EE4800-03
Embedded Systems Design

Lessons 19 - 22
Real Time Operating Systems

Revised: Dec 15, 2003
Overview

- RTOS Concepts
- Data structures
- Dynamic memory allocation
- Task and Task Control Blocks
- RTOS tracking mechanisms
- RTOS scheduling algorithms
- RTOS issues
RTOS Concepts

• A parable - waitron

• What is RTOS?
  – Multiple events handled by a single processor
  – Events may occur simultaneously
  – Processor must handle multiple, often competing events
  – Wide range of RTOS systems
    • Simple polling through multiple interrupt driven systems
RTOS Concepts

• Each system activity designated as Task
• RTOS is a multitasking system where multiple tasks run concurrently
  – system shifts from task to task
  – must remember key registers of each task
    • called its context
RTOS Concepts

• RTOS subdivided into categories based on the criticality of meeting time constraints:
  – Hard Real Time System: failure to meet time constraints leads to system failure
  – Firm Real Time System: low occurrence of missing a deadline can be tolerated
  – Soft Real Time System: performance is degraded by failure to meet time constraints
RTOS Concepts

- RTOS responsible for all activities related to a task:
  - scheduling and dispatching
  - intertask communication
  - memory system management
  - input/output system management
  - timing
  - error management
  - message management
Dynamic Memory Allocation

• RTOS uses abstract data types such as record, linked list, and queue
• These data types normally use RAM dynamic memory allocation techniques
• Data structures are created (allocated) on the fly during program execution and destroyed when no longer needed
  – Requires large RAM memory
Dynamic Memory Allocation

• Memory allocation command `malloc()` used in conjunction with size of `( )`

```c
ptr = (variable_type *) malloc(sizeof (variable_type));
```

• Memory returned to system when no longer needed using `free()` command

• Heap is portion of memory used for dynamic memory allocation

• Must allocate separate RAM spaces for the Heap as well as the Stack
Data Structures - Record

- Record/Structure
  - Custom design a data type
  - Related information but of different data types

```c
struct car
{
  int year;           /* year of manufacture */
  char make[10];      /* BMW, Hummer, Saturn */
  char model[12];     /* coupe, convertible, SUV, pickup */
  char VIN[10];       /* combination of numbers, characters */
  float mileage;      /* odometer reading: 0 to 500,000+ */
  struct car *next;   /* pointer to next car in list */
};

typedef struct car ELEMENT;       /* for a variable type */
typedef ELEMENT *car_temp_ptr;     /* defines pointer to car */
```
Data Structures - Record

• To create (allocate) a record during program execution:

```c
  car_temp_ptr new_car-entry;
  new_car_entry = (car_temp_ptr) malloc(sizeof(ELEMENT));
```
Data Structures - Linked List

• Linked list consists of a node with two parts:
  – data portion: information about node
  – link field: pointer (address) to the next node in list
• Beginning of list called head
• End of list called tail
  – contains null character in link field
Data Structures - Linked List

Link List Operations

- initialize_link_list
- insert_linked_list
- delete_linked_list
- search_linked_list
- print_linked_list
Data Structures - Linked List
Data Structures - Queue

- Specially configured linked list
- First-in-first-out (FIFO) buffer
- Elements added to rear
- Elements extracted from front
- Queue length variable dependent upon system activity
Data Structures - Circular Queue
Data Structures - The Stack

• Last-in-first-out (LIFO) data structure
• RTOS requires multiple stacks - one for each task
• Stack operations
  – initialize
  – push
  – pull
  – stack_empty
  – stack_full
  – print_stack
Data Structures - The Stack
Task and Task Control Blocks

• In RTOS program consists of independent, asynchronous, and interacting tasks
• All tasks are competing for precious processing time
• Task: independent, asynchronous activities
  – small independent program that completes a specific activity
  – Must have capability to store task context
Controlling a Task

- Dormant - task has no need for computer time
- Ready - task is ready to go active, waiting processor time
- Active - task is executing associated activities
- Waiting - task put on temporary hold to allow lower priority task chance to execute
- Suspended - task is waiting for resource
- Resceduled - task is complete, need not be repeated right away
Task Control Block (TCB)

- Task uses TCB to remember its context
- RTOS updates TCB when task is switched
Multitasking System Components - RTOS Tracking Mechanisms

- Task Control Block (TCB)
  - track individual task status
- Device Control Block (DCB)
  - tracks status of system associated devices
- Dispatcher/Scheduler
  - primary function is to determine which task executes next
RTOS Scheduling Algorithms
Polled Loop System

- Sequentially determines if specific task requires processor time
- When task associated actions are complete, operating system continues polling for tasks requiring operating time
- Simple, easy to write and debug
- Cannot handle burst of events, multiple tasks occurring simultaneously
RTOS Scheduling Algorithm
Polled Loop System

- System sequentially polls remote and front panel for switch activation
- Completes selected task
RTOS Scheduling Algorithms
Polled Loop System w/interrupts

- Polling system good fit; however, several time sensitive critical tasks exists
- Example: transistor amplifier overheat
  - employ interrupts
RTOS Scheduling Algorithms

Round-robin System

- Sequences from task to task
- Tasks may run to completion or time-slicing techniques may be used
  - Time-slicing: each task has fixed amount of processor time allocated
- Used for equal priority tasks
- Example: missile patch
RTOS Scheduling Algorithms
Hybrid Systems

• Round-robin scheduling equipped with interrupts
  – Background: round-robin scheduler
  – Foreground: higher priority interrupts

• Example: missile patch with flooded launch tube, fire, etc.
RTOS Scheduling Algorithms

Interrupt Driven System

• Main program consists of system initialization activities
• System then placed in continuous loop to wait for interrupt driven events
• System prioritizes multiple interrupts and handles highest priority tasks first
• Example: Wall-following Robot
RTOS Scheduling Algorithms
Cooperative Multitasking

• Highest priority ready task executes for some amount of time
• Task then relinquishes control back to operating system at convenient break point
  – TCB updated when control relinquished
• Task re-enters ready state
• System then determines next task for execution
• Implemented with series of linked lists
RTOS Scheduling Algorithms
Cooperative Multitasking
RTOS Scheduling Algorithms
Pre-emptive Priority Multitasking

- Operating system determines when a task should relinquish control
  - Examines linked lists of ready tasks and chooses task with highest priority to place in active state
RTOS Issues

- Concurrency: prevent two tasks from using the same critical resource simultaneously
- Reentrancy: a function is said to be reentrant if it always works correctly and preserves data even if interrupted and restarted
- Communication: intertask communication
  - employ global variables or mailbox techniques
- Safety, verification, fail-safe operation