Real-Time/Embedded Operating Systems & Resource Management

Tei-Wei Kuo, Ph.D.
ktw@csie.ntu.edu.tw
Dept. of Computer Science & Information Engineering
National Taiwan University
Taipei, Taiwan, ROC

Contents

- Overview
- A General Architecture of Real-Time/Embedded Operating Systems
- Scheduling Strategies & System Analysis
- Process Synchronization over IPC
- Summary

* All rights reserved. Tei-Wei Kuo, National Taiwan University, 2002
Overview

- The Purposes of Operating Systems
  - Convenience
  - Efficiency

- Characteristics of Many Real-Time/Embedded Applications
  - More specific in their applications.
  - More drastic for their failures.

Overview

- A Typical Control System Example

- Rates - sensors & actuators, peripheral, control program
- Phases - takeoff, cruise, and landing, etc.
Overview

- Potential Timing Hazards:
  - Loop
    - ???Multiprogramming???
      - Sensor();
      - computation……
      - t = time();
      - SleepTime := ReadyTime + PERIOD - t;
      - ReadyTime = ReadyTime + PERIOD;
      - Sleep(SleepTime);
    - EndLoop;

- General Concerns:
  - Could I verify the performance of my system?
  - How to avoid timing hazards?
  - Is there any good way in scheduling processes or allocating resources?
  - Understand your operating system and hardware, and use resources intelligently!
Contents

- Overview
- A General Architecture of Real-Time/Embedded Operating Systems (RTOS’s)
- Scheduling Strategies & System Analysis
- Process Synchronization over IPC
- Summary

A General Architecture of RTOS’s

- Various Requirements
  - Predictability – Verifiability
  - Reliability – Strictness of Deadline Violations
  - Reconfigurability – System Size and Functionality
  - Efficiency of System Components – Time Granularity, Threads, and Resource Management
  - Variable Models of Task Communication – Characteristics of Applications

* All rights reserved, Tei-Wei Kuo, National Taiwan University, 2002.
A General Architecture of RTOS’s

- Objectives in the Design of Many RTOS’s
  - Efficient Scheduling Mechanisms
  - Good Resource Management Policies
  - Predictable Performance
- Common Functionality of Many RTOS’s
  - Task Management
  - Memory Management
  - Resource Control, including devices
  - Process Synchronization

A General Architecture

User Space

----------
OS

processes

Top Half

Bottom Half

hardware

System calls such as I/O requests which may cause the releasing CPU of a process!

Timer expires to
- Expire the running process’s time quota
- Keep the accounting info for each process
Interrupts for Services

* All rights reserved, Tei-Wei Kuo, National Taiwan University, 2002.
A General Architecture

- **2-Step Interrupt Services**
  - Immediate Interrupt Service
    - Interrupt priorities > process priorities
    - Time: Completion of higher priority ISR, context switch, disabling of certain interrupts, starting of the right ISR (urgent/low-level work, set events)
  - Scheduled Interrupt Service
    - Usually done by preemptable threads
  - Remark: Reducing of non-preemptable code, Priority Tracking/Inheritance (LynxOS), etc.

A General Architecture

- **Scheduler**
  - A central part in the kernel
  - The scheduler is usually driven by a clock interrupt periodically, except when voluntary context switches occur – thread quantum?

- **Timer Resolution**
  - Tick size vs Interrupt Frequency
    - 10ms? 1ms? 1us? 1ns?
  - Fine-Grained hardware clock
A General Architecture

- Memory Management
  - No protection for many embedded systems
  - Memory-locking to avoid paging
- Process Synchronization
- Sources of Priority Inversion
  - Nonpreemptible code
  - Critical sections
  - A limited number of priority levels, etc.

Contents

- Overview
- A General Architecture of Real-Time/Embedded Operating Systems (RTOS’s)
- Scheduling Strategies & System Analysis
- Process Synchronization over IPC
- Summary
Scheduling Strategies & System Analysis

- Why a process in my application does not meet its deadline?
- Factors:
  - Impacts from the executions of higher-priority processes – preemption cost
  - Lengthy execution time of the process
  - Blocking time from lower-priority processes – priority inversion

Possible Questions:

- How do I assign priorities to processes?
- How are my processes scheduled by the OS?
- How long is the blocking time/non-preemptable critical sections (from lower-priority processes or interrupts)?

→ Understand your schedulers
- Fixed-Priorities or Dynamic Priorities
- Preemptive or Non-Preemptive Scheduling
Scheduling Strategies & System Analysis Scheduling Strategy

- Major Components of a Scheduler
  - Priority Assignment Policy
    - The number of priority levels, e.g., 256?
    - Aging Effects?
  - Priority-Driven Scheduling Mechanism
    - Priority Queue
    - Thread Quantum?
    - Preemption Lock – Disabling of Preemption
    - etc.

Rate Monotonic Scheduling Algorithm

- Assumptions:
  - all periodic fixed-priority processes
  - relative deadline = period
  - independent process - no non-preemptable resources
- Rate Monotonic (RM) Scheduling Algorithm
  - RM priority assignment: priority ~ 1/period.
  - preemptive priority-driven scheduling.
- Example: T1 (p1=4, c1=2) and T2 (p2=5, c1=1)
Rate Monotonic Scheduling Algorithm

- Critical Instant
  - An instant at which a request of the process have the largest completion/response time.
  - An instance at which the process is requested simultaneously with requests of all higher priority processes

- Usages
  - Worst-case analysis
  - Fully utilization of the processor power
  - Example: T1 (p1=4, c1=2) and T2 (p2=5, c1=1)

- Schedulability Test:
  - A sufficient but not necessary condition
  - Achievable utilization factor $\alpha$
    - of a scheduling policy $P$ -> any process set with total utilization factor $\sum \frac{c_i}{p_i}$ no more than $\alpha$ is schedulable.
    - Given n processes, $\alpha = \left\lceil \frac{1}{n} \sum_{i=1}^{n} \frac{1}{c_i} \right\rceil$

- Stability:
  - Let processes be sorted in RM order. The ith process is schedulable if
    $$\sum_{j \neq i} \frac{D_j}{p_j} \leq \left( 1 - \frac{1}{n} \right)$$

- An optimal fixed priority scheduling algorithm

---

2. All rights reserved. Tzu-Wei Kuo, National Taiwan University, 2002.
Schedulability Tests – A Sufficient Condition

Theorem 0 [Liu&Layland 73]: A set of \( n \) periodic processes is schedulable if the total utilization factor of the process set is no larger than

\[
\sum_{i=1}^{n} \frac{1}{2^{n-i}} - 1
\]

Theorem 1 [Kuo, et al. 00]: Suppose that \( T_i \) is schedulable. Let \( k \) be the number of roots in \( T_i \). If the total utilization factor of \( T_i \) is no larger than

\[
k \left( \frac{1}{2^{k}} - 1 \right)
\]

then \( T_i \) is schedulable.

Schedulability Tests – Effects of Interrupts

Task \( \Gamma_1 \) : \( C_1 = 20 \text{ms}, P_1 = 100 \text{ms}, U_1 = 0.2 \)
Task \( \Gamma_2 \) : \( C_2 = 40 \text{ms}, P_2 = 150 \text{ms}, U_2 = 0.267 \)
Interrupt : \( C_{\text{int}} = 60 \text{ms}, P_{\text{int}} = 200 \text{ms}, U_{\text{int}} = 0.3 \)
Task \( \Gamma_3 \) : \( C_3 = 20 \text{ms}, P_3 = 350 \text{ms}, U_3 = 0.057 \)

The last task was not affected!
Schedulability Tests – Effects of Priority Mapping

- Ready Queue

Let processes be sorted in RM order. The ith process is schedulable if

\[ \sum_{k=i}^{2n} \frac{C_k}{D_k} \leq \frac{D_i}{C_i} \]

Rate Monotonic Analysis – A Sufficient & Necessary Condition

- Rate Monotonic Analysis (RMA)²
  - Basic Idea:
    Before time \( t \) after the critical instance of process \( \tau_i \), a high priority process \( \tau_j \) may request \( \frac{C_j}{D_j} \) amount of computation time.

  \[ \tau_j \]

  \[ \tau_i \]

  \[ \text{deadline of } \tau_i \]

  \[ \text{Time} \]

  \[ \text{for some } t \text{ in } [0, T] \text{ for } \tau_i \]

- Formula:
  \[ \psi_i(t) = \sum_{j=1}^{n} \epsilon_j \left( \frac{t - t_j}{P_j} \right) \leq t \leq d_i \]

  A sufficient and necessary condition and many extensions...

² Sha, “An Introduction to Rate Monotonic Analysis,” Tutorial Notes, SEI, CMU, 1992
Rate Monotonic Analysis – A Sufficient & Necessary Condition

- A RMA Example:
  - T1(20,100), T2(30,150), T3(80,210), T4(100,400)
  - T1
    - c1 <= 100
  - T2
    - c1 + c2 <= 100 or
    - 2c1 + c2 <= 150
  - T3
    - c1 + c2 + c3 <= 100 or
    - 2c1 + c2 + c3 <= 150 or
    - 2c1 + 2c2 + c3 <= 200 or
    - 3c1 + 2c2 + c3 <= 210
  - T4
    - c1 + c2 + c3 + c4 <= 100 or
    - 2c1 + c2 + c3 + c4 <= 150 or
    - ...

Rate Monotonic Scheduling Algorithm

- RM was chosen by
  - Space Station Freedom Project
  - FAA Advanced Automation System (AAS)
- RM influenced
  - the specs of IEEE Futurebus+
- RMA is widely used for off-line analysis of time-critical systems.
Earliest Deadline First Scheduling Algorithm

- Assumptions (similar to RM):
  - all periodic dynamic-priority processes
  - relative deadline = period
  - independent process - no non-preemptable resources

- Earliest Deadline First (EDF) Scheduling Algorithm:
  - EDF priority assignment: priority ~ absolute deadline. i.e., arrival time $t + $ relative deadline $d$.
  - preemptive priority-driven scheduling

- Example: $T_1(c_1=1, p_1=2), T_2(c_2=2, p_2=7)$

Earliest Deadline First Scheduling Algorithm

- Schedulability Test:
  - A sufficient and necessary condition
  - Any process set is schedulable by EDF iff
    \[
    \sum_{i} \frac{D_i}{C_i} \leq 1
    \]

- EDF is optimal for any independent process scheduling algorithms
- However, its implementation has considerable overheads on OS’s with a fixed-priority scheduler and is bad for (transiently) overloaded systems.
Contents

- Overview
- A General Architecture of Real-Time/Embedded Operating Systems (RTOS’s)
- Scheduling Strategies & System Analysis
- Process Synchronization over IPC
- Summary

Synchronization & IPC

- Why Inter-Process Communication (IPC)?
  - Exchanging of Data and Control Information!

- Why Process Synchronization?
  - Protect critical sections!
  - Ensure the order of executions!
Synchronization & IPC

- Common Facility for IPC?
  - Shared Memory
  - Message Transmission

- Common Facility for Synchronization?
  - Semaphores
  - Signals

Example 1:
```c
flag[i]=TRUE;
while flag[i];          # Deadlock?
  ...
flag[j]=FALSE;
```

Example 2: (initially, `lock=FALSE`)
```c
key=TRUE;
for(swap(lock,key); key==TRUE; )
  swap(lock,key);
  ...
lock=FALSE;
```

- Shared Memory
  - Characteristics: High bandwidth and low latency, but very primitive!
  - Needs: Ways to synchronize its access to avoid racing conditions!
    - Potential deadlocks/livelocks/blocking problems
    - e.g., semaphores, bakery algorithm, etc.
**Synchronization & IPC**

- **Message Transmission**
  - Characteristics: Simple & clean interface with various extensions: m:m communication, multi-machines
  - Needs: A priority-based message transmission/notification/processing mechanism
    - e.g., message priority, non-blocking library functions, notification of msg arrivals, servicing order of msgs (with respect to other threads in the system), etc.

**Synchronization & IPC**

- **General Concerns:**
  - How to enforce mutual exclusion?
    - We should not rely on OS scheduling to avoid race conditions!
  - How to process critical messages first?
    - There is a tight coupling between message processing and OS scheduling!
  - After all, we want to manage the priority inversion problem!
    - How to let critical jobs be done with minimized interferences from less important jobs!
Synchronization & IPC

- Solutions??
  - Synchronization Methods
  - Priority-Driven Resource Scheduling Support!

- Popular Approaches
  - Semaphore-Based Synchronization?!
    - Signals are too slow.
  - Priority Inheritance?!
    - Ceiling?

Synchronization & IPC – Signals?

- The Design of Signal Mechanisms
  - Inform processes/threads of the occurrences of exceptions or events

- Posting of a signal to a process
  - An appropriate signal is added to the set of pending signals for the process.

- Delivering within the context of the receiver
  
  ```
  if (sig = CURSIG(p))
  postsig(sig)
  ```

- Signal handlers: user-mode routines
Synchronization & IPC – Signals

- Questions?
  - How fast can a signal be delivered?
    - Complicated actions done by OS
  - The length of a signal handler?
    - Signal handlers are executed prior to returning control to the user code.

- Real-Time Support
  - Application-defined signals
  - Queuing of signals while being blocked.
  - Signals are delivered in the priority order.
  - etc

Synchronization & IPC – Can we manage the priority inversion problem?

- What are the sources of priority inversion?
  - Synchronization and mutual exclusion
  - Nonpreemptable regions of code
  - FIFO of any other non-priority-based queues
  - Interrupts
  - A limited number of priorities
Synchronization & IPC – Can we manage the priority inversion problem?

- Popular Synchronization Methods
  - Nonpreemptable Critical Sections
  - Highest Locker’s Priority
  - Priority Inheritance
  - Priority Ceiling

Synchronization Protocols

1. Nonpreemptible Critical Section
   Critical sections are executed at an “infinitely” high priority!

   \( \tau_L \)
   \( \tau_M \)
   \( \tau_H \)

   Note that \( \tau_H \) & \( \tau_M \) have no intention to enter a critical section!
2. Highest Locker’s Priority Protocol

Execute critical section at the priority of the highest-priority task that may lock the semaphore(/resource); higher-priority tasks may preempt the critical section.

\[ \tau_L \quad \tau_M \quad \tau_H \quad \tau_{vH} \]

Note that \( \tau_{vH} \) is no longer blocked by \( \tau_L \).

3. Basic Inheritance Protocol (BIP) [Sha87]

Execute the critical section at the priority of the highest-priority task being blocked; higher-priority tasks may preempt the critical section.

\[ \tau_L \quad S1 \quad \tau_M \quad \tau_H \quad \tau_{vH} \]

- Note that \( \tau_M \) is no longer blocked until necessary.
- However, system may be deadlocked or have chained blocking!
4. Priority Ceiling Protocol [Sha87]

BIP + a “Priority Ceiling” rule about when to grant lock requests (see [Sha 87, 90])

- No deadlock & chained blocking at the cost of reducing the concurrency level of the system.
- Blocked-at-most-once.

Summary of Synchronization Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Blocked at Once</th>
<th>Deadlock Avoidance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-preemptible Critical Section</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Highest Locker’s Priority</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Priority Inheritance</td>
<td>Bounded</td>
<td>No</td>
</tr>
<tr>
<td>Priority Ceiling</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Contents

- Overview
- A General Architecture
- Scheduling Strategies
- Inter-Process Communication
- Summary

Summary

- Understand your operating systems and hardware!
- There is no substitute for intelligent resource allocation methods!