Real-Time Systems

Real-Time Operating Systems

Hermann Härtig

TECHNISCHE UNIVERSITÄT DRESDEN

WS 2018/19

Outline

- Introduction
  - Basic variants of RTOSes
  - Real-Time paradigms
  - Common requirements for all RTOSes
  - High level resources
  - Non-Real-Time on RTOS
- Scheduling
- Memory Management
- Example
  - POSIX and Real-Time

Basic Variants of RTOSes

Cyclic Executive

- Only one task as infinite loop
- Time driven, polling for external events

Set of Interrupt Handlers

- Event driven
- Handlers usually have priorities
- Stacked execution: see stack-based priority ceiling protocol

Basic Variants of RTOSes

Thread Packages: iRMX, FreeRTOS, eCos, …

- Use a form of scheduling
  - Preemptive or cooperative
  - Priorities
- Provide synchronization primitives (e.g., semaphores)
  - Some with priority inheritance/ceiling
- No address-space protection, no virtual memory

Basic Variants of RTOSes

Microkernels: QNX, VxWorks, L4/Fiasco, …

- Memory protection: address spaces
  - With or without virtual memory
  - More robustness (fault isolation)
- Extensive functionality provided by services

Basic Variants of RTOSes

Monolithic RTOS: LynxOS, MontaVista Linux, …

- Monolithic kernel with RT API, like POSIX RT
- Often non-real-time APIs as well, e.g., Linux compatibility
- Device drivers etc. usually run in privileged mode
- Real-time applications usually run in user mode

Monolithic with RT executive underneath:
RTLinux, RTAI, XtratuM, …

- Combination of a legacy OS with some form of an RT thread package, usually without memory protection
- Real-time applications run in kernel mode

Basic Variants of RTOSes

Partitioned System

- All resources are statically allocated to partitions: CPU, Memory, Devices
- Isolation in space and time
- Multiple threads/processes in a partition
- Scheduling:
  - Partitions: time driven
  - Threads/processes: any local scheduling scheme possible
- ARINC 653-1 standard for avionics

ARINC 653-1 standard for avionics

Statically partitioned system (time-driven scheduling)

- Execution in one partition must not influence execution in another partition (strict isolation)
- Strictly time-driven scheduling of partitions
- No transfer of idle CPU time among partitions
- Additionally defines
  - System health monitoring
  - Intra/inter-partition communication
  - Time management

Virtual Machines and RT

- Hypervisor provides virtual machines with Guest OS
- Used as partitioned system
  - Address space isolation within virtual machines
  - Sometimes virtual machine is just marketing-speak for virtual memory
- Used to co-locate existing systems
  - Can RT-Properties of Guest OS be preserved?
  - No global scheduling, but hierarchical
  - Enlightened VMs can use hints to influence global schedule

Common Requirement for RT

Time as First Class Citizen

- Periodic processes or absolute timeouts
- POSIX Interface: `clock_gettime`, `clock_getres`
- High clock resolution necessary
  - Special CPU event counters
  - Linux: some clocks readable without system call
- Time synchronization

High-Level Resources

- RT is often reduced to CPU scheduling and latencies
- There are more resources to give guarantees about: Disk, Video, Network, …

Operating systems addressing these are

- Linux/RK (resource kernel)
- Redline
- L4/DROPS
- …

Non-Real-Time on RTOS

Non-RT API on RT kernel

- Unix emulation on QNX
- Linux emulation on LynxOS

Run Non-RT OS on RT kernel

- Xtratum
- Radisys (Windows-NT)
- RT-MACH
- L4Linux on L4

Scheduling in Real-Time OSes

- Priorities
- Priority inversion and countermeasures
- Time budgets
- Interrupt / event latencies

Real-Time Scheduling

Fixed Priorities

- Sufficient priority levels (e.g., RMS 256 priorities)
- Events/messages with priorities
  - Higher priority events arrive first
  - On some systems priority is donated to the receiver
- Signals are queued (predictability)

Real-Time Scheduling

Dynamic Priorities

- Application based: `set_priority`
  - Good for mode changes
  - Not suitable for EDF
- OS-driven EDF scheduling

Scheduling – Priority Inversion

Priority Ceiling

- Set priority of lock
- Critical sections as parameter for process creation

Priority Inheritance

- Borrowing of CPU time (priority)

- Non-preemptive critical sections

Scheduling – Budgets

What if processes abuse their priorities?

Overload situations?

Periodic threads and time quanta

- Assign budgets per period to threads:
  thread = (period, priority, budget)
- Control overuse of budgets:
  - Periodic threads as first class object
  - Watchdog timers to signal budget overruns

Interrupt Latencies

Key Property of RTOSes:

Predictable and low interrupt latency

Interrupt Latency Reduction

- No interrupt blocking for synchronization (preemptivity)
- Short interrupt service routines (‘top halves’)
- Schedule more complex interrupt handling in a thread-like fashion (Linux: tasklets)
- Partition data and instruction caches
- …

Real-Time and Memory Management

Avoid demand paging/swaping

(disk access is orders of magnitude slower than main memory)

- However:
  - Address space isolation useful for robustness/debugging
  - Some scenarios need paging
- Interface
  - `mlock(…)` lock pages in memory (prevent swaping)
  - `munlock(…)` allow demand paging again

Real-Time and Memory Management

Static Memory Allocation

- Good for predictability
- Inflexible

Dynamic memory management

- Use real-time capable memory allocator
- e.g. TLSF

POSIX and Real-Time

POSIX (Portable OS Interface): IEEE 1003.1
REALTIME extensions

- semaphores
- process memory locking
- priority scheduling
- realtime signal extension
- clocks/timers
- interprocess communication
- synchronized I/O
- asynchronous I/O

POSIX: Memory Locking

- Memory ranges can be locked
  (excluded from swaping)
- Provide latency guarantees for memory accesses

POSIX: Real-Time Scheduling

Multiple scheduling policies

- SCHED_FIFO: non-preemptive FIFO
- SCHED_RR: preemptive/time-sliced FIFO
- SCHED_SPORADIC: sporadic server with budget and replenishment interval
- SCHED_OTHER: threads without RT policy
- At least 32 RT priorities

POSIX: Real-Time Signals

Difference to non-realtime signals:

- Queued (also for the same signal number)
- Carry user data
- Ordered delivery

Specific properties

- RT signals are in the range SIGRTMIN to SIGRTMAX
- Handler gets siginfo_t with additional data
- Lowest pending signal (by number) is delivered first

POSIX: Asynchronous I/O

- Initiate I/O
  - `aio_read(struct aiocb *aiocbp)`
  - `aio_write(struct aiocb *aiocbp)`
- POSIX signals for completion
- `aio_suspend(…)` to wait for completion

POSIX: Asynchronous I/O

```
struct aiocbp {
    Int     aio_filedes;          /* file descriptor */
    off_t   aio_offset;           /* absolute file offset */
    Void    *aio_buf;             /* pointer to memory buffer */
    size_t  aio_nbytes;           /* number of bytes to I/O */
    Int     aio_reqprio;          /* prio of request */
    struct  sigevent aio_sigevent; /* signal */
    Int     aio_lio_opcode;       /* opcode for lio_listio */
};
```

POSIX: Real-Time Clocks

Clocks

- Minimum granularity of 20ms (clock_getres())
- Multiple clocks
  - CLOCK_REALTIME (wall clock time)
  - CLOCK_MONOTONIC (system-wide monotonic clock)
  - CLOCK_PROCESS_CPUTIME_ID
  - CLOCK_THREAD_CPUTIME_ID

Timers

- Associated to a specific clock (see Clocks)
- Per process timers (generate RT signals)
- Periodic timers supported (struct timespec)

POSIX: Execution Time Monitoring

Clocks measuring thread/process execution time

- CLOCK_PROCESS_CPUTIME_ID
- CLOCK_THREAD_CPUTIME_ID

Timers connected to these clocks

- Signal deadline misses

Real-Time Paradigms

Time driven

- Static partitioning in time slots
- Scheduler dispatches time slots in a fixed fashion (e.g., fixed cyclic scheduler)

Event driven

- Events: messages, signals, interrupts, …
- Priorities