CS452 F23 Lecture Notes

Lecture 03 - 14 Sep 2023

1. Track and Trains

- trains:
  - 0-14 forward, 15 for reverse
  - add 16 to turn on headlights
- trains slow down gradually
  - do not directly reverse a rolling train
- switches:
  - 0x21 straight, 0x22 turn (curved)
  - changes a switch setting actives a solenoid
    - it must be deactivated 100-500ms after sending the switch command
    - deactivate by:
      - sending another switch command (to any switch)
      - send the special solenoid off command (0x20)
        - single byte - turns off solenoid from last switch
  - middle “double” switches
    - avoid “C/C” state
    - if curved left (CS), and want curved right (SC)
      - first set left switch to S (SS)
      - then set right switch to C (SC)
- sensors
  - 5 banks of 16 up to 16 sensors
    - banks labeled as A, B, C, D on track
    - individual sensors labeled A13, A14
    - sensors are directional: A15 ->, A16 <-
    - when trains trigger sensor, changes sensor state in Marklin controller from 0 to 1
  - Can read the state of all sensors by sending command 0x85 (0x80 + 0x05)
    - means read state of all banks up to and including 5th banks
    - then read 10 bytes of data from the Marklin
      - two bytes (16 bits) per bank
  - Can also read individual banks (0xc0 + bank number)
  - Sensors can be in one of two reset modes, “on” or “off”
    - if reset mode is on, the sensor bits are cleared each time they are read’
    - if reset mode is off, reading the sensor bits does not clear them
  - The single-byte command 0xc0 can be used to turn the reset mode to “on”

2. Concurrency

- In A0, single sequential thread of execution

  - inflexible - not sure how often or when different events will arise
  - how to prioritize some work over other work?
- Want to move to a task based parallelism model
  - mulitiple concurrent tasks, each one a sequential thread of execution
- Want to be able to *prioritize* tasks

3. Polling vs. Interrupts

- Want to be able to wait for events using interrupts, rather than polling for them

4. Context Switching: Two Tasks (abstract idea)

- task state includes
  - program text, data, stack (in-memory)
  - processor state (values in registers)
    - 31 general purpose registers (X0-X30)
    - PC
    - SPs (banked, EL 0-3)
    - pstate - condition codes, processor mode, interrupt masks
- context switch from Task 1 (T1) to Task 2 (T2) (high level)
  - save T1's processor state somewhere in memory
    - must save it without changing it
  - restore T2's saved processor state from memory to registers
- T2 should now resume where it left off
- eventually, T2 can switch back to T1 - which will resume where it left off
- But:
  - how to actually save/restore state?
  - what causes these switches?
  - which task to switch to?

5. Kernel and Exceptions

- kernel: the thing that manages tasks
  - creation, scheduling, termination
- exception (informal): event that causes a context switch
  - system call (synchronous exception)
  - interrupt (asynchronous exception)
- timeline:
  - task T1 is running
  - exception occurs, context switch from T1 to kernel
  - kernel runs, decides which task should run next (say T2)
  - kernel returns from exception, context switch from kernel to T2

6. Generic view of general-purpose kernel

- creates and manages tasks
- priviledged code - protected from tasks, protects tasks from each other
    - priviledge is enforced by the hardware
- resource manager
- device manager
- top half handles requests from tasks
- bottom half interacts with devices

7. CS452 Kernel

- uninterruptible micro-kernel
    - minimal functionality:
        - create, destroy, schedule tasks
        - communication between tasks
        - handle interrupts
    - everthing else, including device management, is left to tasks, which can be prioritized
- privileged code, but memory protection is optional
- though no memory protection, avoid sharing memory between tasks
    - sharing requires synchronization

Author: Ken Salem

Created: 2023-09-14 Thu 11:28