TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

REAL-TIME OPERATING SYSTEMS SHORT OVERVIEW

HERMANN HÄRTIG, WS 2017/18

OUTLINE

- Basic Variants of Real-Time Operating System
- Scheduling
- Memory Management
- Real-Time OS Examples
- POSIX and Real-Time
- ???? DROPS: Dresden Real – Time OS
- ARINC 653-1 Standard for Avionics

BASIC VARIANTS OF RTOSES (I)

Simplest

- Cyclic Executive
  - Only one “task” as infinite loop
  - Time driven, polling for external events
- Set of Interrupt Handlers (as “tasks”)
  - Event driven
  - Handlers have priorities
  - “Stack-based scheduling”

BASIC VARIANTS OF RTOSES (II)

Thread Packages (iRMX, FreeRTOS, eCos, …)

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

BASIC VARIANTS OF RTOSES (III)

Microkernels

- Memory protection (address spaces)
  - With or Without page-based virtual memory
  - More robustness (fault isolation)
- Extensive Functionality provided on top
  - using collection of server or
  - Non-RT-OS in a virtual machine
- examples: QNX, VxWorks, L4/Fiasco, …

BASIC VARIANTS OF RTOSES (IV)

Monolithic RTOS,

Variant 1

- Monolithic kernel with RT API (POSIX RT …)
- Often non-real-time APIs as well (e.g., Linux compatibility)
- Device drivers etc. usually run in kernel mode
  Real-time applications usually run in user mode
- examples: Linux-RT(preempt patch), LynxOS, …

BASIC VARIANTS OF RTOSES (IV)

Monolithic RTOS,

Variant 2

- RT Executive/“hypervisor” underneath legacy
- Combination of a legacy OS with some form of an RT thread package (Usually no memory protection)
- Real-time applications run in kernel mode
- examples: RTLinux, RTAI, XtratuM …

PARTITIONS AS SPECIAL CASE FOR TIME-DRIVEN (I)

As we know:

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

PARTITIONS AS SPECIAL CASE FOR TIME-DRIVEN (II)

Partitioned System "Time Driven", Pike OS

- All resources are statically allocated to the "partitions"
  - space: Cores, Memory, Devices, …
  - time: based on preallocated time-slots
- Multiple threads/processes in a partition
- Scheduling:
  - Partitions: Time driven
  - Threads/Processes: any local scheduling scheme possible

NON-REAL-TIME ON RTOS

Non-RT API on RT Kernel:

- Unix emulation on QNX
- Linux emulation on LynxOS
- Xtratum, Windows-NT, RT-MACH, L4Linux on DROPS

OTHER RESOURCES

Active resources besides CPU and Memory

- Disk, Network Bandwidth, Video, …
- Addressed for example in Linux/RK, DROPS

MORE REQUIREMENTS FOR RT OS

Time as “First Class Citizen”

- Periodic processes or absolute timeouts
- Interface: clock_gettime, clock_getres
- High clock resolution
  - Special CPU event counters
  - Non-periodic timers (dynamic ticks in Linux)

Time synchronization

REAL-TIME SCHEDULING (I)

Fixed Priorities

- Sufficient priority levels (e.g., RMS 256 prios [1])
- Protocols to avoid Priority Inversion
- 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 (II)

Dynamic Priorities

- Application based: set_priority(p)
  - Good for mode changes
  - Not suitable for EDF
- OS driven EDF scheduling(Linux: EDF sched class)

What if processes abuse their priorities?

Overload situations?

Coop with NON-RT-Priorities??

REAL-TIME SCHEDULING (III)

Periodic Threads and Time Quanta (bandwidth servers)

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

INTERRUPT LATENCIES

One important property of RTOSes:
Low and predictable interrupt latency

Interrupt latency reduction:

- No/short interrupt blocking for synchronization (preemptivity)
- Short interrupt service routines (“top halves”)
- Schedule more complex interrupt handling in a thread-like fashion
- Partition data and instruction caches

AVOID PRIORITY INVERSION

Priority Ceiling

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

Priority Inheritance

- Borrowing of CPU time and priority (Linux)
- Non-preemptive critical sections

REAL-TIME ON MODERN HARDWARE

Increasing unpredictability through

- TLBs (MMU Caches)
- Caches
- Pipelining (write buffers)
- Multi-Master Busses
- “Intelligent” Devices

REAL-TIME AND MEMORY MANAGEMENT

Avoid demand paging/swaping
(disk access is orders of magnitude slower than main memory)

However:

- Address space isolation needed for robustness/debugging
- Some scenarios need paging

Interface

- mlock(...) lock pages in memory (prevent swapping)
- munlock(...) allow demand paging again

REAL-TIME AND MEMORY MANAGEMENT

Static Memory Allocation

- Always good for real time
- Inflexible

Dynamic Memory Management

- Use real-time capable memory allocator (e.g., [3] TLSF)

Examples:

- RT-Java real-time garbage collection
- RT-Java with separation of static/dynamic

LONG I/O OPERATIONS

Asynchronous I/O

Example: POSIX (IEEE Std 1003.1)

- Initiate I/O
  - aio_read(struct aiocb *aiocbp)
  - aio_write(struct aiocb *aiocbp)
- POSIX Signals for completion
- aio_suspend(...) to wait for completion

LONG I/O OPERATIONS(II)

```
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 AND REAL-TIME

POSIX (Portable OS Interface): IEEE 1003.1

REALTIME extensions: asynchronous I/O plus

- Semaphores
- Process Memory Locking
- Priority Scheduling
- Realtime Signal Extension
- Clocks/Timers
- Interprocess Communication

POSIX: MEMORY LOCKING

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

POSIX: REALTIME SCHEDULING

Multiple scheduling policies

- SCHED_FIFO (non-preemptive FIFO)
- SCHED_RR (preemptive/time-sliced FIFO)
- SCHED_SPORADIC (2 prio levels, replenishment interval, and budget, FIFO on active priority level)
- SCHED_OTHER (threads without RT policy)

At least 32 RT priorities

POSIX: REALTIME SIGNALS

Difference to non-realtime signals:

- Queued (for the same 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 is delivered first

POSIX: REALTIME TIMERS/CLOCKS (I)

Clocks

- Min. resolution 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

POSIX: REALTIME TIMERS/CLOCKS (II)

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

POSIX: ASYNCHRONOUS I/O

- Explicitly overlap I/O operations and processing
- See Asynchronous I/O earlier slide

L4/FIASCO REAL-TIME SCHEDULING (I)

Periodic mode for Real time execution

- Period defines deadline and minimum refresh interval for real-time scheduling contexts

Multiple scheduling contexts per thread

- Scheduling context is tuple (Priority, Timeslice length) = Reservation

Time slice overrun

- Thread exceeded reserved time quantum (reservation time)

L4/FIASCO REAL-TIME SCHEDULING (II)

Time slice overrun and Deadline miss

- Signaled via IPC to a special preempter thread

Execution models

- Strictly periodic (constant interrelease times)
- Periodic (minimal interrelease times)
- Sporadic (random interrelease time, hard deadline)
- Aperiodic (random interrelease time, no deadline)

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

TAKE AWAY

- the impact of RT-Theory on RTOS
- architectural alternatives
- lots of little things