CS245 – Lecture 6

Real-time Operating Systems

Tony Givargis

Introduction

- Operating System (OS)
  - Allows multiple simultaneously executing programs to coexist
  - Responsible for hiding the details of the underlying hardware
- Real-time Operating System (RTOS)
  - In addition to above, provides constructs to address timing needs of multiple simultaneously executing programs

Timing

- Absolute deadline for performing a computation (hard real-time)
- Performing most computations within deadline (firm real-time)
- Average case deadline for performing a computation (soft real-time)
- Interrupt latency
- Context switch latency
  - Preemption granularity
- Scheduling problem!

Scheduling Basics

- Static
  - A priori knowledge of task/timing information *I*
  - Schedule *S* constructed during design time
  - Schedule *S* strictly followed during execution
  - Efficient scheduling and deterministic execution
- Dynamic
  - Task/timing information *I* becomes available during execution (as a function of time)
  - Schedule *S* modified during runtime by the scheduler as task/timing information *I* becomes available
  - Account for changing task/timing information *I*

Scheduling Basics

- Preemptive:
  - A task T_i broken into multiple disjoint segments
  - Segments of different tasks T_i and T_j may be interleaved
  - At regular intervals, the scheduler is invoked to choose the next task segment to be executed
- Non-Preemptive
  - A task t, once scheduled for execution will run to finish
  - On completion of a task t, control is returned to the scheduler to choose the next task
- Don’t confuse preemptive/non-preemptive with static vs. dynamic execution

Scheduling Basics

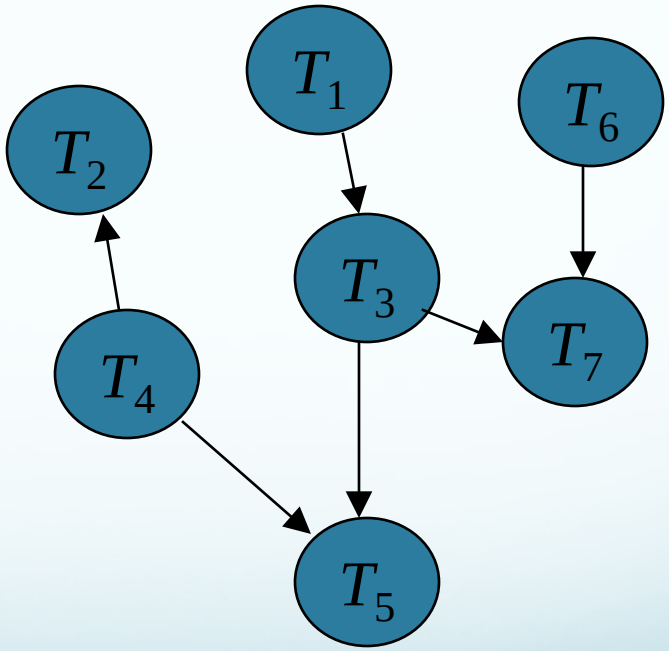
- A task T_i has:
  - Worse case execution time (C)
  - Release time (R)
  - Deadline (D)
  - Period (P)
- In addition, for T_i a scheduler may maintain:
  - Status (S), one of BLOCKED, RUNNING, RUNNABLE
  - Priority (PR)
  - Context, registers, program counter, etc.

Scheduling Basics

- Periodic tasks execute once during each period:
  - Release time (R) = 0
  - Relative Deadline (RD) = Period
  - Static scheduling or quasi-static scheduling
    - Scheduler is invoked when a task is released
  - A notion of a repeating schedule (cycle)
  - Easier to analyze
- Periodic tasks can also be modeled by:
  - Release time (R) = $n \times P$
  - Deadline (D) = $(n + 1) \times P$
  - n is the number of times the task had to execute thus far

Scheduling Basics

- A task graph may impose execution order
- Scheduling is to chose which task to run next (dynamic or static)
- Meet all deadlines and obey execution order defined by the task graph
- Minimize response time
- Minimize energy consumption (DVS)



Priority Scheduling

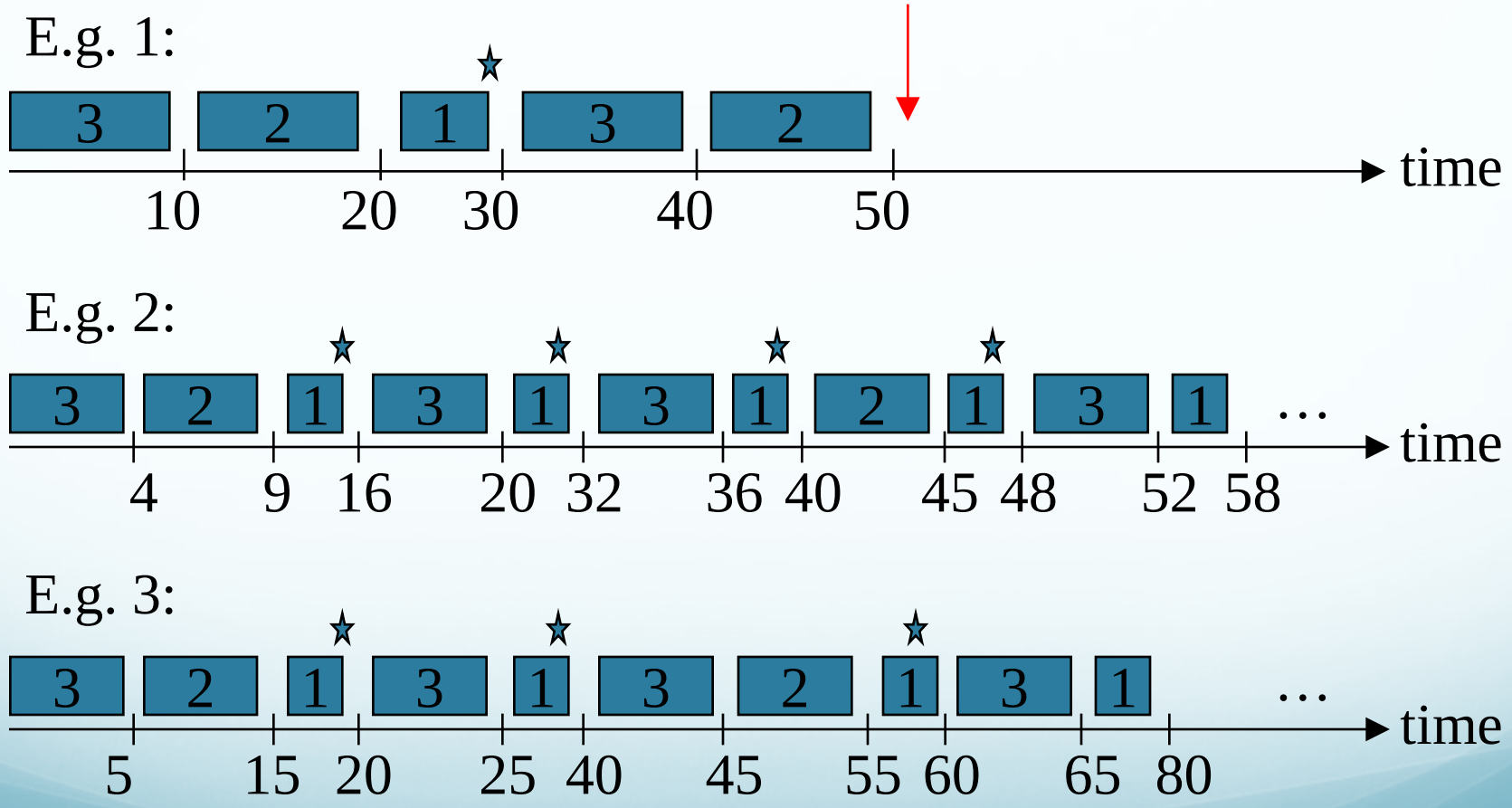
- At any given time, the set of all ready-to-run tasks (i.e., S = RUNNABLE) is maintained in a priority-queue (i.e., using PR to sort the tasks)
- Scheduler, when invoked, chooses the task with highest priority to execute
- Specific scheduling algorithms can be implemented by just selecting the priority

Rate-monotonic Scheduling

- In periodic tasks
  - Priority is inversely proportional to the period (i.e., PR = 1 / P)
- Processor utilization of a single task
  - $U_i = C_i / P_i$
- Total processor utilization
  - $U = U_0 + U_1 + \ldots + U_N$
  - If $U < N \times (2^{1/N} - 1)$ task set can be scheduled
  - As N grows, bound approaches 69.3%!
- Optimal

E.g.	T	P	C	U	U Total	Test
1	1	50	12	0.24	82%	Maybe
	2	40	10	0.25		
	3	30	10	0.33		
2	1	80	32	0.4	77.5%	Yes
	2	40	5	0.125		
	3	16	4	0.25		
3	1	80	40	0.5	100%	Maybe
	2	40	10	0.25		
	3	20	5	0.25		

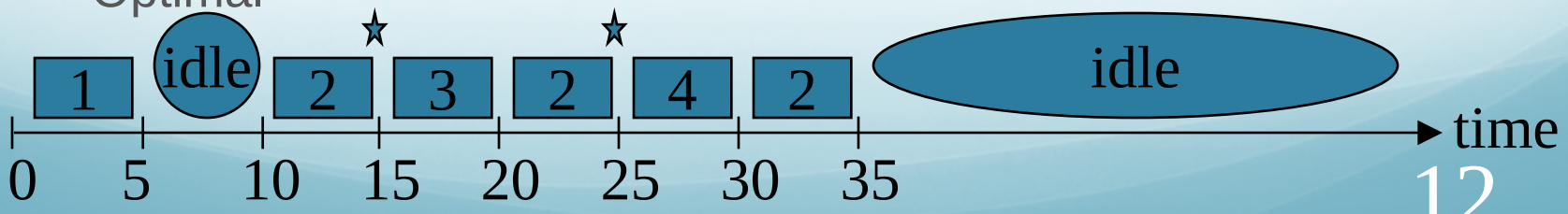
Rate-monotonic Scheduling



Deadline-monotonic Scheduling

- In non-periodic task scheduling
- Priority is equal to 1/deadline (i.e., PR = 1/D)
- A.k.a., earliest deadline first (EDF)
- Utilization test applied to intervals
- Optimal

T	R	D	C
1	0	10	5
2	10	40	15
3	15	20	5
4	25	30	5



Other Scheduling Schemes

- Cooperative
  - Require that the current task to yield to another task
  - Difficult to program
- Round Robin
  - Processor fairly shared among all runnable tasks
  - Default scheduling among tasks with equal priority
  - Not efficient use of processor

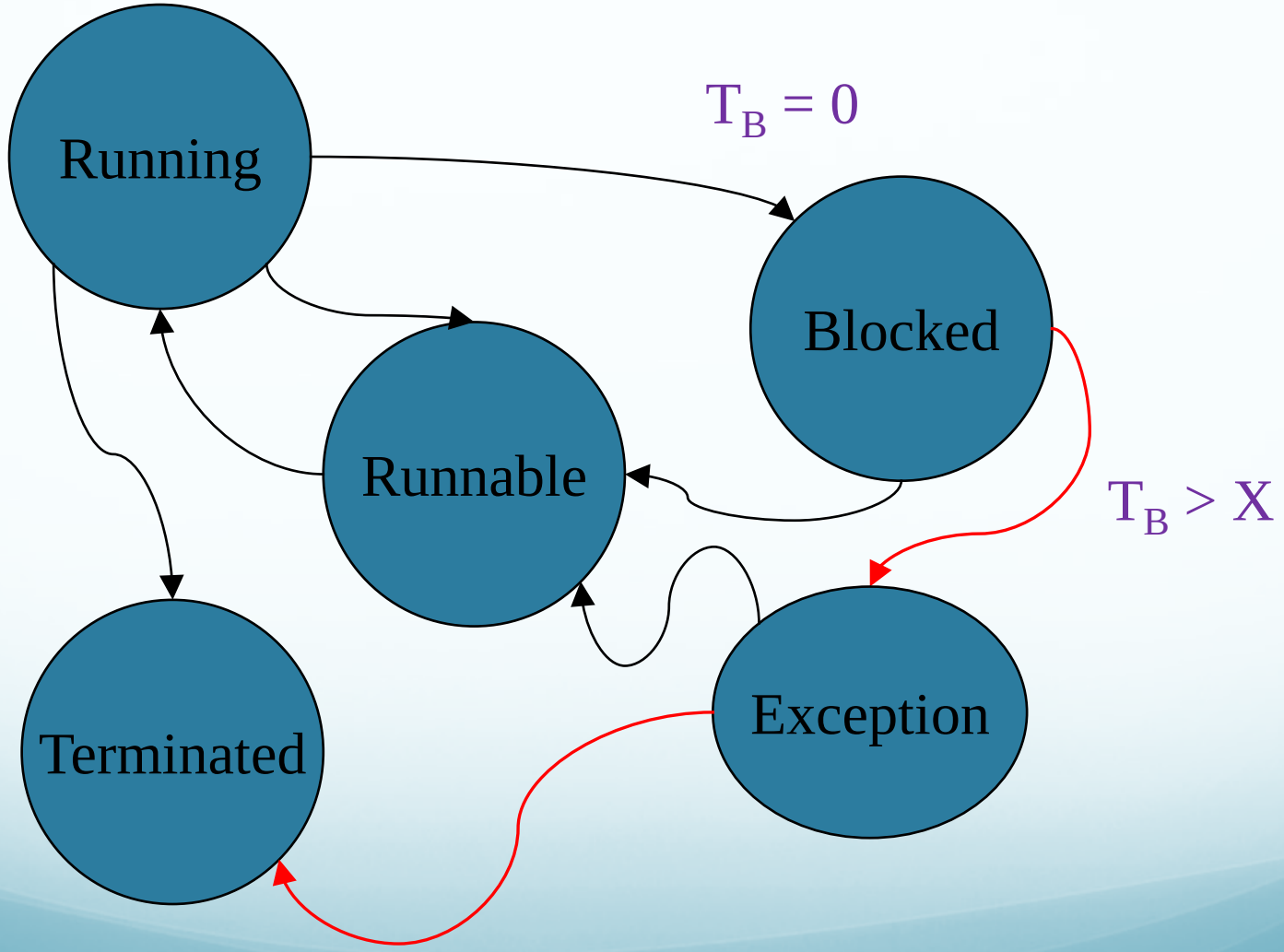
Worse Case Execution Time

- Static profiling of task code
  - Count instructions
  - Assume most number of iterations done by loops
- Dynamic profiling of task code
  - Run code with a large set of input vectors
  - Take the worse case run time
- How about architectural effects?
  - Caches
    - Some success analyzing direct mapped caches
  - DMA and bus contention
  - Branch predictors
- In hard-real time systems, non-deterministic architecture artifacts are avoided

Task Synchronization, Priority Inversion, and Deadlocks

- Most interesting systems have synchronized tasks
  - A Task can be blocked waiting for another task to reach a certain point
- When a task *H* with higher priority is blocked waiting for a lower priority task *L*, *L*'s priority is set to that of *H* (priority inheritance)
- Deadlocks occur when a task *A* is blocked waiting for another task *B*, and task *B* is blocked waiting for task *A*
  - Timeouts can be used to detect and resolve deadlocks
  - Deadlock recovery important in embedded systems

Timeouts



Interrupts

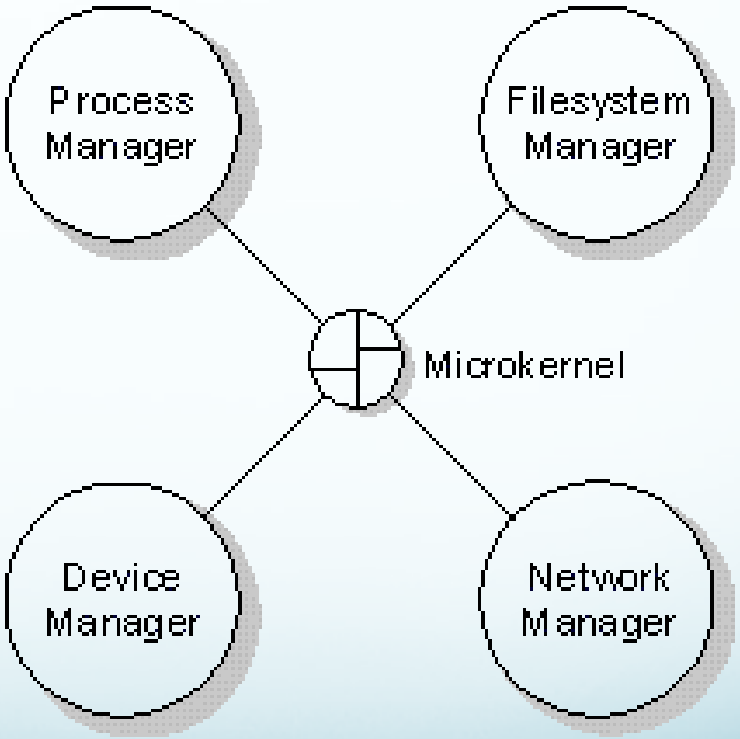
- Interrupt service routine (ISR) is a function responsible for a strategy in handling an interrupt
  - A specially marked routine (close to hardware)
  - An application level routine registered (close to application)
  - Etc.
- Interrupt dispatch time
  - Time taken from the moment an event occurs until the routine with a strategy for handing the event is invoked
  - Processor needs time to poll and detect interrupt
  - RTOS takes time to set flags, and route the interrupt to the application
  - Application may need time to spawn a thread to handle the interrupt
- Interrupt service time
  - Time taken to completely handle interrupt
  - Includes interrupt dispatch time

QNX

- QNX is a Real-time Operating System
  - Provides multitasking
  - Fast context switching
- POSIX Compliant
- QNX + windowing system can fit in less then 1M of flash or ROM
  - Bare-bones' configuration of a kernel with a few small modules to a full-blown network-wide system equipped to serve hundreds of users

QNX Architecture

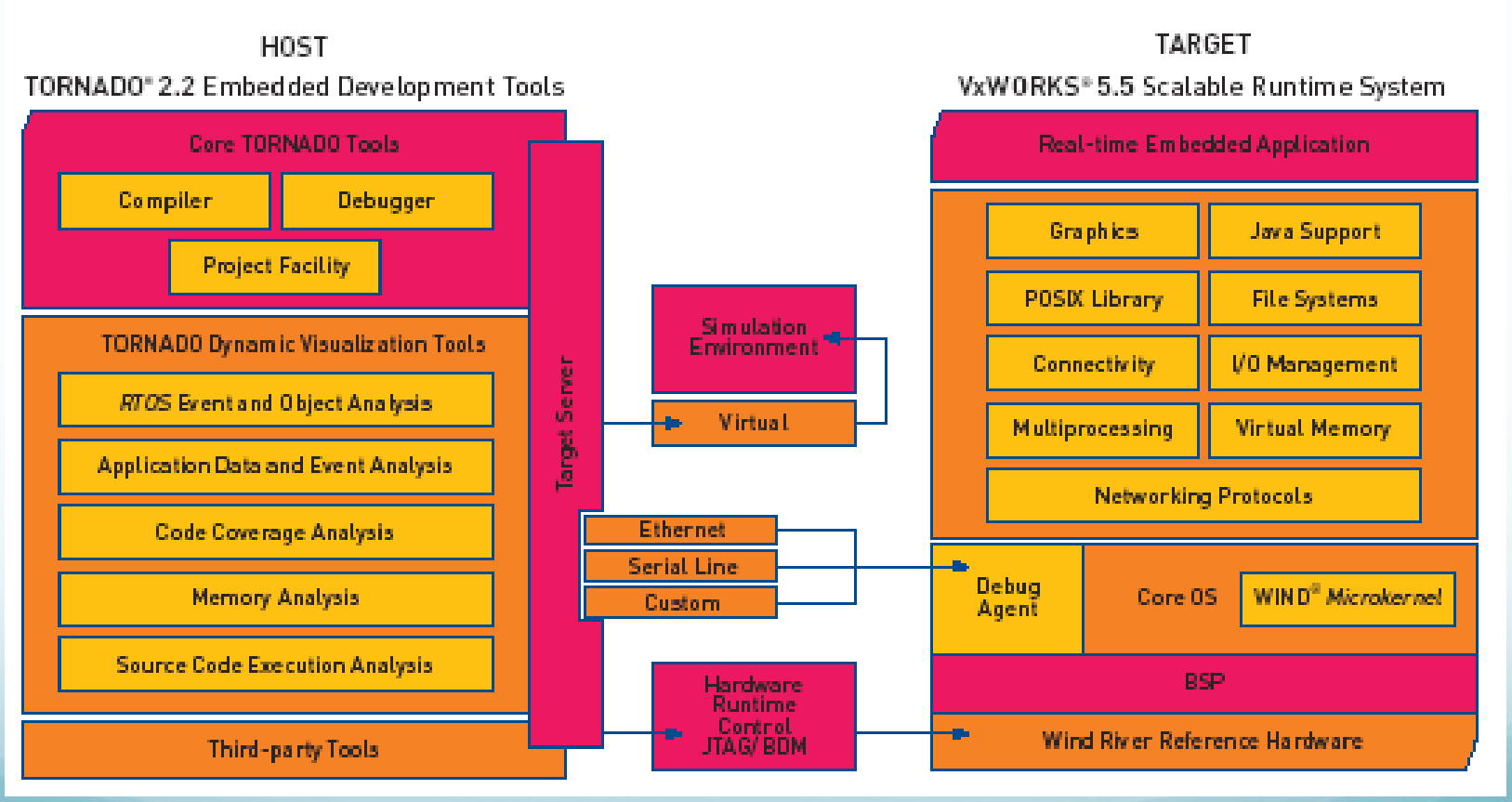
- Microkernel architecture
  - Message-based inter-process communication
  - Priority-based preemptive scheduling
- System processes are no different from any user-written program
- http://www.swd.de/documents/manuals/sysarch/index_en.html



VxWorks

- A popular RTOS for embedded systems
  - Flexibility
    - A large API (1800 of them)
  - Compatibility
    - Runs on all popular embedded processors
    - TCP/IP, POSIX
  - Scalability
- Microkernel Design
  - Preemptive round robin scheduling
  - Shared memory (intertask)
  - Pipes and IPC (intratask)
- http://www.windriver.com/products/vxworks5/vxworks5x_ds.pdf

VxWorks Architecture



Other RTOS

- Windows CE .NET
- RT-Linux
- BlueCat Linux
- eCos
- Embedix RT
- Hard Hat
- uCLinux
- Etc.

Conclusion

- RTOS
  - Hardware abstraction
  - Resource management
  - Process management
    - Scheduling
  - Process communication
  - Interrupts
- A good RTOS is one that behaves deterministically even under heavy/faulty load conditions