28

Real Time Operating Systems

Part II

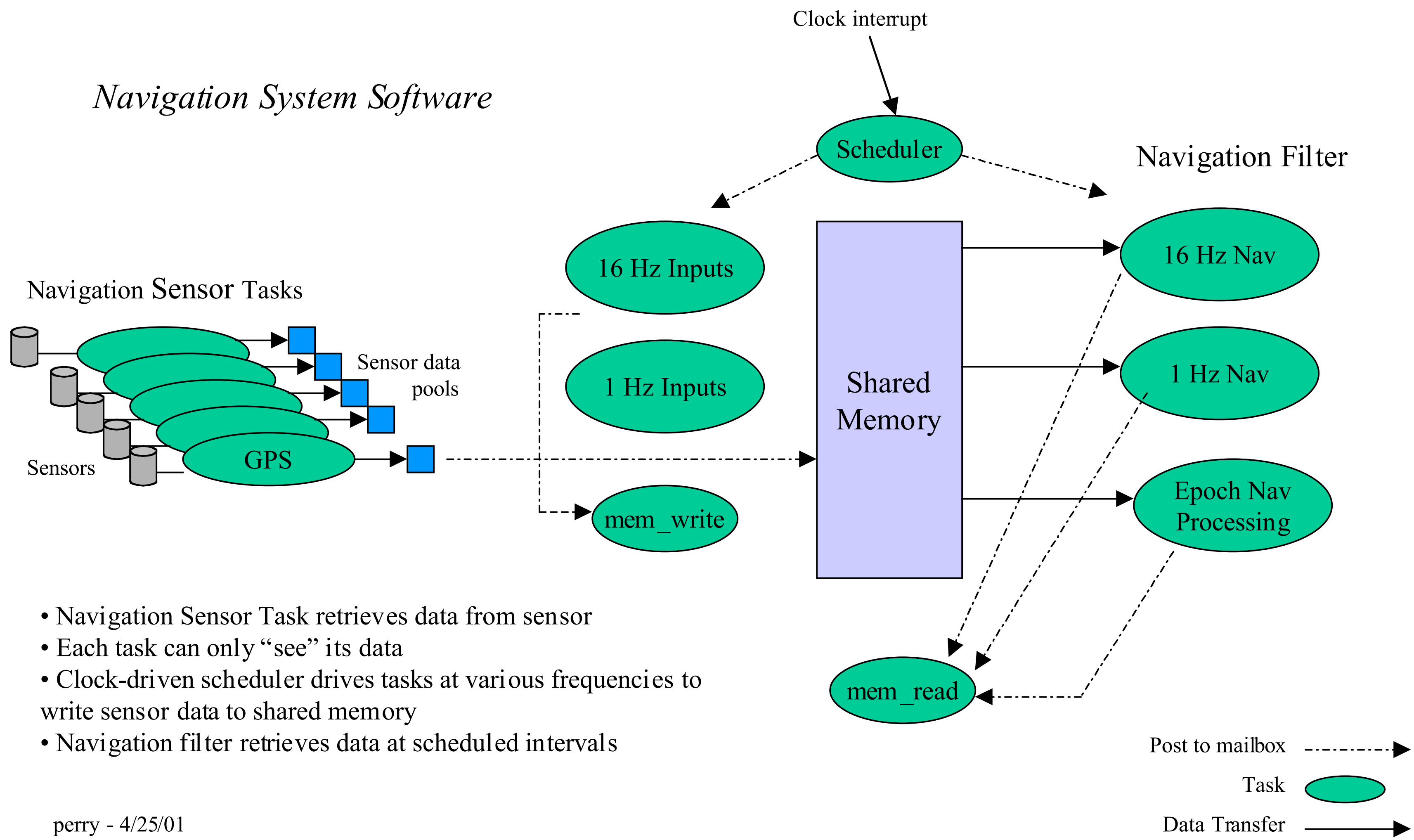
Realtime Kernel Design Strategies

- Polled Loop Systems
- Interrupt Driven Systems
- **Multi-tasking**
- **Foreground / Background Systems**
- **Full Featured RTOS**

Multitasking

- What is Multitasking?
  - Separate tasks share one processor (or processors)
  - Each task executes within its own context
    - Owns processor
    - Sees its own variables
    - May be interrupted
  - Tasks may interact to execute as a whole program

Multitasking Example



- Navigation Sensor Task retrieves data from sensor
- Each task can only "see" its data
- Clock-driven scheduler drives tasks at various frequencies to write sensor data to shared memory
- Navigation filter retrieves data at scheduled intervals

Multitasking

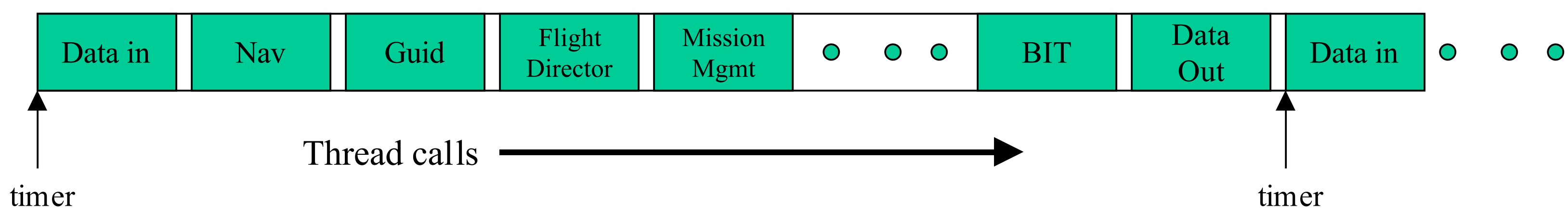
- Context switching
  - When the CPU switches from running one task to running another, it is said to have switched contexts.
  - Save the MINIMUM needed to restore the interrupted process
    - Back to the book example, what might be needed? *(name, page, paragraph , word#)*
  - In a Computer System, the MINIMUM is often
    - contents of registers
    - contents of the program counter
    - contents of coprocessor registers (if applicable)
    - memory page registers
    - memory-mapped I/O
    - special variables
  - During context switching, interrupts are often disabled
  - Real Time Systems require minimal times for context switches

Multitasking

- How do many tasks share the same CPU?
  - Cyclic Executive Systems
  - Round Robin Systems
  - Pre-emptive Priority Systems

Multitasking

- Cyclic Executive
  - Calls to statically ordered threads



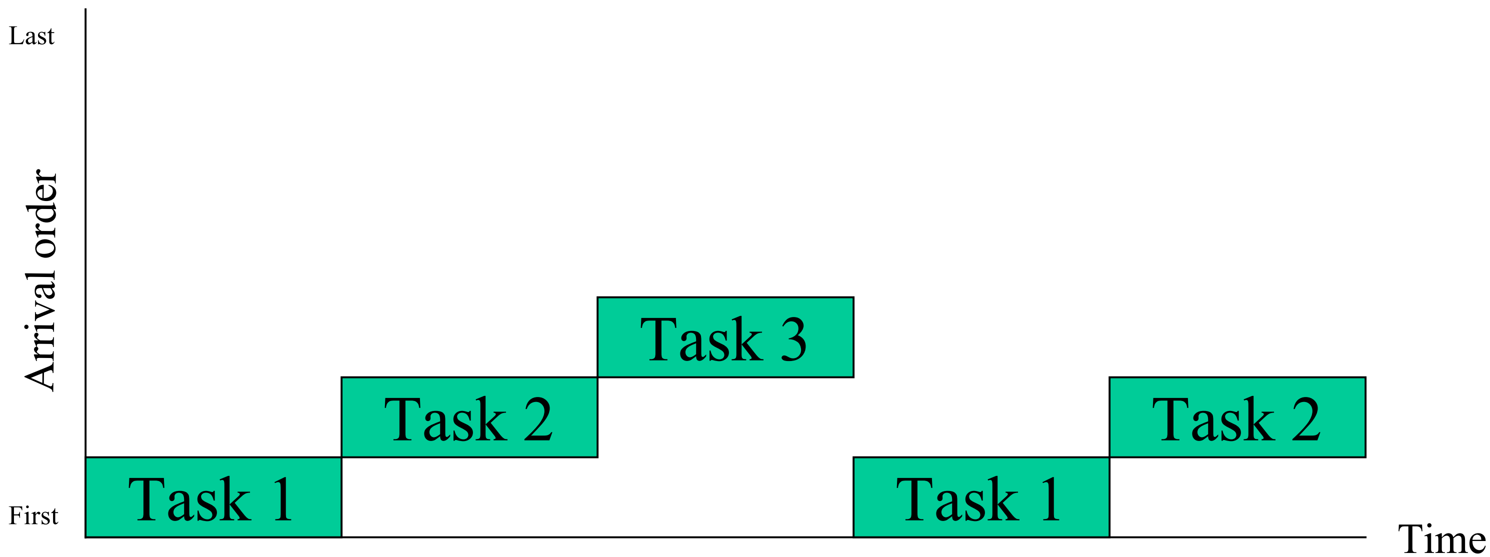
  - Pros:
    - Easy to implement (used extensively in complex safety critical systems)
  - Cons:
    - Not efficient in overall usage of CPU processing
    - Does not provide optimal response time

Multitasking

- Round Robin Systems
  - Several processes execute sequentially to completion
  - Often in conjunction with a cyclic executive
  - Each task is assigned a fixed time slice
  - Fixed rate clock initiates an interrupt at a rate corresponding to the time slice
    - Task executes until it completes or its execution time expires
    - Context saved if task does not complete

Multitasking

- Round Robin Systems - Time Slicing of 3 Tasks

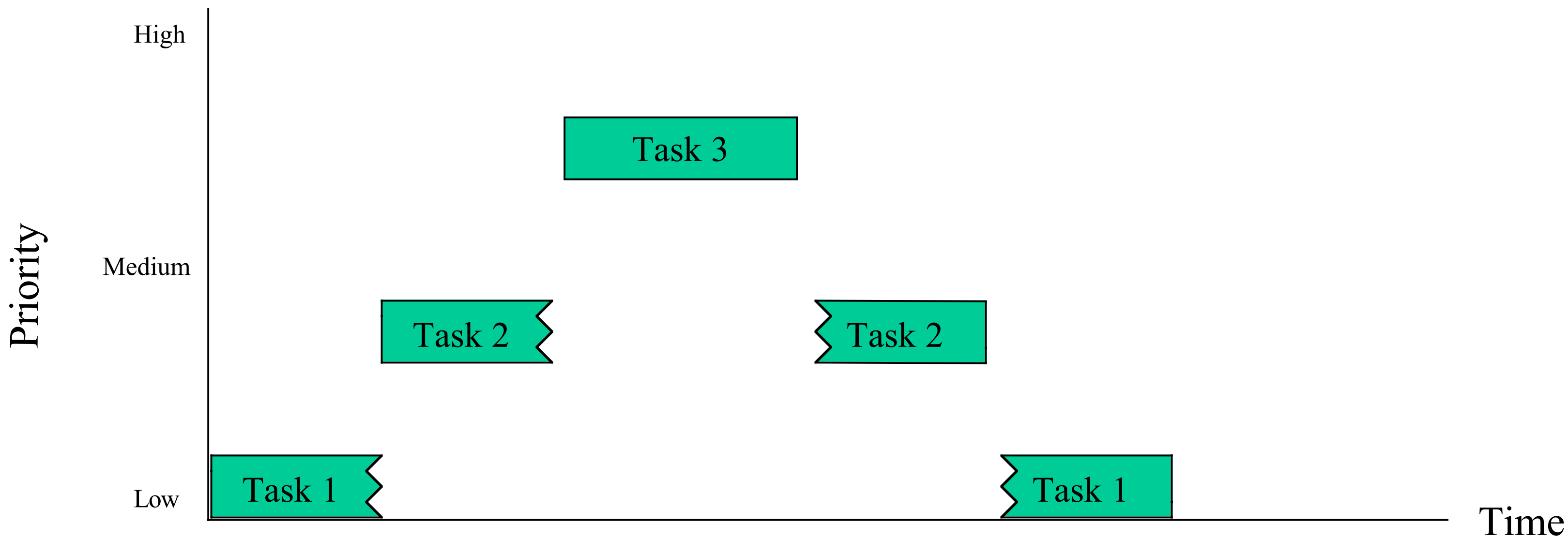


Multitasking

- Pre-emptive Priority Systems
  - Higher priority task can preempt a lower priority task if it interrupts the lower-priority task
  - Priorities assigned to each interrupt are based upon the urgency of the task associated with the interrupt
  - Priorities can be fixed or dynamic

Multitasking

- Round Robin Systems - Preemptive Scheduling of 3 Tasks



Multitasking

- Preemptive Priority Systems - An Example
  - Aircraft Navigation System:
    - High priority task: Task that gathers accelerometer data every 5 msec
    - Medium priority task: Task that collects gyro data and compensates this data and the accelerometer data every 40 msec
    - Low priority task: Display update, Built-in-Test (BIT)

Multitasking

- Multitasking is not perfect
  - High priority tasks hog resources and starve low priority tasks
  - Low priority tasks share a resource with high priority tasks and block high priority task
- How does a RTOS deal with some of these issues?
  - Rate Monotonic Systems (higher execution frequency = higher priority)
  - Priority Inheritance

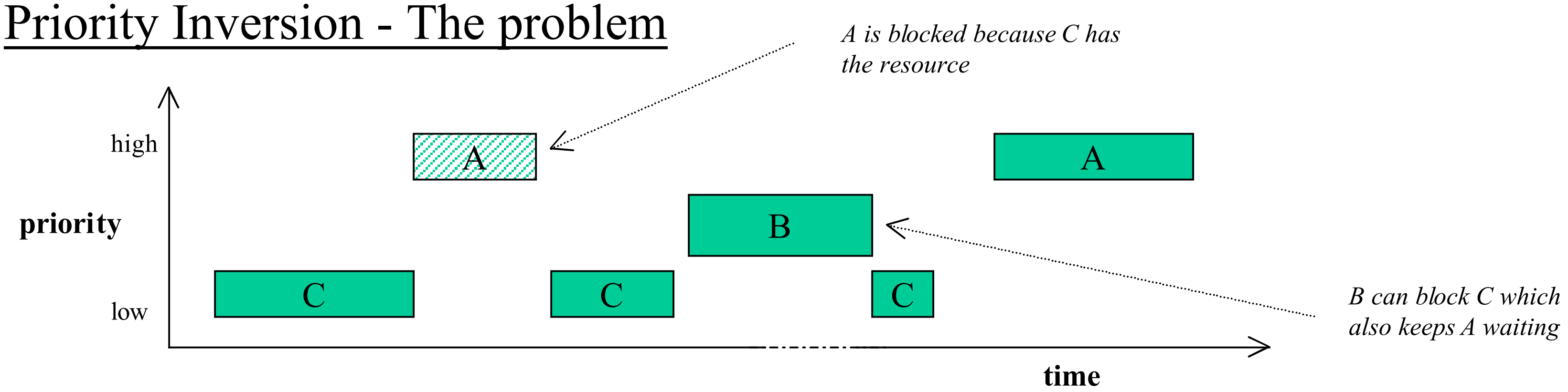
Multitasking

Priority Inversion / Priority Inheritance

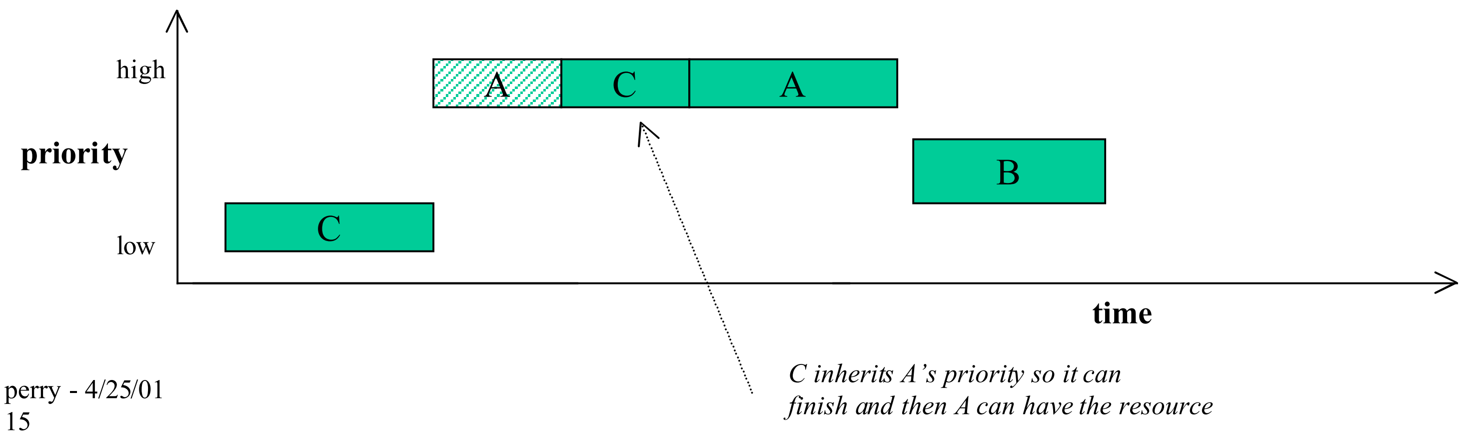
- Task A and Task C share a resource
- Task A is High Priority
- Task C is Low Priority
- Task A is blocked when Task C runs (effectively assigning A to C's priority, hence Priority Inversion)
- Task A will be blocked for longer, if a Task B of medium priority comes along to keep Task C from finishing
- A good RTOS would sense this condition and temporarily promote task C to the High Priority of Task A (Priority Inheritance)

Multitasking

Priority Inversion - The problem



Priority Inheritance - A solution



Realtime Kernel Design Strategies

- Polled Loop Systems
- Interrupt Driven Systems\
- Multi-tasking
- **Foreground / Background Systems**
- Full Featured RTOS

FOREGROUND/BACKGROUND SYSTEMS

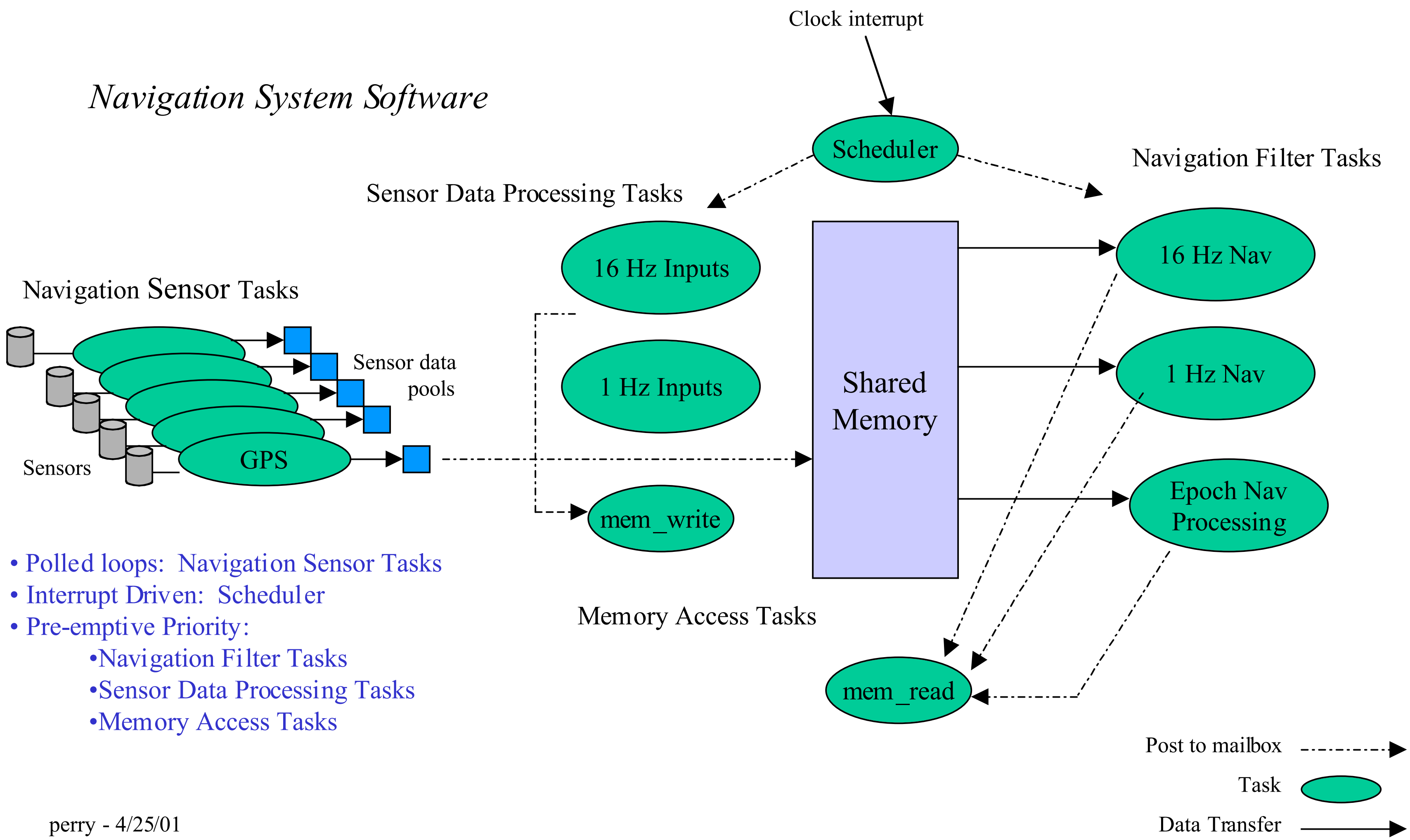
- Most common hybrid solution for embedded applications
- Involve interrupt driven (foreground) AND noninterrupt driven (background) processes.
- All RealTime solutions are just special cases of foreground/background systems
  - Polled loops = Background only system
  - Interrupt-only systems = Foreground only system
- Anything not time-critical should be in background
  - Background is process with lowest priority

Foreground/Background Systems

- Hybrid Systems = Combining what we have learned so far
  - Polled Loops
  - Interrupt-Driven Systems
  - Multi-tasking
    - Pre-emptive Priority or...
    - Round Robin or...
    - Cyclic Executive

A Return to the Multitasking Example

Navigation System Software



- Polled loops: Navigation Sensor Tasks
- Interrupt Driven: Scheduler
- Pre-emptive Priority:
    - Navigation Filter Tasks
    - Sensor Data Processing Tasks
    - Memory Access Tasks

Multitasking Pros & Cons

- Pros
  - Segments the problem into small, manageable pieces
  - Makes more modular software (can reuse portions more easily)
  - Allows software designer to prioritize certain tasks over others
- Cons
  - Depending upon the implementation, timing may not be deterministic (jitter caused by variations in timing of incoming data)
  - Context switching adds overhead

MIT 16.07 - RTOS lecture 2

Realtime Kernel Design Strategies

– Polled Loop Systems
– Phase/State Driven Code
– Coroutines / Cooperative Multi-tasking
– Interrupt Driven Systems
– Foreground / Background Systems
– **Full Featured RTOS**

Full Featured RTOS

- Expand foreground/background solution.
  - Add:
    - network interfaces
    - device drivers
    - complex debugging tools
- Most common choice for complex systems
- Many commercial operating systems available
- Chapter 6.6 in the Real Time Text book goes into more detail

Choosing a RTOS approach

- How do you know which one is right for your application?

  – Look at what is driving your sytem (arrival pattern of data)
    - Irregular (Known but varying sequence of intervals between events)
    - Bursty (Arbitrary sequence with bound on number of events)
    - Bounded (Minimum interarrival interval)
    - Bounded with average rate (Unpredictable event times, but cluster around mean)
    - Unbounded (Statistical prediction only)

  – What is the critical I/O?
  – Are there any absolute hard deadlines?

Choosing a RTOS approach

- How do you know which one is right for your application? Let's look at some real life choices.

  – Reusable Launch Vehicle for satellites. Thrust Vector Control SW requires new attitude data every 40 msec or rocket becomes unstable.
    - *We chose cyclic executive.*

  – Navigation and Control System for submarine. Interface to multiple sensors at multiple data rates. Information from the Inertial Reference Unit is most critical, but exact timing of input data is not essential.
    - *We chose preemptive priority scheme running on a commercial RTOS. Important tasks given highest priority.*

Choosing a RTOS approach

- How do you know which one is right for your application? Let's look at some real life choices.
  - Avionics System requires new data from flight control surfaces, navigation equipment, and radar system every 50 msec.
    - *Cyclic executive. Each task runs to completion. Tasks run in series. Last tasks may not finish before 50msec interrupt occurs.*
  - Microcontroller running to switch radar antennae and check for incoming signal. If the signal is there, power up the signal processing chip.
    - *We chose polled loop.*

Summary

- Multi-tasking involves many tasks sharing resources (CPU)
- Each task executes in its <u>own context</u>
- Multiple tasks can combine to create one program
- There are different ways to implement multi-tasking
  - Cyclic Executive
  - Round Robin
  - Priority-Based Systems
- Some systems are built by combining different RTOS constructs
- There is no one right way to build an embedded system, but there are certainly wrong ones. Carefully choose your RTOS approach!