Embedded System
Lecture Note 5. Embedded System Programming(II)

tags: `Embedded`

編號	主　題	內　容　重　點
1	嵌入式系統程式建構模組與裝置控制	• 嵌入式系統程式建構模組與範例 • 嵌入式系統 I/O 程式設計 • Linux 核心模組：LKM 與 LDD
2	嵌入式系統之即時系統概論	• 即時系統 vs. 嵌入式系統 • 即時系統任務模型(Task Model) • 即時系統任務排程(Scheduling)
3	中斷 (Interrupt) 信號處理	• 中斷信號簡介與中斷信號處理機制 • 中斷服務程式(ISR)設計

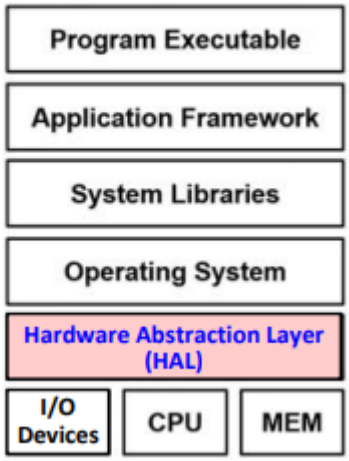
嵌入式系統程式建構模組與裝置控制

嵌入式系統程式建構模組與範例

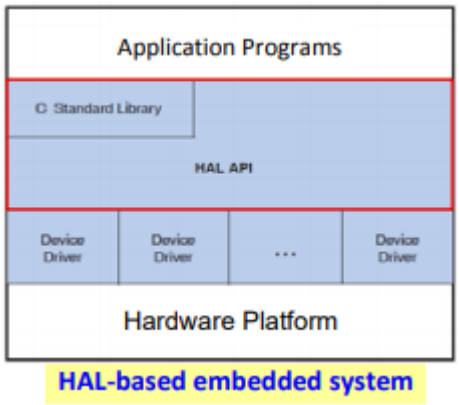
Hardware Abstraction Layer

- Software layer resides between hardware platform and operating system
- Hiding implementation details
- Enabling unified device access
- Reducing development workload
  - Devices: n
  - O/S: m

  $n \times m \Rightarrow n + m$
- Simplifying device management
- Might be O/S-specific
- Help O/S portability & VM



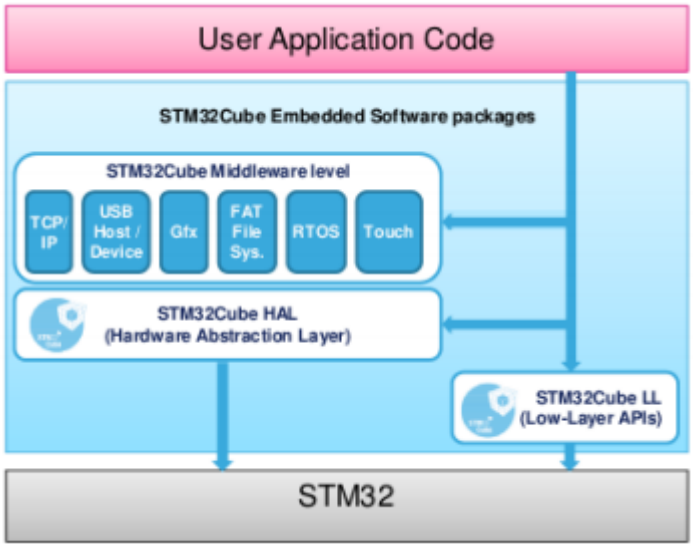
將硬體方面的不同抽離作業系統的核心，核心模式的程式碼就不必因為硬體的不同而需要修改
=> 因此硬體抽象層可加大軟體的移植性



HAL-based embedded system

- HAL serves as a light-weight runtime environment
- HAL provides simple device driver interfaces for application programs
- Fundamental services:

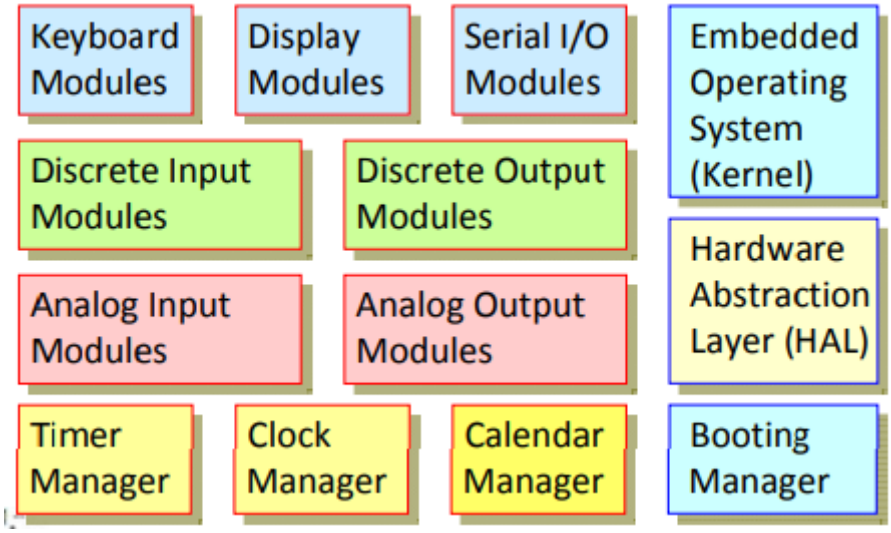
    - API for accessing devices
    - Device drivers
    - Initialization function
- HAL is based on specific hardware system
- HAL: Example - STM32

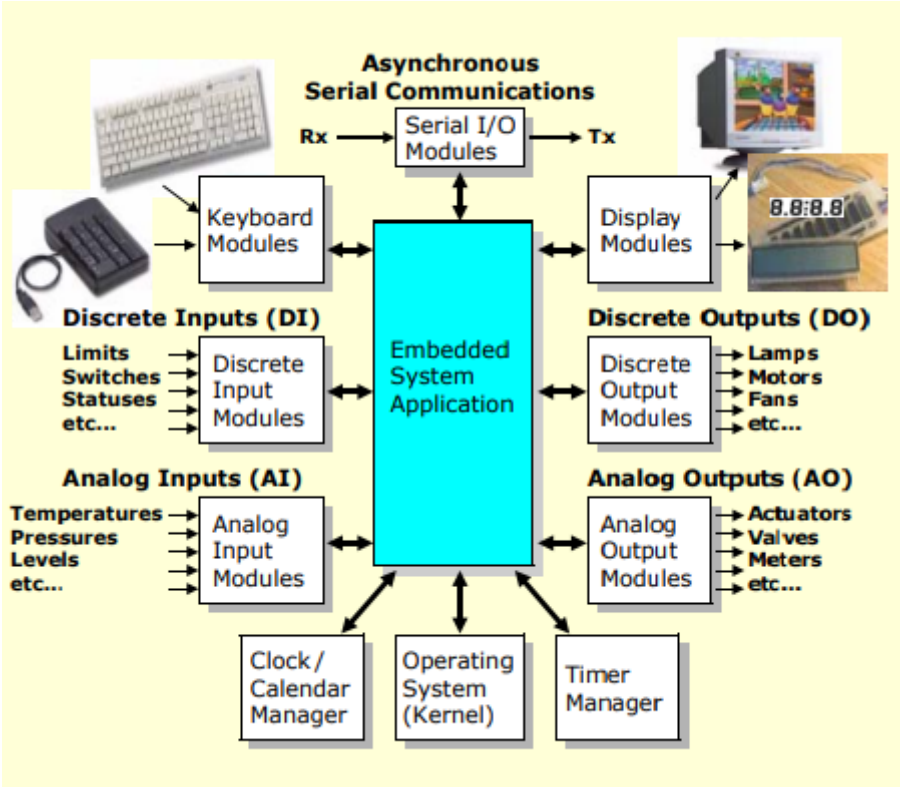


- HAL: Example - Linux 下的HAL
  它不是位於操作系統的最底層，直接操作硬件，相反，它位於操作系統和驅動程序之上，是一個運行在用戶空間中服務程序
- Developing Software Building Blocks for Embedded Systems:
    - Focused on device management
    - Unified access model of devices
    - Functional partition
    - Layered structures
    - Encapsulation of implementation details
    - Isolation of hardware dependency
    - Help portability
    - Application interface of unified access model
    - Common function modules
    - Configuration parameters
    - Device independent function modules

- Device dependent function modules
  - The Unified Access Model:
    - Initialization/Reset
    - Start-Up and Shutdown
      - Open/Close
      - Connect/Disconnect
      - Start/Stop
    - Data transfer:
      - Input – Get/Input/Read
      - Output – Put/Output/Write
    - Set-up/Mode control
    - Status Inquiry

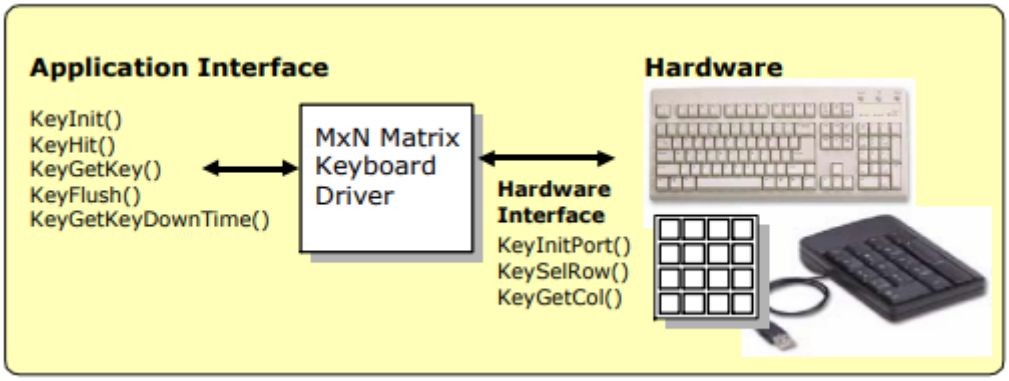
- The Categories:
  - Basic input/output functions (Keyboard & Display)
  - Clock and timer manager functions
  - Discrete input/output functions
  - Analog input/output functions
  - Serial (Asynch/Synch) communication functions
  - Networking functions
  - Operating system support





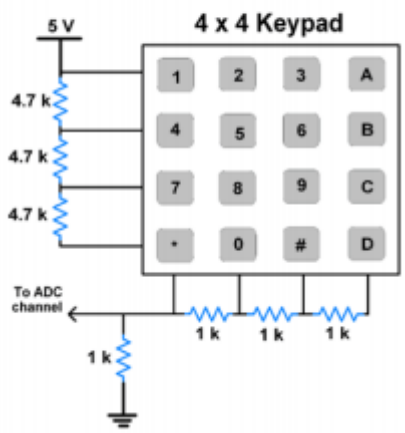
嵌入式系統I/O程式設計

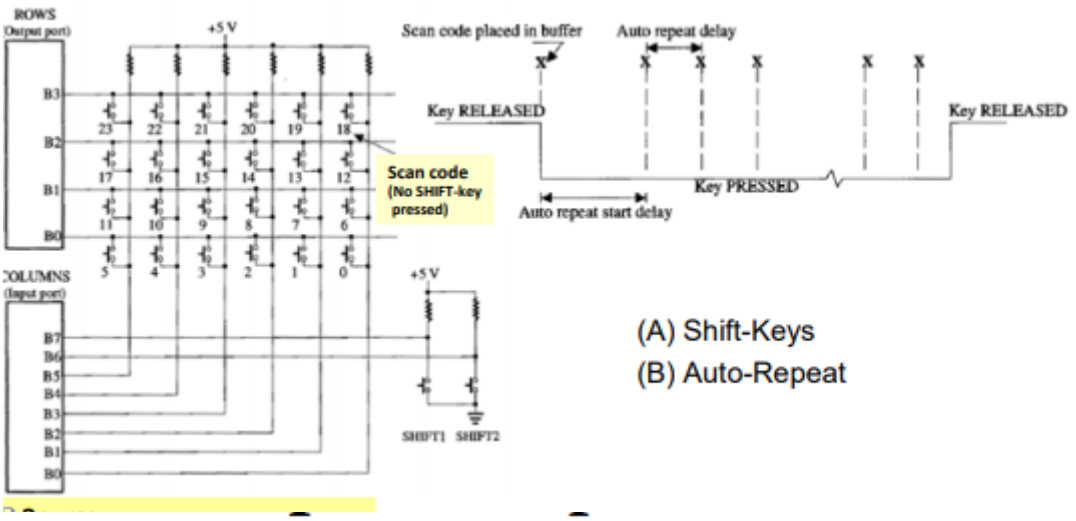
- Special Keyboard Functions

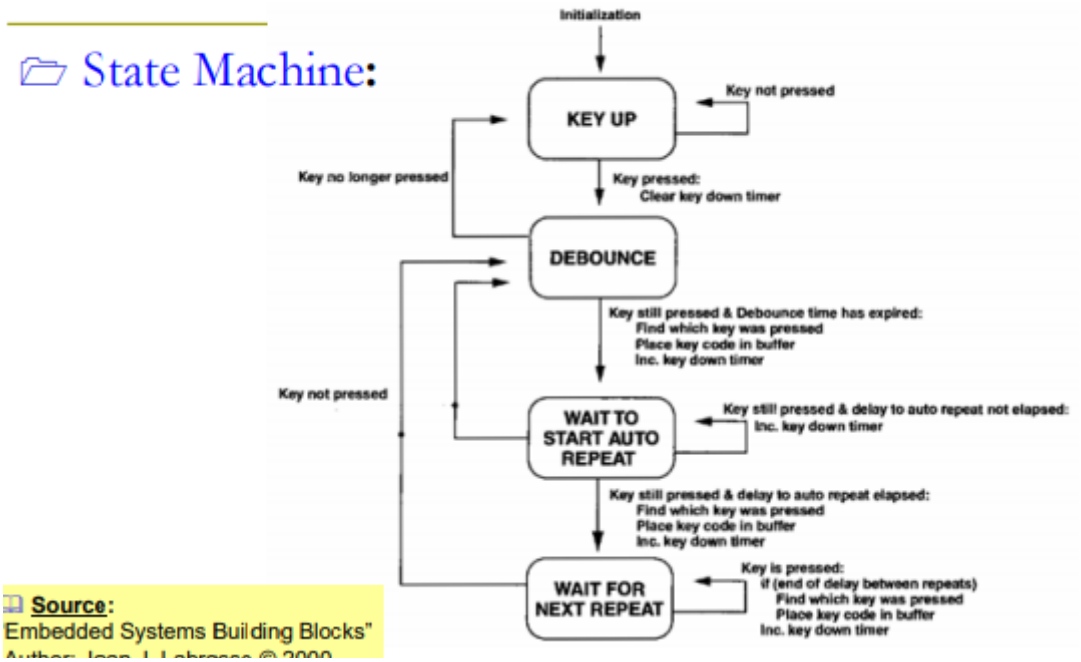


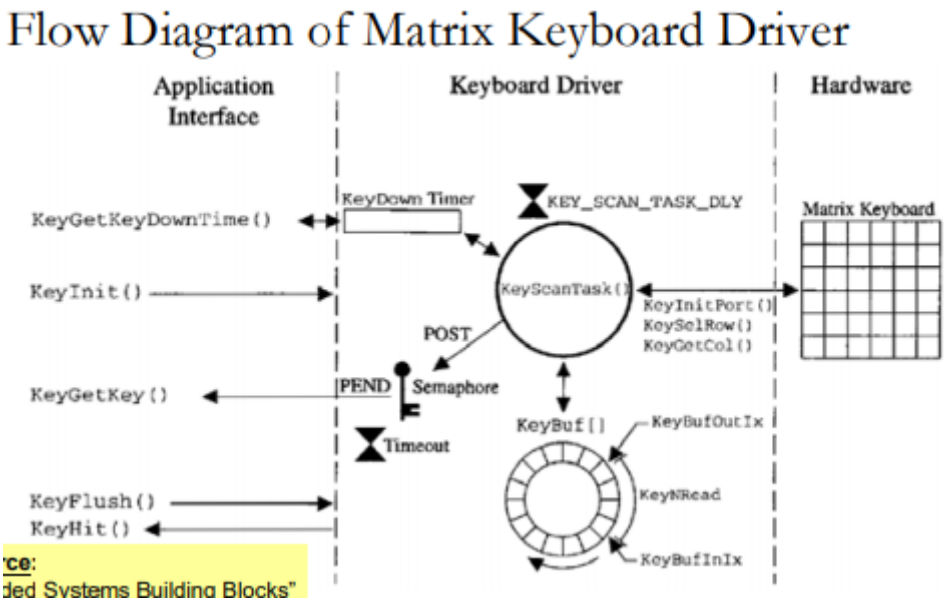
Keyboard Function Interfaces:

- Asynchronous event
- Implementation
  - Keyboard driver module
  - API
- Issues
  - Debouncing
  - Key rollover
  - Interrupt driven or polling?
  - Keyboard buffer/status







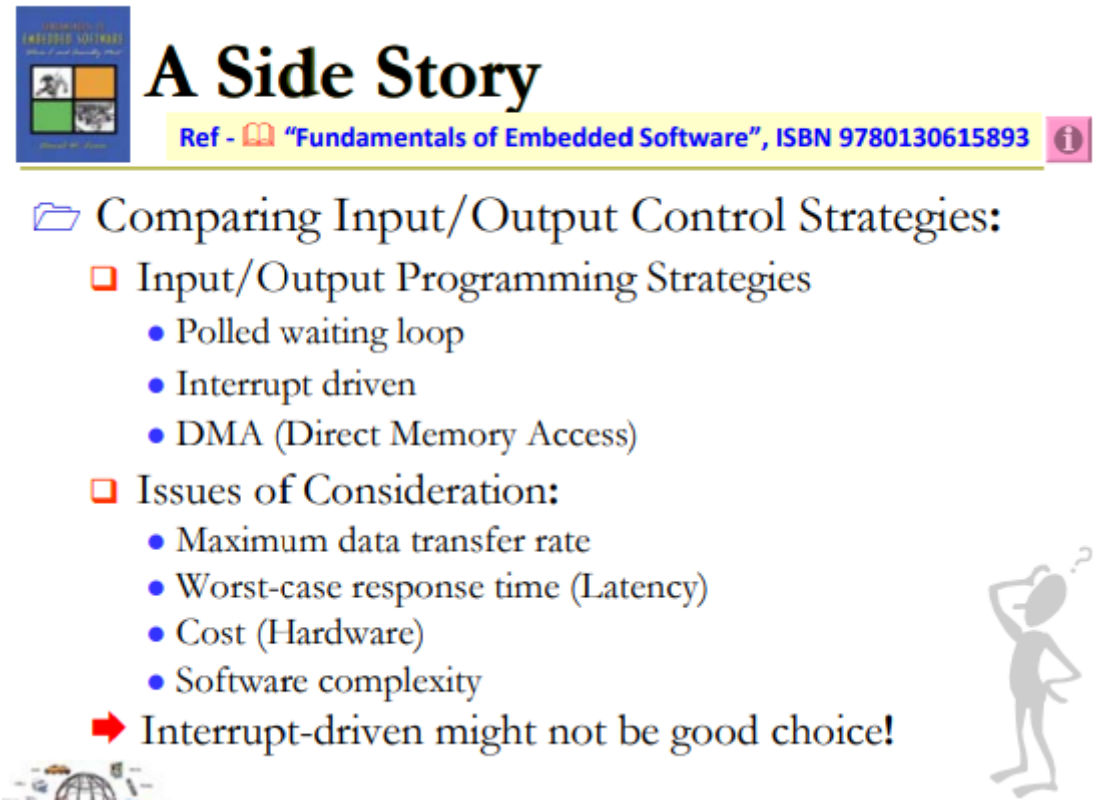


- Input/Output Programming and Device Control
  - Basic input/output architecture:
    - Memory-mapped - Access as normal memory read/write

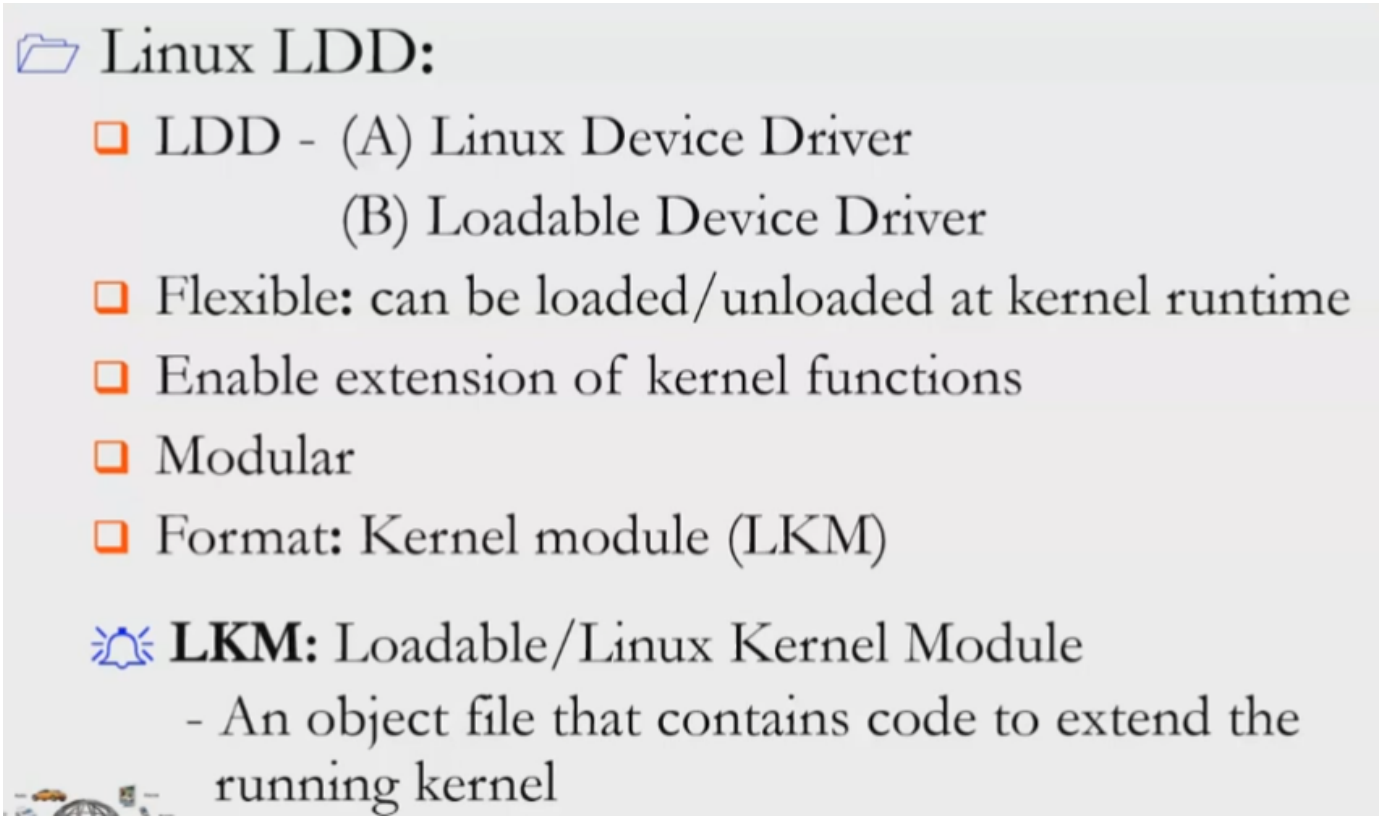
        - Port-mapped - Access using special I/O instructions
    - Common interfaces on input/output devices:
        - Command/Control registers
        - Data registers
        - Status registers
    - (A) Register might be shared by different functions
    - (B) Device might not be directly connected to CPU/Bus
    - Basic input/output control:
        - Initialize/Reset
        - Data transfer operations – Read/Write operations
        - Status inquiry
    - Programming techniques:
        - Blocking/Non-Blocking operations
        - Polled waiting loop
        - Interrupt-driven
        - DMA (Direct Memory Access)
        - Buffering (dual buffer, multiple buffers)
    - Selecting Input/Output Programming Strategy:
        - Maximum data transfer rate
        - Worst-case response time (Latency)
        - Cost (Hardware)
        - Software complexity

A demonstrative comparison:

	Polled Waiting Loop	Interrupt	DMA
Data Transfer Rate	Moderate	Low	High
Latency	Unpredictable	Moderate	Best
Hardware Cost	Least	Low	Moderate
S/W Complexity	Low	Moderate	Moderate



Linux 核心模組：LKM 與 LDD



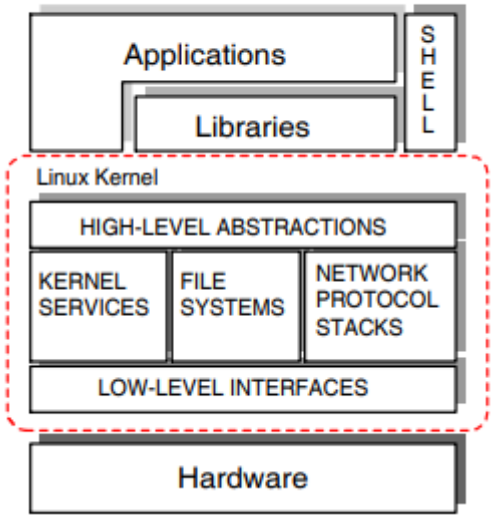
- Device Driver:
  - The program that controls a device
  - The program allows/helps user (high-level) program interact with a device
- A (translation) layer between program and device
- accepts generic commands from programs

- translate generic commands into specialized commands for controlling a device
- Low-level program modules associated with a device
- For flexibility a device driver in Linux usually is implemented as an LKM

Linux Operating System:

- Main Types:
  - Desktop (mainly x86)
  - Server
  - Embedded Linux
  - Non-MMU (μC-Linux)
- Embedded Linux
  - General concepts apply
  - Various hardware
  - Resource constraint
  - Ref: Yocto project

The architecture of generic Linux system

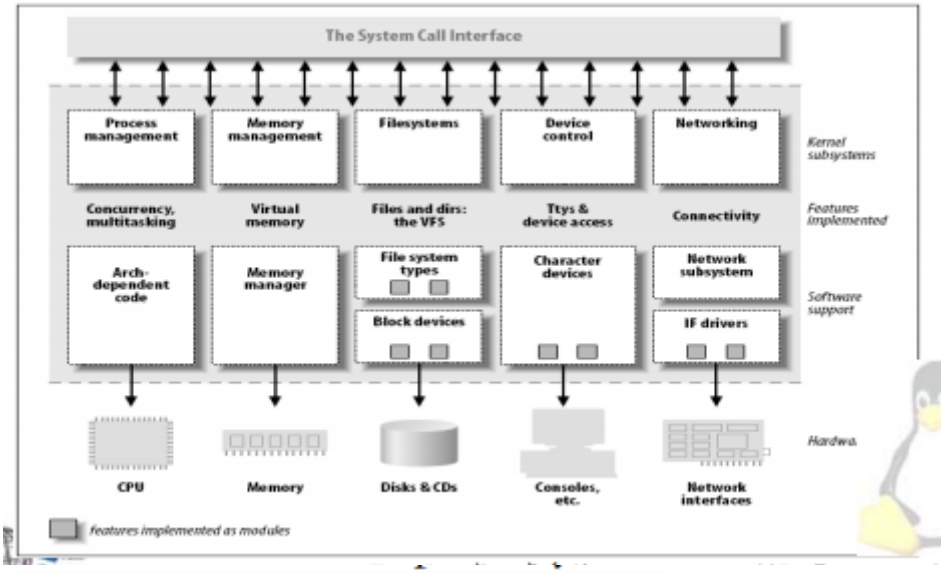


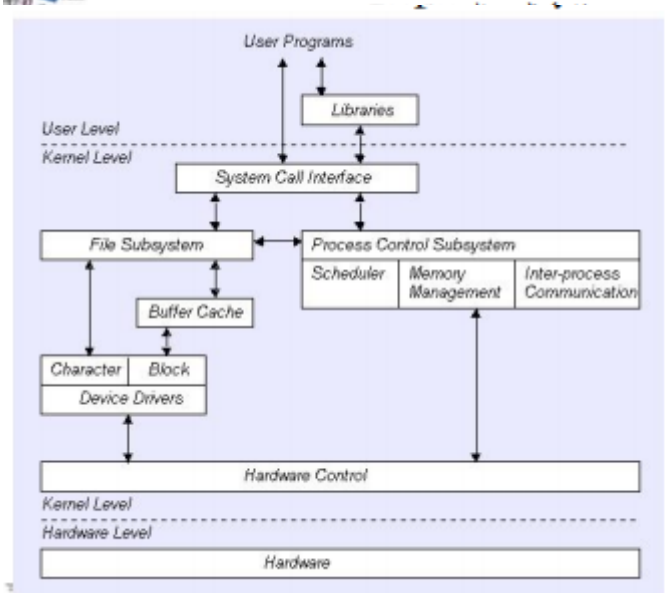
The advantages

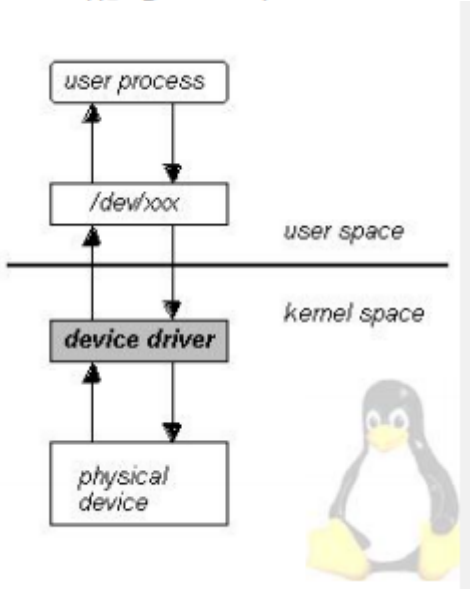
- Availability of source code and tools
- Quality and reliability of code
- Hardware support
- Communication protocol and software standards
- Community support
- Vendor independence
- … and more

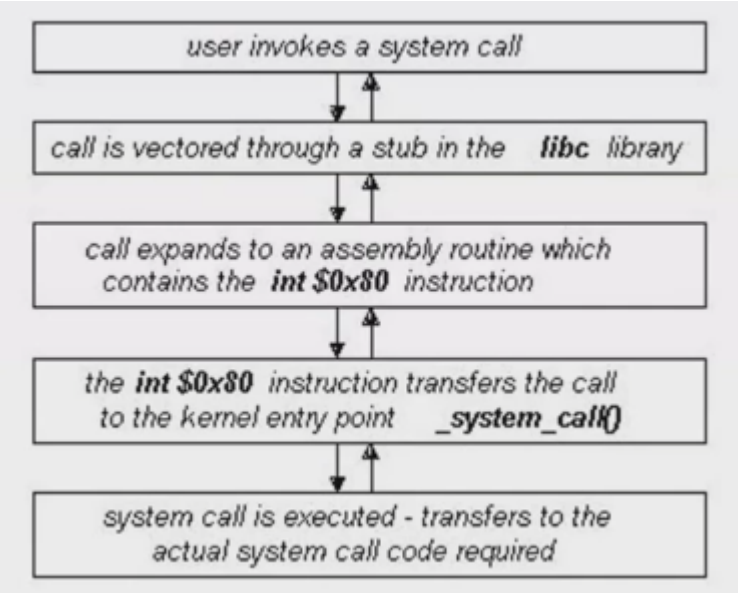
- The challenges
  - Memory management
  - Footprint
  - Boot loader
  - Real-time requirement
  - Device drivers

- Device Driver in Linux





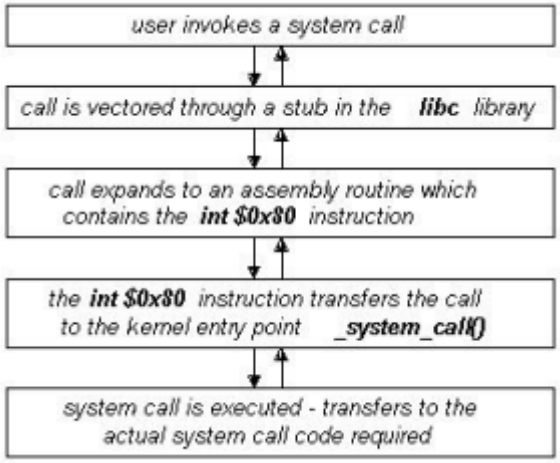




on x86 platform

INT 80H

- Wrapper function:
  (glibc) syscall(...)
  /w <unistd.h>
  <sys/syscall.h>
  <errno.h>
- Kernel function:
  _system_call()



%eax	Name	Source	%ebx
1	sys_exit	kernel/exit.c	int
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs
3	sys_read	fs/read_write.c	unsigned int
4	sys_write	fs/read_write.c	unsigned int
5	sys_open	fs/open.c	const char *
6	sys_close	fs/open.c	unsigned int
7	sys_waitpid	kernel/exit.c	pid_t
8	sys_creat	fs/open.c	const char *
9	sys_link	fs/namei.c	const char *
10	sys_unlink	fs/namei.c	const char *
11	sys_execve	arch/i386/kernel/process.c	struct pt_regs
12	sys_chdir	fs/open.c	const char *
13	sys_time	kernel/time.c	int *

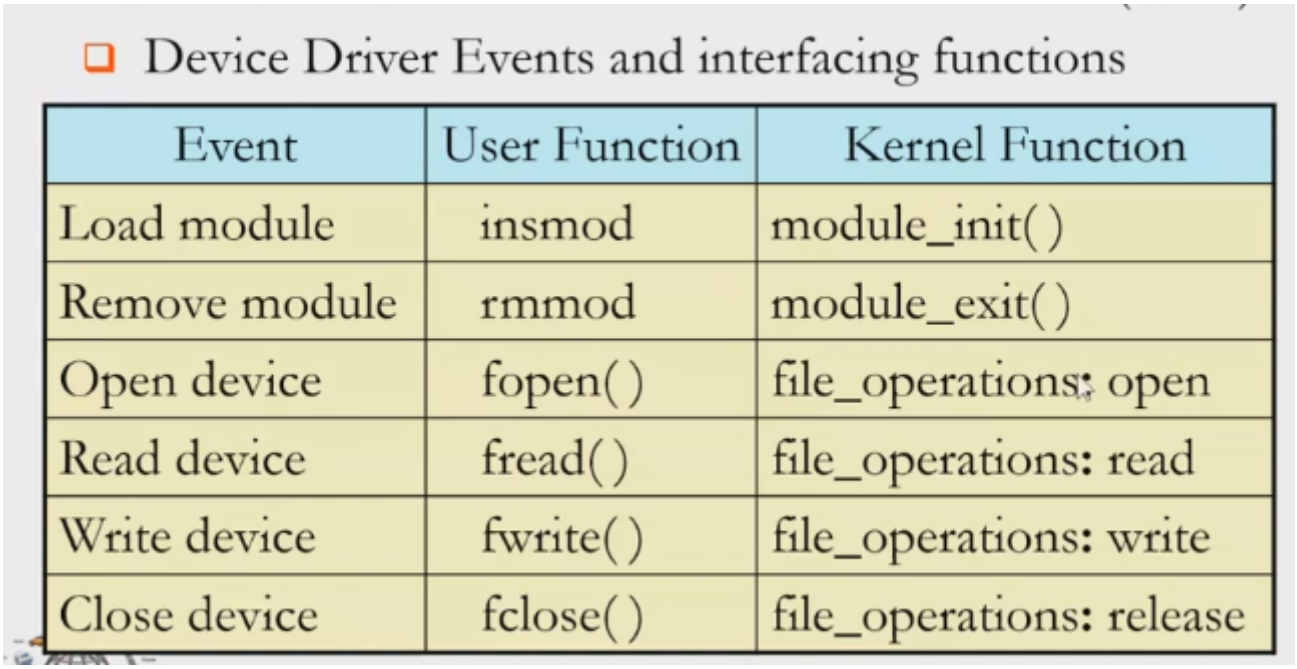
- Devices in Linux Operating System:
  - Classes
    - Character devices (Non-buffered I/O)
    - Block devices (Buffered I/O)
    - Network Interfaces

- Representation
  - Names in /dev
  - Node with (major number, minor number)
    - major number: identifying device group
    - minor number: identifying a specific device in a group
    - Range: 0 ~ 255
- Access – Unified as file manipulation operations
  - open (fopen)
  - close (fclose)
  - read (fget)
  - write (fput)
  - ...
- Example of devices in /dev

```
$ ls -la /dev/hda? /dev/ttyS?
brw-rw---- 1 root disk     3,  1 2004-09-18 14:51 /dev/hda1
brw-rw---- 1 root disk     3,  2 2004-09-18 14:51 /dev/hda2
crw-rw---- 1 root dialout  4, 64 2004-09-18 14:52 /dev/ttyS0
crw-rw---- 1 root dialout  4, 65 2004-09-18 14:52 /dev/ttyS1
```

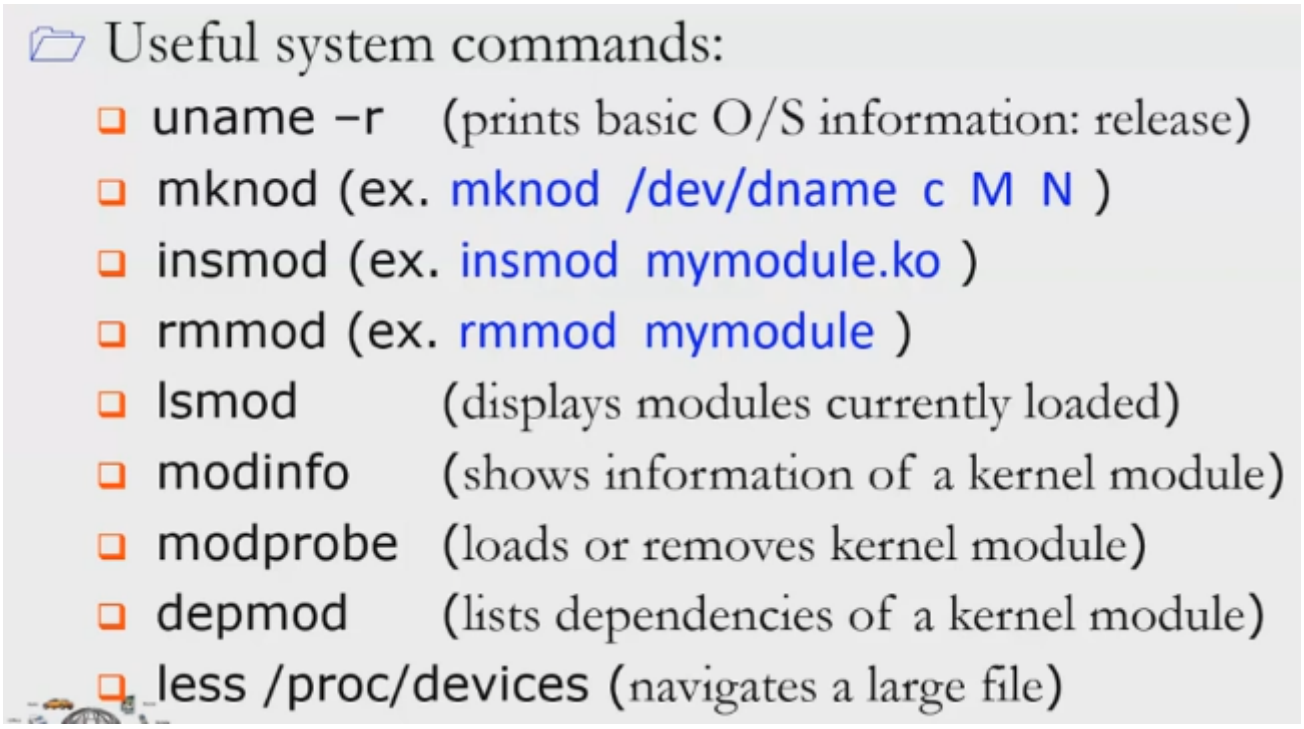
Building Linux Device Drivers

- Fundamentals:
  - Concept: Kernel Space vs. User Space
  - Transition from user space to kernel space
    - System call
    - Interrupt (ISR)
  - Basic components of device driver (module):
    - include file(s)
    - essential macros: module_init()/module_exit()
    - operation function pointers: struct file_operations
    - registration: register_chrdev()/unregister_chrdev()
    - ...



- Device Driver Events and interfacing functions

Event	User Function	Kernel Function
Load module	insmod	module_init( )
Remove module	rmmod	module_exit( )
Open device	fopen( )	file_operations: open
Read device	fread( )	file_operations: read
Write device	fwrite( )	file_operations: write
Close device	fclose( )	file_operations: release



- Useful system commands:
  - uname –r (prints basic O/S information: release)
  - mknod (ex. mknod /dev/dname c M N)
  - insmod (ex. insmod mymodule.ko)
  - rmmod (ex. rmmod mymodule)
  - lsmod (displays modules currently loaded)
  - modinfo (shows information of a kernel module)
  - modprobe (loads or removes kernel module)
  - depmod (lists dependencies of a kernel module)
  - less /proc/devices (navigates a large file)

嵌入式系統之即時系統概論

即時系統與應用簡介

What is real-time computing?

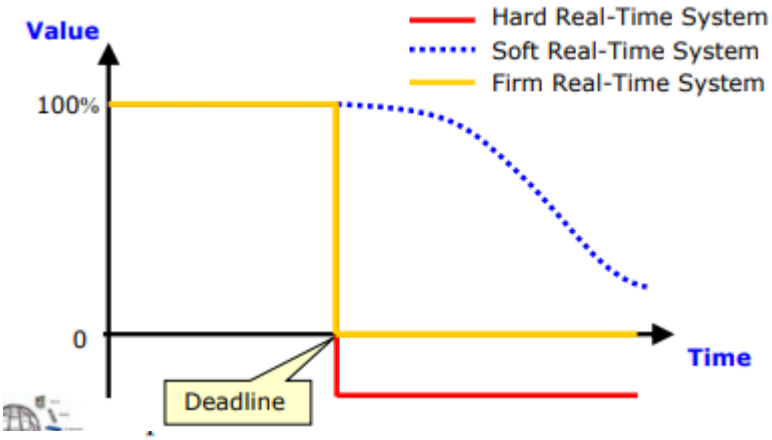
- the correctness of the system depends on not only the logically correct result of the computation but also on the time at which the results are produced. i. e.
  - Logical correctness
  - Timeliness

What is a Real-Time System (RTS)?

- The definition
  - An information processing system which must respond to external input stimuli within a finite and specified time period (deadline)
- Correctness depends on
  - Logical result of computation/processing
  - The time at which the results are produced
- Objectives – Meeting
  - Functional requirements (Mandatory)
  - Timing constraints (Application specific)

Types of Real-Time Systems

- Hard real-time System
  - systems where it is absolutely imperative that responses occur within the required deadline, e.g., flight control systems.
- Soft real-time System
  - systems where deadlines are important but which will still function correctly if deadlines occasionally are missed, e.g., data acquisition system.
- Firm real-time System
  - systems which are soft real-time but in which there is no benefit from late delivery of service.
- Real real-time System
  - systems which are hard real-time and which the response times are very short, e.g., missile system.



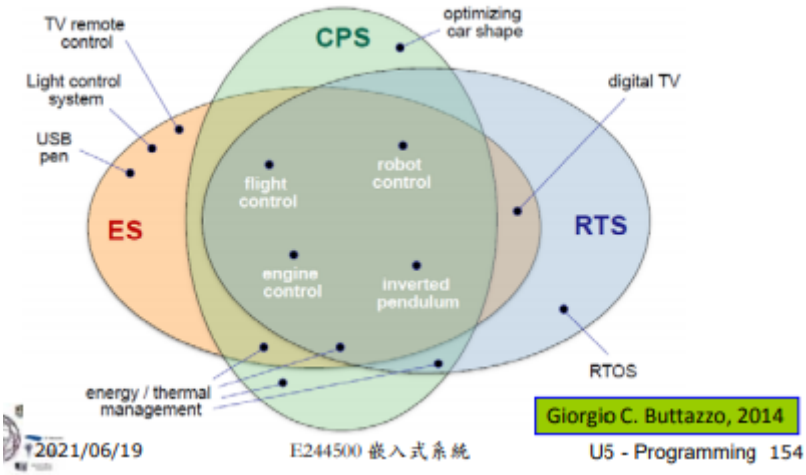
Item	General Computing	Real-Time Computing
Timing Constraint	Generally NO	YES (Hard/Soft/Firm)
Correctness	Logical results	Logical & Timely
Objectives	Optimization in • Performance • Throughput	Meet Deadlines (with guarantee)
Scheduling	• RR(Round-Robin) • Dynamic Priority for fairness or throughput • Preemptive	• Priority-based • Non-preemptive or preemptive • Cyclic executive

Characteristics	Hard Real-Time	Soft Real-Time
Deadlines	Hard	Soft
Pacing	Environment	Computer
Peak-Load Performance	Predictable	Degraded
Safety	Critical	Non-critical
Error Detection	System	User
Redundancy	Active	Standby
Time Granularity	Millisecond (msec.)	Second (sec.)
Data Files	Small/Medium	Large
Data Integrity	Short term	Long term

Real-Time System vs. Embedded System

- Embedded System
  - Any device that
    - includes a programmable computer but
    - is not itself a general-purpose computer
  - Computer as component
- An embedded system might be developed for real-time applications
- Real-Time System is normally application specific

- Common: Real-Time Embedded Systems



- Examples

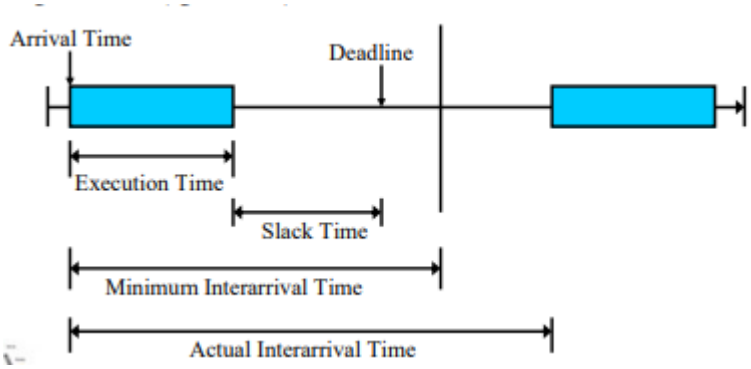
❑ Real-Time and Embedded:
- Nuclear reactor control
- Safety critical system
- Flight control
- Robots
- GPS
- Mobile phone
- MP3 player

❑ Real-Time (but not Embedded):
- Stock trading system
- Networked Multimedia Systems
- Skype

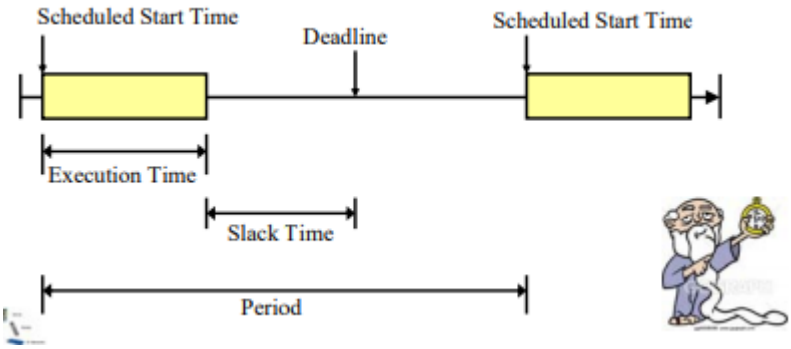
❑ Embedded (but not Real-Time):
- Home appliances
- Traffic light control
- Sprinkler system

即時系統任務模型(Task Model)

Event-Driven Task

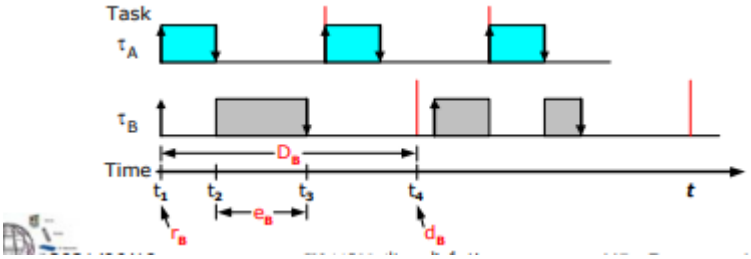


Time-Driven Task



Each job J_i of a task is characterized by

- r_i : release time of J_i
- d_i: absolute deadline
- D_i: relative deadline
- e_i : execution time.



Categories of Tasks

Periodic task:

- A *period* p_i is associated with each task τ_i
- p_i is the minimum time between job releases

Sporadic and aperiodic tasks:

- Released at arbitrary times
- Sporadic Task: hard deadline
- Aperiodic Task: soft deadline or no deadline

- Periodic Task: Modeling

  A periodic task is a sequence of jobs with identical parameters: $\tau_i = \{ J_{i,1}, J_{i,2}, \cdots, J_{i,n} \}$

  - *period* (p_i or T_i)
    - the minimum length of time between the release times of consecutive jobs
  - *execution time* (e_i or C_i)
    - the maximum execution time of any job in the task (**WCET: W**orst **C**ase **E**xecution **T**ime)
  - a *phase* (ϕ_i)
    - the release time of the first job in τ_i (i.e. $J_{i,1}$)

- Periodic Tasks: Some Terms

  The jobs of task τ_i : $J_{i,1}, J_{i,2}, \cdots$

  phase of τ_i : $r_{i,1}$ (the release time of $J_{i,1}$)

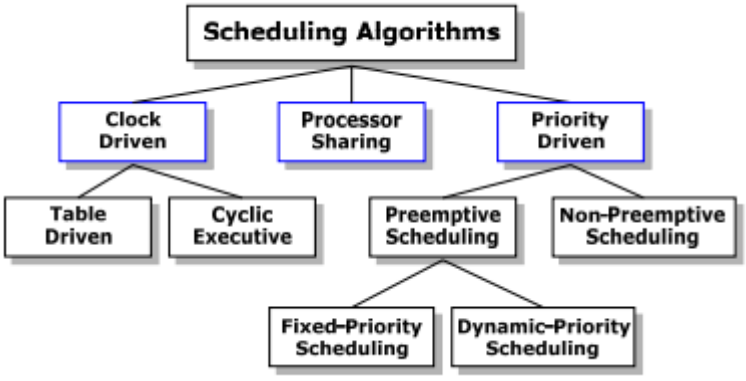
  - **Synchronous System:** $r_{i,1} = 0, 1 \leqq i \leqq n$
  - **Asynchronous System:** Phases are arbitrary

  Hyperperiod: LCM of $\{ p_i \}$

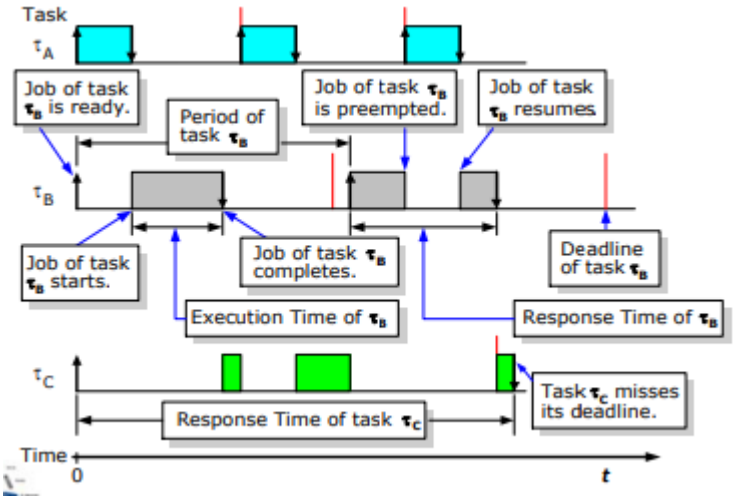
  Task utilization: $u_i = e_i / p_i$

  System utilization: $U = \sum_{i=1..n} u_i$

即時系統任務排程(Scheduling)

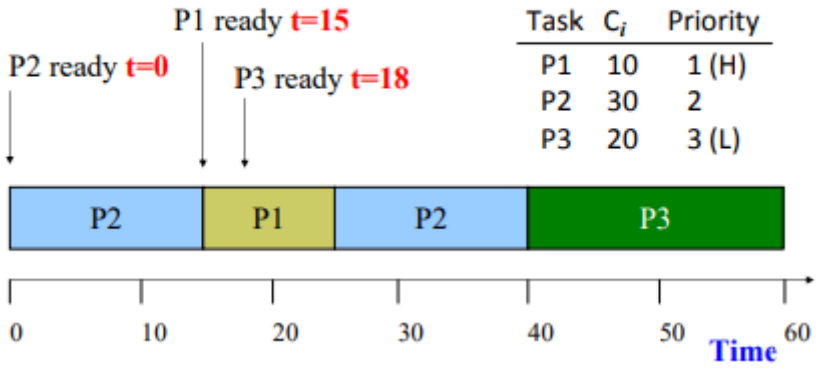


Task Scheduling: Priority-Based

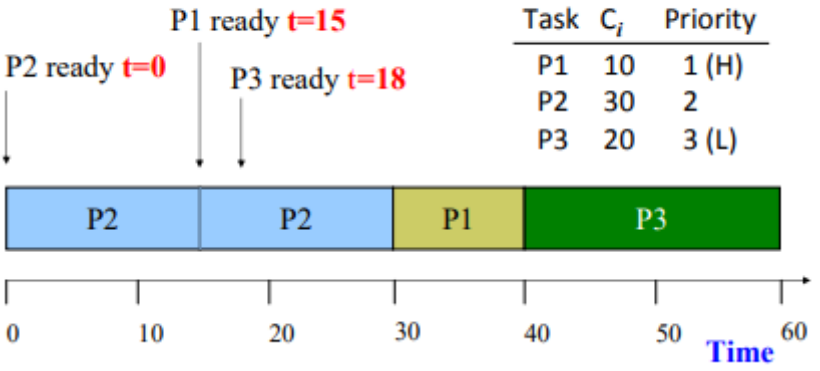


Examples of Non-preemptive vs. Preemptive

Priority-Based Preemptive Scheduling:



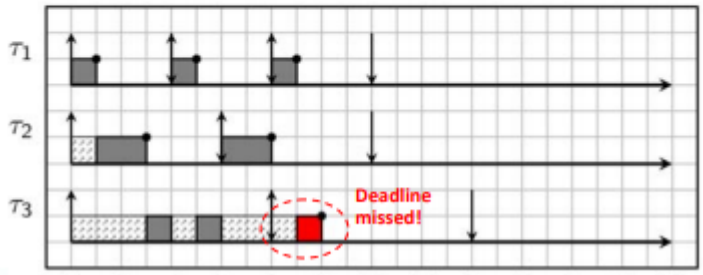
Priority-Based Non-preemptive Scheduling:



Examples of Fixed Priority vs. Dynamic Priority

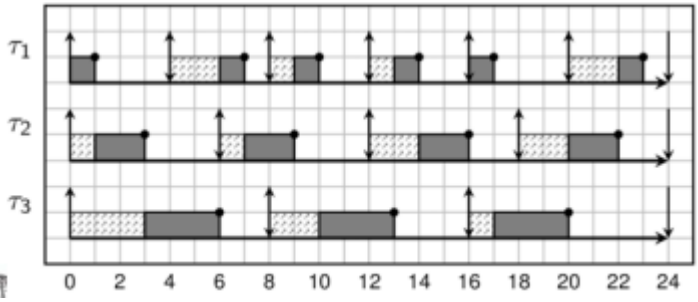
Real-Time Scheduling of Tasks τ_1, τ_2, τ_3

Task	Period	Exec Time	Deadline	Priority
τ_1	4	1	4	H
τ_2	6	2	6	M (Fixed)
τ_3	8	3	8	L



Real-Time Scheduling of Tasks τ_1, τ_2, τ_3

Task	Period	Exec Time	Deadline	Priority
τ_1	4	1	4	
τ_2	6	2	6	(Dynamic)
τ_3	8	3	8	



Priority Assignment

- Priority AssignmentHow are priorities assigned to tasks such that all the deadlines are met?
- Solutions: Good scheduling algorithms
  - Rate-Monotonic (RM) scheduling algorithm
    - 依據頻率高低決定優先權，週期短(頻率高)有高優先權，週期長(頻率低)較低優先權
    - Static (fixed) priority assignment
    - Task with shorter period is assigned higher priority
    - Schedulability bound: n(2^1/n - 1)
  - Deadline-Monotonic (DM) scheduling algorithm
    - 依據誰的deadline先到，誰的優先權就越高
    - Dynamic priority assignment
    - Task with earlier absolute deadline is assigned higher priority
    - Schedulability bound: 100%
  - Earliest-Deadline-First (EDF) scheduling algorithm

Clock-Driven Scheduling (Timer-Driven scheduling)

- schedule a set of periodic tasks
- all job parameters are known a priori
- static scheduling based on system clock/timer
- scheduler execution trigged by timer interrupt
- might not be adaptive to any change at runtime
- in general can be efficient
- Compute static schedule off-line (e.g. at design time) – note that expensive algorithms can be used
- Only periodic tasks are scheduled. Idle times can be used for aperiodic jobs.
- Scheduling table has entries of type (tk, J(tk)), where: tk is the decision time, and , J(tk) is the job to start at time tk
- Input: Schedule (tk, J(tk)), k = 0, 1, ..., n-1

Schedule Table

- Periodic jobs are scheduled using a static schedule computed off-line and stored in a schedule table T
- Timer-driven scheduler (cyclic executive) works based on a given schedule table

中斷 (Interrupt) 信號處理

總結