BLM4021 Gömülü Sistemler

Doç. Dr. Ali Can KARACA

ackaraca@yildiz.edu.tr

Yıldız Teknik Üniversitesi – Bilgisayar Mühendisliği

Lecture 11 – Introduction to Real Time Systems & RTOS

- Real Time Systems
- Hard & Soft Real Time Systems
- Embedded Firmware
- Operating Systems Basics
- GPOS & RTOS Comparison

Ders Materyalleri

Kaynaklar:

- Jiacun Wang, Real-Time Embedded Systems, Wiley, 2017.
- Xiacong Fan, Real-Time Embedded Systems: Design Principles and Engineering Practices, 2015.
- Stefan M. Petters, Real-Time Systems, NICTA.
- Philip Koopman, Real Time Operating Systems, Embedded System Engineering, 2016.
- Colin Perkins, Introduction to Real-Time Systems, University of Glaskow, Lecture 1.
- Kang, G. Shin, Principles of Real-Time and Embedded Systems, Lecture Notes 1-4.
- Özgür Aytekin (Collins Aerospace), Introduction to RT Operating Systems, A training for software development professionals.

Haftalık Konular

Hafta	Teorik	Laboratuvar
1	Giriş ve Uygulamalar, Mikroişlemci, Mikrodenetleyici ve Gömülü sistem kavramlarının açıklanması	Grupların oluşturulması & Kitlerin Testi
2	Bir Tasarım Örneği, Mikroişlemci, Mikrodenetleyici, DSP, FPGA, ASIC kavramları	Kitlerin gruplara dağıtımı + Raspberry Pi Kurulumu
3	16, 32 ve 64 bitlik mikrodenetleyiciler, pipeline	Raspberry Pi ile Temel Konfigürasyon
4	PIC ve MSP430 özellikleri	---- Resmi Tatil ---
5	ARM tabanlı mikrodenetleyiciler ve özellikleri	Uygulama 1 – Raspberry Pi ile Buzzer Uygulaması
6	ARM Komut setleri ve Assembly Kodları-1	Uygulama 2 – Raspberry Pi ile Ivme ve Gyro Uygulaması
7	ARM Komut setleri ve Assembly Kodları-2, Raspberry Pi vers. ve GPIO'ları	Uygulama 3 – Raspberry Pi ile Motor Kontrol Uygulaması
8	*Vize Sınavı*	
9	Haberleşme Protokolleri (SPI, I2C ve CAN)	Proje Soru-Cevap Saati - 1
10	Sensörlerden Veri Toplama, Algılayıcı, ADC ve DAC	Proje Soru-Cevap Saati - 2
11	Zamanlayıcı, PWM ve Motor Sürme	Proje kontrolü - 1
12	Gerçek Zaman Sistemler ve temel kavramlar	Proje Kontrolü - 2
13	Gerçek zaman İşletim Sistemleri	Mazeret sebepli son proje kontrollerinin yapılması
14	Nesnelerin İnterneti	
15	*Final Sınavı*	

For more details -> Bologna page: http://www.bologna.yildiz.edu.tr/index.php?r=course/view&id=9463&aid=3

Definition of RT Systems

Stefan M. Peters:

A real-time system is any information processing system which has to respond to externally generated input stimuli within a finite and specified period

– the correctness depends not only on the logical result but also the time it was delivered
– failure to respond is as bad as the wrong response!

Laplante's book:

A real-time system is a system that must satisfy explicit (bounded) response-time constraints or risk severe consequences, including failure.

J. Wang's book:

Real-time systems are required to compute and deliver correct results within a specified period of time.

Definition of RT Systems

Real-time systems are defined as those systems in which the overall correctness of the system depends on both the functional correctness and the timing correctness.

The timing correctness is at least as important as the functional correctness.

Depending on how serious missing a task deadline is, a timing constraint can be either a hard or a soft constraint:

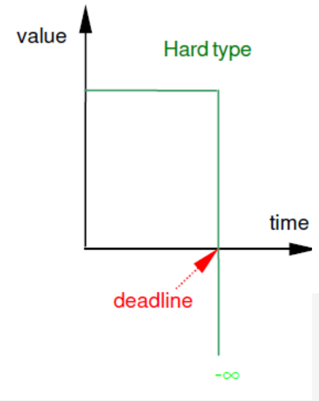
- A timing constraint is hard if the consequence of a missed deadline is fatal. A late response (completion of the requested task) is useless, and sometimes totally unacceptable. «HARD REAL TIME»

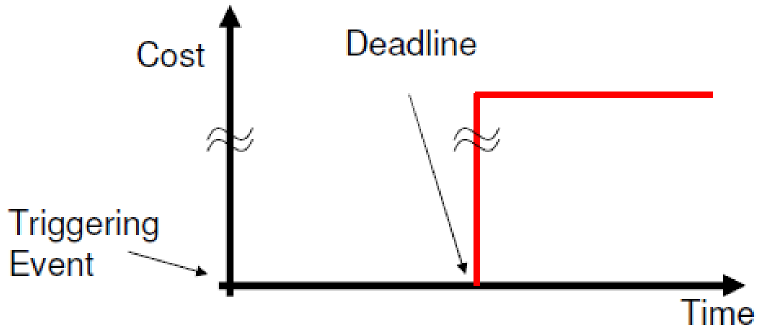
- A timing constraint is soft if the consequence of a missed deadline is undesirable but tolerable. A late response is still useful as long as it is within some acceptable range (say, it occurs occasionally with some acceptably low probability). «SOFT REAL TIME»

Hard Real Time vs. Soft Real Time

Hard Real Time

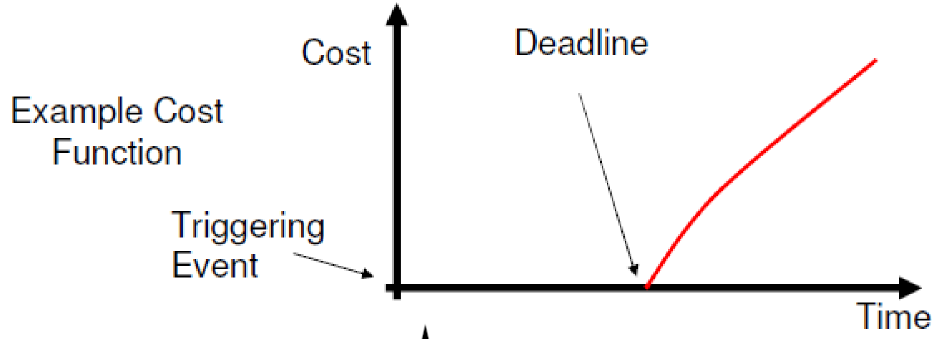
- An overrun in response time leads to potential loss of life and/or big financial damage
- Many of these systems are considered to be safety critical.
- Sometimes they are “only” mission critical, with the mission being very expensive.
- In general there is a cost function associated with the system.

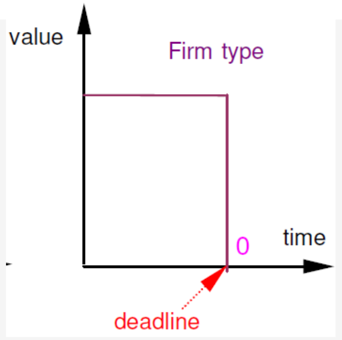




Soft Real Time

- Deadline overruns are tolerable, but not desired.
- There are no catastrophic consequences of missing one or more deadlines.
- There is a cost associated to overrunning, but this cost may be abstract.
- Often connected to Quality-of-Service (QoS)





Credit by Stefan M. Peters

Some Hard Real Time Examples

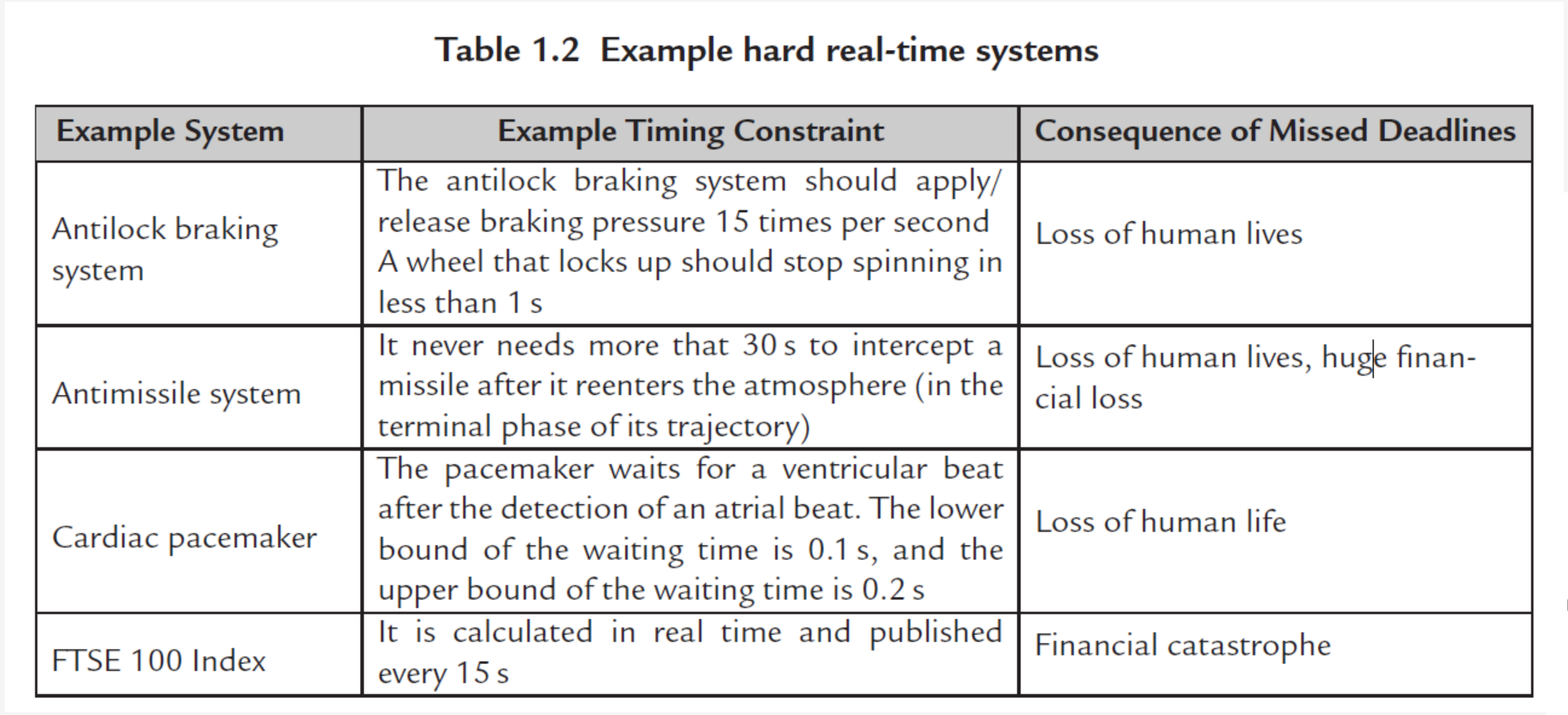
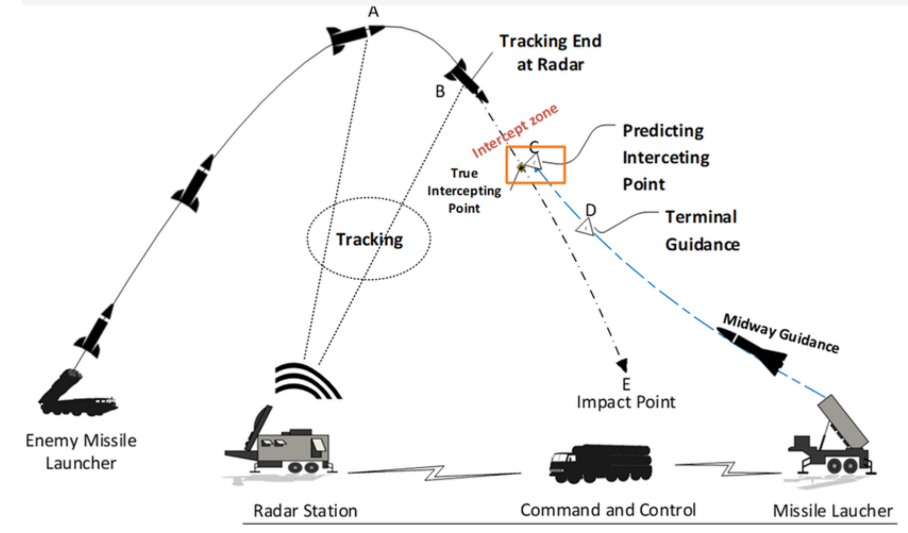


Table 1.2 Example hard real-time systems

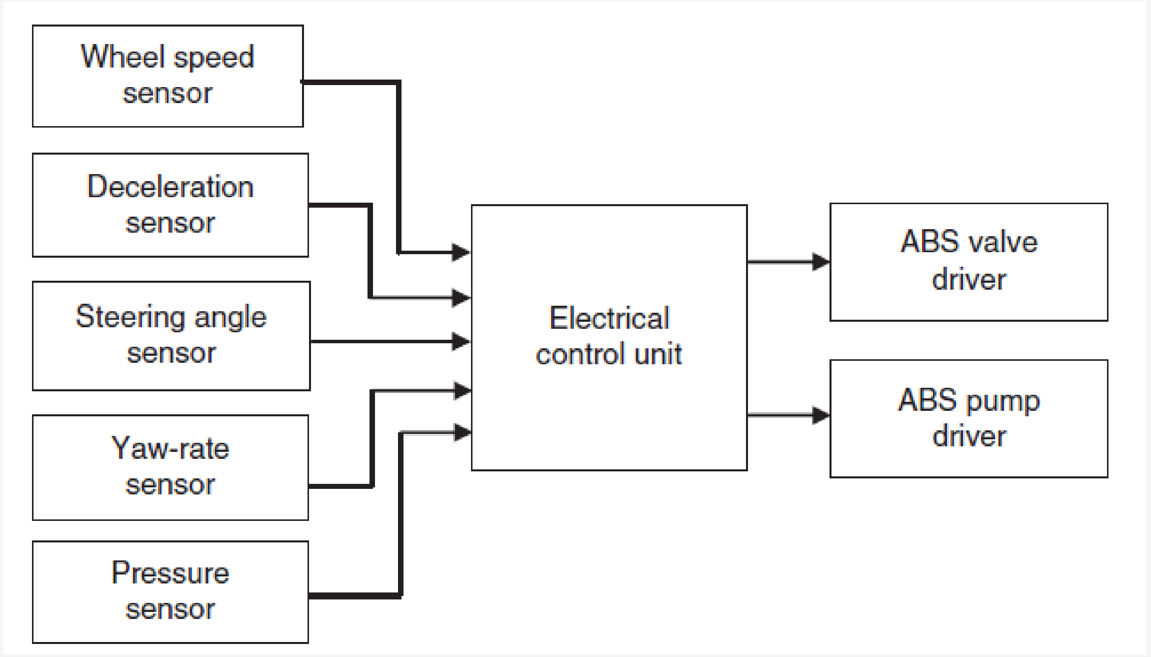
Example System	Example Timing Constraint	Consequence of Missed Deadlines
Antilock braking system	The antilock braking system should apply/release braking pressure 15 times per second A wheel that locks up should stop spinning in less than 1 s	Loss of human lives
Antimissile system	It never needs more that 30 s to intercept a missile after it reenters the atmosphere (in the terminal phase of its trajectory)	Loss of human lives, huge financial loss
Cardiac pacemaker	The pacemaker waits for a ventricular beat after the detection of an atrial beat. The lower bound of the waiting time is 0.1 s, and the upper bound of the waiting time is 0.2 s	Loss of human life
FTSE 100 Index	It is calculated in real time and published every 15 s	Financial catastrophe



Ballistic Missile Defense System

A failed system is a system that cannot satisfy one or more of the requirements stipulated in the system requirements specification.

Example: ABS (Antilock Braking System)



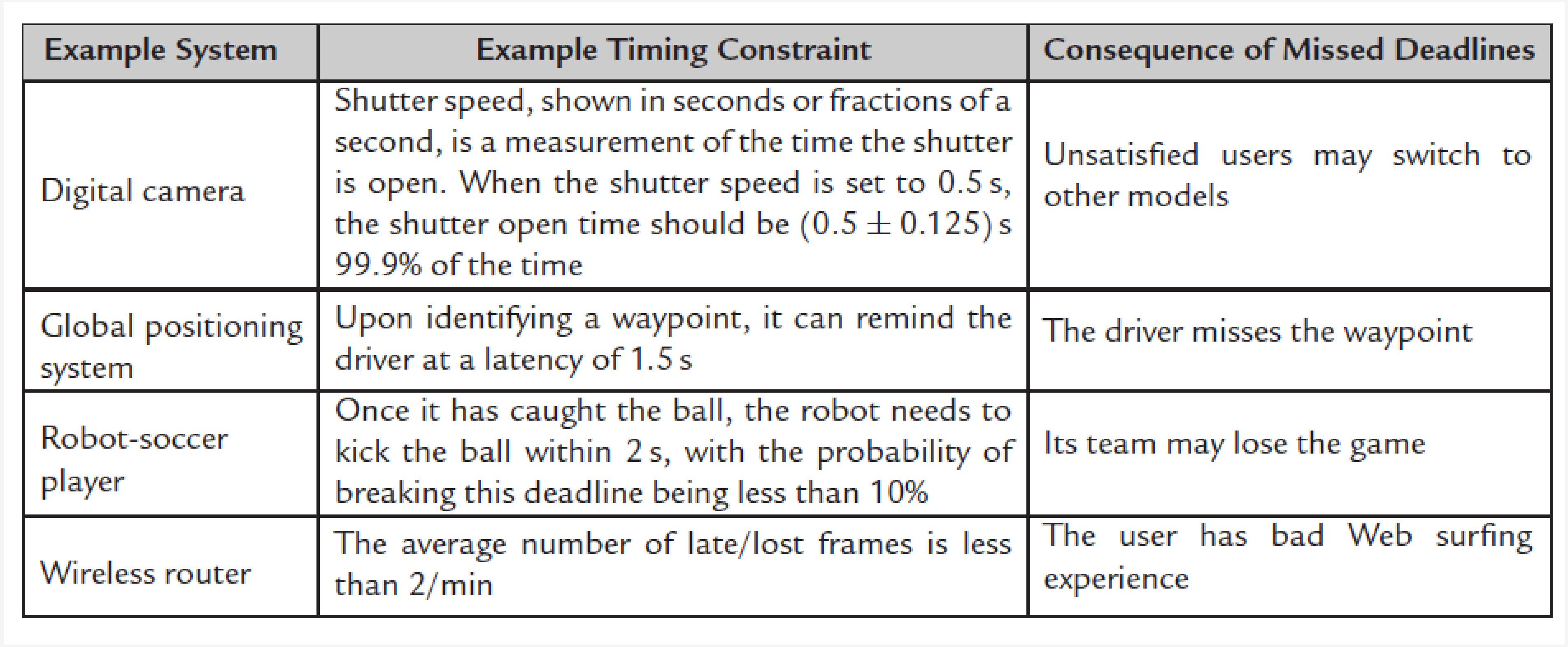
Concurrency refers to a property of systems in which several computations are executing simultaneously and potentially interacting with each other.

For example, the following events in the ABS can occur at the same time:

- Wheel speed sensor event
- Deceleration sensor event
- Brake pedal event
- Solenoid valve movement
- Pump operation

- The wheel speed sensors must be polled every 15 milliseconds.
- Each cycle, the control law computation for wheel speed must be finished in 20 milliseconds.
- Each cycle, the wheel speed prediction must be completed in 10 milliseconds.

Some Soft Real Time Examples

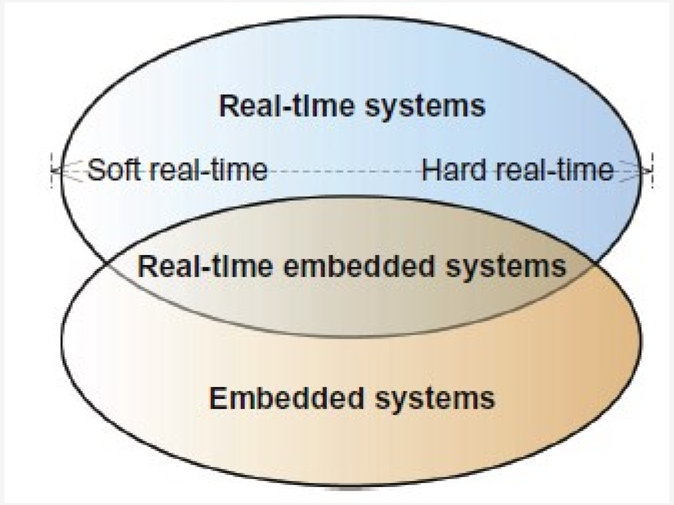


Example System	Example Timing Constraint	Consequence of Missed Deadlines
Digital camera	Shutter speed, shown in seconds or fractions of a second, is a measurement of the time the shutter is open. When the shutter speed is set to 0.5 s, the shutter open time should be (0.5 ± 0.125) s 99.9% of the time	Unsatisfied users may switch to other models
Global positioning system	Upon identifying a waypoint, it can remind the driver at a latency of 1.5 s	The driver misses the waypoint
Robot-soccer player	Once it has caught the ball, the robot needs to kick the ball within 2 s, with the probability of breaking this deadline being less than 10%	Its team may lose the game
Wireless router	The average number of late/lost frames is less than 2/min	The user has bad Web surfing experience

After a credit card or debit card is inserted, the probability that the ATM prompts the user to enter a passcode within 1 second should be no less than 95%.

RT Systems

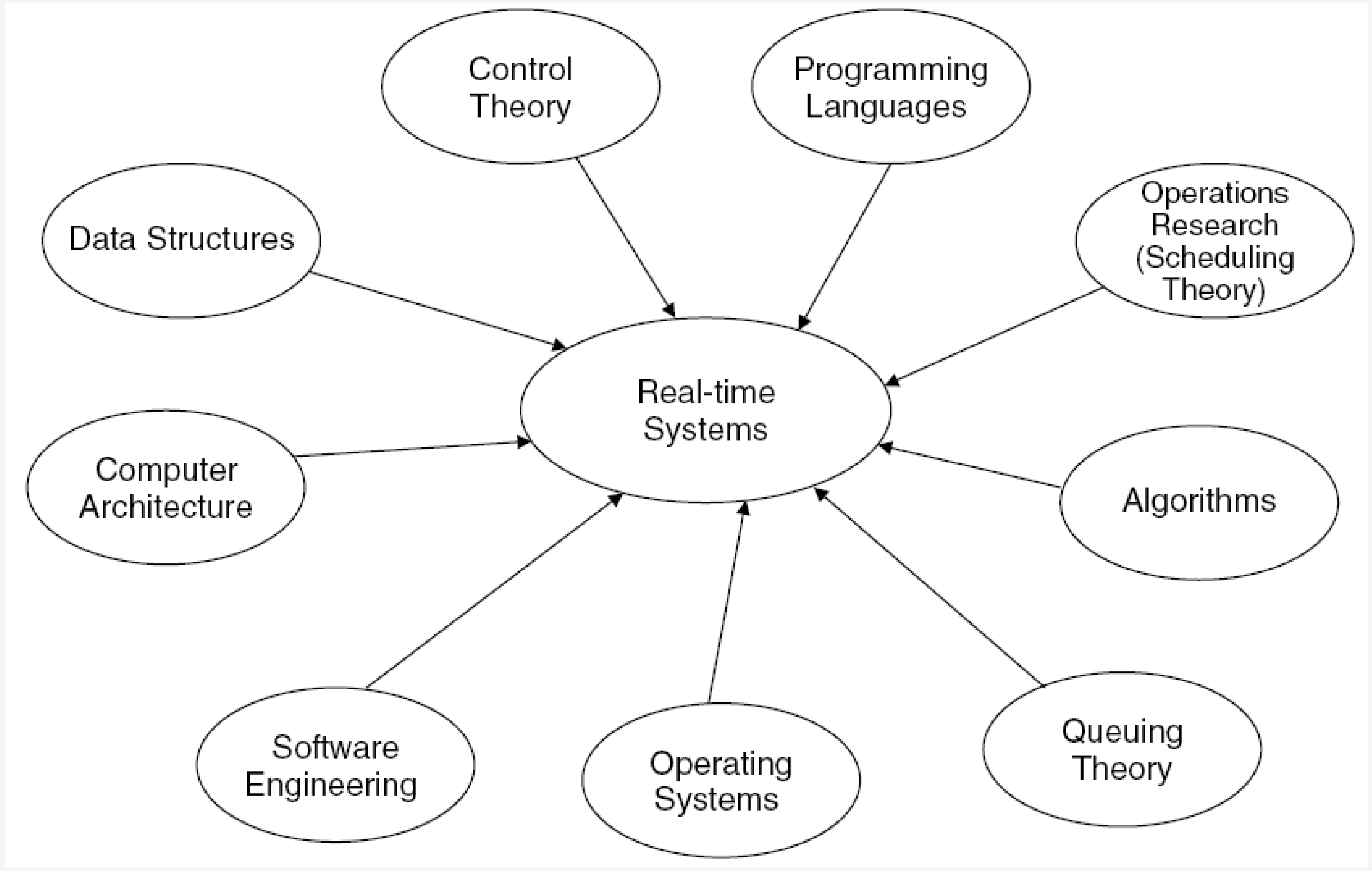
- Response time is deterministic and bounded.
- RT systems provide concurrency.
- They have faulty-tolerances.



Most Common Misconceptions !!!!!

- Real time computing is equal to fast computing. (e.g. pacemaker) ✗
- There is no science in RT system design. ✗
- Advances in hardware will take care of real-time requirements. ✗

RT Systems


Control Theory
Programming Languages
Data Structures
Operations Research (Scheduling Theory)
Real-time Systems
Computer Architecture
Algorithms
Software Engineering
Operating Systems
Queuing Theory

Embedded Firmware

- … is responsible for controlling the various peripherals of the embedded hardware and generating response in accordance with the functional requirements of the product.

- … can be grouped into two approaches:
  - Bare Metal (Super Loop) Based Approach
  - Embedded Operating System Based Approach

The decision on which approach needs to be adopted for firmware development is purely dependent on the complexity and system requirements.

Bare Metal Firmware

- It is Suitable for applications that are not time critical and where the response time is not so important.
- The tasks are executed in a never ending loop.
- In multiple task based system, each task is executed

in a serial way.

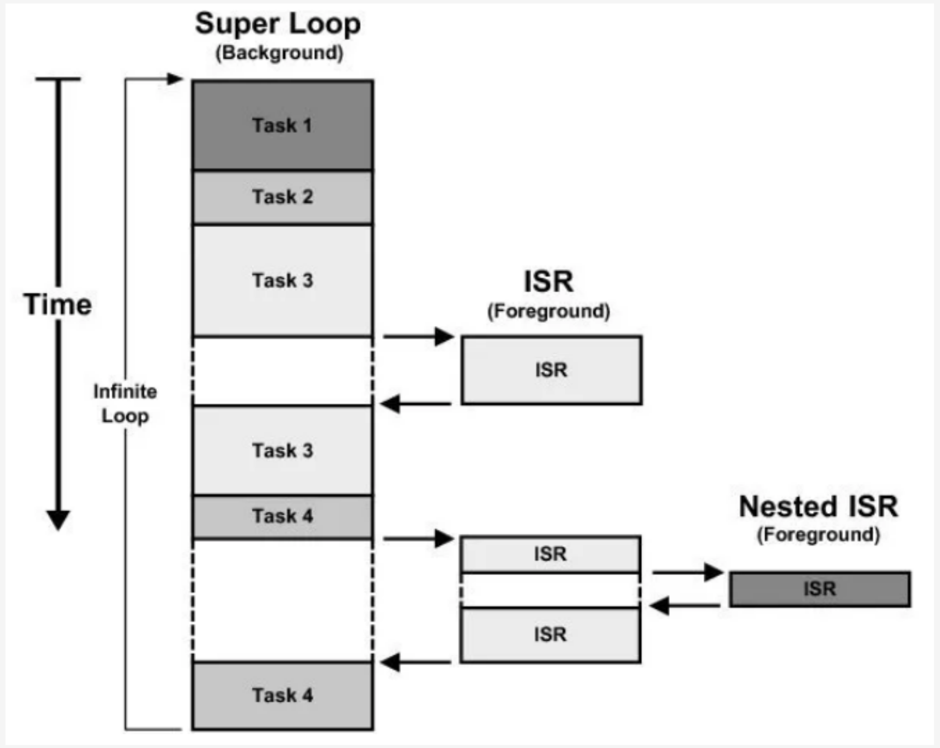
Pros:

Simple design, reduced memory, no cost

Cons:

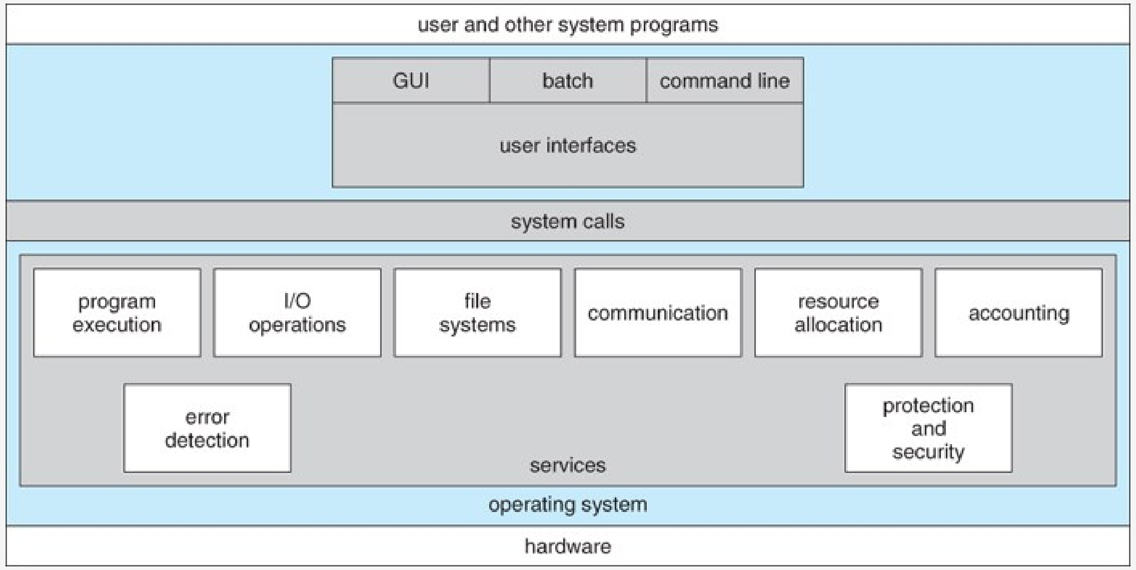
Non-real time execution

Any failure in any part will affect the total system



Embedded OS Firmware

- It can be both Real Time OS and General Purpose OS.
- The Embedded OS is responsible for scheduling the execution of user tasks and the allocation of system resources among multiple tasks.
- It Involves lot of OS related overheads apart from managing and executing user defined tasks.
- It manages the harware resources needed for all the applications in background.

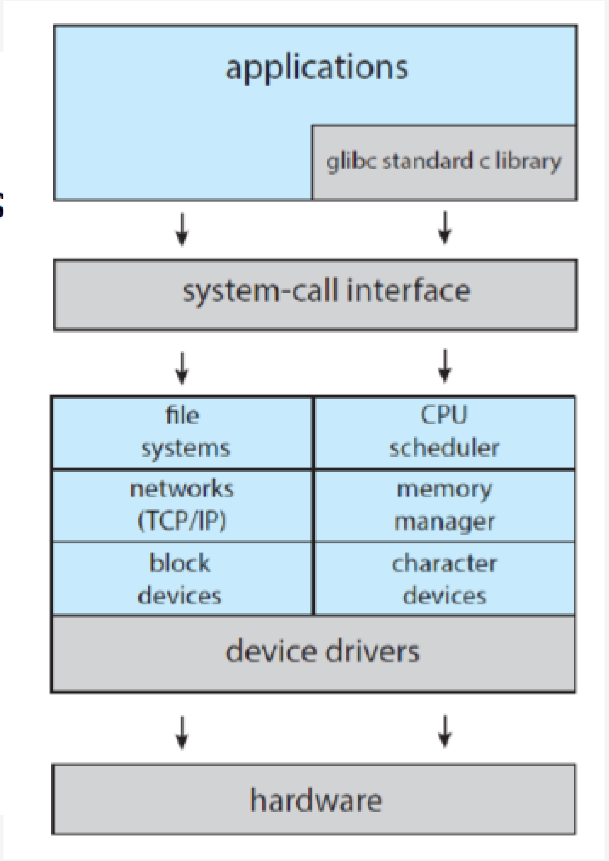


Fundamentals of Operating Systems

Monolithic Kernel

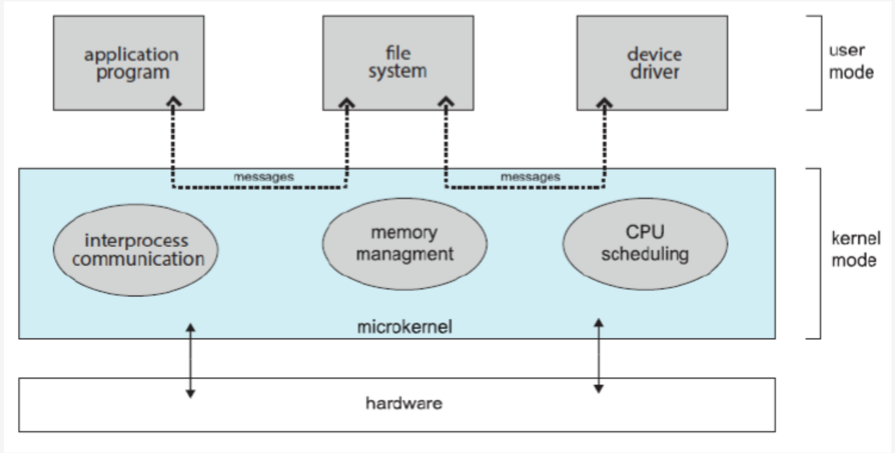
- In the monolithic approach the entire operating system runs as a single program in kernel mode

- The operating system is written as a collection of procedures, linked together into a single large executable binary program.



Microkernel

The near-minimum amount of software that can provide the mechanisms needed to implement an operating system (OS). These mechanisms include low-level address space management, thread management, and inter-process communication (IPC).



Example: QNX - RTOS

It is a commercial Unix-like RTOS.

QNX which allows a system to be scaled to very small sizes and still provide multitasking, preemptive scheduling, and fast context-switching.

QNX Neutrino runs on many modern processor platforms, including PowerPC, x86, MIPS, ARM, and XScale.

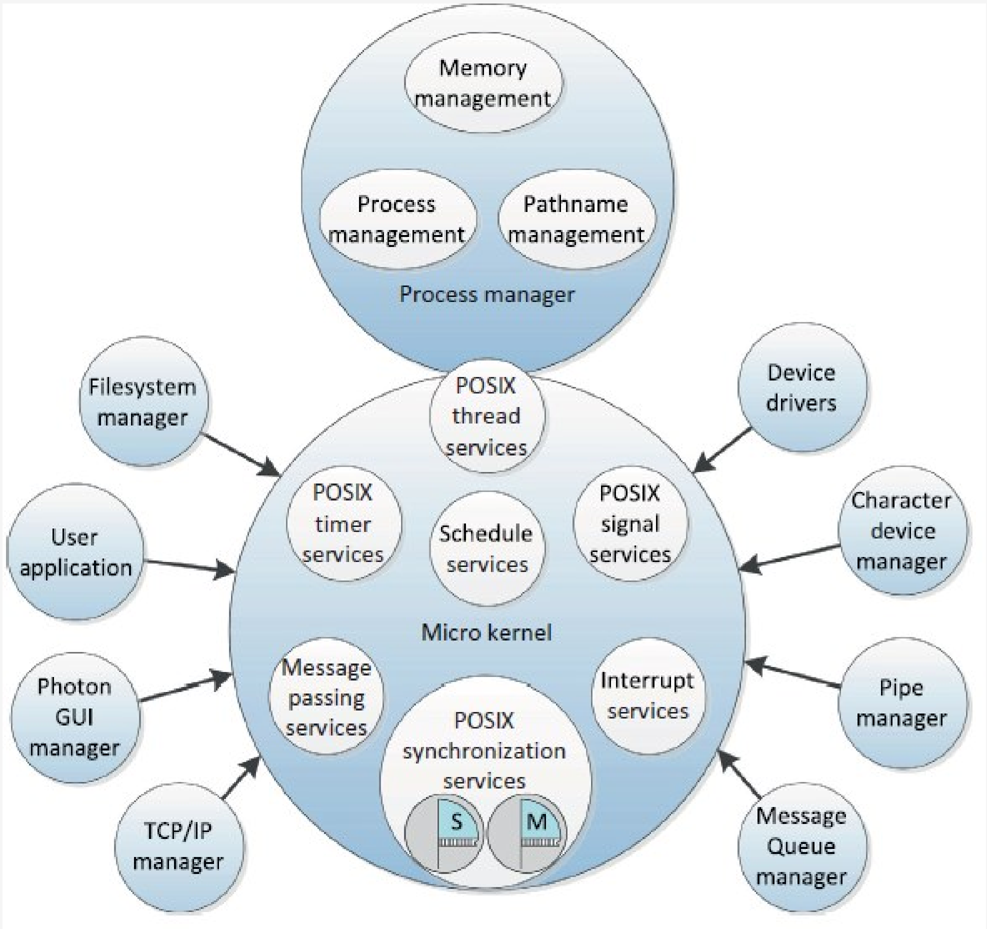
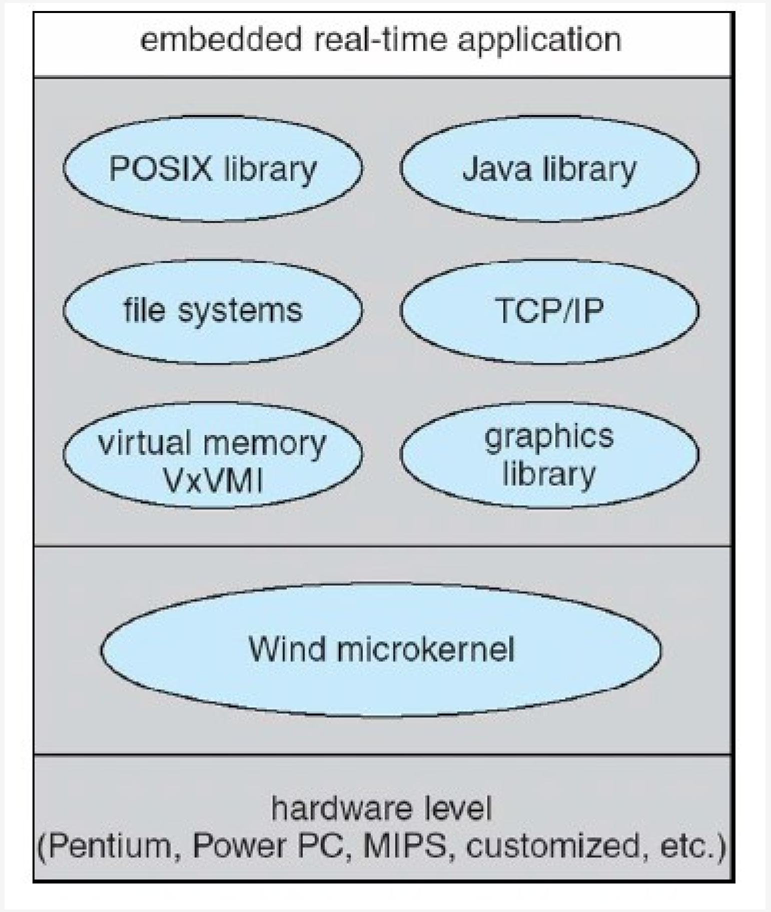


Figure 13.7
The QNX microkernel architecture.

Example: VxWorks 5.0 - RTOS



- The Wind microkernel provides support for the following:

  (1) Processes and threads

  (2) preemptive and non-preemptive round-robin scheduling

  (3) manages interrupts (with bounded interrupt and dispatch latency times)

  (4) shared memory and message passing interprocess communication facilities

Fundamentals of Operating Systems

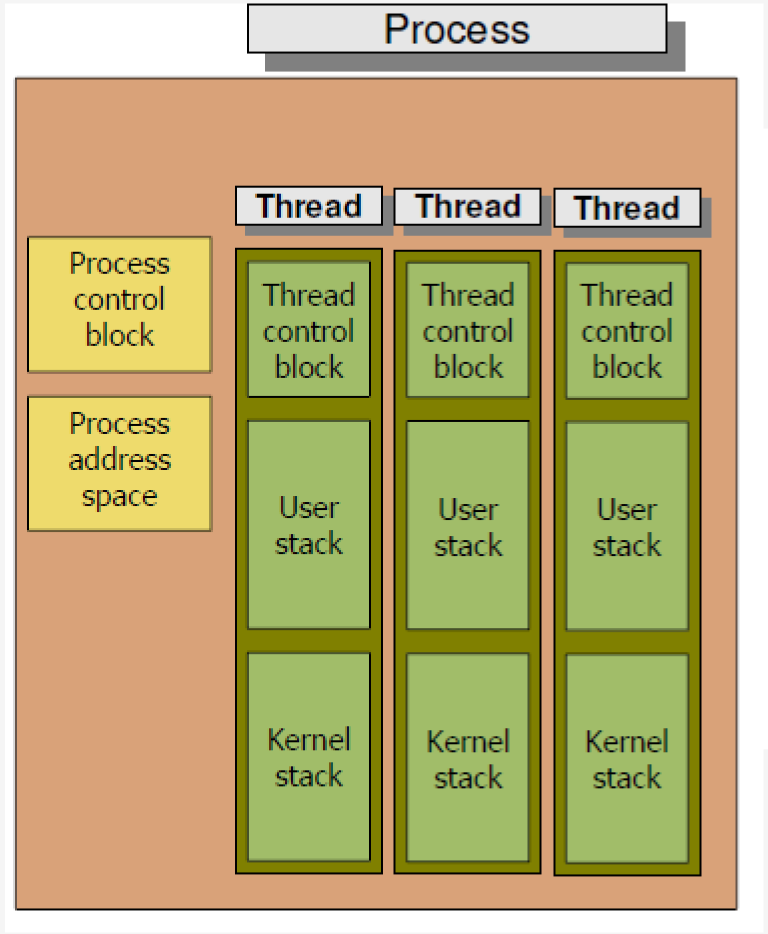
- Process: A *process* is an instance of a program in execution. It is a unit of work within the system. A program is a *passive* entity, while a process is an *active entity*.

- Thread: A *thread* is a path of execution within a process and the basic unit to which the OS allocates processor time. A process can be *single-threaded* or *multithreaded*. Threads within the same process share the same address space, whereas different processes do not.

- Task: A thread is the smallest unit of work managed independently by the scheduler of the OS. In RTOSs, the term *tasks* is often used for threads or single-threaded processes. For example, VxWorks and microC/OS-III are RTOSs that use the term tasks.

Fundamentals of Operating Systems

Multi-Threaded Process Model:



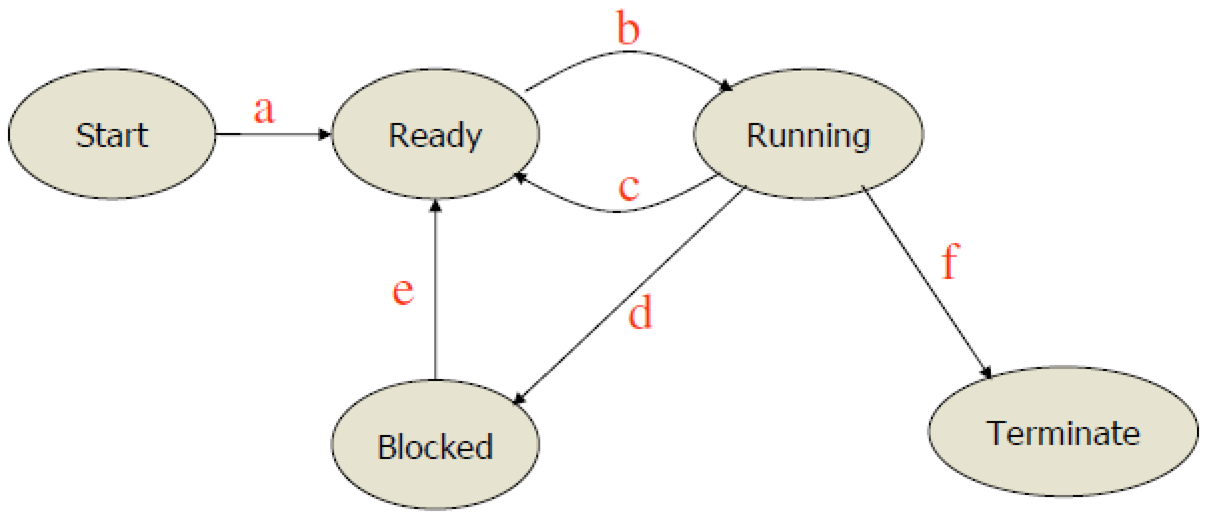
- In the multi-threaded process model each process can have many threads
  - One address space
  - One PCB
  - Many stacks
  - Many TCB (Thread Control blocks)
  - The threads are scheduled directly by the global scheduler

- In Real-Time Operating Systems
  - Depending on the size and type of system we can have both threads and processes or only threads
  - For efficiency reasons, most RTOS only support
    - 1 process
    - Many threads inside the process
    - All threads share the same memory
  - Examples are RTAI, RT-Linux, Shark, some version of VxWorks, QNX, etc.

Credit by Silberschatz et al.

Thread States

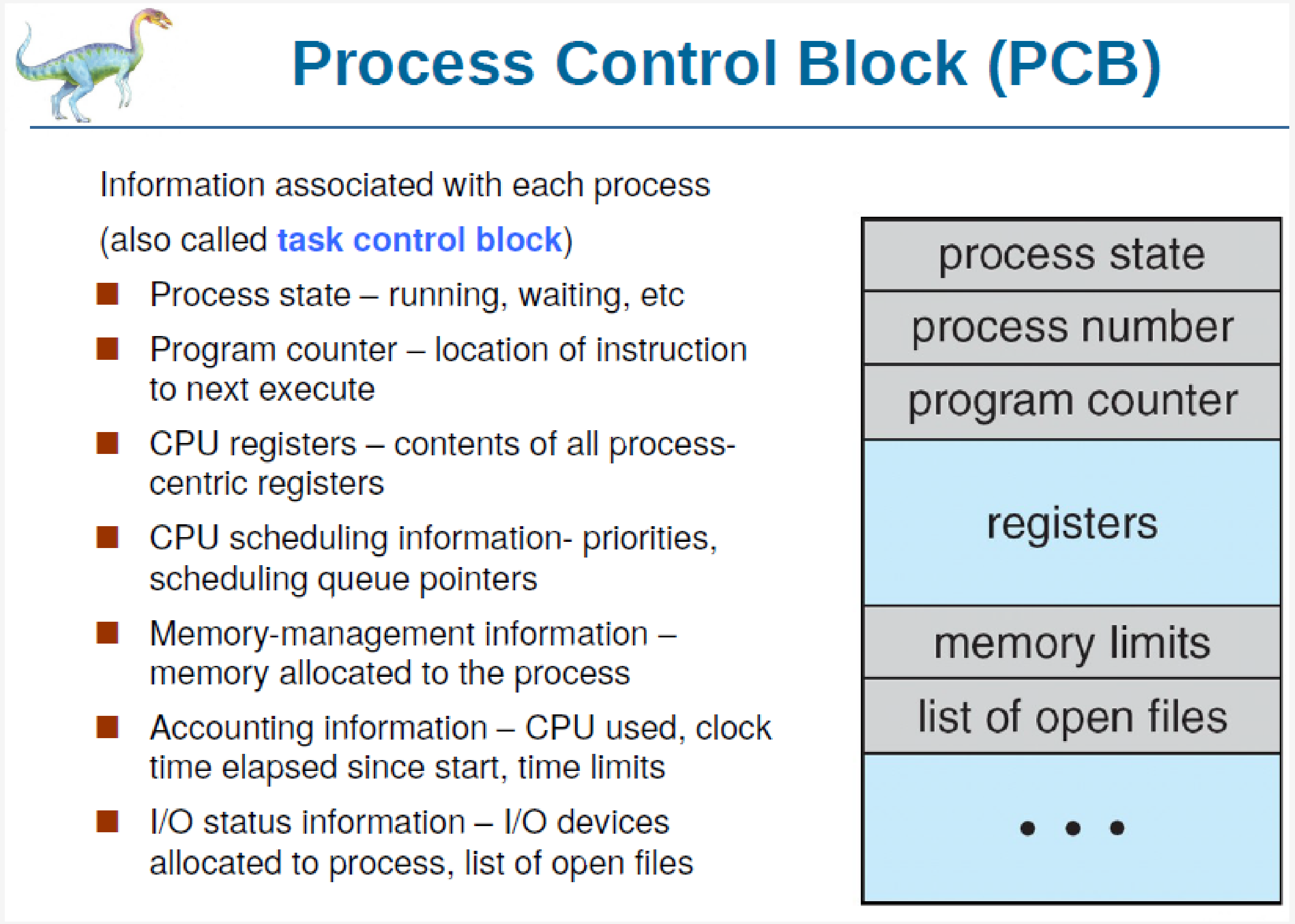
a)	Creation	The thread is created
b)	Dispatch	The thread is selected to execute
c)	Preemption	The thread leaves the processor
d)	Wait on condition	The thread is blocked on a condition
e)	Condition true	The thread is unblocked
f)	Exit	The thread terminates



Context switching happens when:

- The thread has been "preempted" by another higher priority thread
- The thread blocks on some condition
- In time-sharing systems, the thread has completed its "round" and it is the turn of some other thread

Fundamentals of Operating Systems

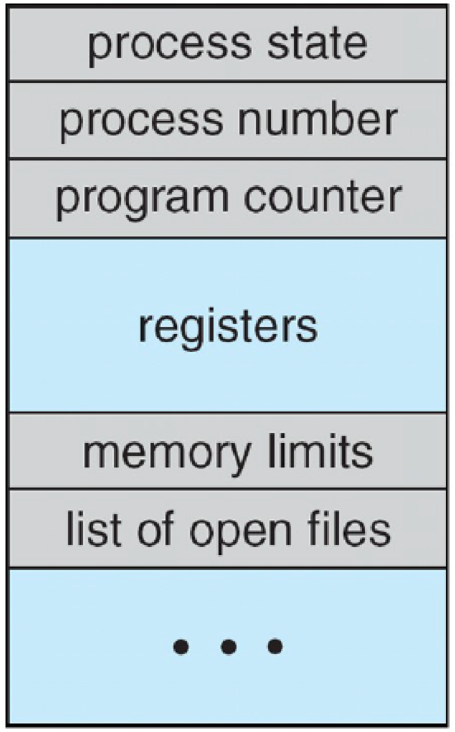


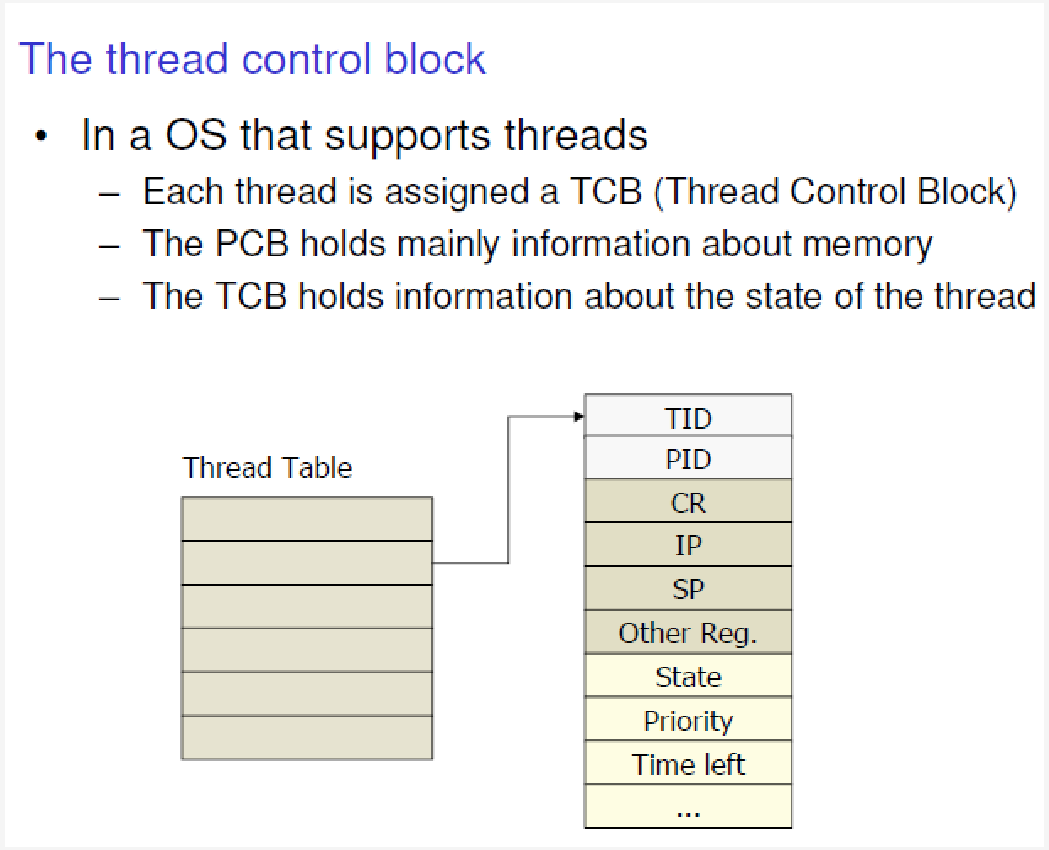
Process Control Block (PCB)

Information associated with each process

(also called **task control block**)

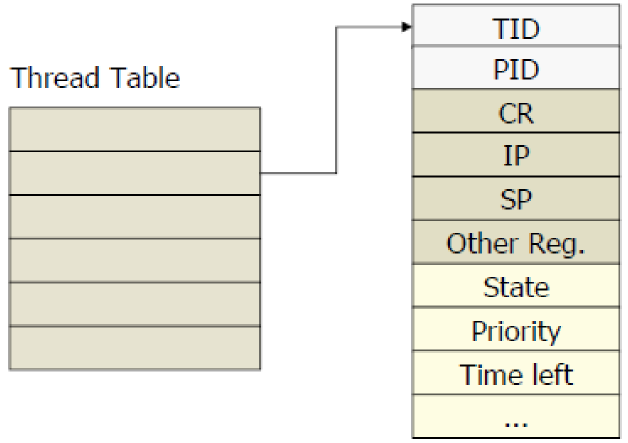
- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files





The thread control block

- In a OS that supports threads
  - Each thread is assigned a TCB (Thread Control Block)
  - The PCB holds mainly information about memory
  - The TCB holds information about the state of the thread

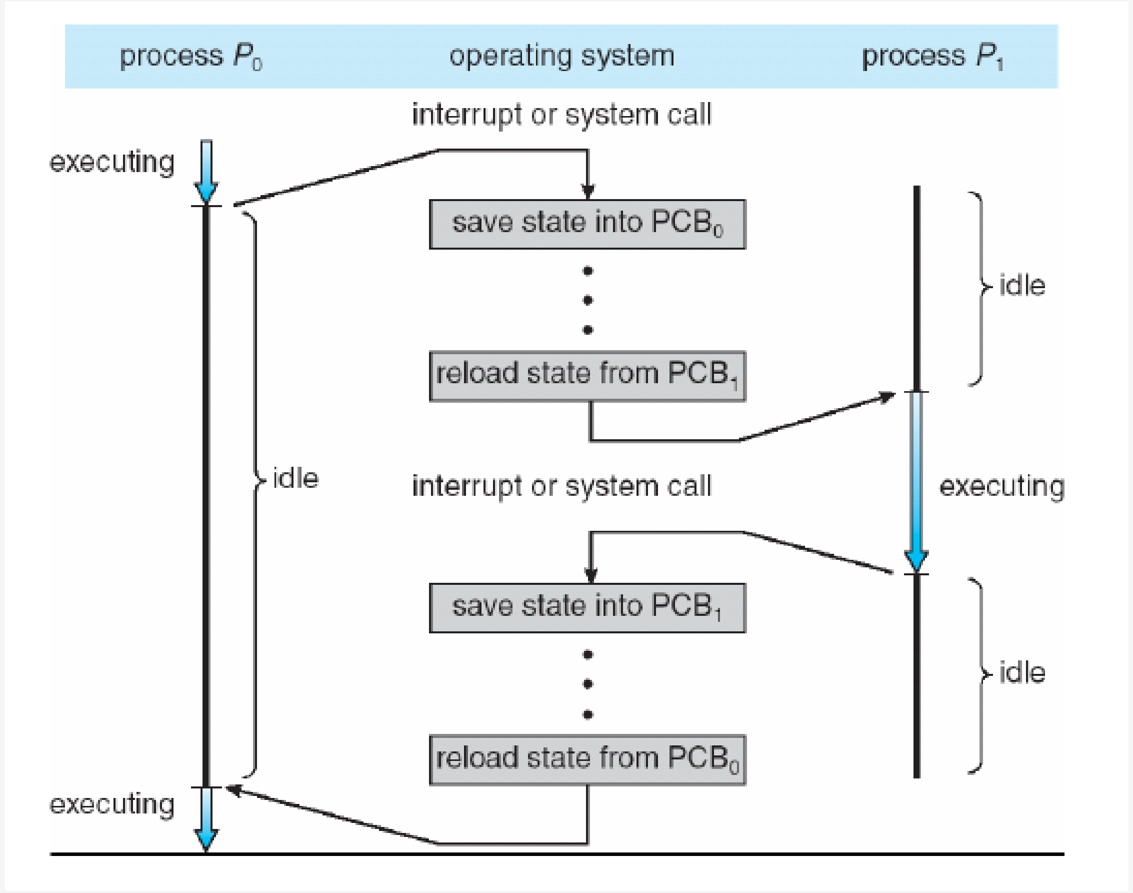


Credit by Silberschatz et al.

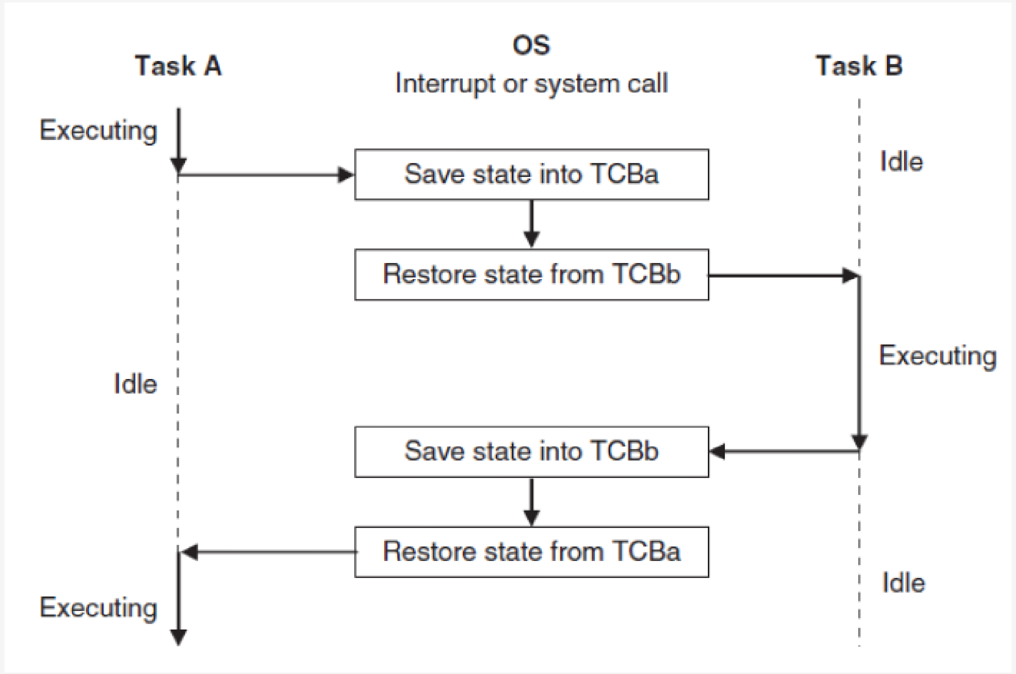
Fundamentals of Operating Systems

Context Switch

When CPU switches to another process/thread , the system must **save the state** of the old process/thread and load the **saved state** for the new process/task via a **context switch**



Context of a process/thread represented in the PCB/TCB.



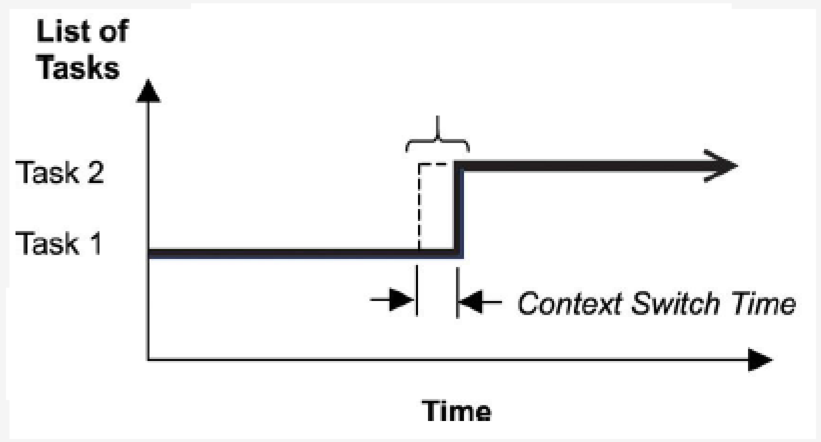
Credit by Silberschatz et al.

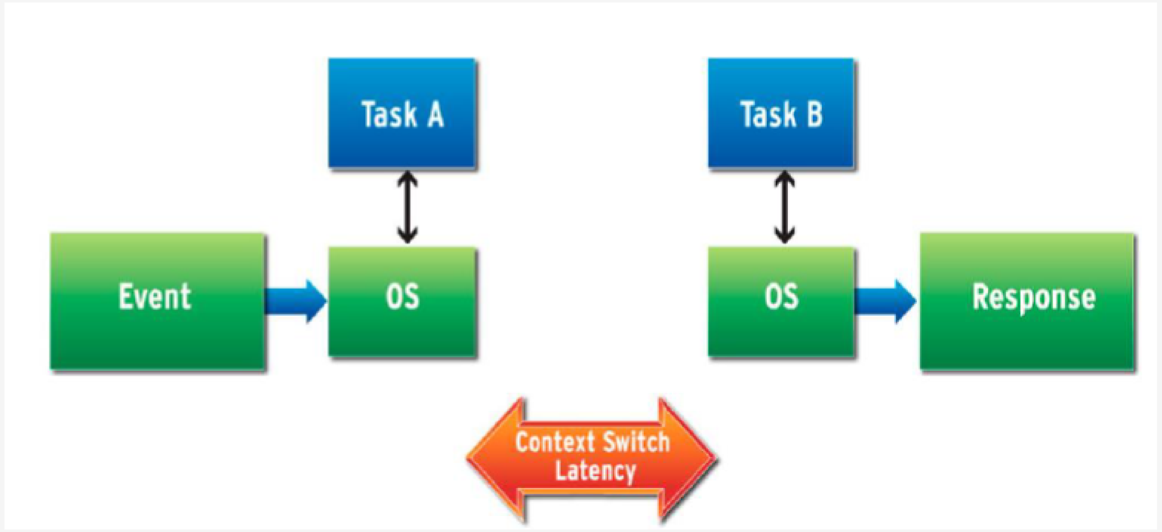
Fundamentals of Operating Systems

Context Switch Time / Dispatch Latency ->

- Context switch time is relatively insignificant compared to most operations that a task performs.

- If an application's design includes frequent context switching the application can incur unnecessary performance overhead.

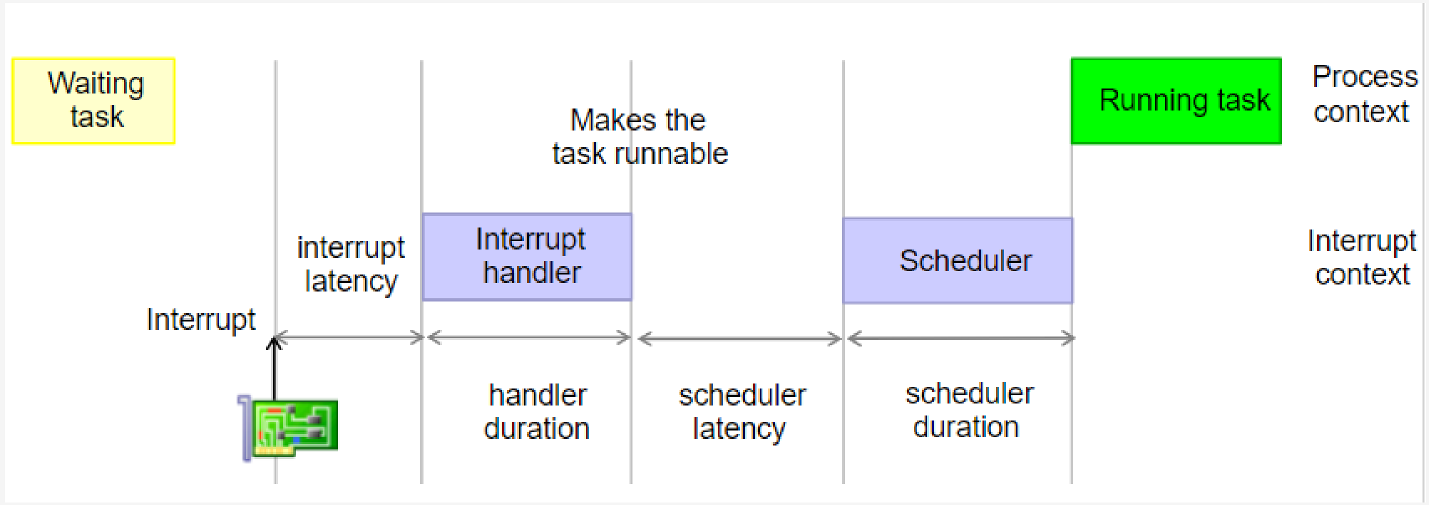




Credit by Özgür Aytekin

Kernel Latency = Task Latency = Event Latency

Task Latency = Interrupt Latency + Handler Duration + Scheduler Latency + Scheduler Duration

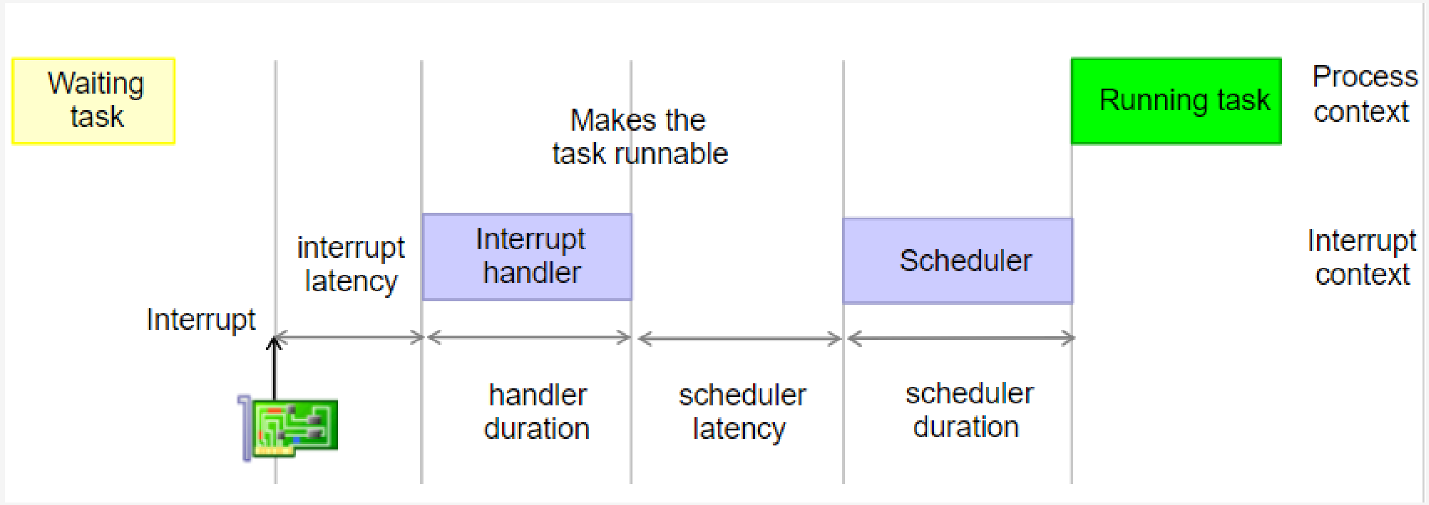


Interrupt:

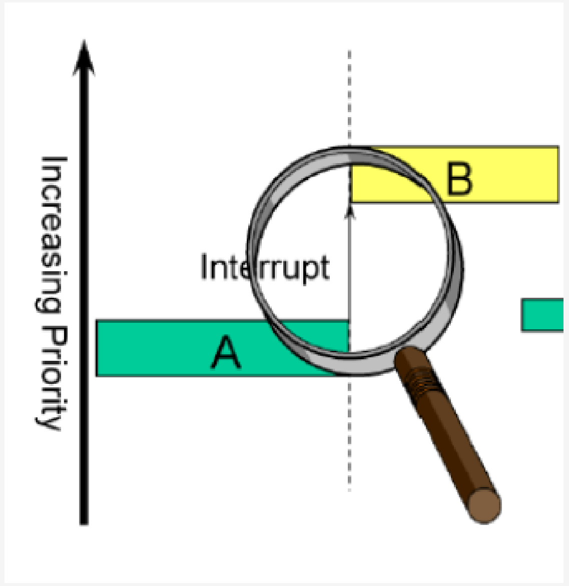
- For example, DMA informs the completion of the memory transfer. (GPIO Interrupt)
- Or timer produces an interrupt.

Task Latency

Task Latency = Interrupt Latency + Handler Duration + Scheduler Latency + Scheduler Duration



A: Task
B: Interrupt



1) Interrput Latency: The time between the occurence of the interrupt and the start of the interrupt handler.

What affects it?

- Time spent with interrupts disabled.
- Time spent in execution of the last instruction in pipeline.
- Time spent in an equal or higher priority interrupt handler
- Time spent in other handlers for this interrupt

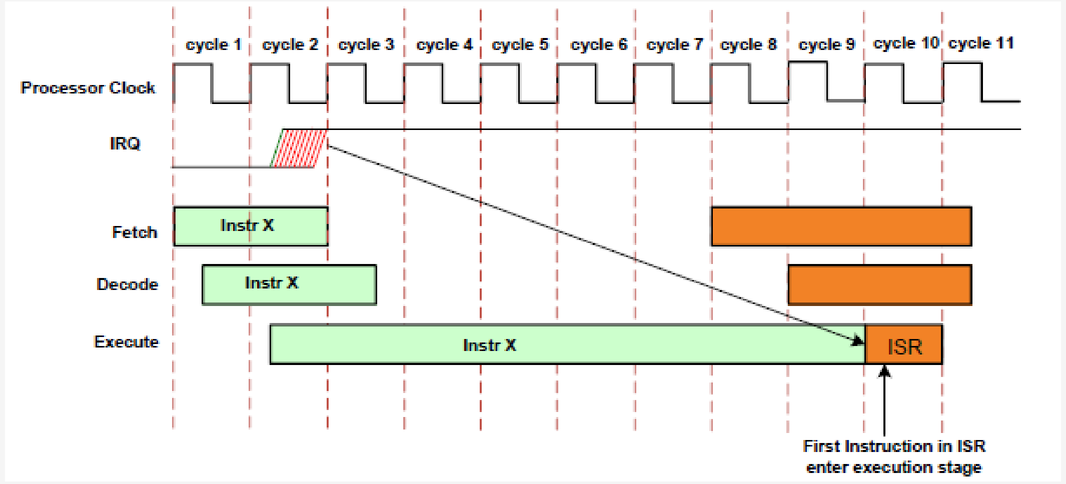
Task Latency

Task Latency = Interrupt Latency + Handler Duration + Scheduler Latency + Scheduler Duration

1) Interrput Latency: The time between the occurence of the intterrupt and the start of the interrupt handler.

What affects it?

- Time spent with interrupts disabled.
- Time spent in execution of the last instruction in pipeline.
- Time spent in an equal or higher priority interrupt handler
- Time spent in other handlers for this interrupt



Task Latency

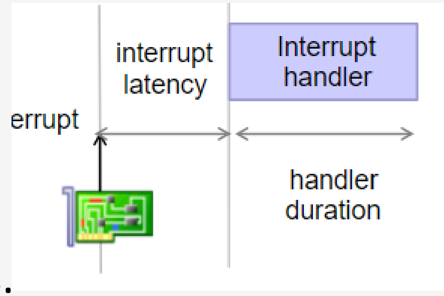
Task Latency = Interrupt Latency + Handler Duration + Scheduler Latency + Scheduler Duration

2) Interrupt Handler:

The interrupt handler recognizes and handles the event, and then wake-up the user-space task that will react to this event.

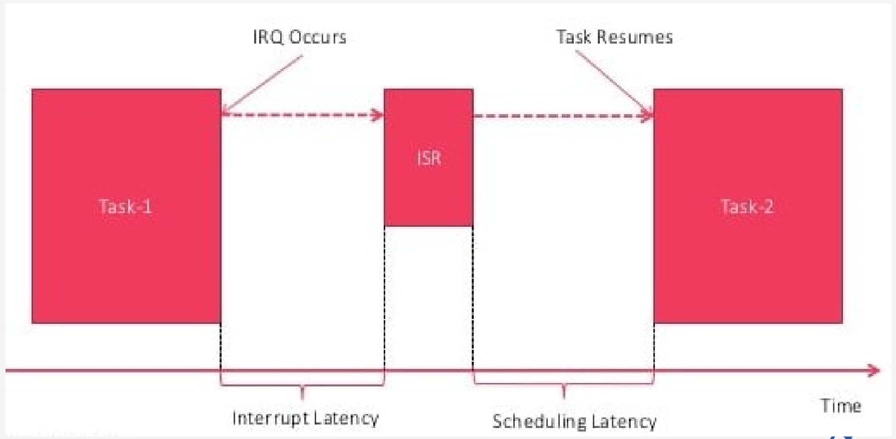
Runs Interrupt Service Routine(ISR).

Interrupt Handler needs to be completed ISR as soon as possible and unmask the other interrupts.



3) Scheduler (Latency+Duration):

- Other interrupts may come in.
- A system call may be executing.
- Higher priority tasks that may be also ready.
- Context switch time (~dispatch latency).
- Priority of the thread scheduled

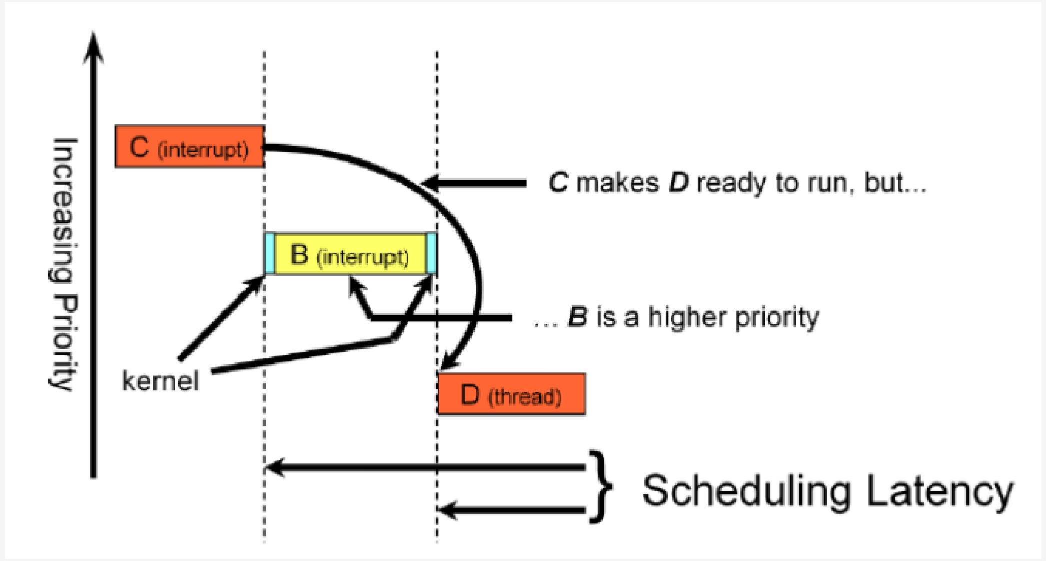


Task Latency

Task Latency = Interrupt Latency + Handler Duration + Scheduler Latency + Scheduler Duration

3) Scheduler (Latency+Duration):

- Other interrupts may come in.
- A system call may be executing.
- Time spent in higher priority tasks that may be also ready.
- Priority of the thread scheduled
- Context switch time.

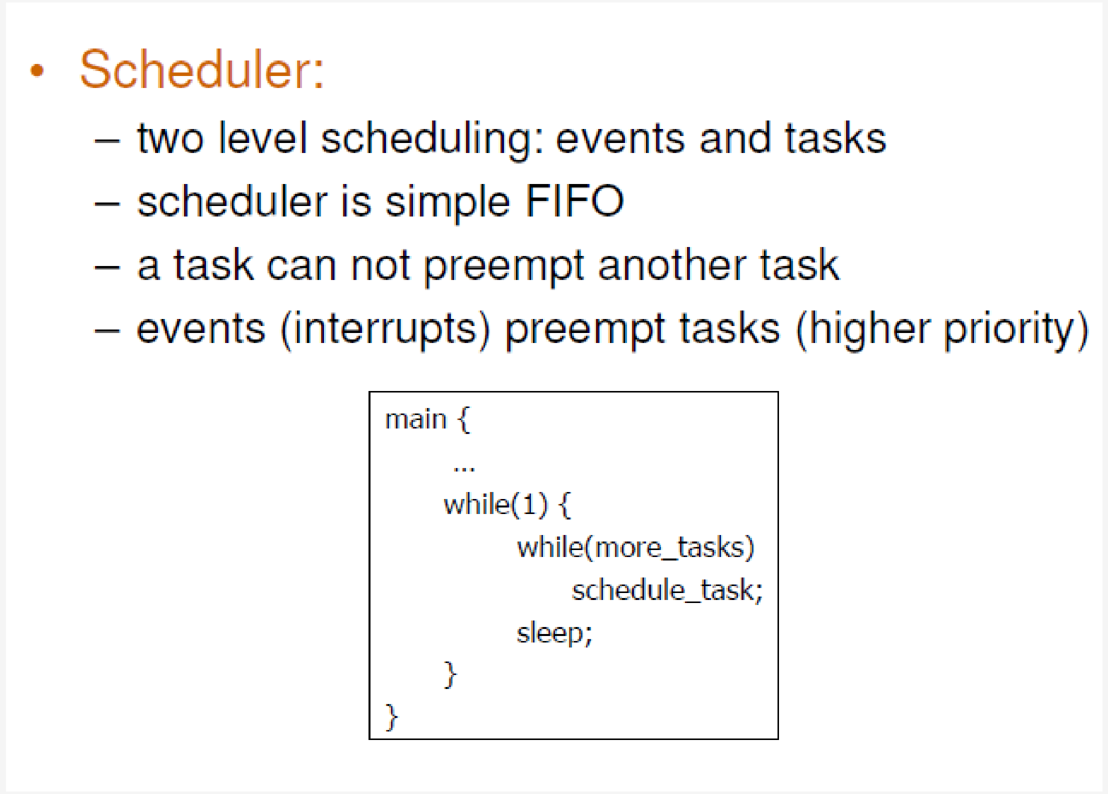


TinyOS: OS for Wide Sensor Networks ~400 bytes

TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices.

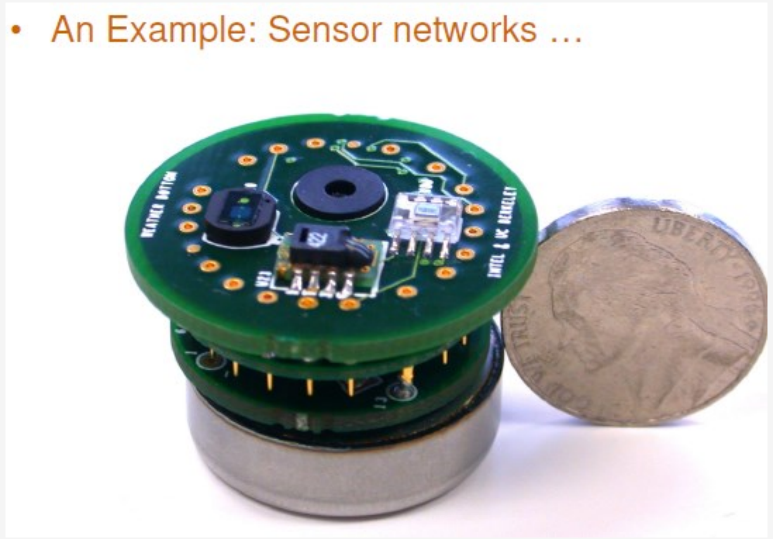
To save energy, node stays asleep most of the time.

TinyOS scheduler continues running until all tasks are cleared, then sends mote back to sleep.



- Scheduler:
  - two level scheduling: events and tasks
  - scheduler is simple FIFO
  - a task can not preempt another task
  - events (interrupts) preempt tasks (higher priority)

```
main {
     ...
     while(1) {
          while(more_tasks)
               schedule_task;
          sleep;
     }
}
```



- An Example: Sensor networks …

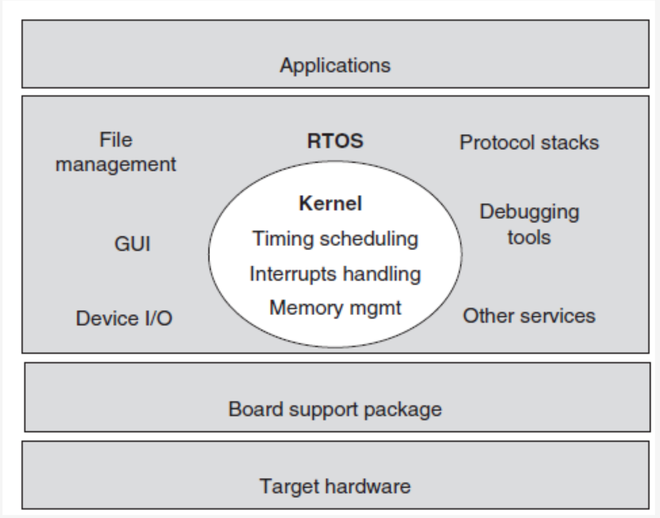
http://www.tinyos.net/

Real-Time Operating Systems

«The ability of the OS to provide a required level of service in a bounded response time.»

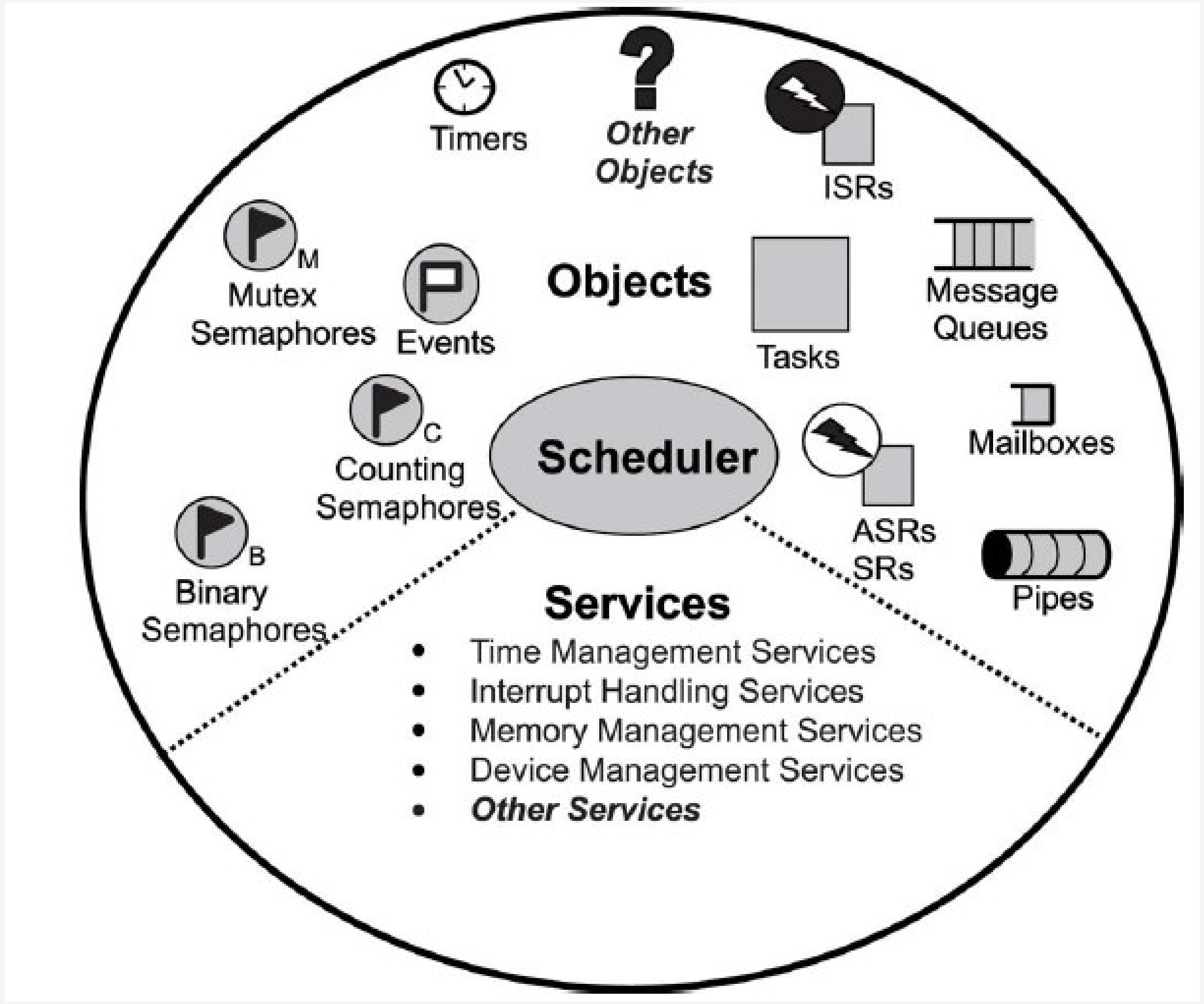
-POSIX Standard 1003.1 (Portable OS Interface specified by IEEE Computer Society.)

- The timing behavior of the OS must be predictable and deterministic. Upper limit is known.
- The OS manage the timing and scheduling according to priorities.
- Context switch times should be small and bounded.
- They must include RT kernels.
- High resolution timers are important.
- They should have controlled kernel size for smaller ESs.
- Sophisticated features of OSs should be removed.



High-level view of an RTOS

Common Components in an RTOS



Basic Functions of RTOS:

- Time management
- Task management (create, terminate)
- Interrupt handling (ISRs)
- Memory management
- Expection handling (deadlock, timeout)
- Task synchronization (sem., mutex)
- Task scheduling (priorty-based)

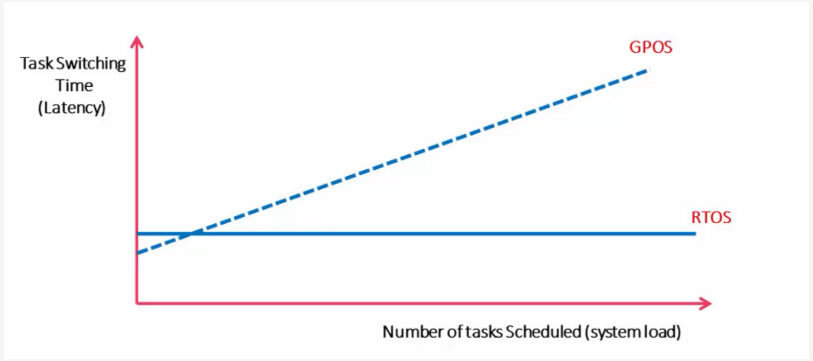
RTOS vs. GPOS

- ❑In an RTOS priority based preemptive scheduling policy is employed
- ❑An RTOS can sacrifice throughput for being deterministic
- ❑In an RTOS <u>dispatch latencies are constant</u>
- ❑RTOSes are best suited for real-time embedded systems

- ❑In a GPOS, the scheduler typically uses a fairness policy to dispatch threads and processes onto the CPU.
- ❑Such a policy enables the high overall throughput
- ❑Most GPOSs have <u>unbounded dispatch latencies</u>
- ❑General Purpose Operating systems are very good at what they are designed for

Some examples to RTOS:

- FreeRTOS,
- LynxOS,
- QNX (Blackberry),
- OSE,
- VxWorks



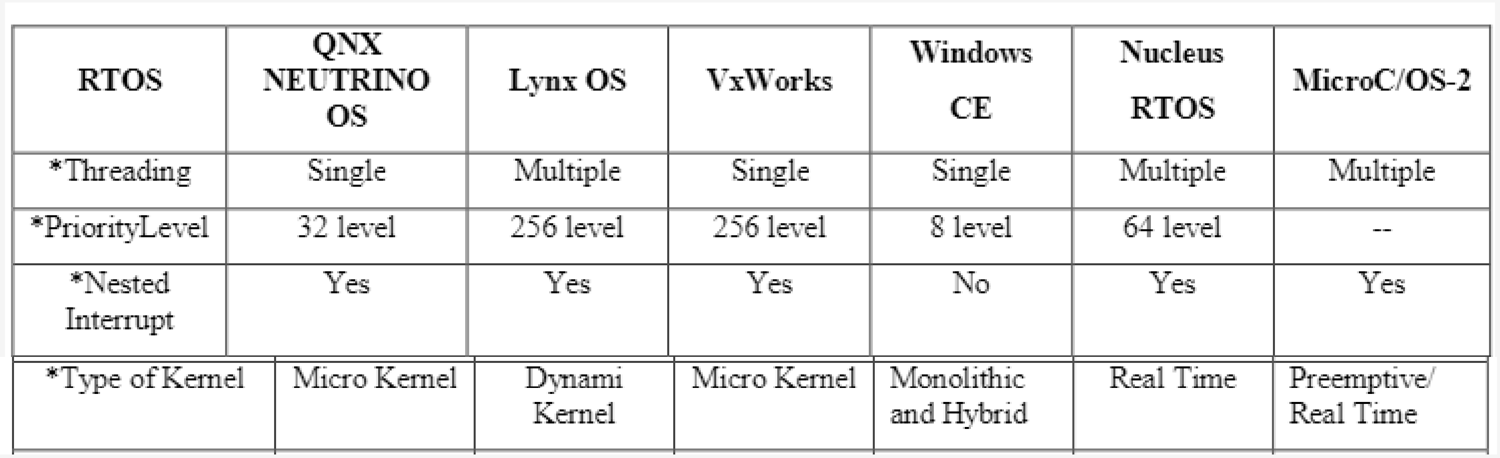
Some examples to GPOS:

- Windows OS,
- Mac OS,
- Linux.

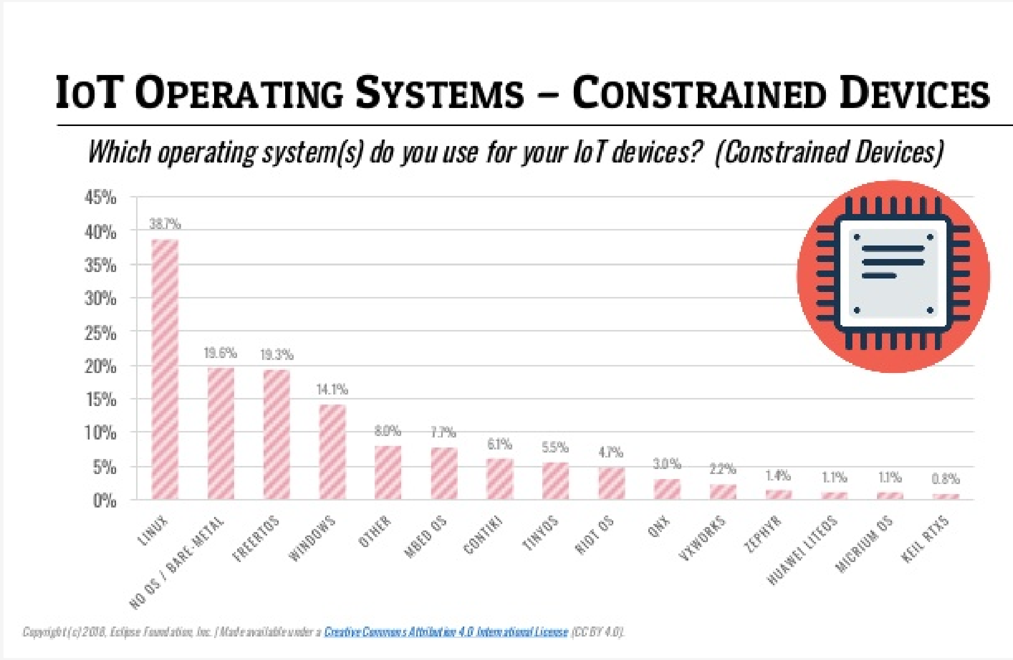
RTOS: 4 categories

- Priority based kernel for embbeded applications e.g. OSE, VxWorks, QNX, VRTX32, pSOS .... Many of them are commercial kernels
  - Applications should be designed and programmed to suite priority-based scheduling e.g deadlines as priority etc
- Real Time Extensions of existing time-sharing OS e.g. Real time Linux, Real time NT by e.g locking RT tasks in main memory, assigning highest priorities etc
- Research RT Kernels e.g. SHARK, TinyOS ... ...
- Run-time systems for RT programmingn languages e.g. Ada, Erlang, Real-Time Java ...

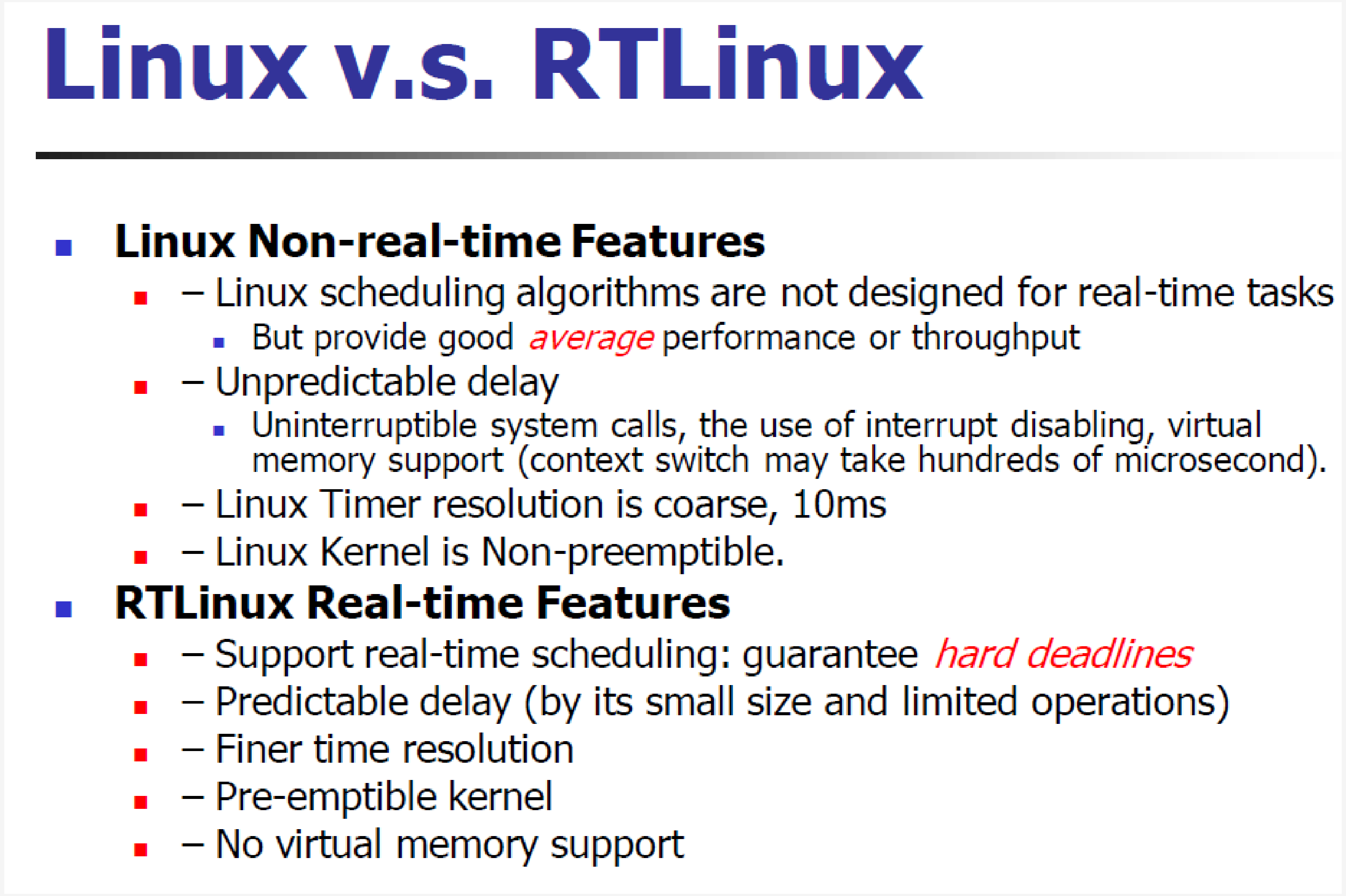
Some comparison about RTOS



RTOS	QNX NEUTRINO OS	Lynx OS	VxWorks	Windows CE	Nucleus RTOS	MicroC/OS-2
*Threading	Single	Multiple	Single	Single	Multiple	Multiple
*PriorityLevel	32 level	256 level	256 level	8 level	64 level	--
*Nested Interrupt	Yes	Yes	Yes	No	Yes	Yes
*Type of Kernel	Micro Kernel	Dynami Kernel	Micro Kernel	Monolithic and Hybrid	Real Time	Preemptive/ Real Time



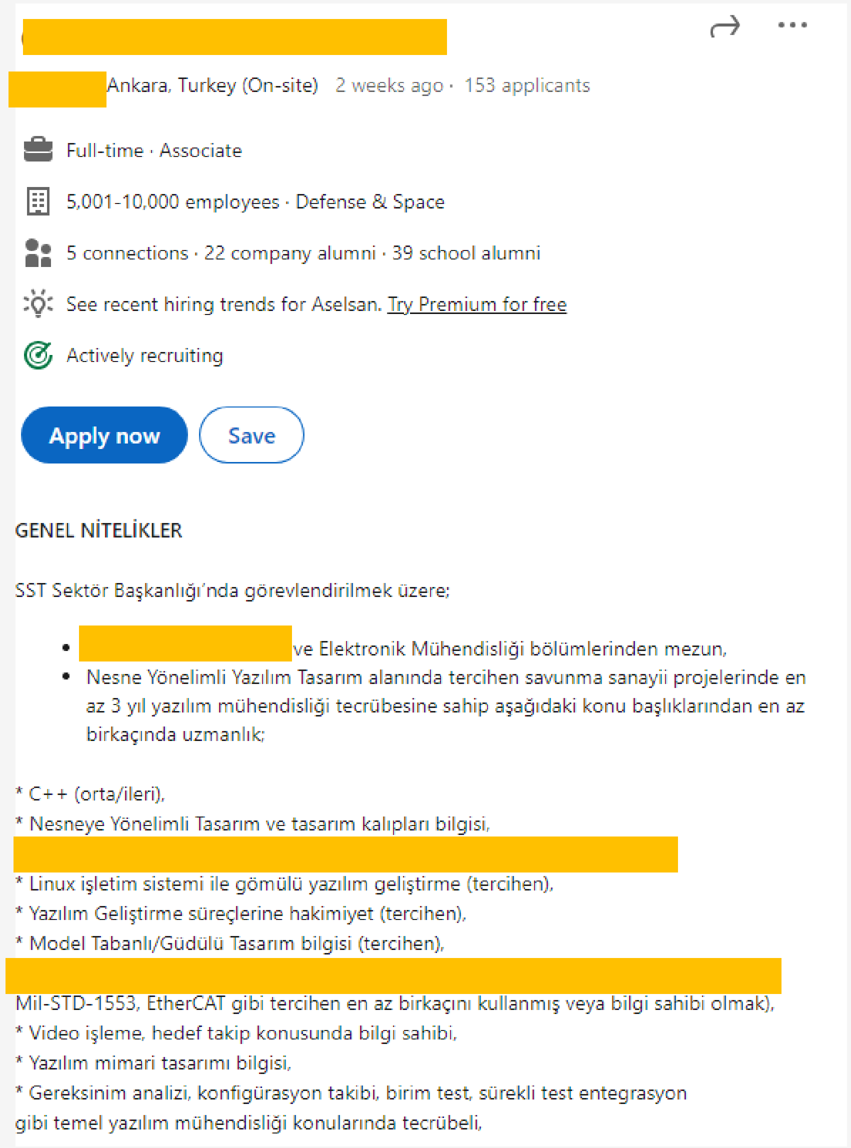
Linux vs RTLinux



Linux v.s. RTLinux

- **Linux Non-real-time Features**
  - – Linux scheduling algorithms are not designed for real-time tasks
    - But provide good *average* performance or throughput
  - – Unpredictable delay
    - Uninterruptible system calls, the use of interrupt disabling, virtual memory support (context switch may take hundreds of microsecond).
  - – Linux Timer resolution is coarse, 10ms
  - – Linux Kernel is Non-preemptible.
- **RTLinux Real-time Features**
  - – Support real-time scheduling: guarantee *hard deadlines*
  - – Predictable delay (by its small size and limited operations)
  - – Finer time resolution
  - – Pre-emptible kernel
  - – No virtual memory support

Embedded Software Developer



[redacted]

[redacted] Ankara, Turkey (On-site) 2 weeks ago · 153 applicants

Full-time · Associate

5,001-10,000 employees · Defense & Space

5 connections · 22 company alumni · 39 school alumni

See recent hiring trends for Aselsan. Try Premium for free

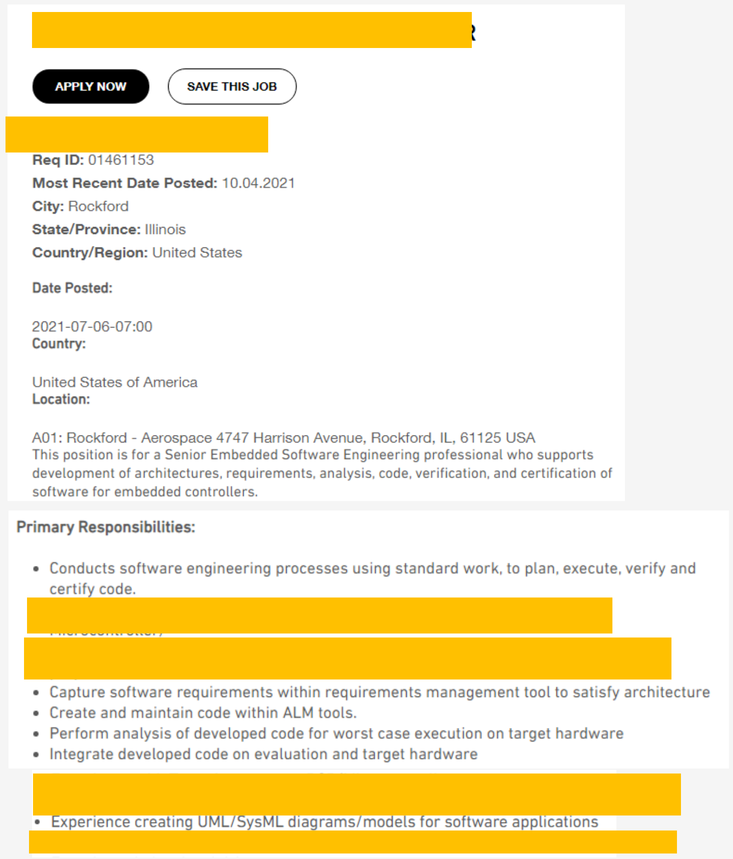
Actively recruiting

GENEL NİTELİKLER

SST Sektör Başkanlığı'nda görevlendirilmek üzere;

- [redacted] ve Elektronik Mühendisliği bölümlerinden mezun,
- Nesne Yönelimli Yazılım Tasarım alanında tercihen savunma sanayii projelerinde en az 3 yıl yazılım mühendisliği tecrübesine sahip aşağıdaki konu başlıklarından en az birkaçında uzmanlık;

* C++ (orta/ileri),
* Nesneye Yönelimli Tasarım ve tasarım kalıpları bilgisi,
[redacted]
* Linux işletim sistemi ile gömülü yazılım geliştirme (tercihen),
* Yazılım Geliştirme süreçlerine hakimiyet (tercihen),
* Model Tabanlı/Güdülü Tasarım bilgisi (tercihen),
[redacted]
Mil-STD-1553, EtherCAT gibi tercihen en az birkaçını kullanmış veya bilgi sahibi olmak),
* Video işleme, hedef takip konusunda bilgi sahibi,
* Yazılım mimari tasarımı bilgisi,
* Gereksinim analizi, konfigürasyon takibi, birim test, sürekli test entegrasyon gibi temel yazılım mühendisliği konularında tecrübeli,



[redacted]

[redacted]

Req ID: 01461153
Most Recent Date Posted: 10.04.2021
City: Rockford
State/Province: Illinois
Country/Region: United States

Date Posted:

2021-07-06-07:00
Country:

United States of America
Location:

A01: Rockford - Aerospace 4747 Harrison Avenue, Rockford, IL, 61125 USA
This position is for a Senior Embedded Software Engineering professional who supports development of architectures, requirements, analysis, code, verification, and certification of software for embedded controllers.

Primary Responsibilities:

- Conducts software engineering processes using standard work, to plan, execute, verify and certify code.

[redacted]

- Capture software requirements within requirements management tool to satisfy architecture
- Create and maintain code within ALM tools.
- Perform analysis of developed code for worst case execution on target hardware
- Integrate developed code on evaluation and target hardware

[redacted]

- Experience creating UML/SysML diagrams/models for software applications

[redacted]