10/07/18

Embedded System Design

System :-

A System is a way of working, organizing or doing one or many tasks according to a fixed plan, program, or set of rules. A System is also an arrangement in which all its units assemble and work together according to the Plan or program.

EX : A watch and a washing machine.

Embedded System :-

An embedded system is a System that has embedded software and computer - hardware, which makes it a system dedicated for an applications or specific part of an application or product or a part of a larger System. (10-15 applications)

Embedded System is defined in several ways.

(1). wayne wolf defined embedded System as,

any device that includes a programmable Computer but is not itself intended to be a general-purpose computer and a fax machine or a clock built from a microprocessor is an embedded Computing System.

(2). Todd D. Morton defined as,

Embedded Systems are electronic systems that contain a microprocessor or microcontroller, but we do not think of them as Computers - the computer is hidden or embedded in the System.

(3) Tim wilmshurst,

An embedded System is a System whose principal function is not computational, but which is Controlled by a computer embedded within it. The word embedded

A computer is a System that has the following components

1. A microprocessor
2. A large memory < Primary memory / Secondaly memory.

Microprocessor – No-onchip memory , 8085, 8086

microcontroller – has on chip memory which is ROM., 8051, ARM.

→ 8051 General representation is 8XYZ. where X represents,
Pre loaded or user can write only one time (ex: vender gives os s/w

0 → PROM / OTP ROM — one time program.
user can Write program to controller.

7 → UVPROM / EEPROM
(Rewrite) 1 to 100 times (ex: emply CD)
↱ on CD we can see 8X → represents speed. = 150 mbps
12X
32X
64X
Data writes by this much speed.

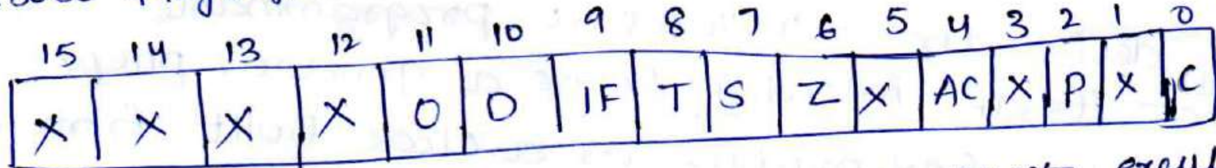
9 → Flash memory (ex: pendeive) 1 to 10,000 times.

→ Y represents : 3 No chip memory
5 onchip memory and it also represents no. of intercepts(i.
5 which are maskable interepts (MI):
INT0, INT1 } external interupts
T0, T1 } Timer interupts
RI, TI → SI Serial interupts
RST → NMI

→ diff. b/w MI & NMI

8086 flagregister.

16bit



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	O	D	IF	T	S	Z	X	AC	X	P	X	C

IF → interupt flag enable then intercept request exeutes → MI / → NMI
↳ ,, ,, disable then NMI exeutes.

But in microcontroller, intercept enable register exeutes NMI.

→ Z represents : 1 → 2 timers T0 & T1

2 → 3 times T0, T1 & T2

on chip internal memory	
ROM	RAM
4KB	128B
8KB	256KB

External memory

ROM	RAM
?64K	?64K

$2^{\text{Add.lines}} \times$ Data lines

$2^{16} \times 8 = 2^{10} \times 2^{6} \times$

1K × 64 Bytes

= 64 KBytes.

→ Both input output device – Touch Screen
↳ Idea given by apple iphone
Tech. implemented by Cypress and.

3. I/O units such as touch screen, modem, fax cum modem, etc.
4. Input units such as keyboard, mice, digitizer, scanner etc.
5. output units such as ~~key~~ LCD screen, video monitor, printer, et
6. Networking units such as an Ethernet card, front end Processor-based server, bus drivers etc
7. An operating System that has general purpose user and application software in the secondary memory.

<u>An Embedded System is a System that has three main Componen embedded into it</u> :-

1. It embeds hardware similar to a computer. the following figure shows the units in the hardware of an Embedded Sys

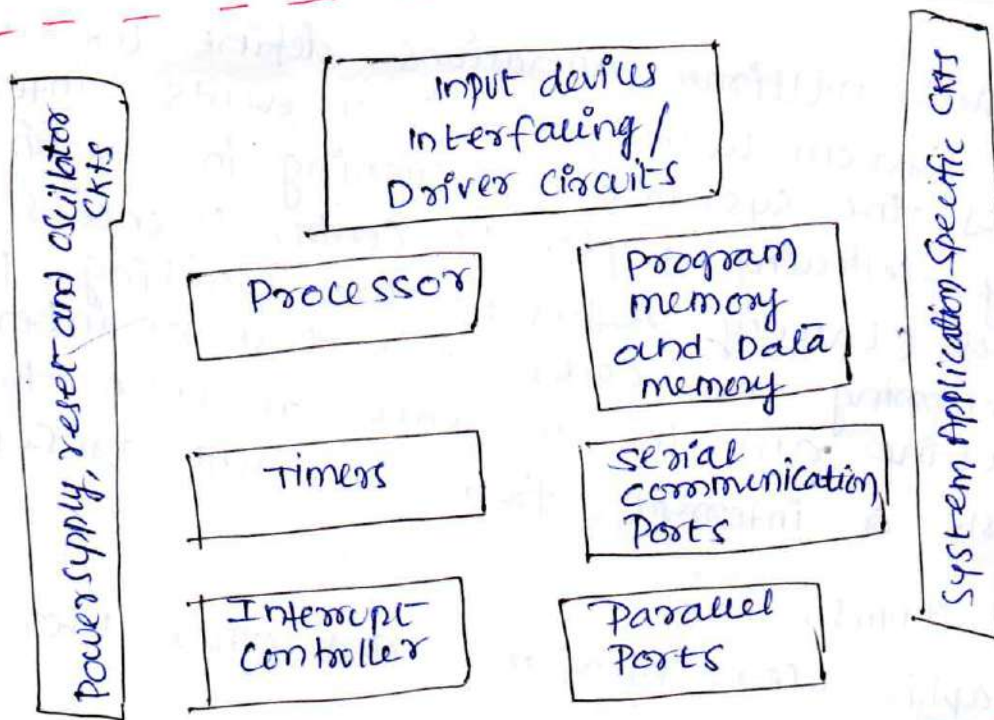


fig : The components of Embedded System Hardware

As its software usually embeds in the ROM or flash memory, it usually do not need a secondary harddisk and CD memory as in a computer.

2. It embeds main application s/w. The application s/w may concurrently perform a series of tasks or processes or threads.

3. It embeds a Real-time operating system (RTOS) that supervises the application software running on hardware and organizes access to a resource according to the priorities of tasks in the system. It provides a mechanism to let the processor run a process as scheduled and context-switch between the various processes. It sets rules during the execution of the application software.

Characteristics of ES

1. Realtime and multirate operations define the way in which the system works, reacts to events, interrupts and schedules the system's functioning in real-time. It does so by following a plan to control latencies and to meet deadlines. (Latency refers to the waiting period between running the codes of a task or interrupt service routine and the instance at which the need for the task or interrupt from an event arises).

2. Complex algorithms.
3. Complex graphic user interfaces and other user interfaces
4. Dedicated functions.

Constraints of ES :

An embedded system is designed keeping in view three constraints :

1. Available system memory
2. Available processor speed
3. the need to limit power dissipation when running the system continuously in cycles of wait for events, run, stop, wake-up and sleep.

The system design or an ES has constraints with regard to performance, power, size and design and manufacturing costs.

In contrast to the general computing systems, ES are highly domain and highly domain application specific in nature, meaning; they are specifically designed for certain set of applications in certain domains like consumer electronics, telecom, automotive, industrial control measurement systems

Embedded system is a combination of specilized hardware and firm ware (software), which is tailored to meet that requirements of the application under consideration.

An embedded system contains a processing unit which can be a microprocessor or a micro controller or a system on chip (soc) or an application specific integrated circuit (ASIC)/ Applications specific standard product (ASSP or a programmable logic device (PLD) like FPGA (or) CPLD, an I/O subsystem which (specificates) the facilitates interfacing of sensors and actuators which acts as the

the embedded system is interacting, on-board and external communications interfaces for communication between the various on-board sub-systems and chips which builds the embedded systems and external systems to which the embedded system interacts, and other supervisory systems and support units like watch-dog timers, reset circuits, Brown-out protection circuits, regulating power supply unit, clock generation circuits etc., which monitor the functioning of the embedded systems.

Our day to day life is becoming more and more depedent on embedded systems and digital techniques Embedded technologies are bonding into our dialy activities even without our knowledge

Ex: Refrigerator, washing machine, microwave oven, Air conditioner, Television, Dvd players, Music system that we use in our home are build around an embedded systems.

In your vehicle it self the presence of speciliz embedded system vary from intelligent head la controllers, engine controllers and ignition contool systems to complex air bag contool systems to protect you in case of a severe accident.

The intelligent embedded systems giving us so much comfort and security

What is an Embedded System ?

An embedded system is an electronic / electromechanical system designed to perform a specific function and is a combination of both hardware and firmware (software)

Every Embedded system is unique, and the hardware as well as the firmware is highly specialized to the application domain. Embedded system are becoming an inevitable part of any product or equipment in all fields including house-hold appliances, telecommunications, medical equipment, industrial control, consumer products etc

Examples of Embedded Systems

Embedded systems have different applications. A few selective applications areas of embedded systems are telecommunication, smart cards, missiles and satellites, computer networking, digital consumer electronics, and automotives.

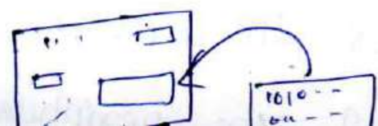
Small scale embedded system applications: few examples of small scale embedded system applications.

1. point of sales terminals: automatic chocolate vending machine.
2. stepper motor controllers for a robotic system
3. washing or cooking systems
4. multitasking toys
5. Microcontroller-based single or multidisplay digital panel meters for voltage, current, resistance and frequency.
6. keyboard controller
7. SD, MMI and n/w access cards
8. CD drive or hard disk drive controller
9. The peripheral controllers of a computer

11. Remote (Controller) of TV
12. Telephone with memory, display and other sophisticated features
13. Motor control Systems
14. Electronic data acquisition and supervisory control system
15. Electronic instruments, such as industrial process controller
16. Digital storage system
17. spectrum analyzer
18. Electronic smart weight display system and an industrial moisture recorder cum controller.
19. Biomedical systems such as an ECG LCD display cum recorder, a blood cell recorder cum analyzer, and a patient monitor syste
20. Computer networking systems
21. For internet appliances,
22. Entertainment Systems such as video game and a music Syste
23. Banking systems, for example bank ATM and credit card transactions.
24. signal tracking systems
25. communication systems such as a mobile communication SIM card, a numeric pager, a cellular phone
26. Image filtering, image processing, pattern recognizer, video processing
27. video games.
28. A system that connects a pocket PC to the automobile driver mobile phone and a wireless receiver.
28 mobile smart-phones and computing Systems
29 mobile computer
30. Embedded systems for wireless LAN

Simple def. of ES:

An ES consists of both H/w and S/w, so when chips are build into a system and a S/w loaded on to it for a particular functionality it becomes an ES.



What is an Embedded System ?

An embedded system is an electronic / electromechanical system designed to perform a specific function and is a combination of both hardware and firmware (software)

Every Embedded system is unique, and the hardware as well as the firmware is highly specialized to the application domain. Embedded system are becoming an inevitable part of any product or equipment in all fields including household appliances, telecommunications, medical equipment, industrial control, consumer products etc

Examples of Embedded Systems

Embedded systems have different applications. A few selective applications areas of embedded systems are telecommunications, smart cards, missiles and satellites, computer networking, digital consumer electronics, and automotives.

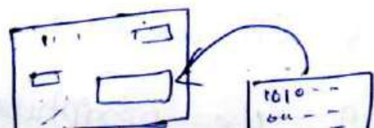
Few examples of small scale embedded system applications.

1. point of sales terminals: automatic chocolate vending machine.
2. stepper motor controllers for a robotic system
3. washing or cooking systems
4. multitasking toys
5. Microcontroller-based single or multidisplay digital panel meter for voltage, current, resistance and frequency.
6. keyboard controller
7. SD, MMI and n/w access cards
8. CD drive or hard disk drive controller
9. The peripheral controllers of a computer

11. Remote (Controller) of TV
12. Telephone with memory, display and other sophisticated features
13. Motor control Systems
14. Electronic data acquisition and Supervisory control System
15. Electronic instruments, such as industrial process Controller
16. Digital Storage System
17. spectrum analyzer
18. Electronic Smart weight display System and an industrial moisture recorder cum Controller.
19. Biomedical Systems such as an ECG LCD display cum recorder, a blood cell recorder cum analyzer, and a patient monitor system
20. Computer networking systems
21. For internet appliances,
22. Entertainment Systems such as video game and a music System
23. Banking systems, for example bank ATM and credit card transactions.
24. Signal tracking Systems
25. communication Systems such as a mobile Communication SIM card, a numeric pager, a cellular phone
26. Image filtering, image processing, pattern recognizer, video processing
27. video games.
28. A System that Connects a pocket PC to the automobile driver mobile phone and a wireless receiver.
28 mobile Smart-phones and computing Systems
29 mobile computer
30. Embedded systems for wireless LAN

Simple def. of ES:

An ES consists of both H/w and S/w, so when chips are build into a System and a S/w loaded on to it for a particular functionality it becomes an ES.

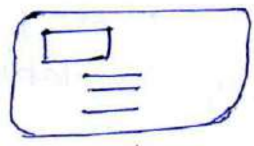


Applications of the Embedded Systems in various areas

1. Telecom

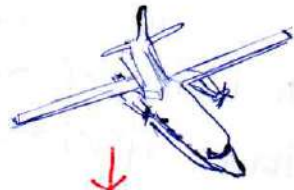
1. mobile computing
2. Mobile access

2. Smart Cards



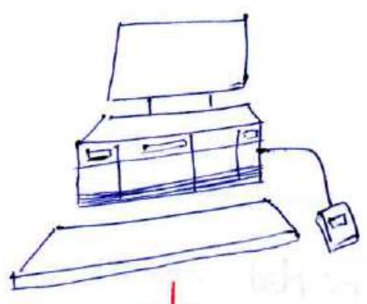
1. Banking
2. Security

3. missiles and Satellites



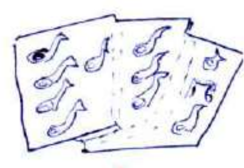
1. Defence
2. Aerospace
3. communication

4. Computer Networking Systems and peripherals



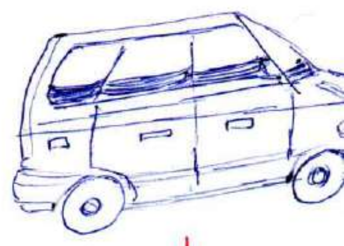
1. Networking Systems
2. image processing
3. printers
4. Networks cards
5. monitors and Displays

5. Digital Consumer Electronics



1. DVD's
2. set top boxes
3. High definition TVs
4. Digital Cameras

6. Automotive



1. Motor control Sy
2. Cruise control
3. Engine/Body Sa
4. Robotics in Ass
5. Car Entertainme
6. Car multimedi

Embedded Systems Vs General Computing Systems.

The General computing requirements are not sufficient for the embedded computing requirements.

The embedded computing requirements demand something speci[al] in terms of response to stimuli, meeting the computational dead li[nes], Power efficiency, limited memory availability, etc.

	General purpose Computer	Vs	Embedded System
Purpose	Multipurpose		Single functioned
Constrained	Low or no resource constrained		size, power, cost, memory, Real time
Performance	Faster and better than E S		Fixed runtime requirements, are predefined
User interface	Keyboard, display, mouse, Touch Screen		Integrated into the real world. Button, Sensors.

General purpose computing System	Embedded System
1. A System which is a combination of a generic hardware and a general Purpose operating system for executing a variety of applications.	1. A System which is a combination of special purp hardware and embedded OS for executing a specific se of applications.
2. contains a general purpose OS (GPOS)	2. May or maynot contain operating System for func
3. Applications are alterable (programmable) by the user.	3. The firmware of the ES is pre-programmed and i is non-alterable by the end-user.
4. Performance is the key deciding factor in the selection of the System. Always faster is better.	4. Application specific requirem (like performance, power req memory usage etc). are th key deciding factors.
5. Less/not at all tailored towards reduced operating Power requirements, options for different levels of Power management.	5. Highly tailored to take advantage of the power saving modes supported the hardware and the ~~followi~~ operating system.
6. Response requirements are not time-critical	6. For certain category of embedded systems like mis critical systems, the response time requirement is highly cr
7. Need not be deterministic in execution behaviour.	7. Execution behaviour is determi nistic for certain types of ES like Hard Real time Sy

History of Embedded Systems :

We already know that how the embedded Systems differ from the general purpose compting system. we know that embedded Systems are designed using special purpose hardware and embedded OS to perform a specific tasks.

It's very interesting to know about the specific tasks done through the ES by exploring its historical events.

Have you ever wonderd how the remarkable development of the embedded history has made our life so simpler, it is all due to the invention and development of Semiconductor technology. bcz without Semiconductors there is no micro Controller & microprocessor. and without these there is no Embedded systems. Hence it is importa to know the importance of various reputed industries.

In 1944 the world's first programmable electronic digital computer was developed It was named as the colossus mark I and mark II computers.

The computer was specially designed and used by the britishes code breakers to Read the encrypted German messages during the second world war.

In 1960's a most crucials NASA'S Lunar machine Program called Apollo Guidance Computer was started. The project was developed by charles stark Draper at the MIT instrumentation Laboratory This is the Apollo Guidance computer and it was the first Computer which is used with 1 MHz clock and 4 Kb-words ROM and 256-words of RAM. This is the user interface which has a 7 segment display unit with a keyboard it acts as a communication bridge b/w the Astronat and the computer. It provided the austronat's to input their

MIT Prof Mr. David Mindell, the author of digital Apollo Says, "The Apollo Guidance computers were eally examples of what we would today call embedded computers - which no appears in everything from iphones to automobiles".

Thus apollo, Guidance computer becames the First recognised modern embedded System to this world.

Later in 1961 another one crucial millitary project wa initiated. it was the minuteman - missile I. for this the Autonetics D-17 guidance computer was developed. It was built from the discrete transistor logic and a hard disk ~~was~~ its main memory. It was also the first embedded system to be produced in Large quantities.

In 1965 the first computer embedded in a commerci instrument was launched. it was a 12 bit PDP-8 minicomput

Conceqenty in 1966 the minuteman missile - II went on production. This came up with several range of technic advancements in the autonetics. It replaces the old Auton guidance computer with the new micro electronic computer which was liter in weight. compared with it's older version.

The second version was developed with a high volume of integrated circuits. This process reduced the cost of integrated circuits, from 1000 $ to 3$ each. due to its affordability integrated circuits came into use ~~exists~~. on various commertial products widly.

In 1968 Intel was founded by Robert Noyce and Gordon Moore. The begining of Intel let to the Evolution of microcontrollers and microprocessors. which is the heart of many modren electronics.

In 1969 microprocessor based fuel injection System was introduced in the volkswagen cars and was break through in the field of Embedded systems

In 1971 intel 4004 the First 4-bit microprocessor was developed. by intel scientists. in the same year TMS10 one of the most successful 4-bit microcontroller was developed by another Semiconductor joint the texas instruments.

~~The texas instruments~~

In 1974 Intel 8080 was developed and it was the first microprocessor to be used in a personal computer. and in 1980's microcontroller became the demanded technology the microprocessors are optimised for speed and memory size where as the microcontrollers are optimised for minimised power consumption and physical size.

In 1983 HP 150 personal computer was introduced by Dave packard & Bill Hewlett. It was claimed to be the first computer to be featured touchscreen sensitivity.

In 1987 Vxworks a Real time operating System (RTOS) was introduced by WIND RIVER systems and this RTOS was used for NASA's mars arm finder machine. in later year. In 1989 one of the embedded system Dot matrix printer was Restricted to 8-bit due to power expenses and tight timing and electrical Constraints.

In 1992 the ES went wireless. and newly 10 million mobile phones were manufactured using ES for Controlling their functioning.

In 1996 microsoft entered into the Embedded market and designed its handheld personal computer using windows CE1.0 operating system.

following with windows OS
Relating to OS, in 1999 the Linux foundation started to develop its roots in the Embedded System.

at the begining of 2000 a second parallel revolution occured in the Embedded Systems. one of the highlights of the launch of the real time version of Linux by "time sys" corporation.

In 2005 the multi dollor companies Intel, IBM and AMD released their first multicore processors and by the year 2007 the reputed Americal multi-national company Apple Inc. released its first iphone mobile. following the apple iphone in 2008. the first Android phone was launched and the Android code became an open source OS.

finally when it comes to our conteporary Senario over 95% of electronic chips producing in a year are Embedded Systems.

These are hidden in familiar objects which you see around and you use them everyday.

position of ES in the contempory Senario

Another one technological breakthrough in current Embedded is the wifi connected smart appliances the range includes, microwave oven, Refrigerator, cofee brewer, washingmachine etc. and all these appliances can be monitored and controlled via your smart phone app. But what will be the picture of embedded systems in the future?

One of the most awaited project in this area is self driving cars, its a fully automated driverless cars. market players believing this technology came in exists soon by 2020 and its going to change everythi
lets hope for the best.

Classification of Embedded Systems :-

Embedded Systems are classified based on different criteria. Some of the criteria used in the classification of ES are

1. Based on generation
2. complexity and performance requirements
3. Based on deterministic behaviour
4. Based on triggering.

I Classification based on Generation :-

This classification is based on the order in which the embedded processing systems evolved from the first version to till now. As per this criteria ES can be classified into:

(1) First Generation :

The early ES were built around 8 bit microprocessors like 8085 and z80, and 4 bit microcontrollers. simple in Hardware ckts with firmware developed in Assembly Code. Digital telephone keypads, stepper motor control units etc. are examples of this.

(2) Second Generation :

These are embedded Systems built around 16 bit microprocessors and 8 or 16 bit microcontrollers, following the first generation embedded systems. The instruction set for the second generation processors/controllers were much more complex and powerful than the first generation processors/controllers. Some of the second generation ES contained Embedded operating systems for their operation. Data Acquisition Systems, SCADA Systems, etc are examples of second Generation ES.

3. Third Generation :

with advances in processor technology, embedded system developers started making use of powerful 32bit processors and 16 bit microcontrollers for their design. A new concept of application and domain specific processors/controllers like Digital signal processors (DSP) and Application specific integrated circuits (ASICS) came into the picture.

The instruction set of processors became more complex and powerful and the concept of instruction pipelining also evolved. processors like Intel, Pentium, motorola 68K, etc gained attention in high performance embedded requirements. Embedded systems spread its ground to areas like robotics, media, industrial proces control, networking etc.

4. Fourth Generation :

The advent of system on chips (SOC), reconfigural processors and multicore processors are bringing high performance, tight integration and miniaturisation into the embedded device market.

The SOC technique implements a total system on a chip by integrating different functionalities with a processor core on an integrated circuit.

The 4th generation ES are making use of high performance real time embedded operating systems for their functioning.

Smart phone devices, mobile internet devices (MIDs) etc are examples of 4th generation ES.

③ Third Generation :

with advances in processor technology, embedded system developers started making use of powerful 32bit processors and 16 bit microcontrollers for their design. A new concept of application and domain specific processors/controllers like Digital signal processors (DSP) and Application specific integrated circuits (ASICS) came into the picture.

The instruction set of processors became more complex and powerful and the concept of instruction pipelining also evolved. processors like Intel, pentium, motorola 68K, etc gained attention in high performance embedded requirements. Embedded systems spread its ground to areas like robotics, media, industrial process control, networking etc.

④. Fourth Generation :

The advent of system on chips (SOC), reconfigurable processors and multicore processors are bringing high performance, tight integration and miniaturisation into the embedded device market.

The SOC technique implements a total system on a chip by integrating different functionalities with a processor core on an integrated circuit.

The 4th generation ES are making use of high performance real time embedded operating systems for their functioning.

Smart phone devices, mobile internet devices (MIDs) etc. are examples of 4th generation ES.

II classification based on complexity & performance :-

According to the classification based on complexity and system performance, embedded systems can be grouped as,

1. Small-scale Embedded Systems :-

Embedded Systems which are simple in application needs and where the performance requirements are not time critical there we use this small-scale ES.

An electronic toy is a typical example of a small-~~Signal~~ scale Embedded systems.

Small-scale embedded systems are usually built around low performance and low cost 8 or 16 bit microprocessors/microcont

A small-scale embedded system may or may not contain an operating system for its functioning.

2. Medium-scale Embedded Systems :-

Embedded systems which are slightly complex in H/w and firmware (s/w) requirements fall under this category.

These systems are usually built around medium performance, low cost 16 or 32 bit MP/MC or digital signal processors. They usually contain an embedded operating system (either general purpose or real time OS) for functioning.

3. Large-scale ES / complex systems :-

Embedded systems which involve highly complex hardware and firmware requirements fall under this category.

They are employed in mission critical applications demanding high performance. Such systems are commonly built around high performance 32 or 64 bit RISC processors/controllers. or Reconfigurable System on Chip (RSoC) or multi core processors an PLD. They may contain multi processors/controllers and co-units/hardware accelerations for offloading the processi requirements from the main processor of the system.

Complex ES usually contain a high performance Real time operating system for task scheduling, prioritization and management.

Major application areas of ES :-

Embedded Systems plays an important role in our day-to-day life starting from home to the computer industry. Embedded technology has acquired a new dimension from its first generation model, the Apollo guidance compu to the latest radio navigation system combined with in-ca entertainment technology and the MP based smart run Shoes launched by Adidas in April 2005.

The application areas and the products in the embedded system domain are countless. few of them are,

1. Consumer electronics : camcorders, cameras, etc.
2. Household appliances : Television, DVD players, washing machi fridge, microwave oven, etc.
3. Home automation and security systems : Air conditioners, sprinklers, Closed ckt television cameras, fire ala etc.
4. Automotive industry : Anti-lock breaking systems (ABS), engine control, ignition systems, automatic navigation systems, etc.

5. Telecom :- Cellular Telephones, Telephone switches, handsets, multimedia applications etc.

6. computer peripherals :- printers, Scanners, fax machines

7. computer networking systems :- N/w routers, switches, hubs, firewalls etc.

8. Healthcare :- Different Kinds of scanners, EEG, ECG machines etc.

9. Measurement & instrumentation :- Digital multi meters, digital CROs, logic analyzers, PLC Systems etc.

10. Banking & Retail :- Automatic teller machines (ATM) and currency counters, Point of Sales (POS)

11. Card Readers :- Barcode, Smart card readers, hand held devices, etc.

Purpose of Embedded Systems

As we already discussed that, embedded systems are used in various domains like consumer electronics, home automation, telecommunications, automotive industry, healthcare, control and instrumentation, retail and banking applications, etc.

Within the domain itself, according to the application usage context, they may have different functionalities.

Each embedded system is designed to serve the purpose of any one or a combination of the following tasks

1. Data collection / ~~and~~ Storage / Representation
2. Data Communication
3. Data Processing

5. control
6. Application specific user interface.

① Data collection / Storage / Representation :-

Data means all kinds of information, for example text, voice, image, video, electrical signals and any other measurable quantities. Data can be either analog or digital.

Data collection performs acquisition of data from the external world.

Embedded systems with analog data capturing techniq collect data directly in the form of analog signals. whereas embedded systems with digital data collection mecha converts the analog signal to corresponding digital signa using analog to digital (A/D) converters. and then collec the binary equivalent of analog signal.

1. If the data is digital, it can be directly captured without any additional interface by digital embedded systems.

Now the collected data is directly stored in the system or may be transmitted to some other systems. or it may be processed by the system or it may be delete instantly after giving a meaningful representation.

All these actions are purely dependent on the purpose for which the embedded system is designed.

2. Embedded systems designed for pure measurement applications without storage, used in control and instrumenta domain, collects data and gives a meaningful representation of the collected data by means of graphical representat and deletes the collected data when new data arrives at

Ex: 1. Analog to digital CROs. &
2. any ~~medical~~ measuring equipment used in the medical domain for monitoring without any storage functionality.

3. Some embedded systems store the collected data for processing and analysis. Such systems incorporate a buit-in/plug-in storage memory for storing the captured data. Some of them give the user a meaningful representation of the collected data by visual or audible means using display units, (LCD, LED), buzzers, alarms etc.

Ex: measuring instruments, with storage memory & monitoring instruments with storage memory used in medical applications.

4 A digital Camera is a typical example of an ES with data collection/ Storage/ representation of data.

(2) Data communication :-

Embedded data communication systems are deployed in applications from complex satellite communication systems to simple home networking systems.

The data collected by embedded terminal may require transferring of the same to some other system located remotely. The transmission is achieved either by a wire-line or by a wireless medium.

In olden days embedded systems wire-line medium was used. As technology changing wireless medium is becoming the standard for data communication in ES.

Modern industry trends are settling towards digital communication.

Certain ES act as a dedicated transmission unit between the sending and receiving terminals, offering

sophisticated functionalities like data packetizing, encrypting and decrypting. N/w hubs, routers, switches etc are typical examples of dedicated data transmission embedded systems. They act as mediators in data communication and provide various features like data security, monitoring etc.

③ Data Processing :-

Data collected by embedded systems may be used for different kinds of data processing. Embedded systems with signal processing functionalities are employed in applications demanding signal processing like speech coding, synthesis, transmission applications etc.

Ex : A digital hearing aid is a typical example of an embedded system employing data processing. it improves the hearing capacity of hearing impaired persons.

④ Monitoring :-

Embedded systems falling under this category are specifically designed for monitoring purpose. Almost all embedded products coming under the medical domain are with monitoring functions only.

Ex : Electro Cardiogram (ECG) machine for monitoring the heartbeat of a patient. The machine is intended to do the monitoring of the heartbeat. The sensors used in ECG are the different electrodes

⑤ Control :-

Embedded systems with control functionalities impose control over some variables according to the changes in input variables.

A system with control functionality contains both sensors and actuators.

Sensors are connected to the input port for capturing the changes in environmental variable or measuring variable.

The actuators connected to the output port are controlled according to the changes in input variable to put an impact on the controlling variable to bring the controlled variable to the specified range.

Ex : Air conditioner system used in our home to contro[l] the room temperature to a specified limit is a typical examp[le] in Embedded system for control purpose.

An air conditioner contains a room temparature sensing element (sensor) which may be a thermistor and a handhe[ld] unit for setting up the desired temparature.

The air compressor unit acts as the actuator. The compressor is controlled according to the current room temperature and the ~~controlled variable is~~ desired temp set by the end user.

Here the input variable is the current room temparature and the controlled variable is also the room temparature.

The controlling variable is cool air flow by th[e] compressor unit. if the controlled variable and input variable are not at the same value, the controlling variable tries to equalise them through taking actions on the cool air flow.

⑥ Application Specific User Interface :-

These are embedded systems with application-specific user interfaces like buttons, switches, keypad, lights, bells, display units etc.

Ex : Mobile phone.

In mobile phone the user interface is provided through the keypad, graphic LCD module, system

Characteristics & Quality Attributes of Embedded Systems.

There must be a set of characteristics for every system. The system may be a embedded or a non-embedded. The non-functional aspects that need to be addressed in embedded system design are commonly referred as Quality Attributes.

Characteristics of an Embedded system :-

Embedded systems possess certain specific characteristics and these characteristics are unique to each embedded systems. Some important characteristics of embedded system are :

1. Application and Domain specific
2. Reactive and Realtime
3. Operates in Harsh environments
4. Distributed
5. Small size and weight
6. Power concerns.

(1) For any embedded System, It has certain functions to Perform and they are developed in such a manner to do the intended functions only. "they can't be used for any other function or purpose".
It is the major criterion which distinguishes an embedded System from a general purpose System.

Ex: you can't replace the embedded control unit of microwave. oven and airconditioner's embedded system control unit.

② Reactive and RealTime :-

Embedded Systems produce changes in output in response to the changes in the input. So they are generally referred as Reactive Systems.

Real time System operation means the timing behavio[r] of the System should be deterministic; means the System should respond to requests or tasks in a known amount of time. A Real time System should not miss any dead[lines] for tasks or operations. It is not necessary that all embedded Systems should be Real time in operations.

Ex Embedded applications or Systems which are mission critical, like flight control Systems, Antilock Brake System (ABS), etc. are examples of Real time Systems.

③ Operates in Harsh Environment :-

It is not necessary that all embedded Systems should be moved into Controlled environments. The environment in which the embedded system deployed may be a dusty one or a high temparature zone or an area subject to vibrations and shock. Systems placed in such areas should be capab[le] to withstand all these adverse operating conditions. The design should take care of the operating conditions of the area where the system is going to impleme[nt].

Ex: If the system needs to be deployed in a high temp. zone, then all the components used in the system should be of high temp. grade.

and also proper shock absorption techniques should be provided to systems which are going to be commissione[d] in places subject to high shock.

Power supply fluctuations, Corrosion and component aging etc. are the other factors that need to be taken into consideration for embedded Systems to work in

④ Distributed :-

Distributed means that embedded System is a part of larger systems. many number of Such distributed embedded Systems form a single large embedded Control unit.

Ex: 1. The vending machine (An automatic vending machine)

↓

It contains Card reader, Vending unit etc. each of these are independent embedded units but they work together to perform the overall vending function.

2. Automatic Teller machine (ATM)

↓

Contains. a Card reader, responsible for reading and validating the users ATM Card,

transaction unit - for performing transactions

a currency counter - for dispacting / vending currency to the authorised person and a printer unit for printing the transaction details.

These are all independent ES. But they work together to achieve a common goal.

⑤ Small Size and weight :-

Product aesthetics is an important factor in choosing a product. For example, when you plan to buy a new mobile phone, you may make a comparative study on the Pros and Cons of the products available in the market. Definitely the product aesthetics (size, weight, shape, style, will be one of the deciding factors to choose a product.

People believe in phrase Small is beautiful. moreover it is Convenient to handle a compact device than a bulky product. In embedded domain also compact ... most of the applications

⑥ power concerns :-

Power management is another important factor that needs to be considered in designing embedded system. Embedded Systems should be designed in such a way as to minimise the heat dissipation by the system.

The production of high amount of heat demands cooling requirements like cooling fans which in turn occupies additional space and make the system bulky.

Nowadays ultra low power components are available in the market. Also power management is a critical constraint in battery operated application. The more the power consumption the less is the battery life.

Quality Attributes of Embedded Systems:-

Quality attributes are the non-functional requirements that need to be documented properly in any system design. The various quality attributes that needs to be addressed in any embedded system development are broadly classified into two, namely

1. operational Quality attributes
2. Non-operational " "

① operational Quality attributes :-

The operational quality attributes represent the relevant quality attributes related to the embedded system when it is in the operational mode or online mode.

The important quality attributes are :

1. Response
2. Throughput
3. Reliability
4. Maintainability
5. Security
6. Safety.

① Response :-

Response is a measure of quickness of the system. It gives you an idea about how fast your system is tracking the changes in input variables.

Most of the embedded systems demand fast response which should be almost Real time.

Ex: An embedded system deployed in flight control application should respond in a Real time manner. Any response delay in the System will create potential damages to the safety of the flight as well as the passengers.

It is not necessary that all embedded systems should be Realtime in response.

Ex: Electronic toy

The response time requirement for an electronic toy is not at all time-critical. There is no specific deadline that this system should respond within this particular time.

② Throughput :-

Throughput deals with the efficiency of a system. In general it can be defined as the rate of production or operation of a defined process over a stated period of time.

rate can be expressed in terms of units of products, batches produced, or any other meaningful measurements. In case of Card Reader throughput means how many transactions the reader can perform in a minute or in an hour or in a day.

Throughput is generally measured in terms of Benchmark. A benchmark is a reference point by which something can be measured.

③ Reliability :-

Reliability means it is a measure of how much % you can rely upon the proper functioning of the system or what is the % susceptibility of the system to failures.

MTBF and MTTR are the terms used in defining system reliability. MTBF gives the frequency of failures in hours/weeks/months. MTTR specifies how long the system is allowed to be out of order following a failure.

④ Maintainability :-

Maintainability deals with support and maintenance to the end user or client in case of technical issues and product failures or on the basis of a routine system checkup.

Reliability, maintainability are the two complementary issues. A more reliable system means that system is less corrective maintainability requirements.

Reliability of the system ↑ ⇒ chance of failure and non-functioning redues. then the need of maintainability also reduces. Maintainability broadly classified into two types,

1. Scheduled or periodic maintenance (Preventive maintenance)
2. Maintenance to unexpected failures (corrective ")

Some embedded products may use consumable components or may contain components ~~which are subject~~ they should be replaced on a periodic basis.

Ex: Printer.

1. An inkjet printer uses ink cartridges, which are consumable components and as per the printer manufacturer the end user should replace the cartridge after n number of printouts to get quality prints. → This is an example for Scheduled or periodic maintenance.

If the paper feeding part of the printer fails, the printer fails to print and it requires immediate repairs to rectify this problem. → It is maintenance to unexpected failure.

In both of the maintenances, the printer needs to be brought offline and during this time it will not available to user. Hence it represents maintainability is simply an indication of the availabity of the product for use.

In any ES the ideal value for availability is expressed as,

$$A_i = \frac{MTBF}{MTBF + MTTR}$$

A_i – Availability in Ideal Condition

MTBF – Meantime B/w failure

MTTR – " to Repair.

⑤ Security :-

Confidentiality, Integrity and Availability are the three major measures of information security.

Confidentiality deals with the protection of data and application from unauthorised disclosure.

Integrity deals with the protection of data and application from unauthorised modification.

Availability deals with the protection of data and application from unauthorized users.
Ex: Personal Digital Assistant.

⑥ Safety :-

Safety and security are two confusing terms. Sometimes you may feel both of them as a single attribute.

Safety deals with the possible damages that can happen to the operators, public and the environment due to the breakdown of an embedded system or due to the emission of radioactive or hazardous materials from the embedded products.

The breakdown of an embedded system may occur due to a hardware failure or a firmware failure.

Safety analysis is a must in product engineering to evaluate the anticipated damages and determine the best course of action to bring down the consequences of the damages to an acceptable level.

Some of the safety threats are sudden like product breakdown and some of them are gradual like hazardous emissions from the product.

(2) Non-operational quality Attributes.

The quality attributes that needs to be addressed for the product not on the basis of operational aspects are called non-operational quality attributes. They categoris as,

1. Testability & Debug-ability
2. Evolvability
3. Portability
4. Time to prototype and market
5. Per unit and total cost.

(1) Testability and debug-ability :-

Testability deals with how easily one can test his/her design application and by which means he/she can test it. For an embedded product, testability is applicable to both the embedded hardware and firmware.

Hardware testing ensures that the peripherals and the total hardware functions in the desired manner, whereas firmware testing ensures that the firmware is functioni in the expected way.

Debug ability has two aspects in the embedded system development. They are hardware level debugging and firmware level debugging.

1. Hardware debugging is used for figuring out the issues created by hardware problems.
2. firmware debugging is employed to figure out the probab errors that appear as a result of flaws in the firmwar

(2) Evolvability :-

Evolvabilty is referred as the non-heritable variatio For an Embedded System, the quality attribute Evolvability refers to the ease with which the embedded product can be modified to take advantage of new firmware or H/W

(2) Non-operational quality Attributes.

The quality attributes that needs to be addressed for the product not on the basis of operational aspects are called non-operational quality attributes. They categorise as,

1. Testability & Debug-ability
2. Evolvability
3. Portability
4. Time to prototype and market
5. Per unit and total cost.

(1) Testability and debug-ability :-

Testability deals with how easily one can test his/her design application and by which means he/she can test it. For an embedded product, testability is applicable to both the embedded hardware and firmware.

Hardware testing ensures that the peripherals and the total hardware functions in the desired manner, whereas firmware testing ensures that the firmware is functioning in the expected way.

Debug ability has two aspects in the embedded system development. They are hardware level debugging and firmware level debugging.

1. Hardware debugging is used for figuring out the issues created by hardware problems.
2. firmware debugging is employed to figure out the probable errors that appear as a result of flaws in the firmware.

(2) Evolvability :-

Evolvability is referred as the non-heritable variation. For an Embedded System, the quality attribute Evolvability refers to the ease with which the embedded product can be modified to take advantage of new firmware or H/W.

③ Portability :-

An embedded product is said to be portable if the Product is capable of functioning as such in various environments target processors/controllers and embedded operating systems.

A standard embedded product should always be flexible and portable. In embedded products, the term 'porting' represents the migration of the embedded firmware written for one target processor to a different target processor.

④ Time to prototype and market :-

Time-to-market is the time elapsed between the conceptualisation of a product and the time at which the product is ready for selling (for commercial product, or use (for non-commercial products).

The commercial embedded product market is highly competitive and time to market the product is a critical factor in the success of a commercial embedded product. There may be multiple players in the embedded industry who develop products of the same category. If you come up with a new design and if it takes long time to develop and market it, the competitor product may take advantage of it with their product.

Product prototyping helps a lot in reducing time to market. Whenever you have a product idea, you may not be certain about the feasibility of the idea. Prototyping is an informal kind of rapid product development in which the important features of the product under consideration are developed.

If the prototype is developed faster, the actual estimated development time can be brought down significantly. In order to shorten the time to prototype, make use of all possible options like the use of off-the-shelf components, re-usable assets etc.

5. Per unit cost and Revenue :-

Cost is a factor which is closely monitored by both end user (those who by the product) and produce manufacturer (those who build the product). Cost is a highly sensitive factor for commercial products.

Proper market study and cost benefit analysis should be carried out before taking a decision on the Per-unit cost of the embedded product. From a designer/Product development company, the ultimate aim of a product is to generate marginal profit. So the budget and total system cost should be properly balanced to provide a marginal profit.

Every embedded product has a product life cycle which starts with the design and development phase. The product idea generation, prototyping, Roadmap definition actual product design and development are the activities carried out during this phase.

During the design & development phase there is only investment and no returns.

Once the product is ready to sell, it is introduced to the market → product introduction stage. During initial period the sales and revenue will be low. There won't be much competition and the product sales and revenue increases with time.

~~Growth phase → The growth and Sales will be Steady and the revenue reaches~~.

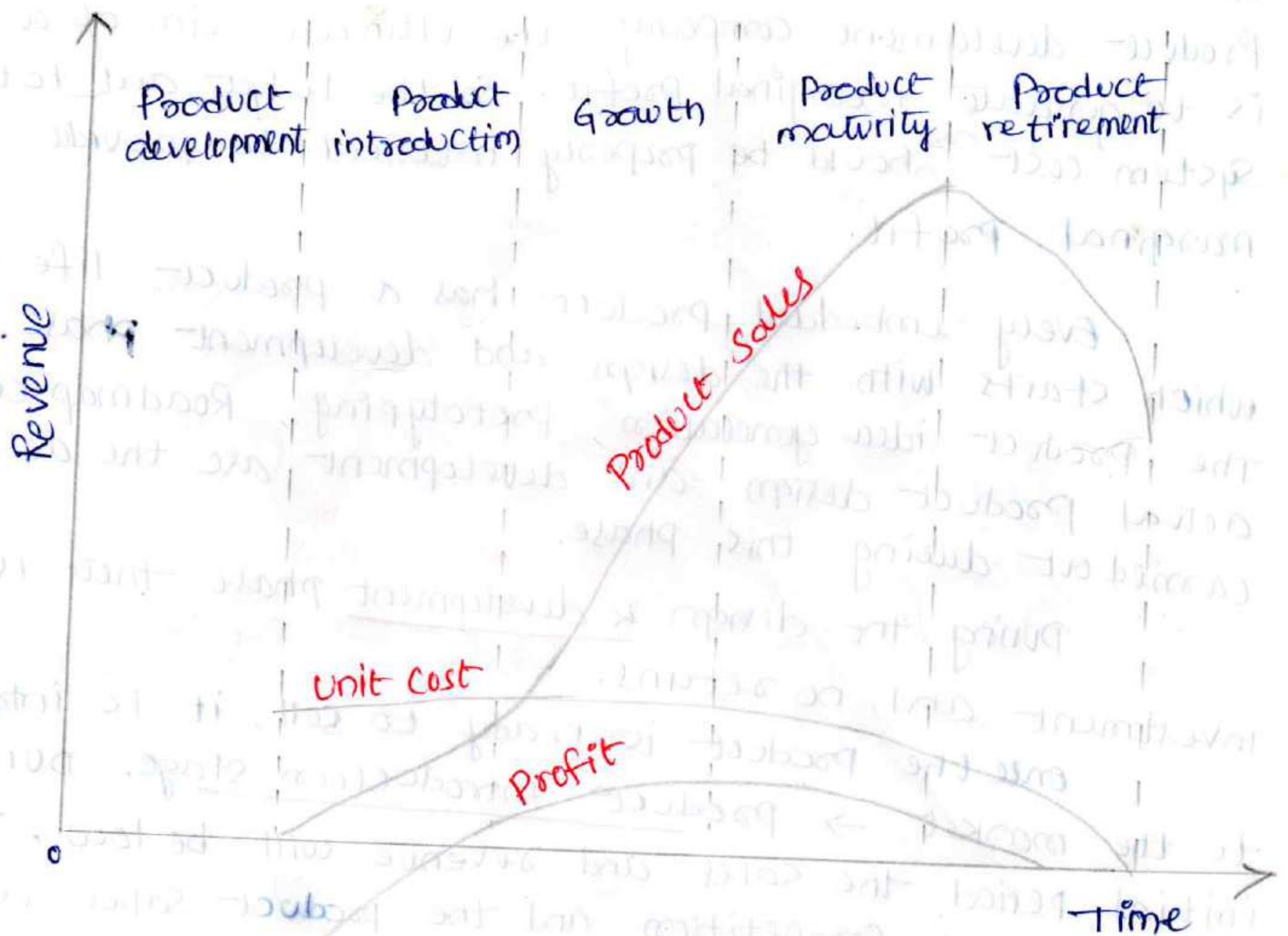
In Growth phase the product grabs high market place.

During maturity phase, the growth and sales will be steady and the revenue reaches its peak.

The product Retirement / Decline phase starts with the drop in sales volume, market share and revenue. The decline happens due to various reasons like competition from simillar product with enhanced features or technology changes etc.

At some point of decline stage the manufacturer announces discontinuing of the product.

fig: Product life cycle (PLC) graph



From the graph, it is clear that the total revenue increases from the product introduction stage to the Product maturity stage. The revenue peaks at the maturity stage and starts falling in the decline/retirement stage.

The unit cost is very high during the introductory stage.

EX: Cell phone

The profit increases with increase in sales and attains a ... falls with a dip in sales.

UNIT-II

The typical Embedded System

The typical embedded system contains a single chip Controller, which acts as the master brain of the system.
↳ controller can be a microprocessor or a microcontroller or a FPGA device or a Digital signal processor or an ASIC / Application specific standard product (ASSP).

Sensors are connected to the input port. Actuators are connected to the o/p port. Embedded hardware/software systems are basically designed to regulate a physical variable or to manipulate the state of some devices by sending some control signals to the Actuators, in response to the input signals provided by the sensors. Hence embedded system can be viewed as a reactive system.

Key boards, push buttonswitches, etc are examples for common user interface input devices where as LEDs, liquid crystal displays, piezoelectric buzzers etc. are examples for common user interface output devices for a typical embedded system. It should be noted that it is not necessary that all embedded systems should incorporate these I/O device user interfaces. It is dependent on the type of application for which the embedded system is designed.

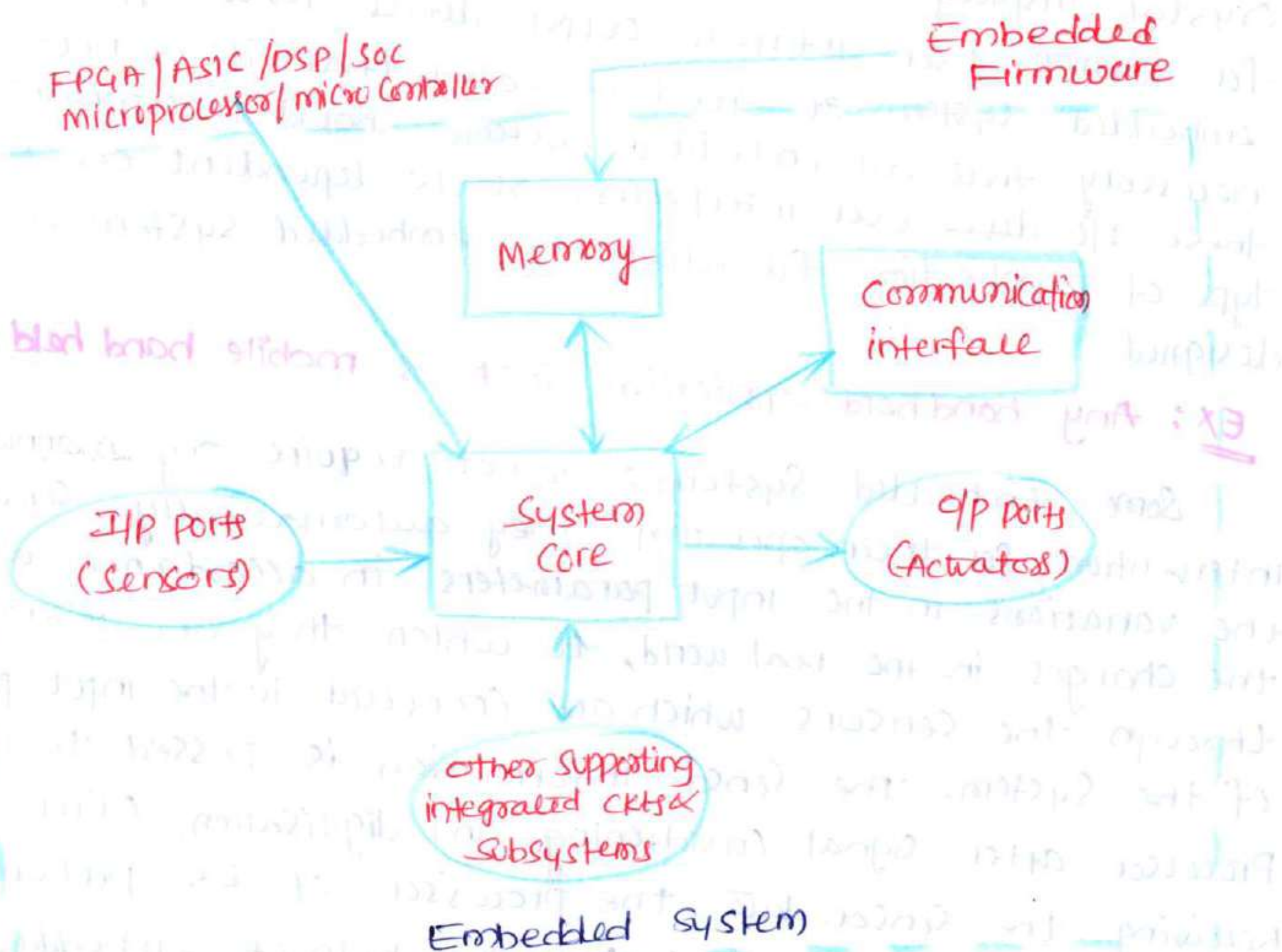
EX: Any hand held application such as mobile hand held app.

Some Embedded Systems do not require any manual intervention for their operation. They automatically sense the variations in the input parameters in accordance with the changes in the real world, to which they are interacting through the sensors which are connected to the Input port of the System. The Sensor information is passed to the Processor after signal conditioning and digitisation. After Receiving the sensor data the processor of ES performs some pre defined operations. With the help of embedded

firmware in the system. and sends some actuating signals to the actuator connected to the o/p port of ES. To make the ES work in the desired manner.

The memory of the system is responsible for holding the control algorithm and other important configuration details. For most of embedded systems, the memory for storing the algorithm or configuration data is of fixed type, which is ROM. and it is not available for the end user for modifications, which means memory is protected. from unwanted user interaction.

The most common types of memories used in ES for algorithm storage are OTP, PROM, UVEPROM, EEPROM & FLASH. Some times the system requires temporary memory for performing arithmetic operations or control algorithm execution and this type of memory is known as working memory. RAM is used as working memory. Various types of RAM like SRAM, DRAM and NVRAM are used for this purpose.



Core of the Embedded System

Embedded Systems are domain and application specific. The core of the embedded System has,

1. General purpose and Domain specific Processors
   - 1.1. Microprocessors
   - 1.2. Micro Controllers
   - 1.3. Digital signal processors
2. Application specific integrated circuits (ASICs)
3. Programmable Logic devices (PLDs)
4. commercial Off-the-shelf Components (COTS)

(1) General purpose and Domain Specific processors :-

Most of the embedded Systems are processor/Controller based. The processor may be a MP or MC or a DSP, depending on the domain and application.

Most of the embedded Systems in the industrial Control and monitoring applications make use of MP or MC. where as domains which require signal processing such as Speech Coding, Speech recognition etc. use DSP.

1.1. Microprocessors

A silicon chip representing a Central Processing unit (CPU), which is capable of performing arithmetic as well as logical operations according to a pre-defined set of instructions.

It is a dependent unit. It requires the combination of other chips like timers, program and data memory Chips, interrupt Controllers, etc. for functioning. Most of the time it is general purpose in design and operation.

It doesn't contain a built in I/O port. The I/O port functionality needs to be implemented with the help of external programmable peripheral interface chips like 8255. It has limited power saving options compared to microcontrollers.

1.2 Microcontrollers

A microcontroller is a highly integrated chip that contains a CPU, scratchpad RAM, special and general purpose register arrays, on chip ROM/FLASH memory for program storage, timer and interrupt control units and dedicated I/O ports.

It is a self contained unit and it doesn't require external interrupt controller, timer, UART, etc. for its functioning. These are mostly application-oriented or domain-specific. Most of the processors contain multiple built-in I/O ports which can be operated as a single 8 or 16 or 32 bit port or as individual port pins. It includes lot of power saving features.

1.3 Digital Signal Processors

These are powerful special purpose 8/16/32 bit microprocessors designed specifically to meet the computational demands and power constraints of today's embedded audio, video and communications applications. DSPs are 2 to 3 times faster than the general purpose microprocessors in signal processing applications. This is becz of the architectural difference b/w the two.

DSPs implement algorithms in hardware which speeds up the execution where as general purpose processors implement the algorithm in firmware and

the speed of ~~operation~~ execution depends primarily on the clock for the processors. A typical DSP has the following units.

1. Program memory - memory required for storing the Program by DSP to process the Data.
2. Data memory - working memory for storing temporary Variables and data/signal to be processed.
3. Computational Engine - Performs the signal processing in accordance with the stored program memory. computational engine incorporates many specialised arithmetic units and each of them operates simultaneously to increase the execution speed. It also incorporates multiple hardware shifters for shifting operands and thereby saves exection time.
4. I/O unit - Acts as an interface b/w the outside world and DSP. It is responsible for capturing signals to be processed and delivering the processed signals. DSP employs a large amount of real-time calculations. Sum of products calulation, convolution, FFT, DFT etc. are some of the operations performed by DSP.

RISC vs CISC processors/controllers

The term RISC stands for reduced instruction set computing. As the name implies, all RISC processors/controllers possess lesser number of instructions, in the range of 30 to 40.

CISC - complex instruction set computing. in this instruction set is complex and instructions are high in number. for example the instruction set contains 255 instructions.

RISC processors are comfortable since s/he needs to learn only a few instructions. whereas for a CISC processor s/he needs to learn more number of instructions and should understand the context of usage of each instruction.

Ex for RISC processor is Atmel AVR microcontroller and its instruction set contains only 32 instructions.

Ex for CISC controller is 8051 microcontroller and its instruction set contains only 255 instructions.

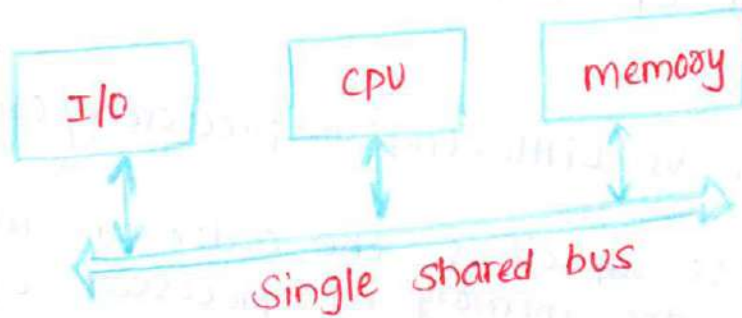
RISC	CISC
1. Lesser number of instructions	1. Greater no. of instructions
2. Instruction pipelining and increased exection speed.	2. Generally no instruction pipelining feature
3. orthogonal instruction set means allow each instruction to operate on any register and use any addressing mode.	3. Non-orthogonal instruction set.
4. A large no. of registers are available	4. Limited number of general purpose registers.
5. Programmer needs to write more code to exeute a task since the instructions are simpler ones.	5. A programmer can achieve the desired functionality with a single instruction which in turn provides the effect of using more simpler single instructions in RISC
6. single, fixed length instructions	6. variable length instructions
7. with Harvard architecture	7 Can be Harvard or Von-neumann architecture

Harvard vs Von-Neumann processor/controller Architecture

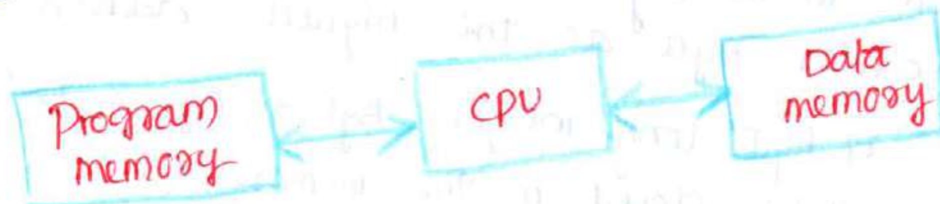
Microprocessors/Controllers based on the Von-Neumann architecture shares a single common bus for fetching both instructions and data. program instructions and data are stored in a common main memory.

Von-Neumann architecture based processors/controllers first fetch an instruction and then fetch the data to support the instruction from code memory. The two seperate fetches slows down the controller's operation.

Von-Neumann architecture also referred as princeton architecture, since it was developed by the princeton university.



Microprocessors/controllers based on the Harvard architecture will have separate data bus and instruction bus. This allows the data transfer and program fetching to occur simultaneously on both buses. With Harvard arch, the data memory can be read and written while the program memory is being accessed. These seperated data memory and code memory buses allow one instruction to execute while the next instr. is fetched (pre-fetching). The pre-fetch theoritically allows much faster execution than von-neumann architecture.



Harvard	Von-Neumann
1. separate buses for instruction and data fetching	1. single bus is shared for inst & data fetching
2. Easier to pipeline, so high performance can be achieved	2. Low performance compared to Harvard.
3. High cost	3. Cheaper
4. No memory alignment problems	4. Allows self modifying codes
5. Sine data memory and program memory are stored physically in different locations, no changes for accidental corruption of program memory	5. In this they are stored in same chip, there is a chance of corruption of program memory.

Big - Endian Vs Little - Endian processors/controllers

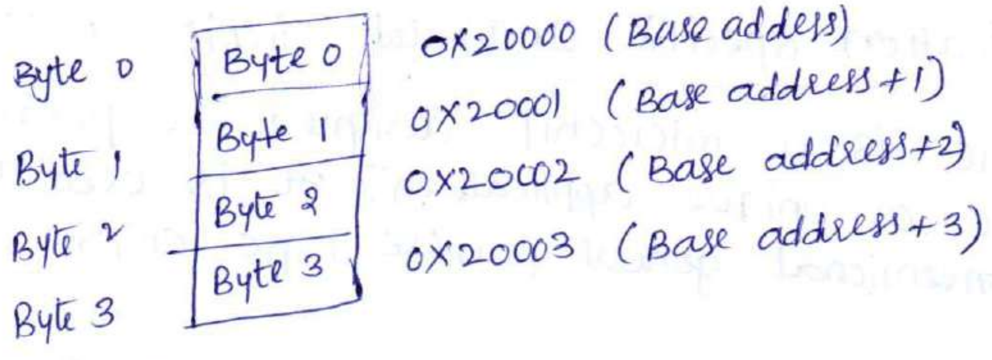
Endianness specifies the order in which the data is stored in the memory by processor operations in a multi byte system. Suppose the world length is two byte then data can be stored in memory in two diff ways.

1. Higher order of data byte at the higher memory and lower order of data byte at location just below the higher memory.
2. Lower order of data byte at the higher memory and higher order of data byte at location just below the higher memory.

Little endian means the lower-order byte of the data is stored in memory at the lowest address, and the higher-order byte at the highest address.

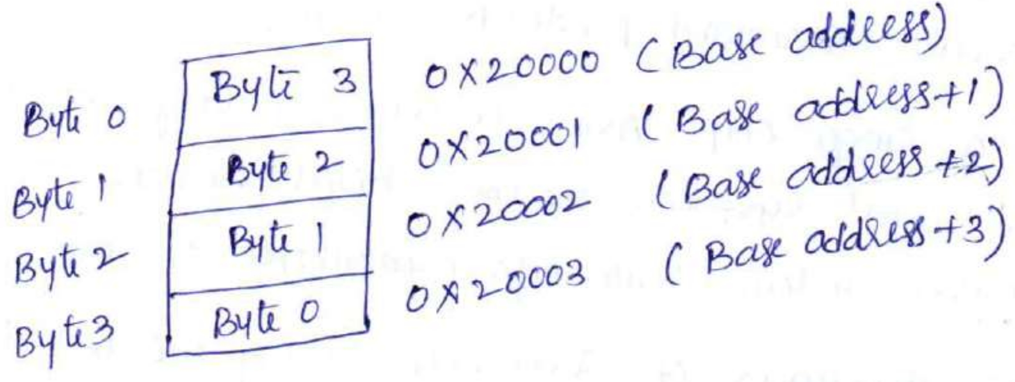
Ex: 4 byte long integer Byte 3, Byte 2, Byte 1, Byte 0 will be stored in the memory as,

Little Endian Operation →



Big endian means the higher order byte of the data is stored in memory at the lowest address, and the lower-order byte at the highest address.

Big Endian Operation →



Loard store operation and instruction pipelining :-

The RISC processor instruction set is orthogonal, meaning it operates on registers. The memory access related operations are performed by the special instructions load & store. If the operand is specified as memory location, the content of it is loaded to a register using the 'load instruction.' The 'instruction store' stores data from a specified register to a specified memory location.

EX:

```
load R1, x
load R2, y
add R3, R1, R2
Store R3, z
```

Application Specific Integrated Circuit (ASICS) :-

- "ASIC is a microchip designed to perform a specific or unique application". It is used as replacement to conventional general purpose logic chips.

- Because of using single chip for integrates several functions there by reduces the system development cost.

- Most of the ASICs are proprietary (which having some specific name) products, it is referred as Application Specific Standard products (ASSP).

- As a single chip ASIC consumes a very small area in the total system. Thereby helps in the design of smaller system with high capabilities or functionalities.

- The developers of such chips may not be interested in revealing the internal data of it.

Programmable Logic devices (PLDs)

1. A PLD is an electronic component. It used to build digital circuits which are reconfigurable.

2. PLDs offer customers a wide range of logic capacity, features, speed, voltage characteristics

3. A logic gate has a fixed function but a PLD does not have a defined function at the time of manufacture.

4. PLDs can be reconfigured to perform any number of functions at any time.

5. A variety of tools are available for the designers of PLDs which are inexpensive and help to develop, simulate and test the designs.

6. PLDs having following two major types.
   1. CPLD complex programmable logic devices offers much smaller amount of logic upto 1000 gates.
   2. FPGAs
      ↓
      Field programmable Gate Array.
      ↓
      It offers highest amount of performance as well as highest logic density, the most features.

Advantages of PLDs :-

1. PLDs offer customer much more flexibility during the design cycle.
2. PLDs do not require long lead times for prototypes or production parts bcz PLDs are already on a distributors shelf and ready for shipment.
3. PLDs can be reprogrammed even after a piece of equipment is shipped to a customer.

Commercial Off-the shelf Components (COTs)

- A COT product is one which is used 'as-is'.
- The COTs components itself may be develop around a general purpose or domain specific processor or an ASICs or a PLDs.

- The major advantage of using COTS is that they are readily available in the market, are chip and a developer can cut down his/her development time to a great extent.

- The major drawback of using COTS components in embedded design is that the manufacturer of the COTS component may withdraw the product or discontinue the production of the COTS at any time if rapid change in technology occurs.

- Advantages :

  1. Ready to use
  2. Easy to integrate
  3. Reduces development time

- Disadvantages :

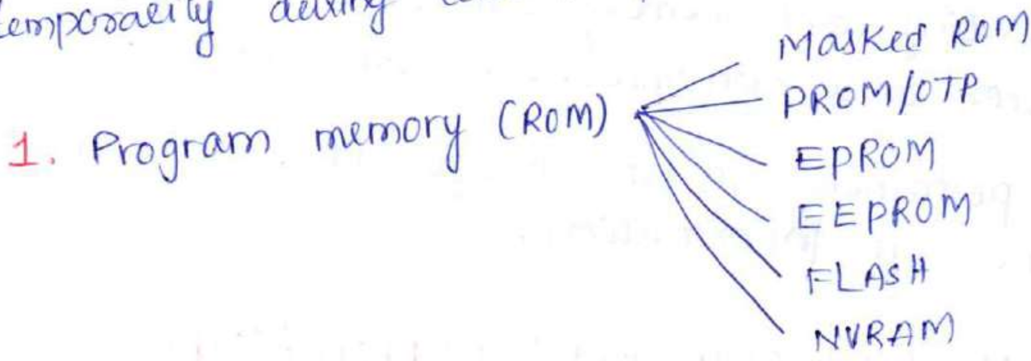
  1. No operational or manufacturing standard
  2. vendor or manufacturer may discontinue production of a particular COTS product.

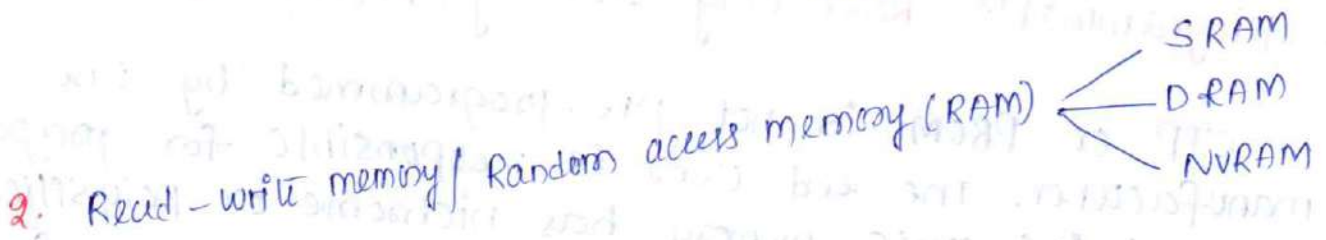
Memory

Memory is an important part of a processor/controller based Embedded Systems. Some processors or controllers have built in memory called 'on chip memory'.

Others do not contain any memory inside the chip and requires external memory to be connected with the controller/processor to store the control algorithm. Called 'off-chip memory'.

Some 'working memory' is required for holding data temporarily during certain operations.





(1) Program Storage memory (ROM) :-

The Program memory or code storage memory of an Embedded System stores the program instructions. The code memory retains its contents even after the power to it is turned off. → Non volatile storage memory

1. Masked ROM (MROM)

Masked ROM is a one-time programmable device. It uses hardwired technology for storing data.
The primary advantage of this is low cost for high volume production. They are least expensive type of solid state memory. Different mechanisms are used for the masking process of the ROM, like

1. Creation of an enhancement or depletion mode transistor through channel implant.
2. By creating the memory cell either using a standard transistor or a high threshold transistor.

MROM is permanent in bit storage, it is not possible to alter the bit information.

2. Programmable Read only memory (PROM / (OTP)

OTP or PROM is not pre-programmed by the
manufacturer. The end user is responsible for programming
these devices. This memory has nichrome or polysilicon
wires arranged in a matrix. These wires can be functionally
viewed as fuses. It is programmed by a PROM programmer
which selectively burns the fuses according to the
bit pattern to be stored.

Fuses which are not blown/burned → logic '1'
" " blown/burned → logic '0'.

The default state is logic '1'.
OTP is widely used for commercial production of ES.
bcz it is a low cost solution. OTPs can not be reprogrammed.

3. EPROM :

OTPs are not useful for development purpose. During the development phase the code is subjected to continuous changes and using an OTP each time to load the code is not economical.

Hence Erasable programmable Read only memory gives the flexibility to re-program the same chip.

EPROM stores the bit information by charging the floating gate of an FET. EPROM contains a quartz crystal window for earasing the stored inf. of the window is exposed to UV rays for a fixed amount of time (20 to 30 min), the entire memory will be erased. and also it is flexible in terms of re-programmability but it is time consuming process to earase the inf.

4. EEPROM :

As the name indicates, the information contained in the EEPROM memory can be altered by using electrical signals at the register/Byte level. They can be erased and reprogrammed in-circuit. These chips include a chip erase mode and in this mode they can be erased in a few milliseconds. It provides greater flexibility for system design. The only limitation is their capacity is limited (few KBytes) when compared with the standard ROM

5. FLASH :

It is the latest ROM technology and is the most popular ROM technology used in today's ES. designs. It combines the re-programmability of EEPROM and the high capacity of standard ROMs.

It is organised as sectors (blocks) or pages. It stores the information in an array of floating gate MOSFET transistors. The erasing of memory can be done at sector level or page level without affecting the other sector or pages.

Each sector/page should be erased before re-programming. The erasable capacity of Flash is 1000 cycles.

6. NVRAM :

Non volatile RAM is a Random access memory with battery backup. It contains Static RAM based memory and a minute battery for providing supply to the memory in the absence of external power supply.

The memory and battery are packed together in a single package. The life span of NVRAM is expected to be around 10 years. DS1644 from maxim/Dallas is an example of 32KB NVRAM.

Read – Write memory / Random Access Memory (RAM)

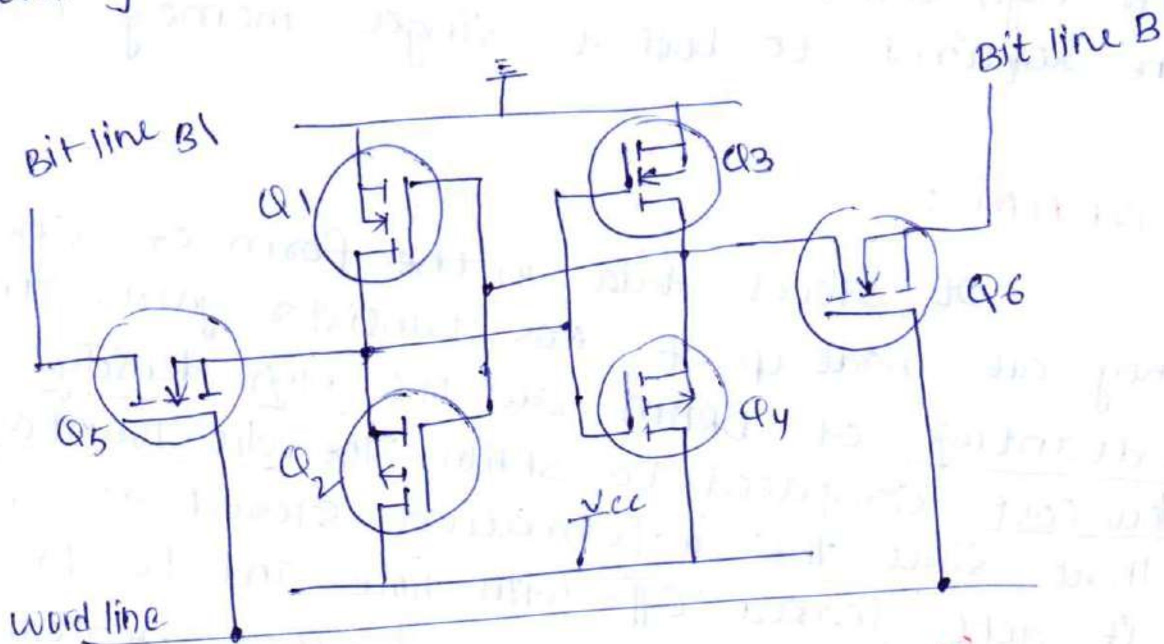
RAM is the data memory or working memory of the controller / processor. controller / processor can read from it and write to it. RAM is volatile, meaning when the power is turned off, all the contents are destroyed. RAM is classified as,

1. Static RAM (SRAM)
2. dynamic RAM (DRAM)
3. Non-volatile RAM (NVRAM)

1. SRAM

Static RAM stores data in the form of voltage. they are made up of flip-flops. SRAM is the fastest form of RAM available. In typical implementation, an SRAM cell (bit) is realised using six transistors (6 MOSFETS). 4 of them used for building the latch (FF) part of the memory cell and two for controlling the access.

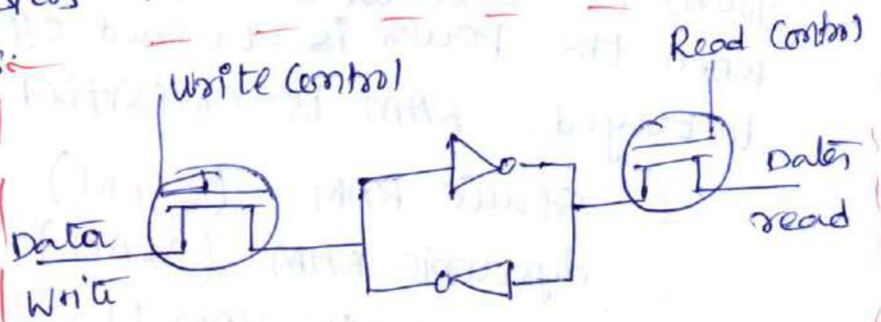
SRAM is fast in operation due to its resistive networking and switching capabilities.



SRAM cell implementation

This implementation in its simpler form can be visualised as two-cross coupled inverters with read/write control through transistors.

The 4 transistors in the middle form a cross coupled inverters:-



In order to write a value to the memory cell, apply the desired value to the bit control lines

For writing 1, make B = 1 and B\ = 0

" 0, B = 0 & B\ = 1

and make word line high.

For reading the contents of the memory cell, assert both B and B\ bit lines to 1 and set the word line to 1.

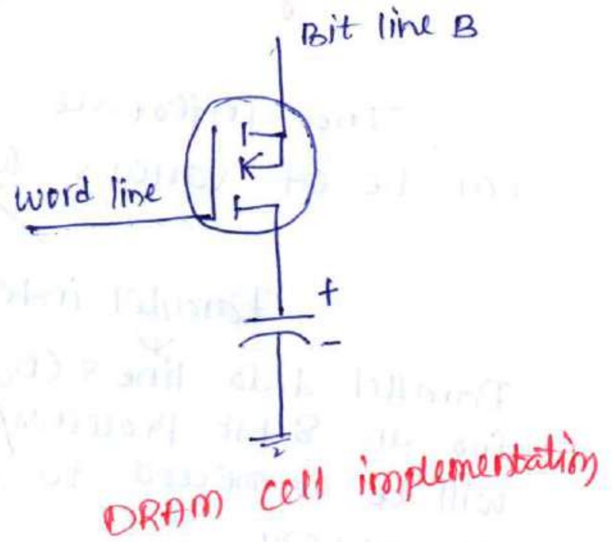
major limitation of SRAM are low capacity and high cost. Since a minimum of six transistors are required to built a single memory cell.

2. DRAM :-

It stores data in the form of charge. They are made up of MOS transistor gates. The advantage of DRAM are its high density and low cost compared to SRAM. The disadvantage is that since the information stored as charge it gets leaked off with time and to prevent this they need to be refreshed periodically.

∴ DRAM controllers are used for the refreshing operation.

The MOSFET acts as the gate for the incoming and outgoing data whereas the capacitor acts as the bit storage unit.



DRAM cell implementation

SRAM	DRAM
1. Made up of 6 CMOS transistors (MOSFET)	1. made up of a MOSFET and a capacitor.
2. Doesn't require refreshing	2. Requires refreshing
3. Low capacity	3. High capacity
4. More expensive	4. less expensive
5. Fast in operation. typical access time is 10ns.	5. Slow in operation. due to refresh requirments. typical access time is 60ns. Write operation is faster than read operation.

3. NVRAM

It is a Random access memory with battery backup. It is used for the non-volatile storage of results of operations for setting up of flags etc.

Memory according to the type of interface:
→ connection

The interface of memory with the processor/controller can be of various types.

- **Parallel interface**
  ↓
  Parallel data lines ($D_0 - D_7$) for an 8 bit processor/controller will be connected to $D_0 - D_7$ of memory.

- **serial interface**
  ↓
  (1) I2C — I square C — two line serial interface
  (2) SPI — serial Peripheral interface
  (3) single wire connection

  It is commonly used for data storage memory like EEPROM.

Memory Shadowing

Generally the execution of a program or a configuration from a Read only memory is very slow (120 to 200 ns) compared to the execution from a random access memory (40 to 70 ns). From the timing ~~diagram~~ parameters it is obvious that RAM access is about three times as fast as ROM access. Shadowing of memory is a technique adopted to solve the execution speed problem in processor-based systems.

Memory Selection for Embedded Systems :

Embedded Systems require a program memory for holding the control algorithm or embedded OS and the applications designed to run on top of it, data memory for holding variables and temporary data during task execution, and memory for holding

Non volatile data. which are modifiable by the application. The memory requirment for an embedded System in terms of RAM and ROM is solely dependent on the type of the embedded system and the applications for which it is designed.

Sensors and Actuators

The changes in the System environment or variables are detected by the sensors connected to the input port of the embedded system. If the ES is designed for any controlling purpose, the system will produce some changes in the controlling variable to bring the controlled variable to the desired value. It is achieved through an actuator connected to the output port of the ES. If the ES is designed for the monitor puepose only. Then there is no need of the actuator in the system.

Ex : ECG machine.

Sensors

A Sensor is a transducer device that converts energy from one form to another for any measurement or control puepose.

Actuators

Actuator is a form of transducer device which converts signals to corresponding physical action. Actuator acts as an output device.

The I/O Subsystem

The I/O subsystem of the embedded System facilitates the interaction of the embedded System with the external world. The interaction happens through the sensors and actuators connected to the input and output ports of the ES.

The Sensors may not be directly interfaced to the input ~~and output ports respectively~~ ports, instead they may be interfaced through signal conditioning and translating systems like ADC, optocouplers etc.

Sensor

- A Sensor is used for taking input.
- It is a transducer that converts energy from one form to another for any measurement or control purpose.
- Ex. A Temparature Sensor.

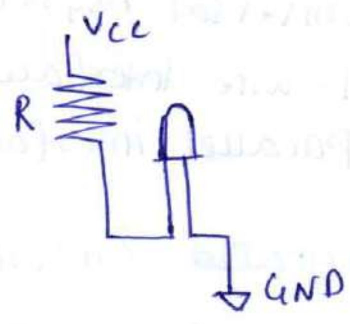
Actuator

- Actuator is used for output.
- It is a transducer that may be either mechanical or electrical which converts Signals to Corresponding Physical actions.
- Ex : LED

LED is a PN Junction diode and Contains a CATHODE and ANODE.

- For functioning the anode is Connected to +ve end of power supply and Cathode is Connected to -ve end of power supply.

- The maximum current flowing through the LED is limited by connecting a Resistor in series between the power supply and LED as shown in the figure below:



- There are two ways to interface an LED to a microprocessor/micro-controller.

1. The Anode of LED is connected to the port Pin and Cathode to Ground: in this approach the port pin sources the current to the LED when it is at logic high
2. The Cathode of LED is connected to the port Pin and Anode to Vcc: in this approach the Port pin Sources the current to the LED when it is at logic high. Here the port pin Sinks the current and the LED is turned ON when the Port pin is at logic low.

Communication Interfaces.

For any Embedded System, the communication interfaces can broadly classified into:

1. Onboard Communication interfaces :-

These are used for internal communication of the embedded System, i.e communication between different components present on the System.

Ex: Common examples of onboard interfaces are:

1. Inter integrated circuit (I2C)
2. Serial peripheral interface (SPI)
3. Universal Asynchronous Receiver Transmitter (UART)
4. 1-wire interface
5. Parallel interface.

Inter Integrated Circuit (I2C)

1. It is synchronous
2. Bi-directional, half duplex, two wire serial interface bus
3. Developed by Phillips Semiconductors in 1980.
4. It comprises of two buses:

   Serial clock - SCL
   Serial Data - SDA

5. SCL generates synchronization clock pulses.
6. SDA transmits data serially across devices
7. I2C is a shared bus system to which many devices can be connected.
8. Devices connected by I2C can act as either master or slave.
9. The master device is responsible for controlling communication by initiating/terminating data transfer.
10. Devices acting as slave wait for commands from the master and respond to those commands.

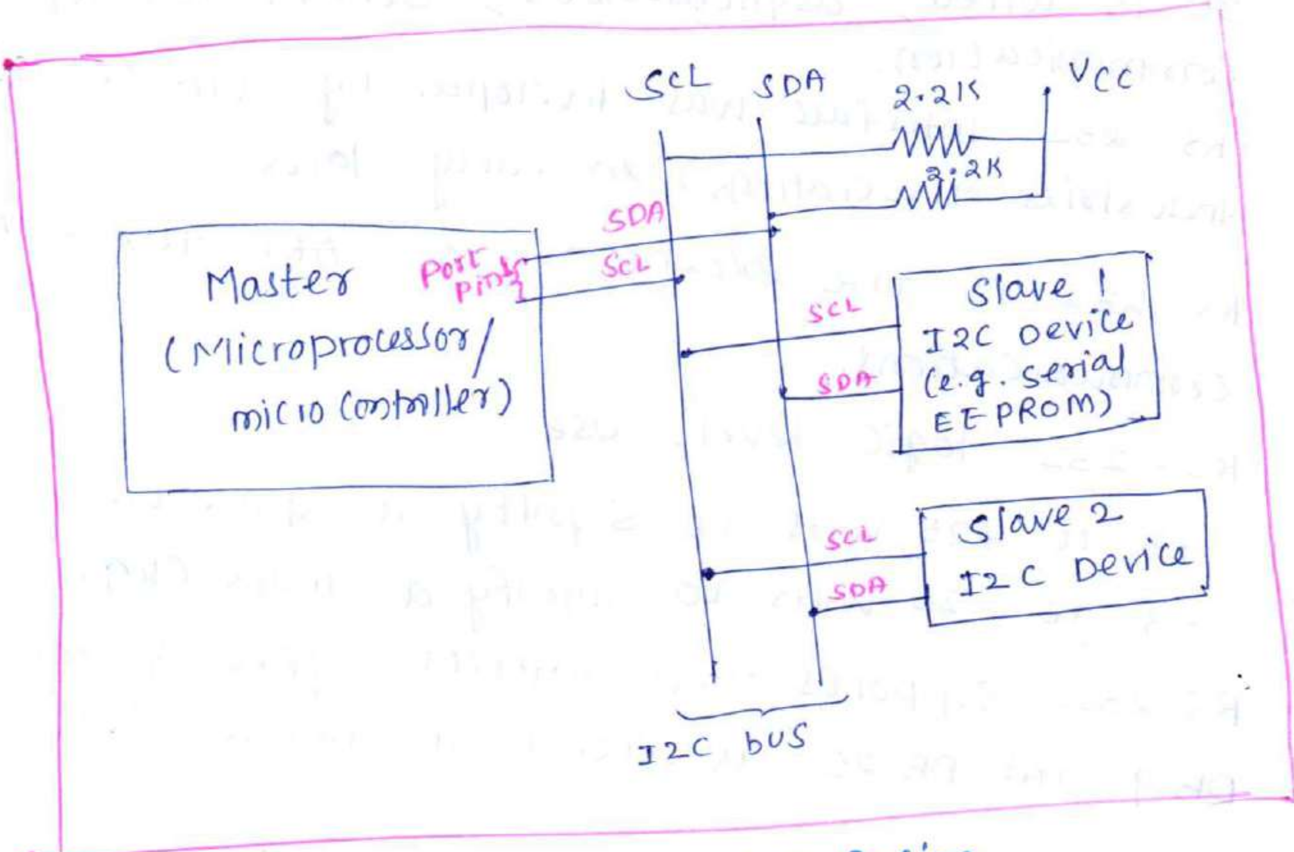


fig : I2C Bus interfacing

2. External or peripheral communication interfaces :-

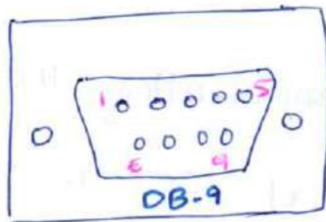
. These are used for external communication of the embedded System i.e : commenication of different components present on the System with external or peripheral components/devices.

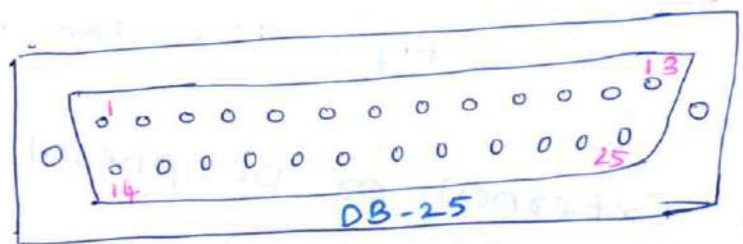
EX: common examples of external interfaces are :

- RS-232 & RS-485
- universal Serial bus (USB)
- IEEE 1394 (Firewire)
- Infrared (IrDA)
- Bluetooth
- Wi-Fi
- Zig Bee
- General packet Radio Service (GPRS)

EX : RS-232C & RS-485

- It is wired, asynchronous, Serial, full duplex communication.
- RS 232 interface was developed by EIA (electronic Industries Associations) in early 1960s.
- RS 232 is the extension to UART for external communications.
- RS-232 logic levels use:
  +3 to +25 volts to signify a space (logic 0) &
  -3 to -25 volts to signify a mark (logic 1).
- RS 232 supports two different types of connectors. DB 9 and DB 25 as shown in below fig.





- RS 232 interface is a point communication interface and the devices involved are called as Data terminating equipment (DTE) and Data communications terminating equipment (DCE)
- Embedded devices contain UART for serial transmission and generate signal levels as per TTL/CMOS logic.
- Converter chips contain converters for both transmitters and receivers.
- RS 232 is used only for point to point connections
- It is susceptible to noise and hence is limited to short distances only.

- RS 422 is another serial interface from EIA
- It supports multipoint connections with 1 transmitter and 10 receivers.
- It supports data rates up to 100kbps and distance up to 400 ft.
- RS 485 is enhanced version of RS 422 and supports up to 32 transmitters and 32 receivers.

UNIT - III

Embedded Firmware

Embedded Firmware refers to the control algorithm (program instructions) and or the configuration settings that an embedded system developer dumps into the code or program memory of the embedded system.

It is an un-avoidable part of an ES.

There are various methods available for developing the embedded Firmware. They are,

1. Write the program in highlevel languages like Embedded C/C++ using an integrated development environment (IDE).

   The IDE contains an editor, compiler, linker, debugger, simulator etc.

   IDE's are different for different family of Processors/ controllers.

   Ex: Keil micro vision3 IDE is used for all family members of 8051 microcontroller, since it contains the generic 8051 compiler C51.

2. Write the program in Assembly language using the instructions supported by your application's target Processor/ controller.

The instruction set for each family of processor/controller is different and the program written in either of the methods given above should be converted into a processor understandable machine code before loading it into the program memory.

The process of converting the program written in either a high level language or processor/controller specific Assembly code to machine readable binary code is called 'HEX file creation'.

The method used for HEX file creation is different depending on the programming techniques used. For a beginner in the embedded software field, it is strongly recommended to use the high level language based development technique. The reason is writing codes in a high level language is easy:

Other System Components :

The other System components refers to the components/circuits/ICs which are necessary for the proper functioning of the embedded system. Some of these circuits may be essential for the proper functioning of the processor/Controller and Firmware execution.

watchdog timer,
Reset IC
brown-out protection IC. etc.

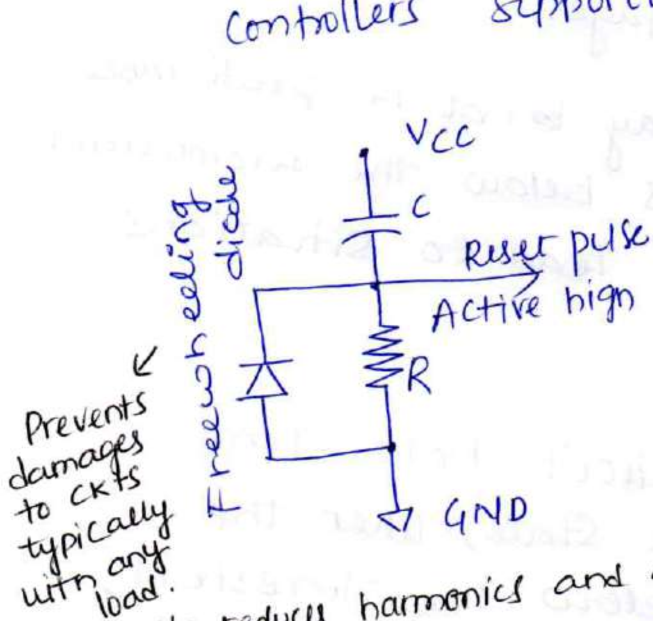
are the examples of circuits/ICs which are essential for the proper functioning of the processor/controllers.

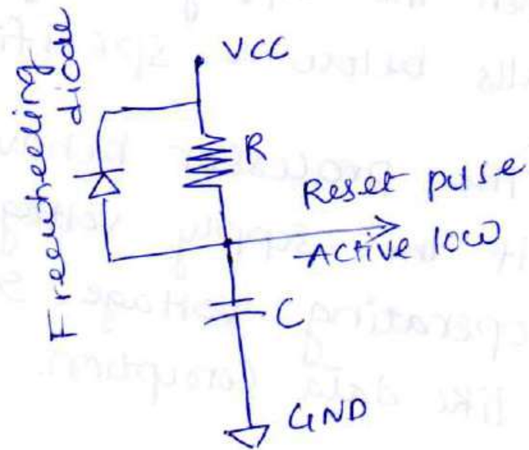
Some of the Controllers or SoCs integrate these components within a single IC and doesn't require such components externally connected to the chip for proper functioning.

Reset circuit :

1. The circuit is essential to ensure that the device is not operating at a voltage level where the device is not guaranteed to operate, during system power ON.

2. The Reset signal brings the internal registers and the different hardware systems of the processor / Controller to a known state and starts firmware execution from the reset vector (Normally from vector address 0x0000 for conventional processors / Controllers. The reset vector can be relocated to an address for processors/ Controllers supporting bootloader).





It reduces harmonics and also suppress the voltage spikes

3. The reset signal can be either active high (The processor undergoes reset when the reset pin of the processor is at logic high) or active low (The processor undergoes reset when the reset pin of the processor is at logic low).

4. The reset signal to the processor can be applied at power ON through an external passive reset ckt comprising a capacitor and resistor or through a standard reset IC like MAX 810 from Maxim Dallas

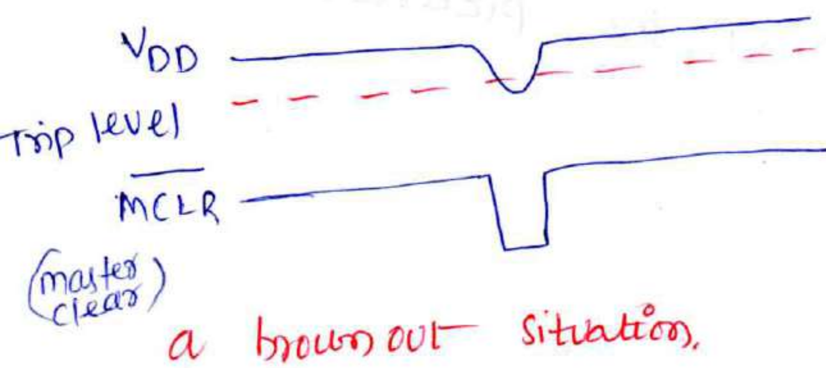
5. Some microprocessors contain inbuilt reset circuitry and they don't need external reset circuitary.

Brown-out protection Circuit :

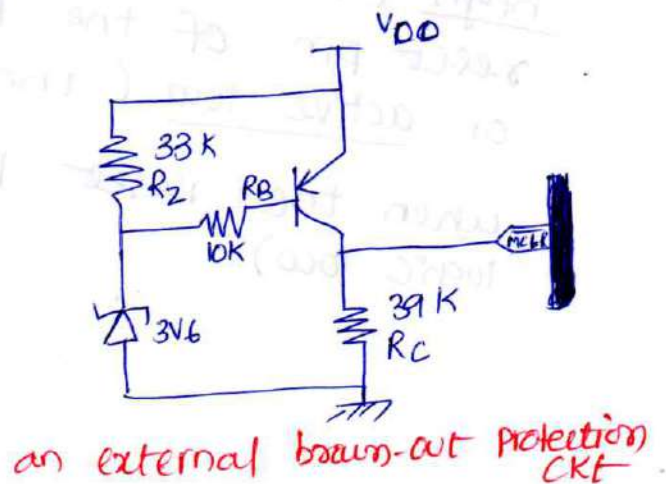
1. Brown-out protection circuit prevents the processor/controller from unexpected program execution behaviour when the supply voltage to the processor/controller falls below a specified voltage.

2. The processor behaviour may not be predictable if the supply voltage falls below the recommended operating voltage. It may lead to situations like data corruption.

3. A brown-out protection circuit holds the processor/controller in reset state, when the operating voltage falls below the threshold, until it rises above the threshold voltage.



a brown out situation.

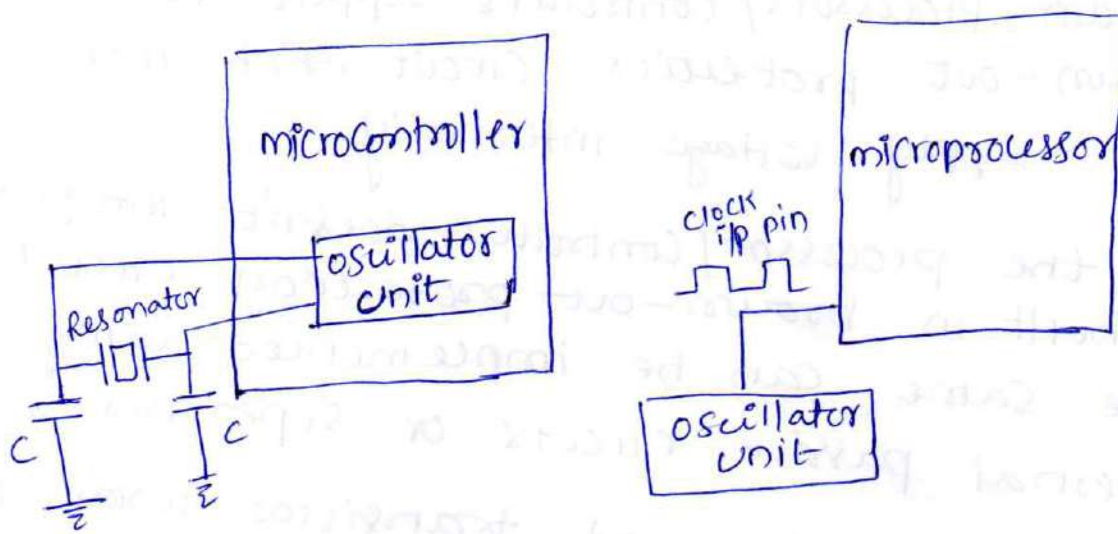


an external brown-out protection ckt

4. Certain processors/controllers support built in brown-out protection circuit which monitors the supply voltage internally.
5. If the processor/controller doesn't integrate a built in brown-out protection circuit, the same can be implemented using external passive circuits or supervisor ICs
6. The zener diode and transistor forms the heart of the circuit. The transistor conducts always when the supply voltage V_{DD} greater than that of the sum of V_{BE} and V_Z.
7. The transistor stops conducting when the supply voltage falls below the sum of V_{BE} and V_Z. The values of the resistors can be selected based on the electrical characteristics.

Oscillator unit:

1. A microprocessor/ microcontroller is a digital device made up of digital combinational and sequential circuits.
2. The instruction execution of a microprocessor/ controller occurs in sync with a clock signal.
3. The oscillator unit of the embedded system is responsible for generating the precise clock for the processor.



4. Certain processor/controller chips may not contain a built in oscillator unit and require the clock pulses to be generated and supplied externally.

5. Quartz crystal oscillators are example for clock pulse generating devices.

6. The total system power consumption is directly proportional to the clock frequency. The power consumption increase with the increase in the clock frequency.

7. The accuracy of the program execution depends on the accuracy of the clock signal. The accuracy of the clock frequency of the crystal oscillator is normally expressed in terms of parts per million.

Real time clock (RTC) :-

1. The system component responsible for keeping track of time. RTC holds information like current time (In hour, minutes and seconds) in 12 hour/24 hour format, date, month, year, day of the week etc and supplies timing reference to the system.

2. RTC is intended to function even in the absence of power. RTCs are available in the form of integrated circuits from different semiconductor manufacturers like maxim/Dallas, ST microelectronics etc.

3. The RTC chip contains a microchip for holding the time and date related information and backup battery cell for functioning in the absence of power, in a single IC package.

4. The RTC chip is interfaced to the processor or controller of the ES.

5. For operating system based embedded devices, a timing reference is essential for synchronizing the operations of the OS kernal. The RTC can interrupt the OS kernal by asserting the interrupt line of the processor/controller to which the RTC interrupt line is connected.

Real time clock (RTC) :-

1. The system component responsible for keeping track of time. RTC holds information like current time (In hour, minutes and seconds) in 12 hour/24 hour format, date, month, year, day of the week etc and supplies timing reference to the system.

2. RTC is intended to function even in the absence of power. RTCs are available in the form of integrated circuits from different Semiconductor manufacturers like maxim/Dallas, ST microelectronics etc.

3. The RTC chip contains a microchip for holding the time and date related information and backup battery cell for functioning in the absence of power, in a single IC package.

4. The RTC chip is interfaced to the processor or controller of the ES.

5. For operating system based embedded devices, a timing reference is essential for synchronizing the operations of the OS kernal. The RTC can interrupt the OS kernal by asserting the interrupt line of the processor/Controller to which the RTC interrupt line is connected.

6. The OS kernal identifies the interrupt in terms of the interrupt request (IRQ) number generated by an interrupt controller.

7. One IRQ can be assigned to the RTC interrupt and the kernal can perform necessary operations like system date time updation, managing software timers etc when an RTC timer tick interrupt occurs.

Watchdog timer (WDT) :-

1. A timer unit for monitoring the firmware execution.
2. Depending on the internal implementation, the watchdog timer increaments or decrements a free running counter with each clock pulse and generates a reset signal to reset the processor if the count reaches zero for a down counting watchdog, or the highest count value for an up counting watchdog.
3. If the watchdog counter is in the enabled state, the firmware can write a zero (for up counting watchdog implementation) to it before starting the execution of a piece of code (subroutine or portion of code which is susceptible to execution hang up) and the watchdog will start counting.

If the firmware execution doesn't complete due to malfunctiong, within the time required by the watchdog to reach the maximum count, the counter will generate a reset pulse and this will reset the processor.

4. If the firmware execution completes before the expiration of the watchdog timer the WDT can be stopped from action.

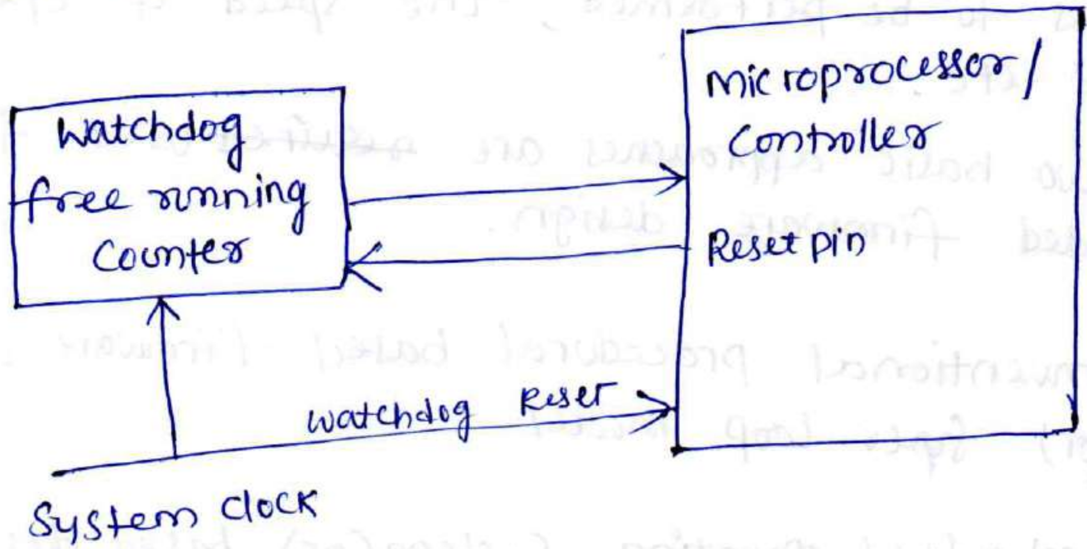


fig: External watchdog timer unit interfacing with processor

5. Most of the processors implement watchdog as a built-in component and provides status register to controll the watchdog timer (like enabling and disabling watchdog functioning) and watchdog timer register for writing the count value.

If the processor or controller doesn't contain a built in watchdog timer, the same can be implemented using an external watchdog timer IC circuit.

Embedded Firmware design Approaches :-

Firmware design approaches for embedded product is purely dependent on the complexity of the functions to be performed, the speed of operation required etc.

Two basic approaches are ~~required~~ used for embedded firmware design.

1. Conventional procedural based firmware design (or) Super Loop model
2. Embedded operating System (OS) based design.

Super Loop model :-

The super loop based firmware development approach is adopted for applications that are not time critical and where the response time is not so important. It is very similar to a conventional procedural programming where the code is executed task by task. The task listed at the top of the program code is executed first and the tasks just below the top are executed after completing the first task.

In a multitask based system, each task is executed in serial in this approach. The firmware execution for this will be.

1. Configure the common parameters and perform initialisation for various hardware components memory, registers etc.
2. Start the first task and execute it.
3. Execute the second task.

4. Execute the next task
5. :
6. :
7. Execute the last defined task.
8. jump back to the first task and follow the same flow.

From the firmware execution sequence, it is obvious that the order in which the tasks to be executed are fixed and they are hard coded in the code itself. also the operation is an infinite loop based approach.

we can visualise the operational sequence listed above in terms of a C program code as .

```
void main ( )
{
Configurations ( );
Initializations ( );
while (1)
{
Task 1 ( );
Task 2 ( );
:
:
Task n ( );
}
}
}
```

Almost all tasks in embedded system applications are non-ending and are repeated infinitely throughout the operation. From the above C code it is clear that the tasks 1 to n are performed one after another and when the last task (nth task) is executed the firmware execution is again re-directed to

task 1 and it is repeated forever in the loop. This approach is also referred as super loop based approach.

Since the tasks are running inside an infinite loop, the only way to come out of the loop is either a hardware reset or an interrupt assertion.

The super loop based design doesn't require an operating system, since there is no need for scheduling which task is to be executed and assigning priority of each task.

In super loop based design, the priorities are fixed and the order in which the tasks to be executed are also fixed.

A typical example of a super loop based product is an electronic video game toy containing keypad and display unit.

The super loop based design is simple and straight forward without any os related overheads.

Disadvantages:

1. The major drawback of this approach is that any failure in any part of a single task will affect the total system.

2.

Unit-4

RTOS Based Embedded System Design

For Operating system based firmware execution in embedded device can address a time critical response for tasks/events.

1) Assign priority to tasks and execute the high priority task when task is ready to execute.
2) Dynamically change the priorities of tasks if required on a need basis.
3) Schedule the execution of tasks based on priorities.
4) Switch the execution of task when a task is waiting for an external event (or) system resource including I/o device operation.

Operating System Basics :-

The operating System acts as a bridge between the user applications/tasks and the underlying system resources through a set of system functionalities and services. The OS manages the system resources and makes them available to the user applications/tasks on need basis. A normal computing system is a collection of different I/o subsystems, working and storage memory.

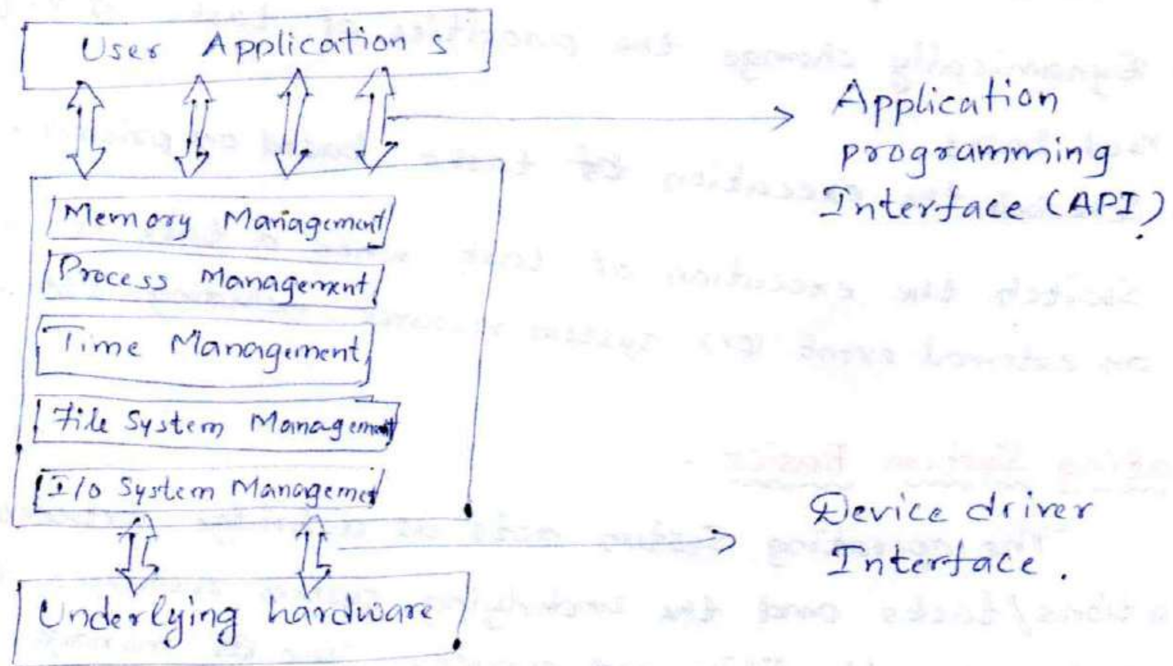
The primary functions of an Operating System is

-> Make the system convenient to use
-> Organise and manage the system resources efficiently and correctly.

The kernal:

The kernel is the core of the operating system and is responsible for managing the system resources and the communication among the hardware and other system services. kernal acts as the abstraction layer between system resources and user applications. kernal contains a set of libraries and services. For a general purpose OS, the kernel contains different services for handling the following.

Process Management: It deals with managing the processes/tasks. Process management includes setting up the memory space for the process, loading the process's code into the memory space, allocating system resources, scheduling and managing the execution of process setting up and managing the Process Control Block (PCB), Inter process communication and synchronization, process termination/deletion etc.



The Operating System Architecture:-

Primary Memory Management: It refers to the volatile Memory (RAM) where processes are loaded and variables are shared data associated with each process are stored. The Memory management unit (MMU) of the kernel is responsible for

- → keeping track of which part of the memory area is currently used by which process.
- → Allocating & Deallocating memory space on a need basis.

Secondary Storage Management: It deals with managing the secondary storage memory devices, if any, connected to the System. Secondary memory is used as backup medium for programs and data since the main memory is volatile. In most of the systems, the secondary storage is kept in disks (Hard disk). It deals with

- → Disk storage allocation
- → Disk Scheduling (Time interval at which the disk is activated to backup data)
- → Free disk space Management.

File System Management: File is collection of related information. A file could be a program (Source code / executable), text files, image files, word documents, audio/video files etc. Each of these files differ in the kind of information they hold and the way in which the information is stored. The file system management service of kernel is responsible for

- → The creation, deletion and alteration of files.
- → Creation, deletion and alteration of directories.
- → Saving of files in the secondary storage memory (eg. Hard disk,
- → Providing automatic allocation of file space based on the amount of free space available.
- → Providing a flexible naming convention for the files.

I/O System (Device) Management: In a well-structured OS, the direct accessing of I/O devices are not allowed access to the are provided through a set of Application Programming Interfaces (APIs) exposed by the kernel.

The service 'Device Manager' of the kernel is responsible for handling all I/O device related operations. The kernel talks to the I/O device through a set of low-level system calls, which are implemented in a service, called device drivers. The device drivers are specific to a device (or) a class of devices. The device manager is responsible for

- → Loading and unloading of device drivers.
- → Exchanging information and the system specific control signals to and from the device.

Protection Systems: It deals with implementing the security policies to restrict the access to both user and system resources by different applications (or) processes / users. In multiuser supported operating systems, one user may not be allowed to view (or) modify the whole / portion's of another user's data (or) profile details. In addition, some application may not be granted with permission to make use of some of the system resources. This kind of protection is provided by protection services running with the kernel.

Interrupt Handler: kernel provides handler mechanism for all external/internal interrupts generated by the system.

The applications/services are classified into two categories.

1) User applications
2) kernel application.

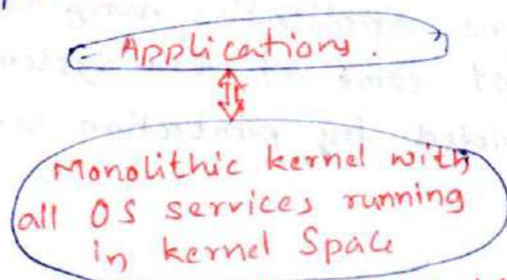
The memory space at which the kernel code is located is known as '**kernel Space**', lly, all user applications are loaded to a specific area of primary memory and this memory area is reffered as '**User Space**'. User space is the memory area where user applications are loaded and executed. The partitioning of memory into kernel and user space is purely Operating System dependent.

Based on the kernel design, kernels can be classified into

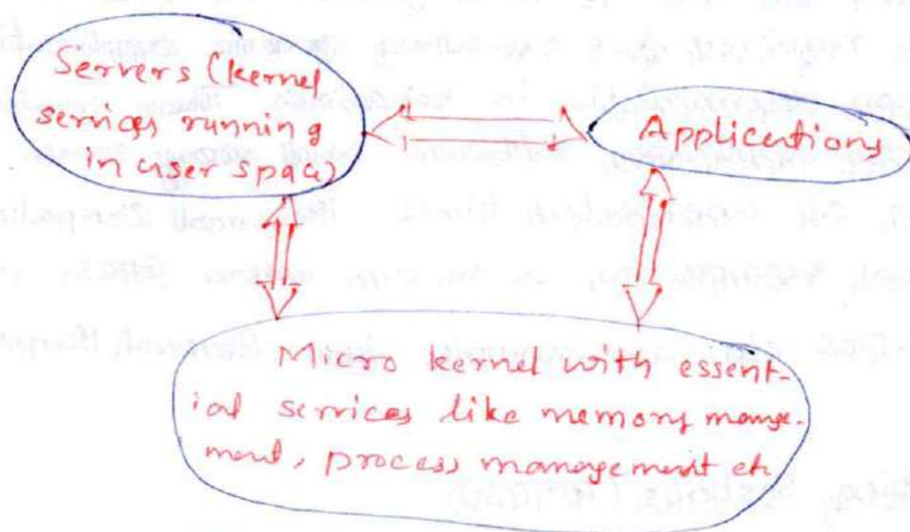
1) Monolithic kernel
2) Micro kernel.

Monlithic kernel:

In this architecture, all kernel services run in kernel space. Here all kernel modules run within the same memory space under a single kernel thread. The tight integration of kernel modules in monolithic kernel architecture allows the effective utilization of the low-level features of the underlying system. The major drawback of monolithic kernel is that any error (or) failure in any of one kernel module leads to the crashing of the entire kernel application. LINUX, SOLARIS, MS-DOS kernels are examples of monolithic kernel.



Micro kernel: The micro kernel design incorporates only the essential set of OS services into the kernel. The rest of the OS services are implemented in programs known as 'Servers' which run in user Space. This provides a highly modular design and OS-neutral abstraction to the kernel. Memory management, process management, timer systems and interrupt handlers are the essential services, which forms the part of the micro kernel. Mach, QNX, Minix3 kernels are examples of micro kernel.



Micro kernel Model.

⇒ It offers the following benefits.

⇒ Robustness: If a problem is encountered with the any of services, which runs as "Server" application, the same can be reconfigured and restarted without the need for restarting the entire OS. Thus, this approach is highly useful for systems, which demands high availability. Since the services which run as "Servers" are running on a different memory space, the chances of corruption of kernel services are ideally zero

⇒ Configurability: Any services, which run as 'Server' application can be changed without the need to restart the whole system. This makes the system dynamically configurable.

Types of Operating Systems:

Depending on the type of kernel and kernel services, purpose and type of computing systems where the OS is deployed and the responsiveness to applications, Operating Systems are classified into different types.

General Purpose Operating System (GPOS)

The operating systems, which are deployed in general computing systems, are reffered as General Purpose Operating Systems (GPOS). The kernel of such an OS is more generalised and it contains all kinds of services required for executing generic applications. GPOS's are often quite non-deterministic in behaviour. Their services can inject random delays into application software and may cause slow responsiness of application at unexpected times. Personal Computer/Desktop system is a typical example for a system where GPOSs are deployed. Windows XP/ MS-DOS etc. are examples for General Purpose Operating Systems.

Real-Time Operating Systems (RTOS) :-

A Real-time Operating System implements policies and rules concerning time-critical allocation of a system's resources. The RTOS decides which applications should run in which order and how much time needs to be allocated for each application. Predictable performance is the hallmark of a well-designed RTOS.

The Real-time kernel :-

In complement to the conventional OS kernel, the Real-Time kernel is highly specialised and it contains only the minimal set of services required for running the user applications/tasks. The basic functions of a Real-Time kernel are listed below.

-> Task/ Process Management
-> Task/ Process Scheduling
-> Task / Process synchronisation
-> Error/ Exception handling
-> Memory Management
-> Interrupt handling
-> Time Management

Task Process Management: Deals with setting up memory space for the tasks, loading the task code into the memory space, allocating the system resources, setting up a Task Contr Block (TCB) information corresponding to a task. TCB contains following set of information.

Task ID :- Task identification Number.

Task State :- The current state of the task (eg: State = "Ready" for a task which is ready to execute)

Task type :- Indicates what is the type for this task. The tas can be hard real time (or) soft real time (or) backgrou task.

Task Priority :- Eg: Task priority = 1 for task with priority = 1.

Task Context Pointer :- Pointer for context saving.

Task Memory Pointer :- Pointers to the code memory, data memory and stack memory for the task.

Task System Resource Pointer :- Pointers to system resources (semaphores, mutex etc) used by the task.

Task Pointers: Pointers to other TCB's (TCB's for preceding, next and waiting tasks).

The parameters and implementation of the TCB is kernel dependent. The TCB parameters vary across different kernels, based on the task management implementation. Task management service utilises the TCB of a task in the following wa.

-> Creates a TCB for a task on creating task.
-> Delete/remove the TCB of task when the task is terminated (or) deleted.
-> Reads TCB to get the state of a task
-> Update the TCB with updated parameters on need basis.
-> Modify the TCB to change the priority of the task dynamically.

Task/Process Scheduling :- Deals with sharing the CPU among various tasks/processes. A kernel application called "Scheduler" handles the task scheduling. It's an algorithm implementation, which performs the efficient and optimal scheduling of tasks to provide a deterministic behaviour.

Task/Process Synchronisation :- Deals with synchronising the concurrent access of a resource, which is shared across multiple tasks and the communication between the various tasks.

Error/Exception Handling :- Deals with registering and handling the errors occured/exceptions raised during the execution of task. Insufficient memory, time out, deadlocks, dead line missing bus error, divide by zero, unknown instruction execution etc. are examples of errors/exceptions. Errors/Exceptions can happen at the kernel level services (or) at task level exception. Deadlock is an example for kernel level exception, where as time out is an example for a task level exception.

Memory Management :- Compared to the General Purpose Operating System, the memory management functions of RTOS kernel is slightly different. In general, the memory allocation time increases depending on the size of the block of memory needs to be allocated and the state of the allocated memory block. RTOS makes use of 'block' based memory allocation technique, instead of the usual dynamic memory allocation techniques used by GPOS. RTOS kernel uses blocks of fixed size of dynamic memory and a block is allocated for a task on need basis. The blocks are stored in "free buffer Queue". To achieve predictable timing and avoid the timing over-heads, most of the RTOS kernels allow tasks to access any of the memory blocks without any memory protection.

A few RTOS kernels implement Virtual memory concept for allocation if the system supports secondary memory storage. The memory allocation can be implemented as constant functions & and it consumes fixed amount of time for memory allocation.

The block memory concept avoids the garbage collection overhead also.

Interrupt Handling: Deals with the handling of various types of interrupts. Interrupts provide Real-Time behaviour to systems. Interrupts inform the processor that an external device (or) an associated task requires immediate attention of the CPU. Interrupt can be either Synchronous (or) Asynchronous.

Interrupts which occurs in sync with the currently executing task is known as Synchronous interrupts. Usually the software interrupts fall under the synchronous interrupt category. Divide by zero, memory segmentation error etc. are examples of synchronous interrupt For synchronous interrupts, the interrupt handler runs in the same context of the interrupting task.

Interrupts which occur at any point of execution of any task, and are not sync with the currently executing task. are known as Asynchronous interrupts. The interrupts generated by external devices connected to the processor/controller, timer overflow interrupts, serial data reception/transmission interrupts etc are examples of asynchronous interrupts. For Asynchronous interrupts kerne uses a seperate task and runs in a different content.

Most of the RTOS kernel implements 'Nested Interrupts' architecture. Interrupt nesting allows the pre-emption (interruption) of an Interrupt service Routine (ISR), servicing an interrupt, by a high priority Interrupt.

Time Management :- Accurate time management is essential for providing precise time reference for all applications. The time reference to kernel is provided by a high resolution Real-Time clock (RTC) hardware chip. The hardware timer is programmed to interrupt the processor / controller at a fixed rate. This timer interrupt is reffered as 'Timer Tick'. "Timer Tick" is taken as the timing reference by the kernel. Usually the 'Timer tick' varies in micro seconds range. The time parameter for task are expressed as the multiples of the 'Timer Tick'.

The system time is updated based on the 'Timer Tick'. If the system register is 32 bit wide and the 'timer tick' interval is 1 us, The system time register will reset in

$$2^{32} \times 10^{-6} / (24 * 60 * 60) = 49700 \text{ Days.}$$

If the Timer tick interval is 1ms, the system will reset in

$$2^{32} \times 10^{-3} / (24 \times 60 \times 60) = 497 \text{ days} = 49.7 \cong 50 \text{ days}$$

'Timer Tick' interrupts is handled by the 'Timer Interrupt' handler of kernel. The 'Timer tick' interrupt can be utilised for implementing the following actions

→ Save the context (context of currently executing task)
→ Increment the System time register by one. Generate timing error and reset the system time register if the timer tick is greater than the maximum range available for system time register.
→ Update the timers implemented in kernel (Increment register with count direction setting = 'Count up' and decrement register with count direction setting = "Count down").
→ Activate the periodic tasks, which are in the idle state.
→ Invoke the schedular and Schedule the tasks again based on the scheduling algorithm.
→ Delete all the terminated tasks and their associated data structures (TCB's)
→ Load the context for the first task in the ready queue. Due to the rescheduling, the ready task might be changed to a new one from the task, which was preempted by the 'Timer Interrupt' task.

Hard Real-Time :- Real-Time Operating Systems that strictly restricted to the timing constraints for a task is reffered as 'Hard-Real-Time' systems. A Hard real-time systems must be meet the deadlines for a task without any slippage. Air bag Control Systems & Anti-lock

brake systems (ABS) of vehicle are typical examples for Hard-Real Time Systems. Hard-Real time Systems doesnot implement the virtual memory for handling the memory. This eliminates the delay in swapping in and out the code corresponding to the task to & from the primary memory.

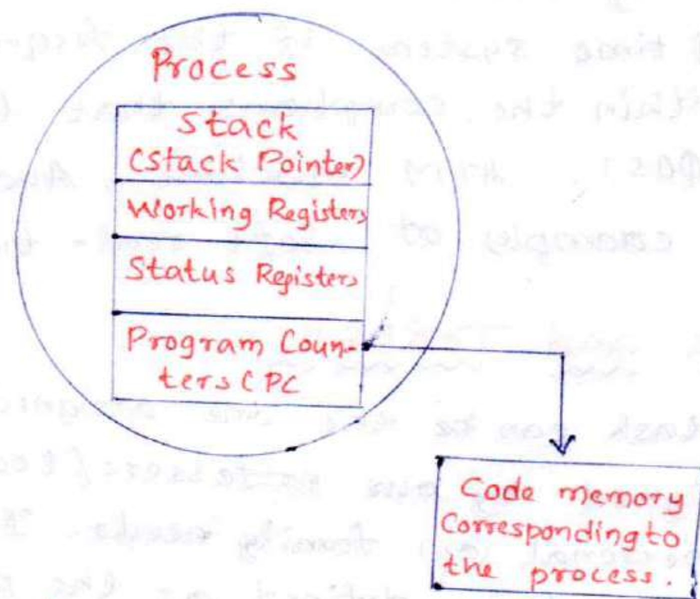
Soft Real-Time: Real-Time Operating Systems that doesnot garantee meeting dead lines, but offers the best effort to meet the dead lines are reffered as 'Soft Real-Time' systems. Missing dead lines for tasks are acceptables for soft real time system if the frequency of deadline missing is within the compliance that limit of Quality of Service (QOS). ATM machine, Audio-video playback systems are examples of Soft real-time Systems.

Tasks, Process and THREADS:-

The task can be the one assigned by our managers (or) the one assigned by our professors/teachers (or) the one related to our personal (or) family needs. In the operating system content, a task is defined as the program in execution and the related information maintained by the operating system for the program. Task is also known as 'Job' in the operating system content. A program (or) a part of it in execution is also called a 'Process'. The terms 'Task', 'Job' & 'Process' refers to same entity in the operating system context and most often they are used interchangeably.

Process: A 'Process' is a program, (or) part of it, in execution. Process is also known as an instance of a program in execution. Multiple instances of the same program can execute simultaniously. A process requires various system resources like CPU for executing the process, memory for storing the code corresponding to the process and associated variables, I/o devices for information exchange, etc. A process is sequential in executic

The structure of a Process:- The concept of "Process" leads to concurrent execution of tasks and there by the efficient utilization of the CPU and other system resources. Concurrent execution is achieved through the sharing of CPU among the processes. A process mimics a processor in propertie and holds a set of registers, process status, a Program Counter (PC) to point to the next executable instruction of the process, a stack for holding the local variables associated with the process and the code corresponding to the process.

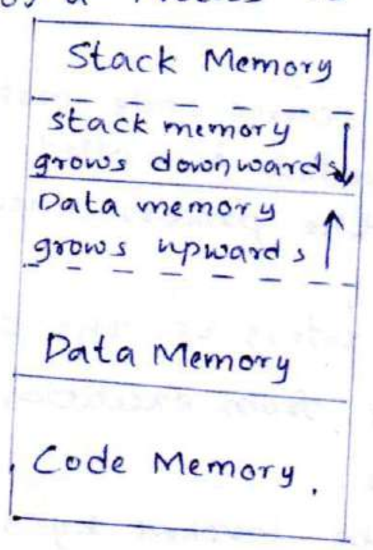


Structure of a Process

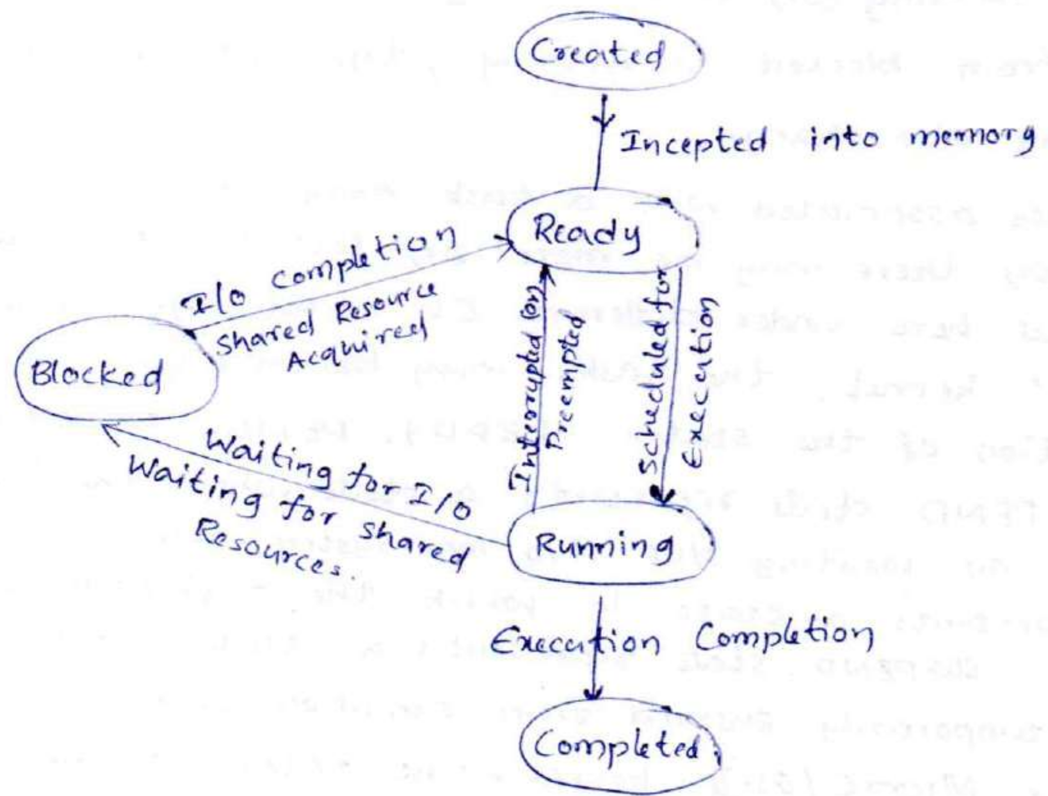
A process which inherits all the properties of the CPU can be considered as a virtual processor, awaiting its turn to have its properties switched into the physical processor. When the process gets its turn, its registers and the program counter register becomes mapped to the physical registers of the CPU. From a memory perspective, the memory occupied by the process is segregated into three regions, namely stack memory, Data mem and code memory.

The stack memory holds all temporary data such as variables local to the process. Data memory holds all global data for the process. On loading a process into the main memory, a specific area of memory is allocated for the process.

The stack memory usually starts at the highest memory addres from the memory area allocated for the process. for example, the memory map of the memory area allocated for the process is 2048 to 2100, the stack at memory starts at address 2100 and grows downwards to accommodate the variables local to the process. Memory organisation of a Process is as shown.



Process States and State Transition:- The creation of a process to its termination is not a single step operation. The process traverses through a series of states during its transition from the newly created state to the terminated state. The cycle through which a process changes its state from 'newly created' to 'execution completed' is known as 'Process Life Cycle'.



:Process States and state transition representation:-

The state at which a process is being Created is referred as 'Created State'. The operating system recognises a process in the 'created State' but no resources are allocated to the process.

The state, where a process is incepted into the memory & awaiting the processor time for execution, is known as "Ready State". At this stage the process is placed in the Ready list queue maintained by the OS.

The state where in the source code instructions corresponding to the process is being executed is called 'Running state'. Running state is the state at which the process execution happens.

Blocked State / wait state refers to the state where a running process is temporarily suspend from execution and does not have immediate access to resources.

The blocked state might be invoked by various conditions like: the process enters a wait state for an event to occur (or) waiting for getting access to a shared resource.

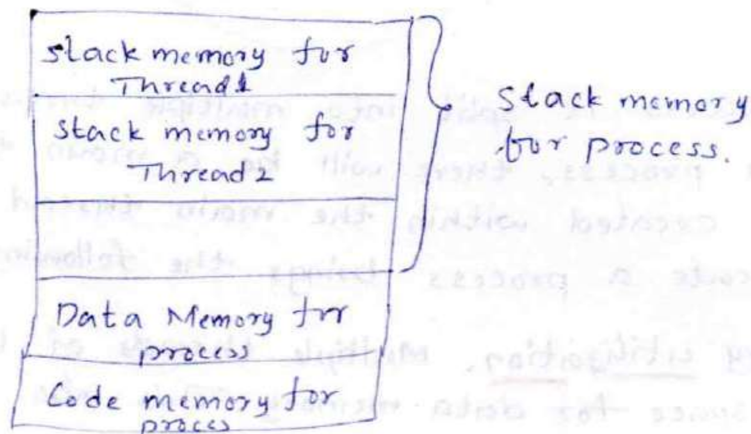
The state where the process completes it's execution is known as "Completed State".

The transition of a process from one state to another is known as 'State transition'. When a process changes it's state from ready to running (or) from running to blocked (or) from Ready to running or from blocked to running, the CPU allocation for the process may also change.

The states associated with a task may be known with a different name (or) there may be more (or) less no. of states than the one explained here under different OS kernel. For example under 'Vx Works' kernel, the tasks may be in either one or a specific combination of the states READY, PEND, DELAY and SUSPEND. The PEND state represents a state where the task/process is blocked on waiting for I/O (or) system resource. The DELAY state represents a state in which the task / process is sleeping and the SUSPEND state represents a state where a task / process is temporarily suspend from execution and not available for execution. Under MicroC/OS-II kernel, the tasks may be in one of the states DORMANT (Created state), READY, RUNNING, WAITING..

Threads :-

A thread is the premitive that can execute code. A thread is a single sequential flow of control within a process. Thread is also known as light wieght process. A process can have many threads of execution. Different threads, which are part of a process, share the same address space, meaning they share the data memory, code memory and heap memory area. Threads maintain their own thread status (CPU register values), Program Counter (PC) and stack.

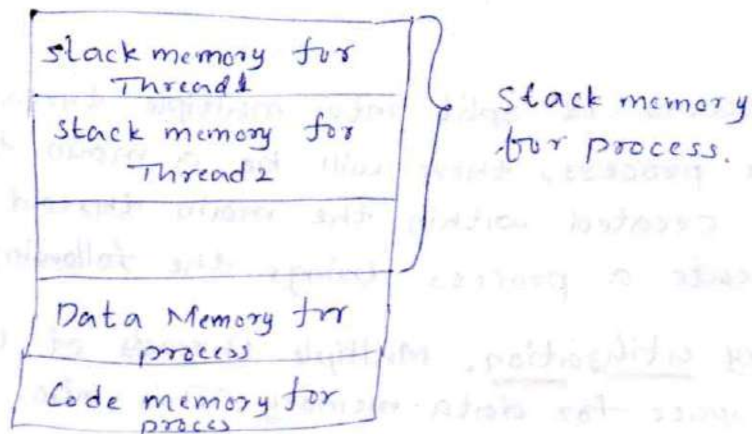


Multi Threading :- A process/task in embedded application may be complex (or) lengthy one and it may contain various suboperations like getting input from I/o devices connected to the processor, performing some internal calculations/operations, updating some I/o devices etc. If all the subfunctions of a task are executed in sequence, the CPU utilisation may not be efficient.

For example, if the process is waiting for a user input, the CPU enters the wait state for the event, and the process execution also enters a wait state. Instead of this, single sequential execution of the whole process, if the task/process is split into different threads carrying out the different subfunctionalies of the process, the CPU can be effectively utilised and when the thread corresponding to the I/O operations enters, the CPU can be effectively utilized and when the thread corresponding to the I/O operation enters the wait state, another threads which do not require the I/O events for their operation can be switched into execution. This leads to more speedy execution of the process & efficient utilization of the processor time and resources.

Threads :-

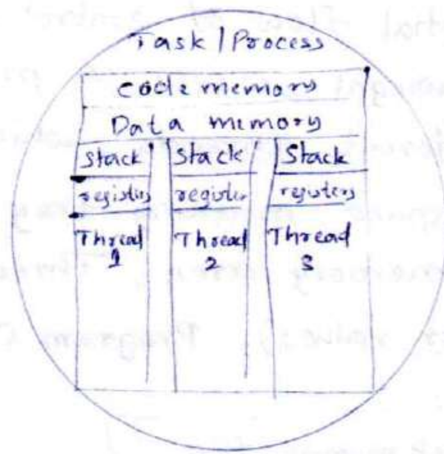
A thread is the premitive that can execute code. A thread is a single sequential flow of control within a process. Thread is also known as light wieght process. A process can have many threads of execution. Different threads, which are part of a process, share the same address space, meaning they share the data memory, code memory and heap memory area. Threads maintain their own thread status (CPU register values), Program Counter (PC) and stack.



Multi Threading :- A process/task in embedded application ma
be complex (or) lengthy one and it may contain various subopera
tions like getting input from I/o devices connected to the proce
ssor, performing some internal calculations/operations, updating some I/o devices etc. If all the subfunctions of a task are executed in sequence, the CPU utilisation may not be efficient.

For example, if the process is waiting for a user input, the CPU enters the wait state for the event, and the process execution also enters a wait state. Instead of this, single sequentia execution of the whole process, if the task/process is split into different threads carrying out the different subfunctionalies of the process, the CPU can be effectively utilised and when the thre corresponding to the I/o operations enters, the CPU can be effectively utilized and when the thread corresponding to the I/o operation enters the wait state, another threads which dono require the I/o events for their operation can be switched int execution. This leads to more speedy execution of the process & efficient utilization of the processor time and resources.

Process with multi threads



If the process is split into multiple threads, which executes a portion of the process, there will be a main thread and rest of the threads will be created within the main thread. Use of multiple threads to execute a process brings the following advantage.

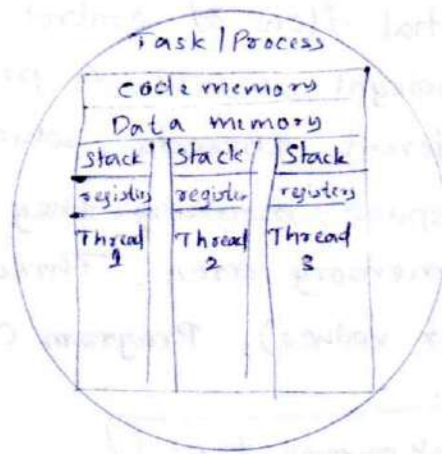
-> Better memory utilization. Multiple threads of the same process share the address space for data memory. This also reduces the complexity of inter thread communication since variables can be shared across the threads.

-> Speeds up the execution of the process.

-> Efficient CPU utilisation. The CPU is engaged all time.

THREAD	PROCESS
-> Thread is a single unit of execution and is part of process.	-> Process is a program in execution and contains one/more threads.
-> The thread does not have it's own data memory & heap memory. It shares with other threads of the same process.	-> Process has it's own code memory data memory and stack memory.
-> A thread cannot live independently it lives within the process.	-> A process contains atleast one thread.
-> Threads are very inexpensive to create.	-> Processes are very expensive to create. Involves many OS overheads
-> Context Switching is inexpensive and fast.	-> Context Switching is complex & involves lot of OS overhead and is comparatively slower.
-> If thread expires, its stack is reclaimed by the process.	-> If the process dies, the resources allocated to it are reclaimed by the OS and all the associated threads of process also dies.

Process with multithreads



If the process is split into multiple threads, which executes a portion of the process, there will be a main thread and rest of the threads will be created within the main thread. Use of multiple threads to execute a process brings the following advantage.

-> Better memory utilization. Multiple threads of the same process share the address space for data memory. This also reduces the complexity of inter thread communication since variables can be shared across the threads.

-> Speeds up the execution of the process.

-> Efficient CPU utilisation. The CPU is engaged all time.

THREAD	PROCESS
-> Thread is a single unit of execution and is part of process.	-> Process is a program in execution and contains one/more threads.
-> The thread does not have it's own data memory & heap memory. It shares with other threads of the same process.	-> Process has it's own code memory data memory and stack memory.
-> A thread cannot live independently it lives within the process.	-> A process contains atleast one thread.
-> Threads are very inexpensive to create.	-> Processes are very expensive to create. Involves many OS overheads
-> Context Switching is inexpensive and fast.	-> Context Switching is complex & involves lot of OS overhead and is comparatively slower.
-> If thread expires, its stack is reclaimed by the process.	-> If the process dies, the resources allocated to it are reclaimed by the OS and all the associated threads of process also dies.

Thread Standard: It deals with different standards available for thread creation and management. These standards utilized by the Operating systems for thread creation and management. It is a set of thread class libraries. The commonly available thread class libraries are

→ POSIX Threads:- POSIX stands for Portable Operating System Interface. POSIX.4 standard deals with the real time extensions and POSIX.4a standard deals with thread extensions. "pthreads" library defines the set of POSIX thread creation & management functions in 'C' language.

The primitive

```
int pthread_create (pthread_t * new_thread_ID, const pthread_attribute * attribute,
void * (*start function) (void *), void * arguments);
```

creates a new thread for running the function start-function.

Here
'pthread_t' is the handle to the newly created thread.
'pthread_attr_t' is data type for holding the thread attributes.
'start function' is the function the thread is going to execute

All the POSIX 'thread calls' returns an integer. A return value zero indicates the success of the call. It is always good to check the return value of each call.

→ Win32 Threads:- These are supported by various flavours of windows Operating Systems. The Win32 Application Programming Interface (Win32API) libraries provide the standard set of Win32 thread creation and management functions.

→ Java Threads:- These are supported by java programming language. The java thread class 'Thread' is defined in the package 'java.lang'. This package needs to be imported for using the thread creation functions supported by the Java thread

class. There are two ways of creating threads in Java: Either by extending the base 'Thread' class or by implementing an interface.

Extending the thread class allows inheriting the methods and variables of the parent class (Thread Class) only whereas interface allows a way to achieve the requirements for a set of classes.

Eg:

```
import java.lang.*;
public class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Hello from MyThread!");
    }
    public static main(String args[])
    {
        (new MyThread()).start();
    }
```

The above piece of code creates a new class Mythread by extending the base class thread.

The method start() moves the thread to a pool of threads waiting for their turn to be picked up for execution by the scheduler. The thread is said to be in the 'Ready' at this stage.

=> MyThread.yield(); will voluntarily give up the execution of the thread and thread is moved to the pool of threads waiting to get their turn for execution.

=> MyThread.sleep(100); sleep for 100 ms : forces the thread to sleep for the duration mentioned by the sleep call, i.e., thread enters into the 'SUSPEND' mode. Once sleep period is expired, the thread is moved to the pool of threads waiting for execution i.e, Ready state.

class. There are two ways of creating threads in Java: Either by extending the base 'Thread' class or by implementing an interface.

Extending the thread class allows inheriting the methods and variables of the parent class (Thread Class) only whereas interface allows a way to achieve the requirements for a set of classes.

Eg:

```
import java.lang.*;
public class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Hello from MyThread!");
    }
    public static main(String args[])
    {
        (new MyThread()).start();
    }
```

The above piece of code creates a new class MyThread by extending the base class thread.

The method start() moves the thread to a pool of threads waiting for their turn to be picked up for execution by the scheduler. The thread is said to be in the 'Ready' at this stage.

⇒ MyThread.yield(); will voluntarily give up the execution of the thread and thread is moved to the pool of threads waiting to get their turn for execution.

⇒ MyThread.sleep(100); sleep for 100 ms : forces the thread to sleep for the duration mentioned by the sleep call, i.e., thread enters into the 'SUSPEND' mode. Once sleep period is expired, the thread is moved to the pool of threads waiting for execution i.e, Ready state.

Multiprocessing and Multitasking:-

In the Operating System context Multiprocessing describes the ability to execute multiple processes simultaneously. Systems, which are capable of performing multiprocessing, are known as multiprocessor systems. Multiprocessor systems posses multiple CPU and can execute multiple processos simultaneously.

The ability of the operating system to have mw hold
multiple processes in memory and switch the processor (CPU) from
executing one process to another process is known as multitasking
Multitasking creates the illusion of multiple tasks executing in
parallel. Multitasking involves the switching of CPU from executing
one task to another.

We know that a process is identical to the physical processor in the sense it has own register set which mirrors the CPU registers, stack and Program Counter (PC). Hence, a 'process' is considered as a 'Virtual processor', awaiting its turn to have its properties switched into the physical processor. The switching of the virtual processor to physical processor is controlled by the scheduler of the OS kernel. Whenever a CPU switching happens, the current content of execution should be saved to retrieve it at a later point of time when the CPU executes the process, which is interrupted currently due to execution switching.

The act of switching CPU among the processes (or) changing the current execution context is known as Context Switching

The act of saving the current content which contains the context details for the currently running process at the time of CPU switching is known as 'Context Saving'.

The process of retrieving the saved context details for a process, which is going to be executed due to CPU switching is known as "Context retrieval".

Multitasking involves 'Context Switching', 'Context Saving' and context retrieval.

Multiprocessing and Multitasking:-

In the Operating System context Multiprocessing describes the ability to execute multiple processes simultaneously. Systems, which are capable of performing multiprocessing, are known as multiprocessor systems. Multiprocessor systems posses multiple CPU and can execute multiple processos simultaneously.

The ability of the operating system to have mw hold multiple processes in memory and switch the processor (CPU) from executing one process to another process is known as multitasking. Multitasking creates the illusion of multiple tasks executing in parallel. Multitasking involves the switching of CPU from executing one task to another.

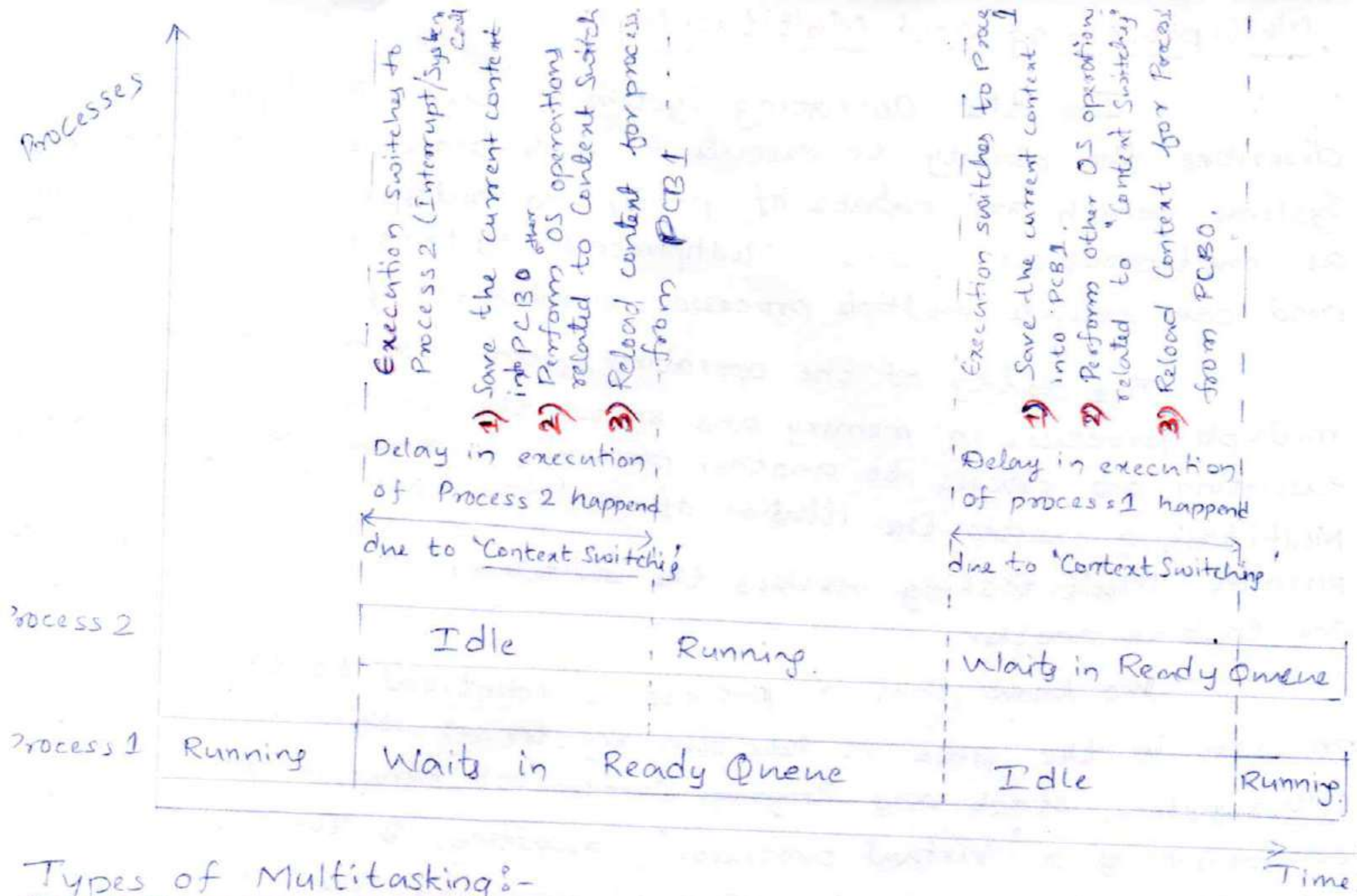
We know that a process is identical to the physical processor in the sense it has own register set which mirrors the CPU registers, stack and Program Counter (PC). Hence, a 'process' is considered as a 'Virtual processor', awaiting its turn to have its properties switched into the physical processor. The switching of the virtual processor to physical processor is controlled by the scheduler of the OS kernel. Whenever a CPU switching happens, the current content of execution should be saved to retrive it at a later point of time when the CPU executes the process, which is interrupted currently due to execution switching.

The act of switching CPU among the processes (or) changing the current execution context is known as Context Switching

The act of saving the current content which contains the context details for the currently running process at the time of CPU switching is known as 'Context Saving'.

The process of retrieving the saved context details for a process, which is going to be executed due to CPU switching is known as 'Context Retrieval'.

Multitasking involves 'Context Switching', 'Context Saving' and context retrieval.



Types of Multitasking:-

Multitasking involves the switching of execution among multiple tasks. Depending on switching act is implemented, multitaski can be classified into different types.

Co-Operative Multitasking:- Co-operative multitasking is the most primitive form of multitasking in which a task/process gets a chance to execute only when the currently executing task/proces voluntarily relinquishes the CPU. In this method, any task/proce can hold the CPU as much time as it wants. Since this type of implementation involves the mercy of the tasks each other for getting the CPU time for execution, it is known as co-operative multitasking. If the currently executing task is non-cooperative, the other tasks may have to wait for a long time to get the CPU.

Preemptive Multitasking:- It ensures that every task/process gets a chance to execute. When and how much time a process gets is dependent on the implementation of the preemptive scheduling. The preemption of task may be based on time slots (or) task/process priority.

Non - Preemptive Multitasking: In non-preemptive multitasking, the process/task, which is currently given the CPU time is allowed to execute until it terminates (or) enters the 'Blocked/Wait' state, waiting for an I/o (or) system resource The co-operative and non-preemptive multi tasking differs in their behaviour when they are in the 'Blocked/Wait' State. In co-operative multitasking, the currently executing process/task need not relinquish the CPU when it enters the "Blocked/wait" state, waiting for an I/o, or a shared resource access (or) an event to occur whereas in non-preemptive multitasking the currently executing task relinquishes the CPU when it waits ~~from~~ for an I/o (or) system resource (or) an event to occur.

Task Scheduling :-

Multitasking involves the execution switching among the different tasks. There should be some mechanism in place to share the CPU among the different tasks and to decide which process/task is to be executed at a given point of time. Determining which task/process is to be executed at a given point of time is known as task/process scheduling.

The process scheduling decision may take place when a process switches its state to

1. 'Ready' state from 'Running' State.
2. 'Blocked/Wait' State from 'Running' State.
3. 'Ready' State from 'Blocked/wait' State.
4. 'Completed' state.

The selection of a scheduling criterion/algorithm should consider the following factors.

⇒ CPU utilisation :- The scheduling algorithm should always make the CPU utilisation high. CPU utilisation is a direct measure of how much percentage of the CPU is being utilised.

⇒ Throughput :- This gives an indication of the no. of processe executed per unit of time. The throughput for a good scheduler should always be higher.

⇒ Turnaround Time :- It is the amount of time taken by a process for completing its execution. It includes the time spent by the process for waiting for the main memory, time spent in the ready queue, time spent on completing the I/O operations, and the time spent in execution. The turnaround time should be a minimal for a good scheduling algorithm.

⇒ Waiting Time :- It is the amount of time spent by a process in the 'Ready' queue waiting to get the CPU time for execution. The waiting time should be minimal for a good scheduling algorithm.

Response Time :- It is the time elapsed between the submission of a process and the first response. For a good scheduling algorithm, the response time should be as least as possible.

The Operating System maintains various queues in connection with the CPU scheduling, and a process passes through these queues during the course of its admittance to execute completion.

The various queues maintained by OS in association wit[h] CPU scheduling are.

Job Queue :- Job queue contains all the processes in the System.

Ready Queue :- Contains all the processes, which are ready for execution and waiting for CPU to get thei[r] turn for execution. The ready queue is empty when there is no process ready for running.

Device Queue :- Contains the set of processes, which are waiting for an I/o device.

⇒ Non-preemptive Scheduling :-

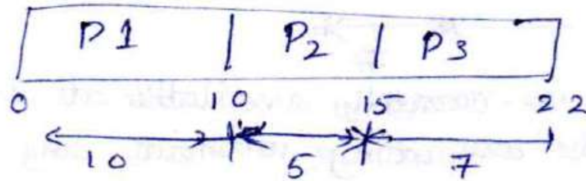
In this scheduling type, the currently executing task process is allowed to run until it terminates (or) enters the 'Wait' state waiting for an I/o (or) system resource.

1) First-Come-First-Served (FCFS) / FIFO scheduling :-

FCFS/FIFO scheduling algorithm allocates CPU time to the processes based on the order in which they enter the 'Ready' queue. The first entered process is serviced first. It is same as any real world application where queue systems are used; eg, Ticketing reservation system where people need to stand in a queue and the first person standing in the queue is serviced first.

Example-1 :- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 ms respectively enters the ready queue together in the order P1, P2, P3. Calculate the waiting time and Turn Around Time (TAT) for each process & the average waiting time and Turn around time.

The Sequence of execution of processes by the CPU is represented as



Waiting time for P1 = 0ms (P1 starts executing first)
Waiting time for P2 = 10ms. (P2 starts executing after completing P1)
Waiting time for P3 = 15ms. (P3 starts executing after completing P1 & P2)

Average Waiting time = $\frac{\text{Waiting time for all processes}}{\text{No. of Processes.}}$

$= \frac{P_1 + P_2 + P_3}{3}$

$= \frac{0+10+15}{3} = \frac{25}{3} = 8.33ms.$

Turn Around time for P1 = 10ms (Time spent in Ready Queue + Execution time)
" P_2 = 15ms
" P_3 = 22ms.

Average turn around time = $\frac{10+15+22}{3} = \frac{47}{3} = 15.66ms.$

Average execution time = $10+5+7/3 = \frac{22}{3} = 7.33$

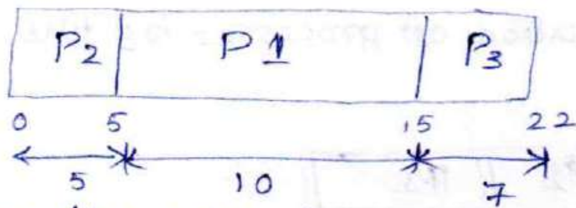
Avg. TAT (turn around time) = Avg. waiting time + Avg. execution time

$= 8.33 + 7.33$

$= 15.66ms.$

Example 2 :- Calculate the waiting time and Turnaround tim (TAT) for the above example if the process enters the 'Ready' quene.

The sequence of execution of the processes by the CPU is represented as



Assuming the CPU is readily available at the time of arrival of P_2, P2 starts executing without any waiting in the Ready Quene. Hence the waiting time for P2 is zero. The waitin time for all processes is given as

Waiting time for P2 = 0ms. (P_2 starts executing first)

Waiting time for P1 = 5ms (P_1 starts executing after completing P_2)

Waiting time for P3 = 15ms (P_3 starts executing after completing P2 & P1).

$$\text{Average Waiting Time} = \frac{\text{Waiting time all processes}}{\text{No. of Processes}}$$

$$= \frac{0+5+15}{3} = \frac{20}{3} = 6.66\text{ ms}.$$

Turnaround Time (TAT) for P_2 = 5ms (Time spent in Ready Queue + Execution Time).

Turnaround Time (TAT) for P_1 = 15ms (")

Turnaround Time (TAT) for P_3 = 22 ms (")

$$\text{Average Turn Around Time} = \frac{\text{TAT for all processes}}{\text{No. of processes}}$$

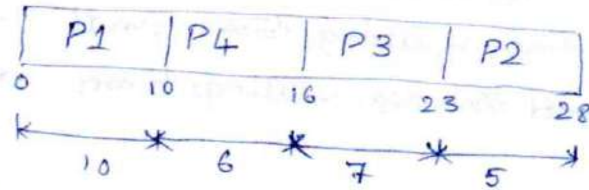
$$= \frac{5+15+22}{3} = \frac{42}{3} = 14\text{ ms}.$$

From the above two examples it is clear that the average waiting time and turn around Time improve if the process with shortest execution completion time is scheduled first.

The major drawback of FCFS algorithm is that it favours monopoly of process. A process, which does not contain any I/O operation, continues its execution until it finishes its task. The average waiting time is not minimal for FCFS scheduling algorithm.

Last - Come - First Served (LCFS)/LIFO Scheduling :- The last come first served (LCFS) scheduling algorithm also allocates CPU time to the processes based on the order in which they are entered in the 'Ready' queue. The last entered process is serviced first.

Example :-



Waiting Time for P1 = 0ms (P1 starts executing first)

Waiting Time for P4 = 5 ms (P4 starts executing after completing P1. But P4 arrived after 5ms of execution of P1.
∴ it's waiting Time = Execution start time − Arrival time = 10 − 5 = 5)

Waiting Time for P_3 = 16ms (P3 starts executing after completing P1 & P_4).

Waiting Time for P_2 = 23 ms (P2 starts executing after completing P_1, P_4 & P_3)

Average Waiting Time = Waiting time for all processes / No. of processes

$$= \frac{0+5+16+23}{4} = \frac{44}{4} = 11 \text{ ms}.$$

TAT for P1 = 10ms

" P4 = 11ms

" P3 = 23ms

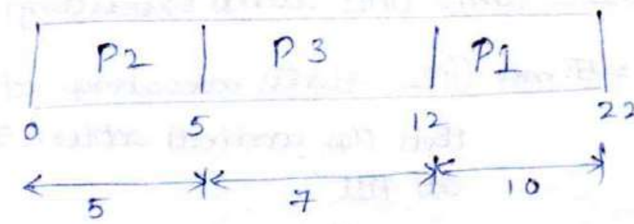
" P2 = 28ms

Average Turnaround Time = TAT for $(P_1 + P_2 + P_3 + P_2)/4$

$$= (10+11+23+28)/4 = \frac{72}{4} = 18 \text{ ms}.$$

Shortest Job First (SJF) Scheduling :- SJF scheduling algorithm sorts the 'Ready' queue each time a process relinquishes the CPU to pick process with shortest estimated completion/run time. In SJF, the process with the shortest estimated run time is scheduled first, followed by the next shortest process, and so on.

Example-1 :- Three processes with process ID's P1, P2, P3 with estimated completion time 10, 5, 7 ms respectively enters the ready queue together. Calculate the waiting time and Turn Around Time (TAT) for each process and Average waiting time and Turn Around Time in SJF algorithm.



Waiting Time for P2 = 0 ms (P2 starts executing first)

Waiting Time for P3 = 5 ms (P3 starts executing after completing P2)

" " " P1 = 12 ms (P1 " " " P_2 & P_3)

Average waiting time = Waiting time for (P2+P3+P1) / 3

= 0+5+12/3 = 17/3. = 5.66 ms.

Turn Around Time (TAT) for P2 = 5 ms

" (TAT) " P3 = 12 ms

" " " " P1 = 22 ms.

Average Turn Around Time = $\frac{5+12+22}{3} = \frac{39}{3} = 13$ ms.

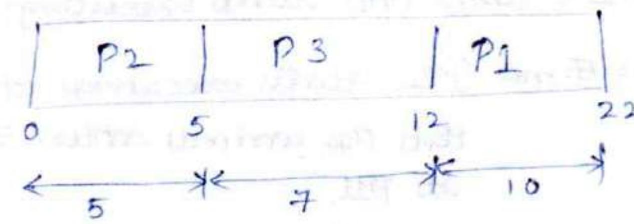
The average Execution Time = $(10+5+7)/3 = \frac{22}{3} = 7.33$.

Average Turn Around Time = 5.66 + 7.33

= 13 ms.

Shortest Job First (SJF) Scheduling :- SJF scheduling algorithm sorts the 'Ready' queue each time a process relinquishes the CPU to pick process with shortest estimated completion / run time. In SJF, the process with the shortest estimated run time is scheduled first, followed by the next shortest process, and so on.

Example-1 :- Three processes with process ID's P1, P2, P3 with estimated completion time 10, 5, 7 ms respectively enters the ready queue together. Calculate the waiting time and Turn Around Time (TAT) for each process and Average waiting time and Turn Around Time in SJF algorithm.



Waiting Time for P2 = 0 ms (P2 starts executing first)

Waiting Time for P3 = 5 ms (P3 starts executing after completing P2)

" " " P1 = 12 ms (P1 " " " P_2 & P_3)

Average waiting time = Waiting time for (P2 + P3 + P1) / 3

= 0+5+12 / 3 = 17/3 = 5.66 ms.

Turn Around Time (TAT) for P2 = 5 ms

" " (TAT) " P3 = 12 ms

" " " " P1 = 22 ms.

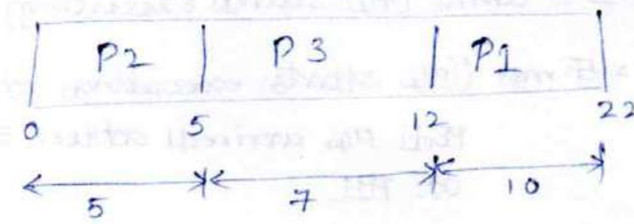
Average Turn Around Time = $\frac{5+12+22}{3} = \frac{39}{3} = 13$ ms.

The average Execution Time = $(10+5+7)/3 = \frac{22}{3} = 7.33$.

Average Turn Around Time = 5.66 + 7.33
= 13 ms.

Shortest Job First (SJF) Scheduling :- SJF scheduling algorithm sorts the 'Ready' queue each time a process relinquishes the CPU to pick process with shortest estimated completion / run time. In SJF, the process with the shortest estimated run time is scheduled first, followed by the next shortest process, and so on.

Example-1 :- Three processes with process ID's P1, P2, P3 with estimated completion time 10, 5, 7 ms respectively enters the ready queue together. Calculate the waiting time and Turn Around Time (TAT) for each process and Average waiting time and Turn Around Time in SJF algorithm.



Waiting Time for P2 = 0 ms (P2 starts executing first)

Waiting Time for P3 = 5 ms (P3 starts executing after completing P2)

" " " P1 = 12 ms (P1 " " " P_2 & P_3)

Average waiting time = Waiting time for (P2+P3+P1) / 3

= 0+5+12 / 3 = 17/3 = 5.66 ms.

Turn Around Time (TAT) for P2 = 5 ms

" (TAT) " P3 = 12 ms

" " " " P1 = 22 ms.

Average Turn Around Time = $\frac{5+12+22}{3} = \frac{39}{3} = 13$ ms.

The average Execution Time = $(10+5+7)/3 = \frac{22}{3} = 7.33$.

Average Turn Around Time = 5.66 + 7.33
= 13 ms.

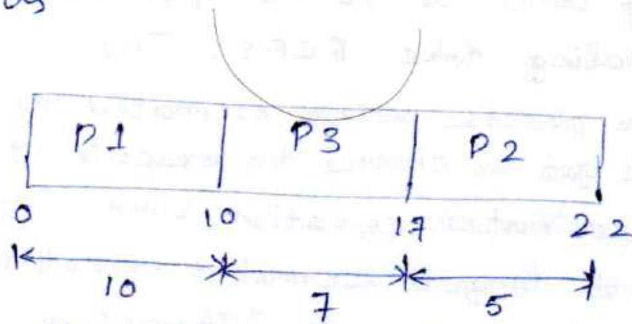
The average waiting time for a given set of process is minimal in SJF scheduling and so it is optimal compared to other non-preemptive scheduling like FCFS. The major drawback of SJF algorithm is that a process whose estimated execution completion time is high may not get a chance to execute if more and more processes with least estimated execution time enters the 'Ready queue before the process with longest estimated execution time startee its execution. This condition is known as 'Starvation'. Another drawback of SJF is that it is difficult to know in advance the next shortest process in the 'Ready' queue for scheduling since new processe with diff estimated execution time keep entering the Ready queue at any point of time.

Priority Based Scheduling :- The turn around time (TAT) and waiting time for processes in non-preemptive scheduling varies with the type of scheduling algorithm. Priority based non-preemptive scheduling algorithm ensures that a process with high priority is serviced atth earliest compared to other low priority processes in the 'Ready' queue. The priority of the task/process can be indicated through various mechanisms. The shortest job First (SJF) algorithm can be viewed as priority based scheduling where each task is prioritised in the order of the time required to complete the task. The lower the time required for completing a process the higher is its priority in SJF algorithm. Another way of priority assigning is associating a priority to the task/process at the time of creation of the task/process. The priority is a no. of ranging from 0 to the maximum priority supported by the OS. The maximum level of priority is OS dependen For example Windows CE supports 256 levels of priority (0 to 255 priority numbers). For Windows CE OS a priority number 0 indicates the highest priority and 255 indicates the lowest priority.

Example 1 :- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 ms and priorities 0, 3, 2 (0- highest priority 3- lowest priority) respectively enters the ready queue together. Calculate the waiting time Turn Around Time (TAT) for each process & the Avg. Waiting time Turn Around Time in priority based scheduling algorithm.

The scheduler sorts the 'Ready' queue based on the priority and schedules the process with the highest priority and next highest priority as second, and so on.

The order in which the processes are scheduled for execution is represented as



The waiting time for all the processes are given as

Waiting time for P1 = 0ms (P1 starts executing first)

Waiting time for P3 = 10ms (P3 starts executing after completing P1)

Waiting time for P2 = 17ms (P2 starts executing after completing P1 & P3)

Average Waiting time = $\frac{\text{Waiting time for all processes}}{\text{No. of processes}}$

$= \frac{0+10+17}{3} = \frac{27}{3} = 9ms$.

Turn Around Time (TAT) for P1 = 10ms

" " P2 = 17ms

" " P3 = 22ms

Average TAT = $\frac{\text{Turn Around Time for all processes}}{\text{No. of Processes}}$

$= \frac{10+17+22}{3} = \frac{49}{3} = 16.33ms$

Similar to SJF scheduling algorithm, non-preemptive priority based algorithm also possess the draw back of 'Starvation' where a process whose priority is low may not get a chance to execute if more and more processes with higher priorities enters 'Ready' queue before the process with lower priority started its execution.

Pre-Emptive Scheduling:- Which implements preemptive multitasking model. In preemptive scheduling, every task in the 'Ready' queue gets chance to execute. When and how often each process gets a chance to execute is dependent on the type of preemptive scheduling algorithm used for scheduling the processes. In this kind of scheduling, the scheduler can preempt (stop temporarily) the currently executing task/process and select another task from the 'Ready' queue for execution. When to pre-empt a task and which task is to be picked up from the 'Ready' queue for execution after preempting the current task is purely dependent on the scheduling algorithm. The act of moving a 'Running' process/task into the 'Ready' queue by the scheduler, without the processes requesting for it is known as 'Preemption'.

The two important approaches adopted in preemptive scheduling are time based pre-emption and priority based pre-emption.

Pre-Emptive SJF scheduling / Shortest Remaining Time (SRT):- The non-preemptive SJF scheduling algorithm sorts the 'Ready' queue only after completing the execution of the current process (or) when the process enters 'Wait' state, whereas the preemptive SJF algorithm sorts the 'Ready' queue when a new process enters the 'Ready' queue and checks whether the execution time of the new process is shorter than the remaining of the total estimated time for the currently executing process. If the execution time of the new process is less, the currently executing process is preempted & the new process is scheduled for execution. Thus preemptive SJF scheduling always compares the execution completion time of a new process entered the 'Ready' queue with the remaining time for completion of the currently executing process and schedules the process with shortest remaining time for execution. Preemptive SJF scheduling is also known as Shortest Remaining Time (SRT) Scheduling.

Example-1:- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 ms respectively enters the ready queue together. A new process P4 with estimated completion time 2ms enters the 'Ready' queue after 2ms.

Pre-Emptive Scheduling:- Which implements preemptive multitasking model. In preemptive scheduling, every task in the 'Ready' queue gets chance to execute. When and how often each process gets a chance to execute is dependent on the type of preemptive scheduling algorithm used for scheduling the processes. In this kind of scheduling, the scheduler can preempt (stop temporarily) the currently executing task/process and select another task from the 'Ready' queue for execution. When to pre-empt a task and which task is to be picked up from the 'Ready' queue for execution after preempting the current task purely dependent on the scheduling algorithm. The act of moving a 'Running' process/task into the 'Ready' queue by the scheduler, without the processes requesting for it is known as 'Preemption'.

The two important approaches adopted in preemptive scheduling are time based pre-emption and priority based pre-emption.

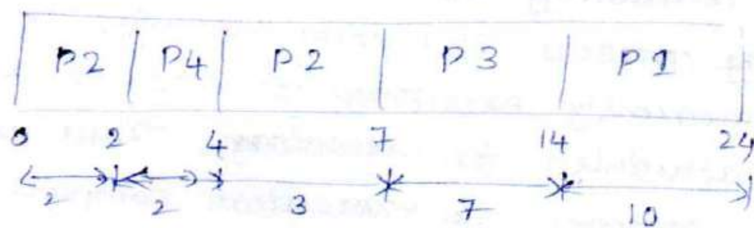
Pre-Emptive SJF scheduling / Shortest Remaining Time (SRT):- The non-preemptive SJF scheduling algorithm sorts the 'Ready' queue only after completing the execution of the current process (or) when the process enters 'Wait' state, whereas the preemptive SJF algorithm sorts the 'Ready' queue when a new process enters the 'Ready' queue and checks whether the execution time of the new process is shorter than the remaining of the total estimated time for the currently executing process. If the execution time of the new process is less, the currently executing process is preempted & the new process is scheduled for execution. Thus preemptive SJF scheduling always compares the execution completion time of a new process entered the 'Ready' queue with the remaining time for completion of the currently executing process and schedules the process with shortest remaining time for execution. Preemptive SJF scheduling is also known as Shortest Remaining Time (SRT) Scheduling.

Example-1:- Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 ms respectively enters the ready queue together. A new process P4 with estimated completion time 2ms enters the 'Ready' queue after 2ms.

At the begining, there are only three processes available in the 'Ready' queue and SRT scheduler picks up the process with the shortest remaining time for execution completion for scheduling.

Now process P4 with estimated execution completion time 2ms enters the 'Ready' queue after 2ms of start of execution of P2. Since the SRT algorithm is preemptive, the remaining time for completion of process P2 is checked with the remaining time for completion of process P4. The remaining time for completion of P2 is 3ms which is greater than that of the remaining time for completion of the newly entered process P4 (2ms). Hence P2 is preempted and P4 is scheduled for execution. P4 continues its execution to finish since there is no new process entered in the 'Ready' queue during its execution. After 2ms of scheduling P4 terminates and now the scheduler again sorts the sorts the 'Ready' queue based on the remaining time for completion of the processes present in the 'Ready' queue. Since the remaining time for P2 (3ms), which is preempted by P4 is less than that of the remaining time for other processes in the 'Ready' queue, P2 is scheduled for execution. Due to the arrival of the process P4 with execution time 2ms, the 'Ready' queue is re-sorted in the order of P2, P4, P2, P3, P1. A begining it was P2, P3, P1.

The execution sequence now changes as per the following diagram.



The waiting time for all the processes are given as

→ Waiting time for P2 = 0 + (4 - 2) = 2ms.
(P₂ starts executing first and is interrupted by P4 and has to wait till the completion of P4 to get the next CPU slot).

→ Waiting time for P4 = 0ms (P4 starts executing by preempting P2 since the execution time for completion of P4 (2ms) is less than that of the remaining time for execution completion of P2 (Here it is 3ms)).

→ Waiting time for P3 = 7ms (P3 starts executing after completing P4 & P2).

→ Waiting time for P1 = 14ms (P1 starts executing after completing P4, P2 & P3)

$$\text{Average Waiting time} = \frac{\text{Waiting time for all the processes}}{\text{No. of Processes}}$$

$$= (0+2+7+14)/4 = \frac{23}{4} = 5.75\text{ ms.}$$

Turn Around Time (TAT) for P2 = 7ms.

" " " " P4 = 2ms

" " " " P3 = 14ms.

" " " " P1 = 24ms.

$$\text{Average Turn Around Time} = \frac{7+2+14+24}{4}$$

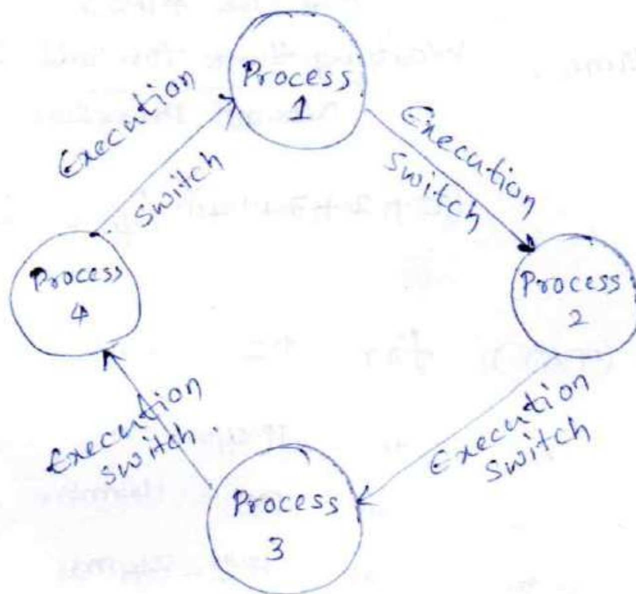
$$= \frac{47}{4} = 11.75\text{ ms.}$$

Round Robin (RR) Scheduling:-

The term Round Robin is very popular among the sports and games activities. In the 'Round Robin' league each team in a group gets an equal chance to play against the rest of the teams in the same group whereas in the 'Knockout' league the lossing team in a match moves out of the tournament.

In the process scheduling content also, 'Round Robin' brings the same message "Equal Chance to all". In Round Robin scheduling, each process in the 'Ready' queue is executed for a pre-defined time slot. The execution starts with picking up the first process in the 'Ready' queue. It is executed for a pre-defined time and when the pre-defined time elapses (or) the process completes, the next process in the 'Ready' queue is selected for execution. This is repeated for all the processes in the 'Ready' queue. Once each process in the 'Ready' queue again for execution. The sequence is repeated. This reveals that the Round Robin scheduling is similar to the FCFS scheduling and the only difference is that a time slice based preemption is added to switch execution b/w the processes in the 'Ready' queue. The 'Ready' queue can be considered as a circular queue in which the scheduler picks up the first process for execution

and moves to the next till the end of the queue and then comes back to the begining of the queue to pick up the first process



∴ Round Robin Scheduling :-

The time slice is provided by the 'timer tick' feature of the time management unit of the OS kernel. time Slice is kernel dependent and it varies in the order of a few microseconds to milliseconds. Certain OS kernels may allow the time slice as user configurable. Round Robin scheduling ensures that every process gets a fixed amount of CPU time for execution. When the process gets its fixed time for execution is determined by the FCFS policy if a process terminates before the elapse of the time slice, the process releases the CPU voluntarily and the next process in the queue is scheduled for execution by the scheduler, The implementation of RR scheduling is kernel dependent.

Priority Based Scheduling :-

Priority based preemptive scheduling algorithm is same as that of the non-preemptive priority based scheduling except for the switching of execution between tasks. In preemptive sheduling, any high priority process entering the 'Ready' queue is immediately scheduled for execution whereas in the non-preemptive scheduling any high priority process entering the 'Ready' queue is scheduled only after the currently executing process completes its execution only when it voluntarily relinquishes the CPU. The priority of a task/process in preemptive scheduling is indicated in the same way as that of the mechanism adopted for non-preemptive multi-tasking.

UNIT - V

Task communication

In a multitasking system, multiple tasks/processes run concurrently and each process may or may not interact between. Based on the degree of interaction, the processes running on an OS are classified as

1. Co operating processes
2. Competing processes.

1. Co operating processes -

In the co-operating interaction model one process requires the inputs from other processes to complete its execution.

2. Competing processes -

The competing process do not share anything among themselves but they share the system resources. The competing processes compete for the system resources such as file, display device etc.

Co operating processes exchanges information and communicate through the following methods.

1. Co operation through sharing

The cooperating process exchanges data through some shared resources.

2. Co operation through communication

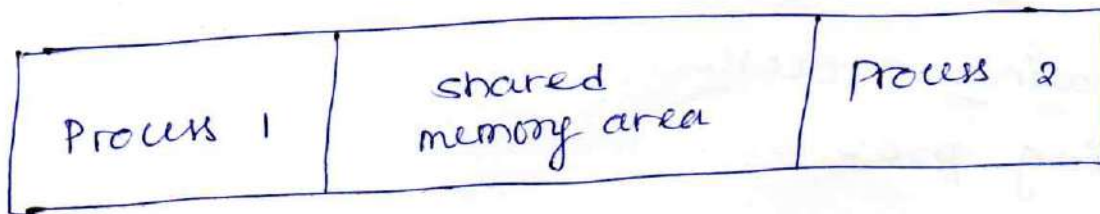
No data is shared between the processes. but they communicate for synchronisation.

The mechanism through which processes/tasks communicate each other is known as inter process/task communication (IPC).

Inter process communication is essential for Process co ordination.

Some of the important IPC mechanisms adopted by ~~process~~ various ~~are~~ Kernels ~~dependent~~. are

1. Shared Memory



Processes share some area of the memory to communicate among them. Information to be communicated by the process is written to the shared memory area. other processes which require this information can read the same from the shared memory area. EX : Notice board.

The implementation of shared memory concept is kernel dependent. Different mechanisms are adopted by different Kernels for implementing this. few of them are

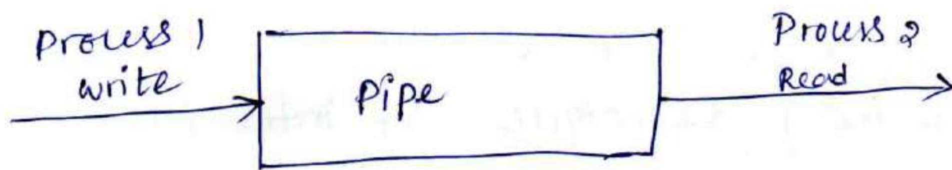
1. Pipes -

Pipe is a section of the shared memory used by processes for communicating. Pipes follow the client server architecture.

A process which creates a pipe is known as a pipe server and a process which connects to a pipe is known as pipe client.

It has two conceptual ends. It can be unidirectional or bidirectional.

Unidirectional pipe → allows the process connecting at one end of the pipe to write to the pipe and the process connected at the other end of the pipe to read the data.



The implementation of pipes is also OS dependent. Microsoft windows desktop operating systems support two ~~wau~~ types of pipes for inter process communication. They are 1. Anonymous pipes.
2. Named pipes.

Anonymous pipes are unnamed, unidirectional pipes used for data transfer between two processes.

Named pipes - It is a unidirectional or bidirectional pipe for data exchange between processes.
Like anonymous pipes, the process which creates the named pipe is known as pipe server. A process which connects to the named pipe is known as pipe client.

with named pipes, any process can act as both client and server allowing point to point communi cation.

2. memory mapped objects :-
It is a shared memory technique adopted by certain Real-time operating systems for allocating a shared block of memory which can be accessed by multiple process simultaneously.
In this approach a mapping object is created and physical storage for it is reserved and committed. A process can map the entire committed physical area or a block of it to its virtual address space. All read and write operations to this virtual address space by a processes can map the physical memory area of the mapped object

Windows CE 5.0 RTOS uses the memory mapped object based shared memory technique for inter process communicat?

Message passing :-

Message passing is an asynchronous information exchange mechanism used for inter process/Thread communication.

The major difference between shared memory and message passing technique is that, through shared memory lots of data can be shared whereas only limited amount of info/data is passed through message passing.

Also message passing is relatively fast and free from the synchronisation overheads compared to shared memory.

Based on the message passing operation between the process, Message passing is classified into 3 types:

1) Message Queue
2) Mail box
3) Signalling.

Message queue :- usually the process which wants to talk to another process post the message to a first in first - out queue called message queue. which stores the messages temporarily in a system defined memory object to pass it to the designed process.

Messages are sent and received through Send and received Methods. The messages are exchanged through a message queue. the implementation of the message queue send and received methods are OS kernel dependent. The windows xp OS kernel maintains a singel message queue and one process / thread

A thread which wants to communicate with another thread post the messages to the system message queue.

The kernel picks up the message from the system message queue one at a time examine the messa for finding the destination thread and then posts th message to the message queue of the corresponding thread.

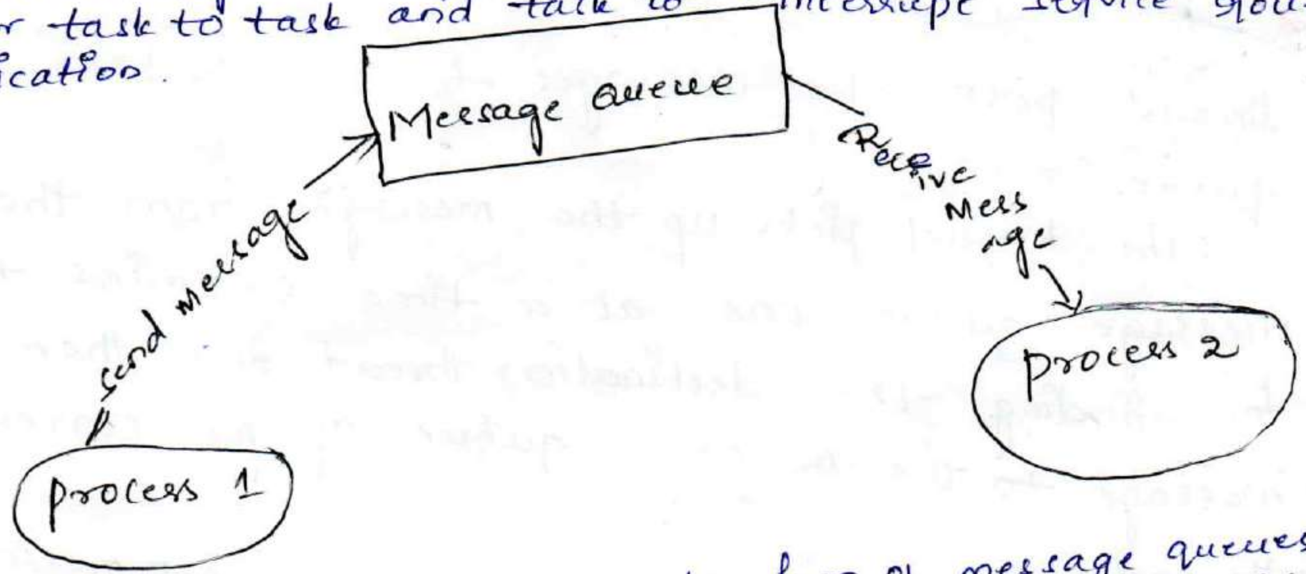
For posting a message to a threads message queue the kernel fills a message structure MSG. and cop it to the message queue of the thread. The messa structure MSG contains the handle of the process/ thread for which the message is intended, the me parameters, the time at which the message is pos

A thread can simply post a message through anot thread and can continue its operation or it may wa for a response from the thread to which the messa is posted.

The messaging mechanism is classified into synchro and asynchronous based on the behaviour of the mes posting thread.

In asynchronous messaging, the message posting thread just posts the message to the queue and it w not wait for an acceptance from the thread to wh the message is posted, where as in synchronous messa the thread which posts a message enters waiting sta and waits for the message results from the thread t which the message is posted. Message queue is the prima interthread communication mechanism under Vx works kernel message q

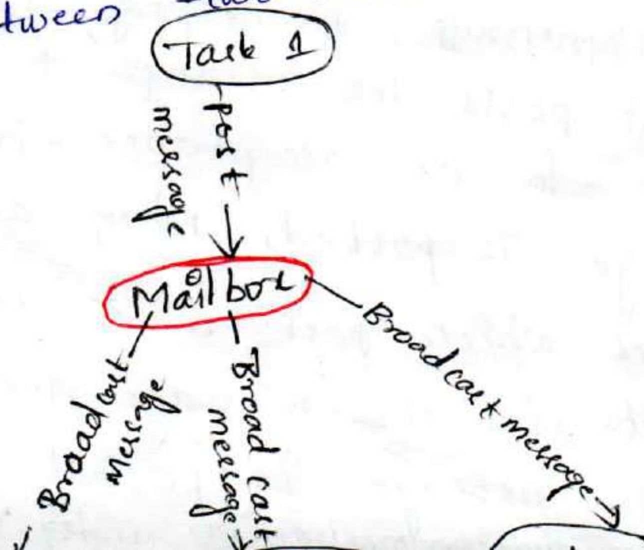
Two way messaging between tasks can be implemented using one message queue for incomming and other for out going messages. this mechanism is used for task to task and task to interrupt service routine communication.



Mail Box :- Mail box is an alternate form of message queues, and it is used in certain real time operating system for IPC. Mail box technique for IPC in RTOS is usually used for one way messaging.

The task/thread which wants to send a message to other tasks threads creats a mail box for posting the messages. The thread which creats the mail box is known as mail box server and the threads which subscribe to the mail box are known as mail box clients.

Mailbox server posts messages to the mailbox and notifies it to the clients which are subscribed to the mail box the clients reads the message from the mailbox and receiving the notification Mailbox and message queues are same in functionality the only difference is in the no. of messages supported by them. Both of them are used for passing data in the form of messages from one task to another task Mailbox is used for exchanging a single message between two tasks or between and interrupt routine and a task



3. Signalling :-

Signalling is a primitive way of communication between processes/threads. Signals are used for asynchronus notifications, whereas one process/thread fires a signal, indicating the occurence of scenar which the other process/thread is waiting.

Signals are not qued and they do not carry any data. The communication mechanisms used in RTX51 Tiny OS is an example for signalling. The OS_send_signal kernel call under RTX-51 sends signal from one task to specified task similarly. os-wait kernel call waits for a specified signal.

Remote Procedure call (RPC) and Sockets

Remote procedure call or RPC is the inter process communication mechanism used by a process to call a procedure of another process running on the same CPU or on a different CPU which is interconnecte in a network.

In the object oriented terminology RPC is also known as Remote invocation or Remote method invocatio

RPC is mainly used for distributed applications like client - server applications. with RPC it is possible to communicate over a heterogeneous network. (i.e N/w where client and server applications are running on different operating systems).

The cpu/process containing the procedure which needs to be invoked remotely is known as server.

The cpu/process which initiates an RPC request is known as client.

3. Signalling :-

Signalling is a primitive way of communication between processes/threads. Signals are used for asynchronus notifications, whereas one process/thread fires a signal, indicating the occurence of scenar which the other process/thread is waiting.

Signals are not qued and they do not carry any data. The communication mechanisms used in RTX51 Tiny OS is an example for signalling. The OS_send_signal kernel call under RTX-51 sends signal from one task to specified task similarly. os_wait kernel call waits for a specified signal.

Remote Procedure Call (RPC) and Sockets

Remote procedure call or RPC is the inter process communication mechanism used by a process to call a procedure of another process running on the same CPU or on a different CPU which is interconnecte in a network.

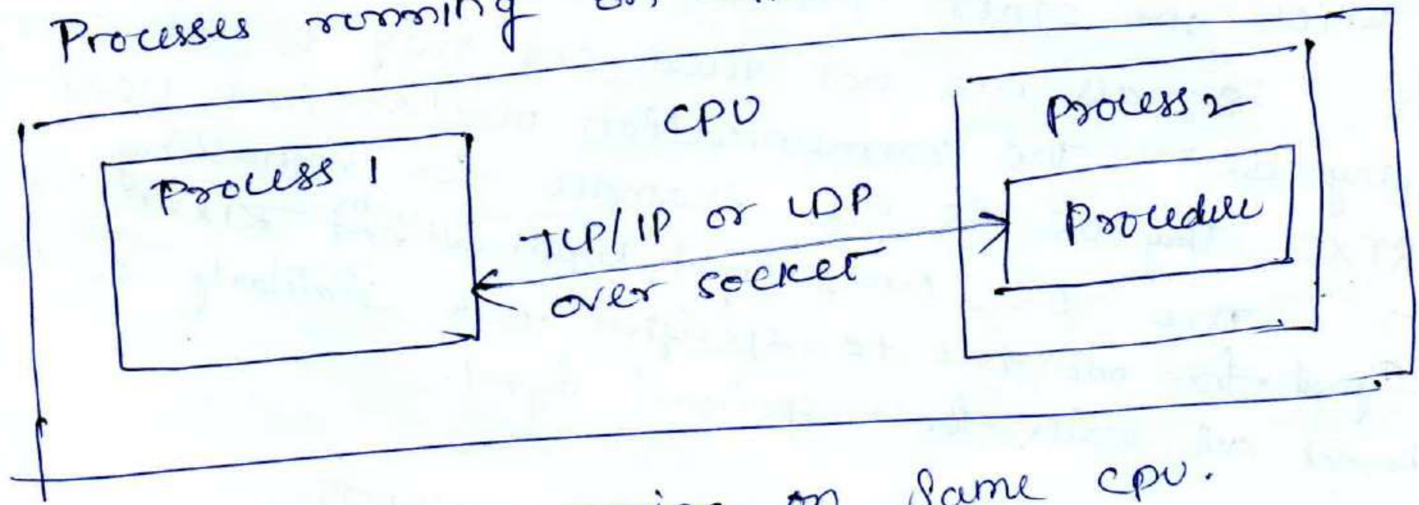
In the object oriented terminology RPC is also known as Remote invocation or Remote method invocati

RPC is mainly used for distributed applications like client - server applications. with RPC it is possible to communicate over a heterogeneous network. (i.e N/w where client and server applications are running on different operating systems).

The cpu/process containing the procedure which needs to be invoked remotely is known as server.

The cpu/process which initiates an RPC request is known as client.

Processes running on different CPU which are networked



Processes running on same CPU.

It is possible to implement RPC communication with different invocation interfaces. In order to make the RPC communication compatible across all platforms it should stick on to certain standard formats.

The RPC communication can be either synchronous or asynchronous. In the synchronous communication, the process which calls the remote procedure is blocked until it receives a response back from the other process.

In asynchronous RPC calls, the calling process continues its execution while the remote process performs the execution of the procedure.

On security front, RPC employs authentication mechanisms to protect the systems against vulnerabilities. The client applications should authenticate themselves with the server for getting access. Authentication mechanisms like IDs, public key cryptography etc are used by the client for authentication. without

Sockets are used for RPC communication. Socket is a logical end point in a two way communication link betwee two applications running on a network.

Sockets are of different types, namely, internet socke UNIX sockets etc.

The INET socket works on internet communication Protocol. TCP/IP, UDP etc. are the communication protoco used by INET sockets. INET sockets are classified into

1. Stream sockets
2. Datagram sockets.

Stream sockets are connection oriented and they use TCP to establish a reliable connection. Datagram sockets are rely on UDP for establishing a connection. The UDP connection is unreliable when compared to TCP. The client-server communication model uses a socket at the client side and a socket at the server side. A port number is assigned to both of these socket The client and server should be aware of the port numbe associated with the socket. In order to start the commun cation, the client needs to send a connection request to the server at the specified port number. The client sho be aware of the name of the server along with its port number.

The server always listens to the specified port number on the network. Upon receiving a connection request from the client, based on the success of authe tication, the server grants the connection request an a communication channel is established b/w the clien and server. The client uses the host name and Port number of server for sending requests and server uses the client's name and port number for sending responses.

Task synchronization :-

In a multitasking environment, multiple processes run concurrently and share the system resources. Apart from this, each process has its own boundary wall and they communicate with each other with different IPC mechanisms including shared memory and variables.

Imagine a situation where two processes try to access display hardware connected to the system or two processes try to access a shared memory area where one process tries to write to a memory location when the other process is trying to read from this. then the result we get an unexpected or wrong one. To eliminate this problem, the solution is, make each process aware of the access of a shared resource either directly or indirectly.

The act of making processes aware of the access of shared resources by each process to avoid conflicts is known as Task/Process synchronisation.

Task communication / Synchronisation Issues :-

1. Racing :- Let us have a look at the following piece of code:

```
#include < windows.h>
# include < stdio.h>
// ************
// counter is an integer variable and buffer is a byte array
// shared between two processes process A and process B
char Buffer [10] = {1,2,3,4,5,6,7,8,9,10};
short int counter = 0;
// ***********
// Process A
void process_A (void) {
int i;
```

Task synchronization :-

In a multitasking environment, multiple processes run concurrently and share the system resources. Apart from this, each process has its own boundary wall and they communicate with each other with different IPC mechanisms including shared memory and variables.

Imagine a situation where two processes try to access display hardware connected to the system or two processes try to access a shared memory area where one process tries to write to a memory location when the other process is trying to read from this. then the result we get an unexpected or wrong one. To eliminate this problem, the solution is, make each process aware of the access of a shared resource either directly or indirectly.

The act of making processes aware of the access of shared resources by each process to avoid conflicts is known as Task/process synchronisation.

Task communication / Synchronisation Issues :-

1. **Racing** :- Let us have a look at the following piece of code:

```
#include < windows.h>
# include < stdio.h>
// * * * * * * * * * * * *
// counter is an integer variable and buffer is a byte array
// shared between two processes process A and process B
char Buffer [10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
short int counter = 0;
// * * * * * * * * * * *
// Process A
void process_A (void) {
int i;
```

```
if (Buffer [i] >0)
    Counter++;
}
}
// *************
// process B
void process_B(void) {
int j;
for (j=5; j<10; j++)
{
 if (Buffer[j]>0)
    counter++;
}
}
// **********
// main thread.
int main () {
    DWORD id;
CreateThread (NULL,0,
              (LPTHREAD_START_ROUTINE) process_A,
               (LPVOID) 0, 0, &id);

CreateThread (NULL, 0,
             (LPTHREAD_START_ROUTINE) process_B,
              (LPVOID) 0, 0, &id);

sleep (100000);
return 0;
}
```

From a programmer perspective the value of counter will be 10 at the end of execution of processes A & B. But it need not be true under a multitasking kernel. The results depending on the process scheduling policies adopted by the OS kernel.

From the above program, the program statement counter++; looks like a single statement from high level programming language perspective. The low level implementation of this statement is dependent on the underlying processor instruction set and the compiler in use.

The low level implementation of the high level program statement counter++; as follows.

```
mov eax, dword ptr [ebp-4]; Load counter in accumulator
add eax, 1; increment accumulator by 1.
mov dword ptr [ebp-4], eax; store counter with accumulator.
```

At the processor instruction level, the value of the variable counter is loaded to the accumulator register (EAX register). The memory variable counter is represented using a pointer. The base pointer register (EBP register) is used for pointing to the memory variable counter. After loading the contents of the variable counter to accumulator, the accumulator content is incremented by one using the add instruction. Finally the content of accumulator is loaded to the memory location.

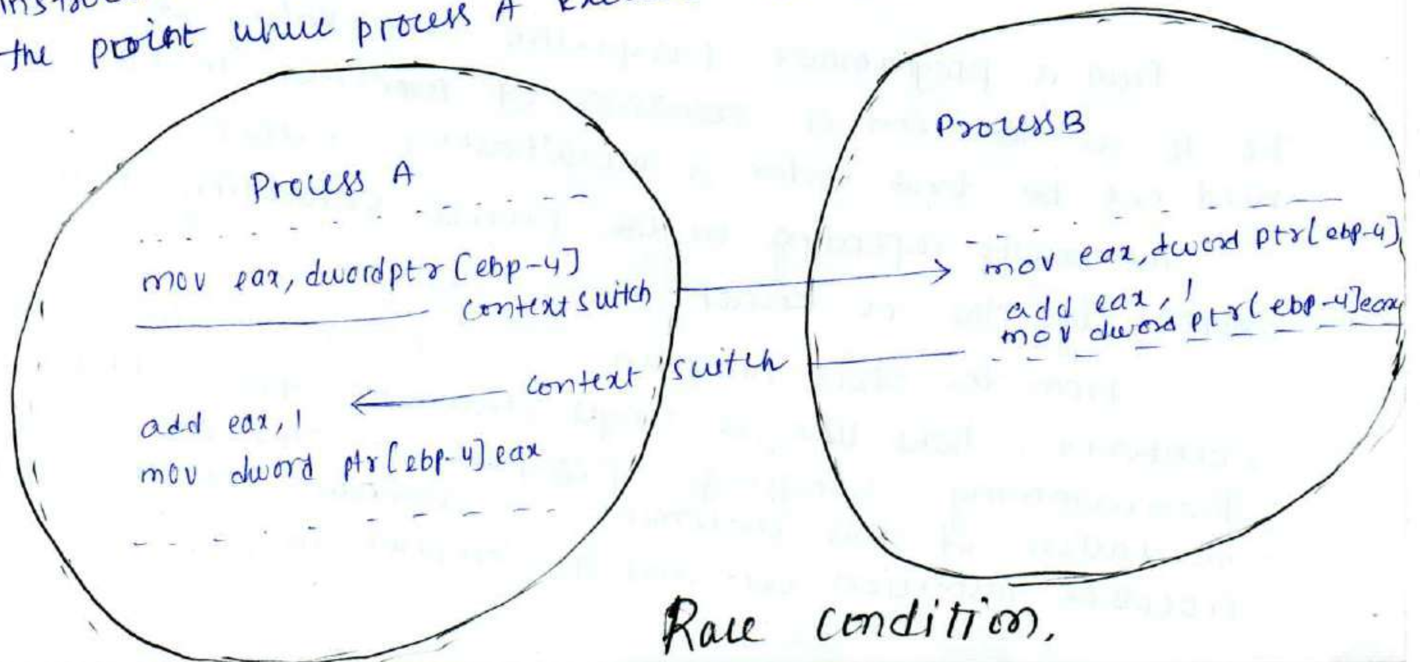
Process A

```
mov eax, dword ptr [ebp-4]
add eax, 1
mov dword ptr [ebp-4], eax
```

Process B

```
mov eax, dword ptr [ebp-4]
add eax, 1
mov dword ptr [ebp-4], eax
```

Imagine a situation where a process switching (context switching) happens from process A to process B when process A is executing the counter++; statement. Process A accomplishes the counter++; statement through three different low level instructions. Now imagine that the process switching happed at the point where process A executed the low level instruction,



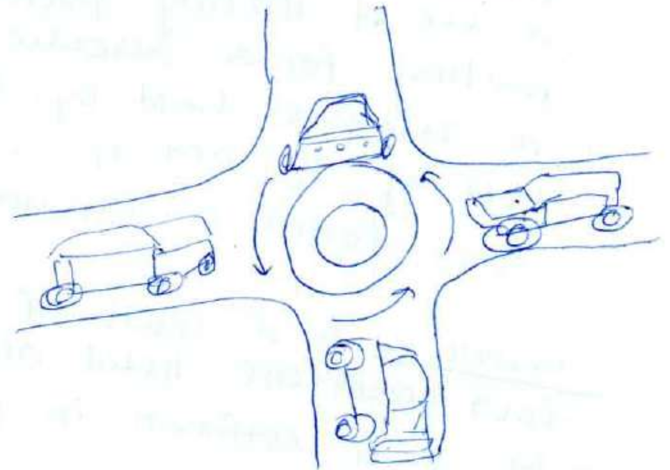
Race condition.

Race or Race condition is the situation in which multiple processes compete (race) each other to access and manipulate shared data concurrently. In a Race condition the final value of the shared data depends on the process which acted on the data finally.

2. Deadlock :-

A race condition produces incorrect results whereas a deadlock condition creates a situation where none of the processes are able to make any progress in their execution, resulting in a set of deadlocked processes.

Ex: traffic jam.



Dead lock visualisation.

In its simplest form deadlock is the condition in which a process is waiting for a resource held by another process which is waiting for a resource held by the first process.

To elaborate: process A holds a resource x and it wants a resource y held by process B.

Process B is currently holding resource y and it wants the resource x which is currently held by process A. Both hold the respective resources and they compete each other to get the resource held by the respective processes. The result of competition is deadlock. None of the competing process will be able to access the resources held by other processes since they are locked by the respective processes.

The different conditions favoring a deadlock situation are,

1. Mutual Exclusion :
   The criteria that only one process can hold a resource at a time. meaning processes should access shared resources with mutual exclusion.

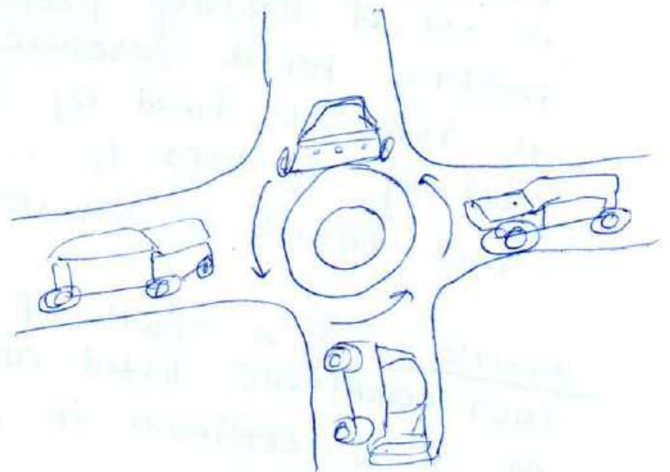
2. Hold and wait :
   The condition in which a process holds a shared resource by acquiring the lock controling the shared access and waiting for additional resources held by other processes.

Race or Race condition is the situation in which multiple processes compete (race) each other to access and manipulate shared data concurrently. In a Race condition the final value of the shared data depends on the process which acted on the data finally.

2. Deadlock :-

A race condition produces incorrect results whereas a deadlock condition creates a situation where none of the processes are able to make any progress in their execution, resulting in a set of deadlocked processes.

Ex: traffic jam.



Dead lock visualisation.

In its simplest form deadlock is the condition in which a process is waiting for a resource held by another process which is waiting for a resource held by the first process.

To elaborate: process A holds a resource x and it wants a resource y held by process B. Process B is currently holding resource y and it wants the resource x which is currently held by process A. Both hold the respective resources and they compete each other to get the resource held by the respective processes. The result of competition is deadlock. None of the competing process will be able to access the resources held by other processes since they are locked by the respective processes.

The different conditions favoring a deadlock situation are,

1. Mutual Exclusion:
The criteria that only one process can hold a resource at a time. meaning processes should access shared resources with mutual exclusion.

2. Hold and wait:
The condition in which a process holds a shared resource by acquiring the lock controlling the shared access and waiting for additional resources held by other processes.

3. No resource preemption

The criteria that operating system cannot take back a resource from a process which is currently holding it and the resource can only be released voluntarily by the process holding it.

4. Circular wait

A process is waiting for a resource which is currently held by another process which in turn is waiting for a resource held by the first process. In general there exists a set of waiting process $P_0, P_1, \ldots P_n$ with P_0 is waiting for a resource held by P_1 and P_1 is waiting for a resource held by $P_0, \ldots P_n$ is waiting for a resource held by $P_n \ldots$ held by P_0 and P_0 is waiting for a resource held by... This forms a circular wait queue.

Deadlock is a result of the combined occurrence of these four conditions listed above. These conditions are described by E.G. Coffman in 1971 and it is popularly known as Coffman conditions.

Deadlock Handling :-

The reaction to deadlock condition by OS is nonuniform. The OS may adopt any of the following techniques to detect and prevent deadlock conditions.

Ignore deadlocks

Always assume that the system is deadlock free. This is acceptable for the reason the cost of removing a deadlock is large compared to the chance of happening a deadlock.

Detect and Recover

This approach suggests the detection of a deadlock situation and recovery from it. This is similar to the deadlock condition that may arise at a traffic junction. When the vehicles from different directions compete to cross the junction, deadlock (traffic jam) condition is resulted. Once a deadlock is happened at the junction, the only solution is

to back up the vehicles from one direction and allow the vehicles from opposite direction to cross the junction. If the traffic is too high, lots of vehicles may have to be backed up to resolve the traffic jam. This technique is also known as 'back up cars technique.'

A deadlock condition can be detected by analysing the resource graph by graph analyser algorithms. once a deadlock condition is detected, the system can terminate a process or preempt the resource to break the deadlocking cycle.

Avoid Deadlocks :

Deadlock is avoided by the careful resource allocation techniques by the operating system. It is similar to the traffic light mechanism at junctions to avoid the traffic jams.

Prevent Deadlocks :

Prevent the deadlock condition by negating one of the four conditions favouring the deadlock situation.

Ensure that a process does not hold any other resources when it requests a resource. This can be achieved by implementing set of rules as fallows.

1. A process must request all its required resource and the resources should be allocated before the process begings its execution.
2. Grant resource allocation requests from processes only if the process does not hold a resource currently.

Livelock :

The livelock condition is similar to the deadlock condition except that a process in livelock condition changes its state with time. When in deadlock a process enters in wait state for a resource and continues in that state forever without making any progress in the execution, in a livelock condition a process always does something but is unable to make any progress in the exection completion.

Ex: two people attempting to cross each other in a narrow corridor.

Starvation :-

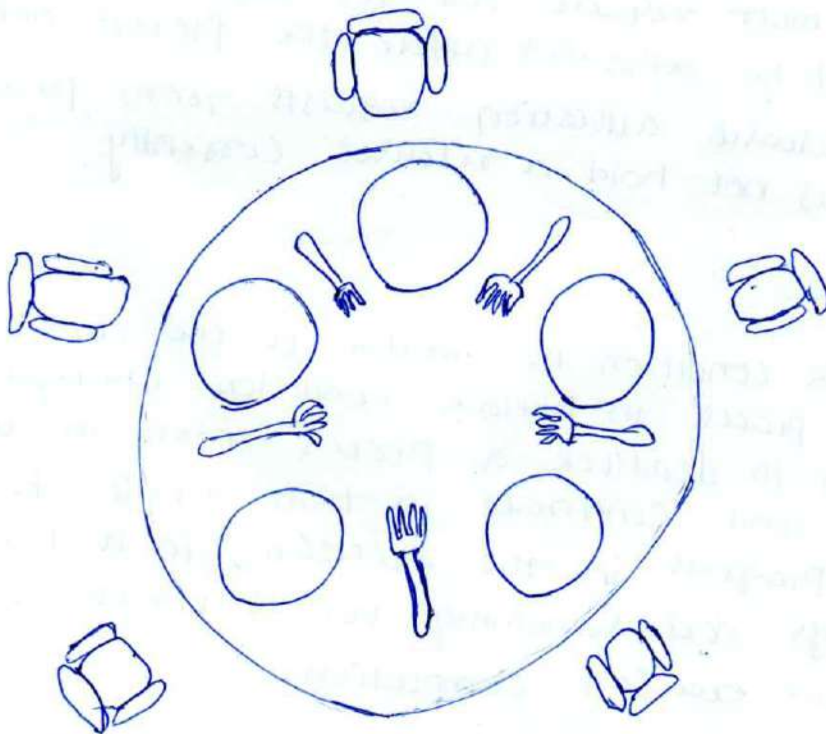
In the multitasking context, starvation is the condition ~~except that a pr~~ in which a process does not get the resources required to continue its execution for a long time. As time progresses the process starves on resource. Starvation may arise due to various conditions like byproduct of Preventive measures of deadlock, scheduling policies favouring high priority tasks and tasks with shortest execution time. etc.

The Dining philosopher's problem:

The dining philosopher's problem is an ~~att~~ interesting example for synchronisation issues in resource utilisation.

Five philosophers are sitting around a round table, involved in eating and brainstorming. At any point of time each philosopher will be in any one of the three states: eating, hungry or brainstorming.

For eating each philosopher requires 2 forks. There are only 5 forks available on the dining table and they are arranged in a fashion one fork in between two philosophers. The philosopher can only use the forks on his/her immediate left and right that too in the order pickup the left fork first and then the right fork.



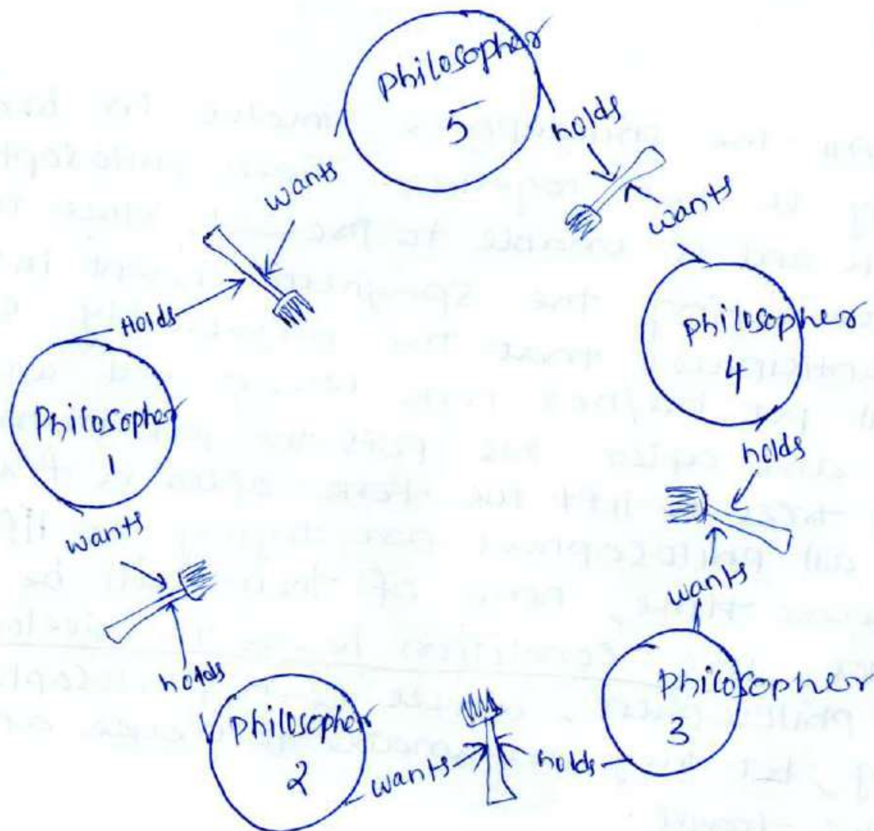
In the OS context, Philosophers represents the Processes and forks represent the resources.

Visualisation of dining philosopher's problem.

Let us analyse the various scenarios that may occur in this situation,

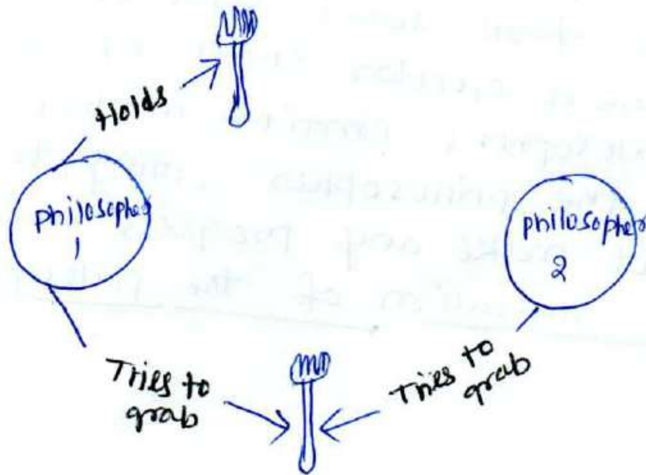
Scenario 1:

All the philosopher's involve in brainstorming together and try to eat together. Each philosopher picks up the left fork and is unable to proceed since two forks are required for eating the spagetti (soft noodles) present in the plate. philosopher 1 think that philosopher 2 sitting to the right of him/her will put the fork down and waits for it. Philosopher 2 thinks that philosopher 3 sitting to the right of him/her will put the fork down and waits for it, and so on. this forms a circular chain of un-granted requests. of the philosophers continue in this state waiting for the fork from the philosopher sitting to the right of each, they will not make any progress in eating and this will result in starvation of the philosophers and deadlock.

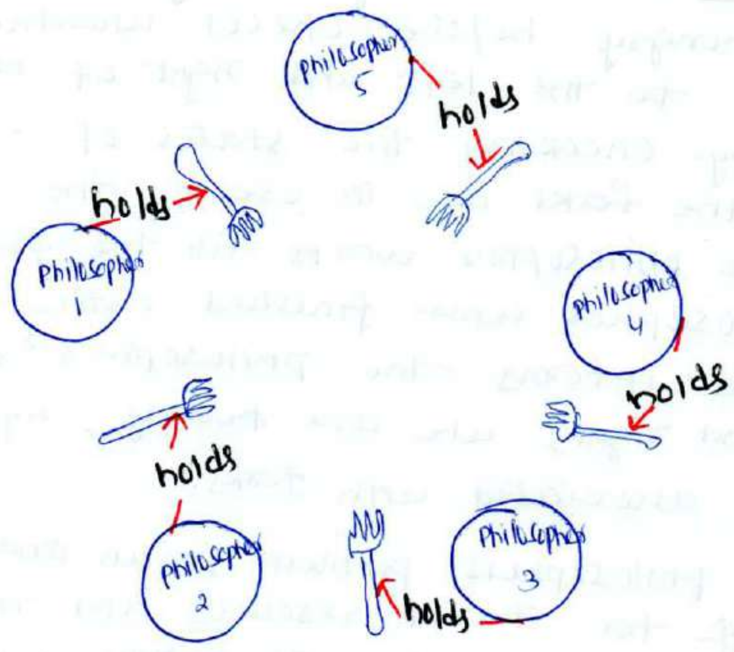


Starvation and deadlock.

Scenario 2 : All the philosophers start brainstorming together. one of the philosophers is hungry and he/she picks up the left fork. when the philosopher is about to pick up the right fork, the philosopher sitting to his right also become hungry and tries to grab the left fork which is the right fork of his neighbouring philosopher who is trying to lift it, resulting in a race condition.



Scenario 3 : All the philosophers involve in brainstorming together and try to eat together. Each philosopher picks up the left fork and is unable to proceed, since two forks are required for eating the spaghetti present in the plate. Each of them anticipates that the adjacently sitting philosopher will put his/her fork down and waits for a fixed duration and after this puts the fork down. Each of them again tries to lift the fork after a fixed duration of time. Since all philosophers are trying to lift the fork at the same time, none of them will be able to grab two forks. This condition leads to livelock and starvation of philosophers, where each philosopher tries to do something, but they are unable to make any progress in achieving the target.



Livelock and Starvation.

Solution: we need to find out alternative solutions to avoid the deadlock, livelock, racing and starvation condition that may arise due to the concurrent access of forks by philosophers. This situation can be handled in many ways by allocating the forks in different allocation techniques including Roundrobin allocation, FIFO etc.

But the requirement is that the solution should be optimal, avoiding deadlock and starvation of the philosophers and allowing maximum number of philosophers to eat at a time. One solution that we could think of is:

(1). Imposing rules in accessing the forks by philosophers like: The philosophers should put down the fork he/she already have in hand after waiting for a fixed duration for the second fork and should wait for a fixed time before making the next attempt.

This solution works fine to some extent, but, if all the philosophers try to lift the forks at the same time, a livelock situation resulted.

(2). Another solution which gives maximum concurrency that can be thought of is each philosopher acquires a

semaphore (mutex) before picking up any fork. When a philosopher feels hungry he/she checks whether the philosopher sitting to the left and right of him is already using the fork, by checking the status of the associated semaphore. If the forks are in use by the neighbouring philosophers, the philosopher waits till the forks are available. A philosopher when finished eating puts the forks down and informs the philosophers sitting to his/her left and right, who are hungry, by signaling the semaphores associated with forks.

The dining philosophers problem is an analogy of processes competing for shared resources and the different problems like racing, deadlock, starvation and livelock arising from the competition.

Priority Inversion :-

Priority inversion is the condition in which a high priority task needs to wait for a low priority task to release a resource which is shared between the high priority task and the low priority task, and a medium priority task which does not require the shared resource continue its execution by preempting the low priority task.

Priority based preemptive scheduling technique ensures that a high priority task is always executed first, whereas the lock based process synchronisation mechanism ensures that a process will not access a shared resource, which is currently in use by another process.

Priority inversion is better explained with the following scenario:

Let Process A, Process B and process C be the three processes with priorities High, medium and low respectively. Process A and Process C share a variable x and the access to this variable is synchronised through a mutual exclusion mechanism like binary semaphore S. This situation is explained with following figure.

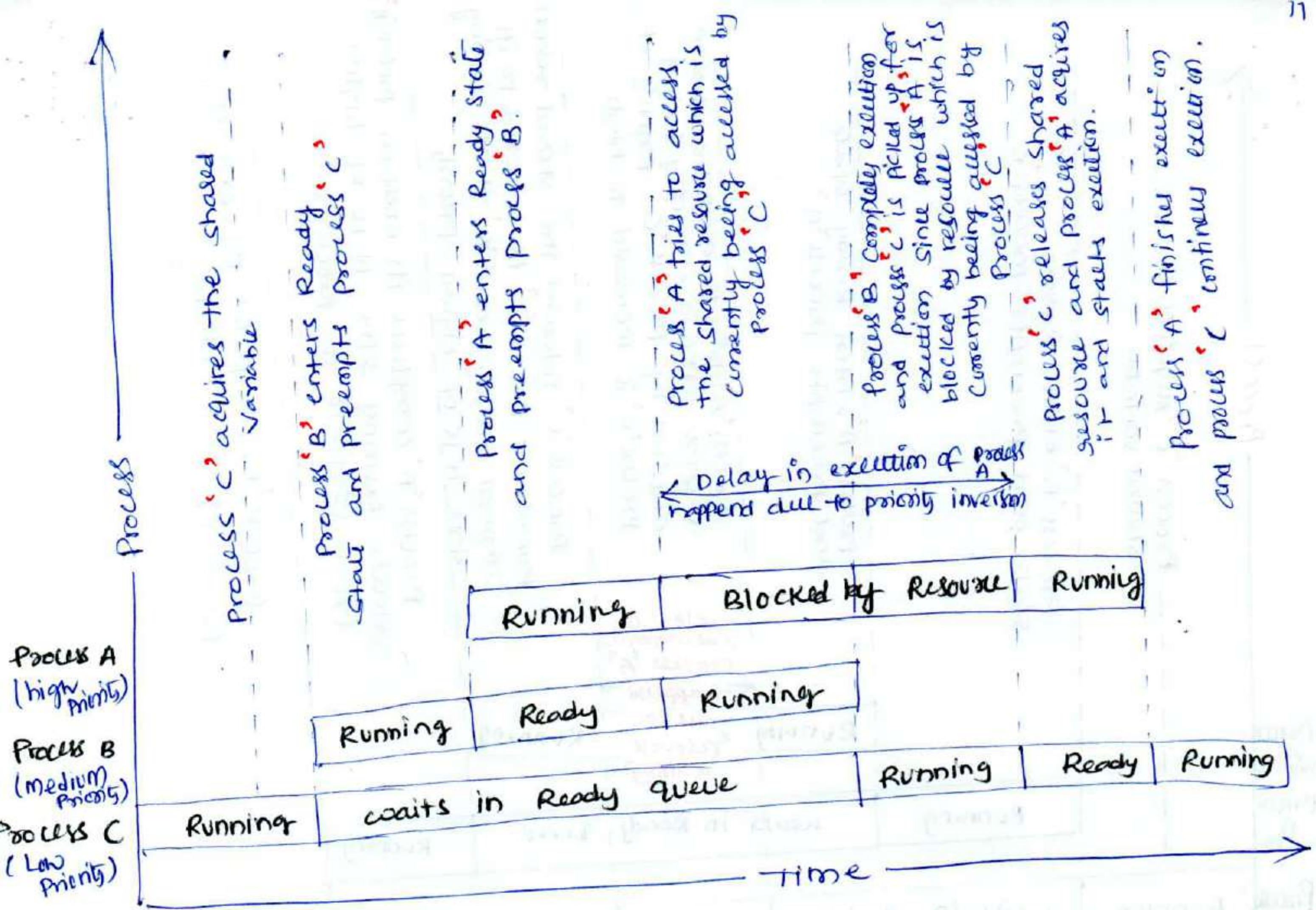


fig: Priority inversion problem.

Priority inheritance :-

A low priority task that is currently accessing a shared resource requested by a high priority task temporarily inherits the priority of that high priority task from the moment the high priority task rises the request. Boosting the priority of the low priority task to that of the priority of the task which requested the shared resource holding by the low priority task eliminates the preemption of the low priority task by other task whose priorities are below that of the tasks requested the shared resource and there by reduces the delay in waiting to get the resource requested by the high priority task.

The priority of the low priority task which is temporarily boosted to high is brought to the original value when it releases the shared resource. which is explained with the following figure.

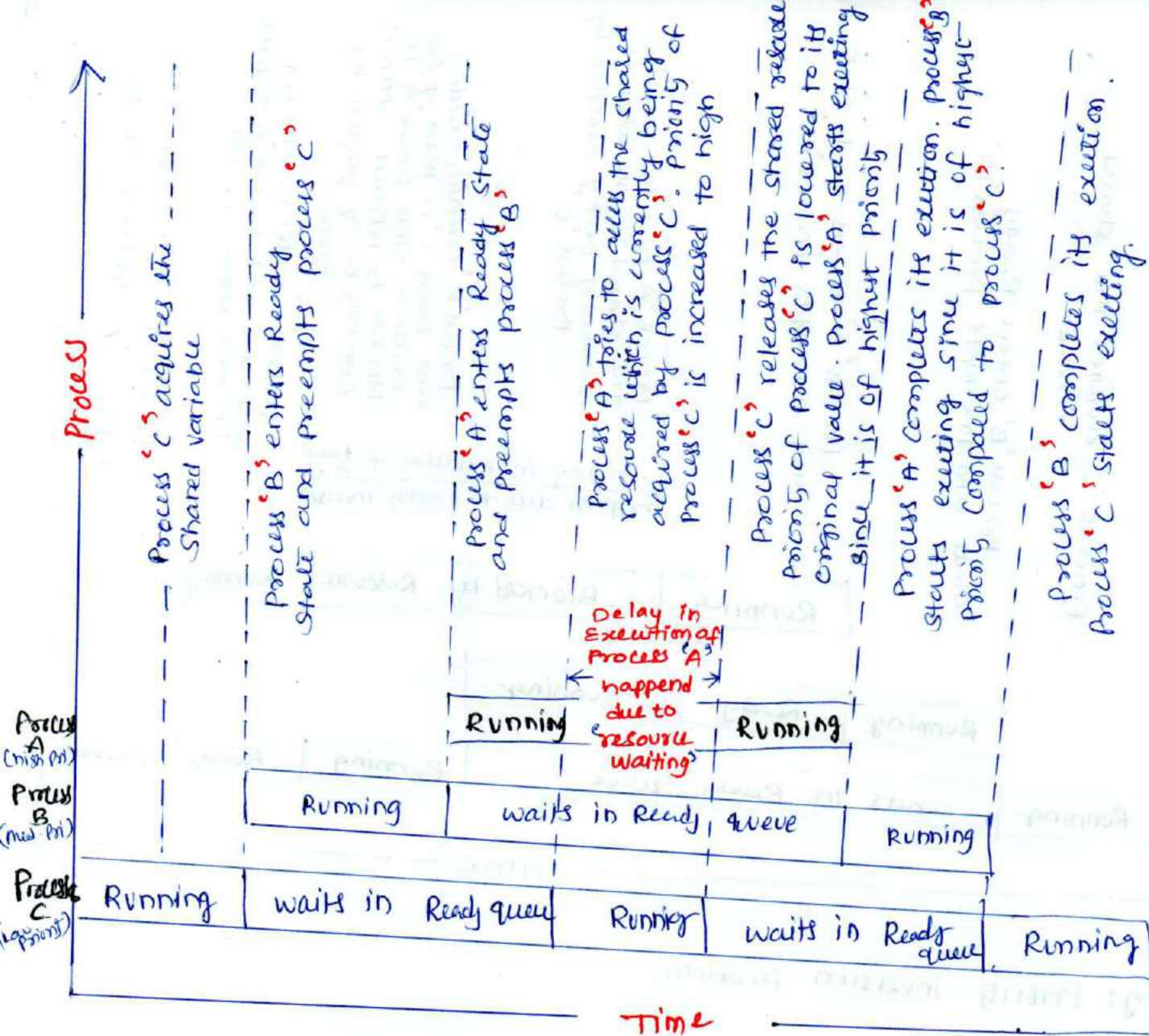


fig: Handling priority inversion problem with priority inheritance.

Priority inheritance is only a work around and it will not eliminate the delay in waiting the high priority task to get the resource from the low priority task. The only thing is that it helps the low priority task to continue its execution and releases the shared resource as soon as possible. The moment, at which the low priority task releases the shared resource, the high priority task kicks the low priority task out and grabs the CPU.

Priority ceiling :-

In priority ceiling, a priority is associated with each shared resource. The priority associated to each resource is the priority of the highest priority task which uses this shared resource. This priority level is called ceiling priority. It is explained with following fig.

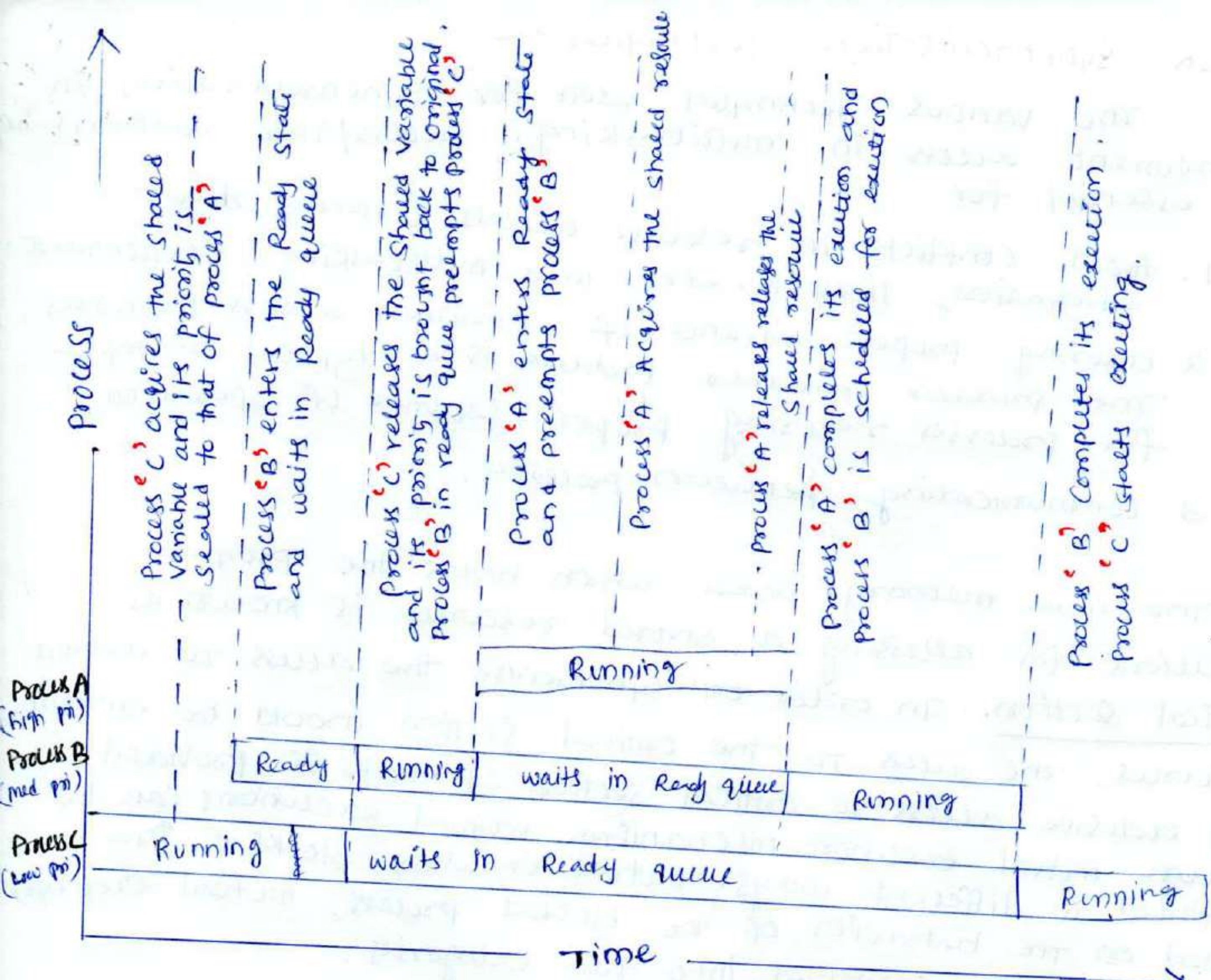


fig: Handling priority inversion problem with priority ceiling

The biggest drawback of priority ceiling is that it may produce hidden priority inversion. With priority ceiling technique, the priority of a task is always elevated no matter another task wants the shared resources. This unnecessary priority elevation always boosts the priority of a low priority task to that of the highest priority tasks among which the resource is shared and other tasks with priorities higher than that of the low priority task is not allowed to preempt the low priority task when it is accessing a shared resource. This always gives the low priority task the luxury of running at high priority when accessing shared resources.

Task Synchronisation Techniques :-

The various Techniques used for Synchronisation in concurrent access in multitasking. Process/Task synchronisation is essential for

1. Avoid conflicts in resource access (racing, deadlock, starvation, livelock, etc). in a multitasking environment.
2. Ensuring proper sequence of operation across processes. The producer consumer problem is a typical example for processes requiring proper sequence of operation.
3. Communicating between processes.

The code memory area which holds the program instructions for accessing a shared resource is known as critical section. In order to synchronise the access to shared resources, the access to the critical section should be exclusive. The exclusive access to critical section of code is provided through mutual exclusion mechanism. mutual exclusions can be enforced in different ways. Mutual exclusion blocks a process. Based on the behavior of the blocked process, mutual exclusion methods can be classified into two categories.

1. mutual exclusion through busy waiting/spin lock
2. mutual exclusion through sleep and wakeup.

The busy waiting mutual exclusion enforcement mechanism used by processes makes the CPU always busy by checking the lock to see weather they can proceed. This results in the wastage of CPU time and leads to high power consumption. This is not affordable in embedded systems powered on battery, since it affects the battery backup time of the device.

An alternative to 'busy waiting' is the 'sleep & wakeup' mechanism. when a process is not allowed to access the critical section, which is currently beeing locked by another process, the process under goes 'sleep' and enters the 'blocked' state. The process which is blocked on waiting for access to the critical section is awake and by the process.

which currently holds the critical section. The process which holds the critical section sends a wakeup message to the process, which is sleeping as a result of waiting for the access to the critical section.

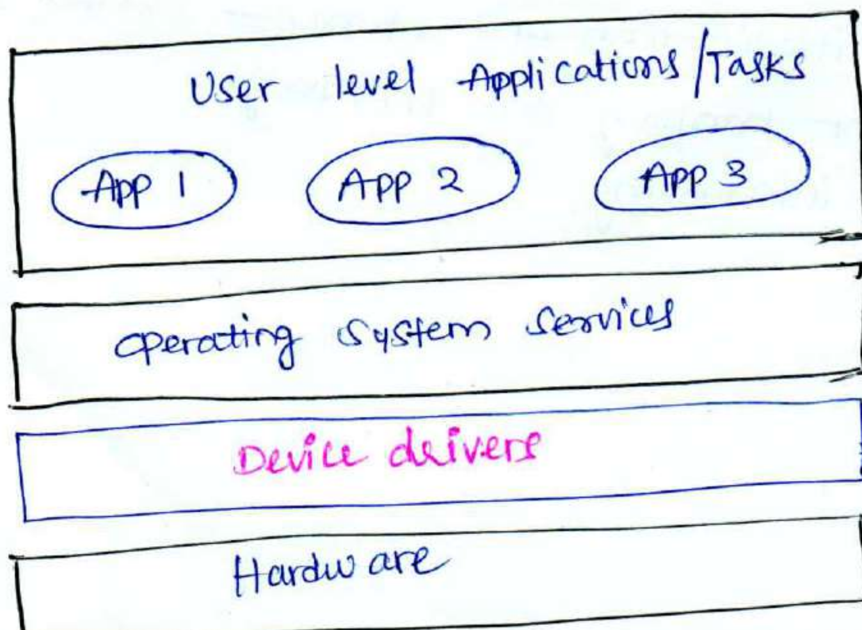
Semaphore :

Semaphore is a sleep & wakeup based mutual exclusion implementation for shared resource access. semaphore is a system resource and the process which wants to access the shared resource can first acquire this system ~~access~~ object to indicate other process which wants the shared resource. That the shared resource is currently acquired by it. The resource which are shared among a process can be eaither for exclusive use by a process or for using by a no. of process at a time.

Binary Semaphore (Mutex) :

It is a synchronisation object provided by OS for process/thread synchronisation. any process/thread can create a mutex object and other processes/threads of these system can use this mutex object for synchronising the access to critical section. only one process/thread can own the mutex object at a time. The state of a mutex object is set to signal when it is not owned by any process/thread, and set to not signaled when it is owned by any process/thread.

Device drivers :-



Device deiver is a piece of software that acts as a bridge between the operating system and hardware. In an OS based product architecture, the user applications talk to the OS kernel for all information exchange including communication with the hardware peripherals. The architecture of the OS kernel will not allow direct device access from the user application. All the device related access should flow through the OS kernel and the OS kernel routs it to the concerned H/w peripherals.

OS provides interfaces in the form of application programming interfaces for accessing the hardware. The device deiver abstracts the h/w from user applications.

The topology of user applications and h/w interactions in an RTOS based systems is shown in figure.

Device drivers are responsible for intiating and managing the communication with the h/w peripherals. They are responsible for establishing the connectivity initialising the h/w (seting up various registers of the h/w device). and transfering data. An embedded product may contain different types of h/w components like wi-fi module, file system, storage device interface etc.

However regardless of OS types a device deiver implements the following.

1. Device initialisation and intrrupt configuration.
2. Interrupt handling and processing
3. Client interfacing.

How to choose an RTOS

RTOS can be either functional or non-functional.

Functional requirements.

Processor support :-

It is not necessary that all RTOS's support all kind of processor architecture. it is essential to ensure the processor support by the RTOS.

Memory requirments :-

OS requires ROM memory for holding the OS files and its normally stored in a non volatile memory like flash. OS also requires working memory RAM for loading the OS services

Real time capabilites :-

The task and process scheduling policies place an important Role in the real time behaviour of OS. it is not mandatory that OS for all embedded system need to be real time and all OS are real time in behaviour.

Kernel and interrupt Latency :-

The kernel of OS may disable interrupts while executing certain services and it may led to interrupt latency, whose response requirements are high, this latency should be minimal.

Inter process communication and task synchronisation :-

The implementation of inter process comm. & synch. is OS kernel dependent. certain kernels may provide a bunch of options where as other provide a limited of options.

Modularisation support :-

Most of the OS provides a bunch of features. at times it may not be necessary for an embedded product for its functioning. It is very useful if the OS supports modularisation where in which the developer can choose the essential module and recombine the OS image for functioning.

Non-functional requirements :-

Custom developed or off the shelf :-

depending on the OS requirement, it is possible to go for the complete development of an operating system suiting the embedded system needs are use and off the shelf, ready available OS, which is eaither commertial Product or an open source product. which is in close match with the system reqirment.

Cost :- The total cost for developing or buying the OS and maintaining it in terms of commertial product and custom build needs to be evaluated before talking a decision on the selection of OS.

Development and debugging tools availability :-

~~The avai~~ It is a critical decision making factor in the selection of an OS for embedded design. Certain OS's may be superior in performance, but the availability of tools for supporting the development may be limited.

Ease of use :-

How easy it is to use a commertial RTOS is another important feature that needs to be considered in the RTOS selection.

After Sales :-

For a commertial RTOS, after sales in the form of e-mail, on-call services etc. for bug fixes, critical Patch updates and support for production issues etc. should be analysed thoroughly.