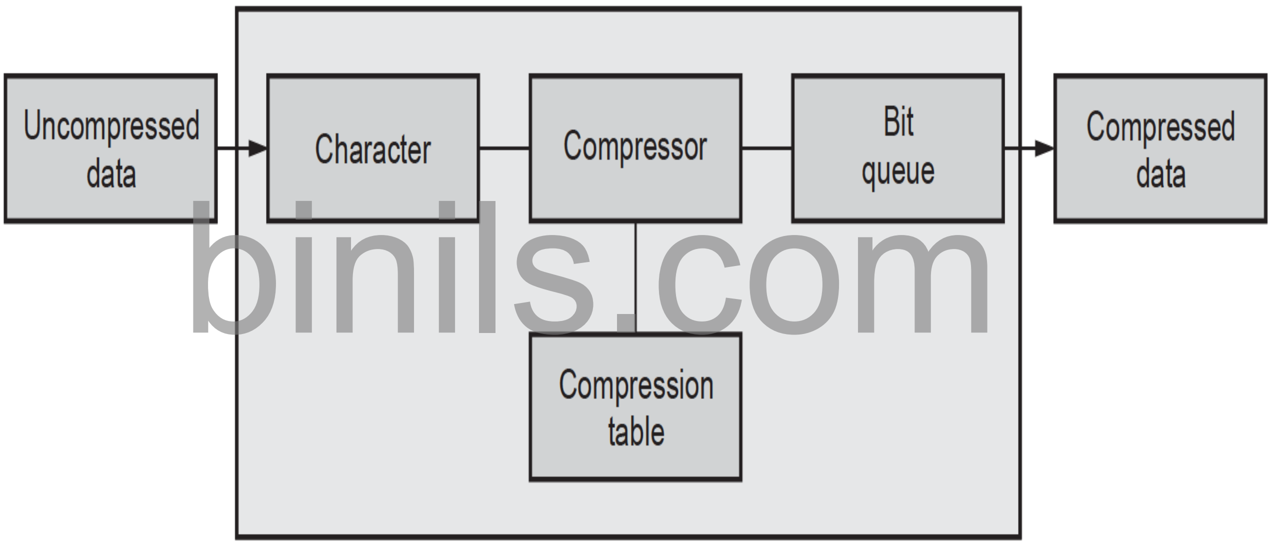
Multiple Tasks and Multiple Processes

Multiple Tasks and Multiple Processes

- Tasks are units of sequential code implementing the system actions and executed concurrently by an OS.
- Real time systems require that tasks be performed within a particular time frame. Task is related to the performance of the real time systems.
- A task, also called a thread, is a simple program that thinks it has the CPU all to itself. The design process for a real-time application involves splitting the work to be done into tasks responsible for a portion of the problem.
- Each task is assigned a priority, its own set of CPU registers and its own stack area.
- In the specified time constraint, system must produce its correct output. If system fail to meet the specified output, then the system is fail or quality decreases.
- Real time systems are used for space flights, air traffic control, high speed aircraft, telephone switching, electricity distribution, industrial processes etc.
- Real time system must be 100 % responsive 100 % of the time. Response time is measured in fractions of second, but this is an ideal not often achieved in the field.
- Real time database is updated continuously. In aircraft example, flight data is continuously changing so it is necessary to update. It includes speed, direction, location, height etc.
- A process is a sequential program in execution. Terms like job and task are also used to denote a process.
- A process is a dynamic entity that executes a program on a particular set of data. Multiple processes may be associated with one program.
- Task is a single instance of an executable program.

- In a multiprogramming environment, usually more programs to be executed than could possibly be rim at one time. In CPU scheduling, it switches from one process to another process. CPU resource management is commonly known as scheduling.
- Objective of the multiprogramming is to increases the CPU utilization. CPU scheduling is one kind of fundamental operating system functions.
- Each process has an execution state which indicates what process is currently doing. The process descriptor is the basic data structure used to represent the specific state for each process. A state diagram is composed of a set of states and transitions between states.



On-the-fly compression box

- Input and output of the compressor box is serial ports. It takes uncompressed data and processes it. Output of the box is compressed data. Given data is compressed using predefined compression table. Modem is used such type of box.
- The program's need to receive and send data at different rates. It is an example of rate control problems. It uses asynchronous input. You can provide a button for compressed mode and uncompressed mode.

- When user press uncompressed mode, the input data is passed through unchanged.
- The button will be depressed at a much lower rate than characters will be received, since it is not physically possible for a person to repeatedly depress a button at even slow serial line rates.
- Keeping up with the input and output data while checking on the button can introduce some very complex control code into the program.
- Sampling the button's state too slowly can cause the machine to miss a button depression entirely, but sampling it too frequently and duplicating a data value can cause the machine to incorrectly compress data.
- This problem is solved by maintaining counter.

Priority Based Scheduling

Earliest-Deadline-First Scheduling

Earliest Deadline First (EDF) is one of the best known algorithms for real time processing. It is an optimal dynamic algorithm. In dynamic priority algorithms, the priority of a task can change during its execution. It produces a valid schedule whenever one exists.

EDF is a preemptive scheduling algorithm that dispatches the process with the earliest deadline. If an arriving process has an earlier deadline than the running process, the system preempts the running process and dispatches the arriving process.

A task with a shorter deadline has a higher priority. It executes a job with the earliest deadline. Tasks cannot be scheduled by rate monotonic algorithm.

EDF is optimal among all scheduling algorithms not keeping the processor idle at certain times. Upper bound of process utilization is 100 %.

Whenever a new task arrive, sort the ready queue so that the task closest to the end of its period assigned the highest priority. System preempt the running task if it is not placed in the first of the queue in the last sorting.

If two tasks have the same absolute deadlines, choose one of the two at random (ties can be broken arbitrarily). The priority is dynamic since it changes for different jobs of the same task.

EDF can also be applied to aperiodic task sets. Its optimality guarantees that the maximal lateness is minimized when EDF is applied.

Many real time systems do not provide hardware preemption, so other algorithm must be employed.

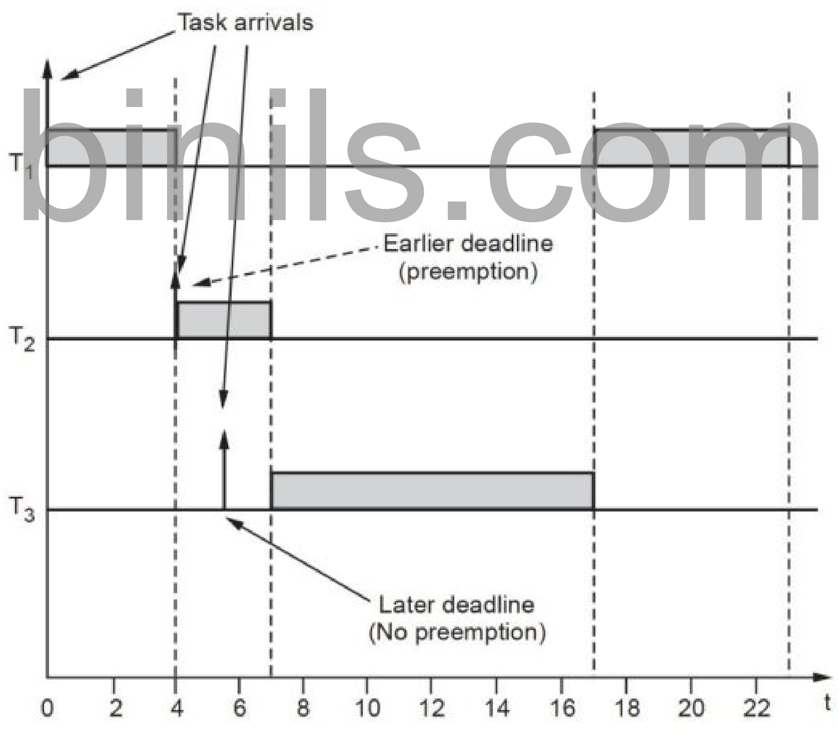
In scheduling theory, a real-time system comprises a set of real-time tasks; each task consists of an infinite or finite stream of jobs. The task set can be scheduled by a number of policies including fixed priority or dynamic priority algorithms.

The success of a real-time system depends on whether all the jobs of all the tasks can be guaranteed to complete their executions before their deadlines. If they can, then we say the task set is schedulable.

The schedulability condition is that the total utilization of the task set must be less than or equal to 1.

Implementation of earliest deadline first : Is it really not feasible to implement EDF scheduling ?

Task	Arrival	Duration	Deadline
T1	0	10	33
T2	4	3	28
T3	5	10	29



Problems for implementations :

1. Absolute deadlines change for each new task instance, therefore the priority needs to be updated every time the task moves back to the ready queue.
2. More important, absolute deadlines are always increasing, how can we associate a finite priority value to an ever increasing deadline value.

3. Most important, absolute deadlines are impossible to compute a-priori.

EDF properties :

1. EDF is optimal with respect to feasibility (i.e. schedulability).
2. EDF is optimal with respect to minimizing the maximum lateness.

Advantages

1. It is optimal algorithm.
2. Periodic, aperiodic and sporadic tasks are scheduled using EDF algorithm.
3. Gives best CPU utilization.

Disadvantages

1. Needs priority queue for storing deadlines
2. Needs dynamic priorities
3. Typically no OS support
4. Behaves badly under overload
5. Difficult to implement.

Rate Monotonic Scheduling

Rate Monotonic Priority Assignment (RM) is a so called static priority round robin scheduling algorithm.

In this algorithm, priority is increases with the rate at which a process must be scheduled. The process of lowest period will get the highest priority.

The priorities are assigned to tasks before execution and do not change over time. RM scheduling is preemptive, i.e., a task can be preempted by a task with higher priority.

In RM algorithms, the assigned priority is never modified during runtime of the system. RM assigns priorities simply in accordance with its periods, i.e. the priority is as higher as shorter is the period which means as higher is the activation rate. So RM is a scheduling algorithm for periodic task sets.

If a lower priority process is running and a higher priority process becomes available to run, it will preempt the lower priority process. Each periodic task is assigned a priority inversely based on its period :

1. The shorter the period, the higher the priority.
2. The longer the period, the lower the priority.

The algorithm was proven under the following assumptions :

1. Tasks are periodic.
2. Each task must be completed before the next request occurs.
3. All tasks are independent.
4. Run time of each task request is constant.
5. Any non-periodic task in the system has no required deadlines.

RMS is optimal among all fixed priority scheduling algorithms for scheduling periodic tasks where the deadlines of the tasks equal their periods.

Advantages :

1. Simple to understand. 2. Easy to implement. 3. Stable algorithm.

Disadvantages :

1. Lower CPU utilization.
2. Only deal with independent tasks.
3. Non-precise schedulability analysis

Comparison between RMS and EDF

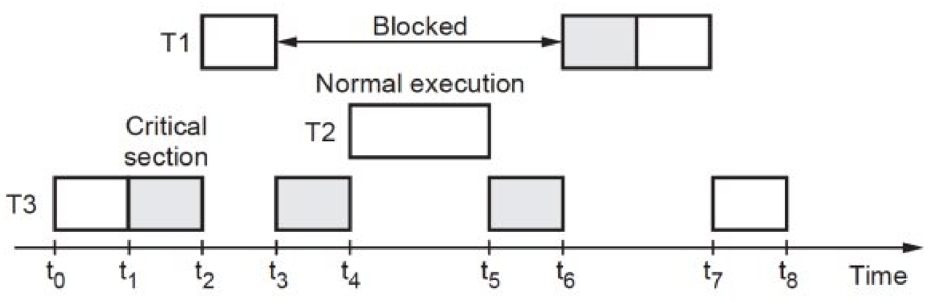
Parameters	**RMS**	**EDF**
Priorities	Static	Dynamic
Works with OS with fixed priorities	Yes	No
Uses full computational power of processor	No	Yes
Possible to exploit full computational power of	No	Yes
Processor without provisioning for slack		

Priority Inversion

Priority inversion occurs when a low-priority job executes while some ready higher-priority job waits.

Consider three tasks Tl, T2 and T3 with decreasing priorities. Task T1 and T3 share some data or resource that requires exclusive access, while T2 does not interact with either of the other two tasks.

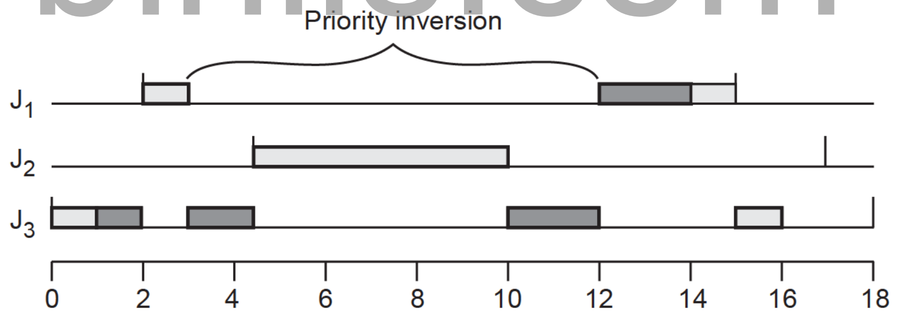
Task T3 starts at time t0 and locks semaphore s at time tv At time t2, Tl arrives and preempts T3 inside its critical section. After a while, Tl requests to use the shared resource by attempting to lock s, but it gets blocked, as T3 is currently using it. Hence, at time t3 continues to execute inside its critical section. Next, when T2 arrives at time t4, it preempts T3, as it has a higher priority and does not interact with either Tl or T3.



Priority inversion method

The execution time of T2 increases the blocking time of Tl, as it is no longer dependent solely on the length of the critical section executed by T3.

When tasks share resources, there may be priority inversions.



Priority inversion example

Priority inversion is not avoidable; However, in some cases, the priority inversion could be too large.

Simple solutions :

1. Make critical sections non-preemptable.
2. Execute critical sections at the highest priority of the task that could use it.

The solution of the problem is rather simple; while the low priority task blocks an higher priority task, it inherits the priority of the higher priority task; in this way, every medium priority task cannot make preemption.

Timing anomalies

As seen, contention for resources can cause timing anomalies due to priority inversion and deadlock. Unless controlled, these anomalies can be arbitrary duration, and can seriously disrupt system timing.

It cannot eliminate these anomalies, but several protocols exist to control them :

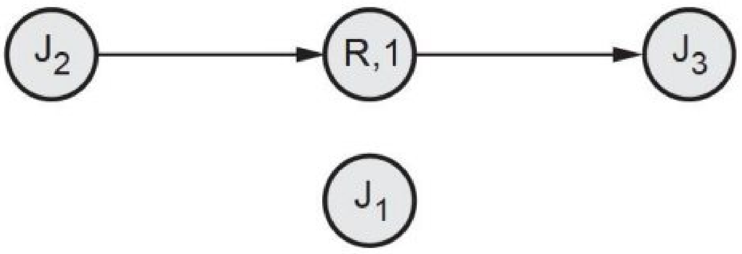
1. Priority inheritance protocol
2. Basic priority ceiling protocol
3. Stack-based priority ceiling protocol

Wait for graph

Wait-for graph is used for representing dynamic-blocking relationship among jobs. In the wait-for graph of a system, every job that requires some resource is represented by a vertex labeled by the name of the job.

At any time, the wait-for graph contains an (ownership) edge with label x from a resource vertex to a job vertex if x units of the resource are allocated to the job at the time.

Wait-for-graph is used to model resource contention. Every serial reusable resource is modeled. Every job which requires a resource is modeled by vertex with arrow pointing towards the resource.



Wait for graph

Every job holding a resource is represented by a vertex pointing away from the resource and towards the job. A cyclic path in a wait-for-graph indicates deadlock.

J3 has locked the single unit of resource R and J2 is waiting to lock it.

A minimum of two system resources are required in a deadlock.

Interprocess Communication Mechanism

Exchange of data between two or more separate, independent processes/threads is possible using IPC. Operating systems provide facilities/resources for Inter-Process Communications (IPC), such as message queues, semaphores and shared memory.

A complex programming environment often uses multiple cooperating processes to perform related operations. These processes must communicate with each other and share resources and information. The Kernel must provide mechanisms that make this possible. These mechanisms are collectively referred to as interprocess communication.

Distributed computing systems make use of these facilities/resources to provide Application Programming Interface (API) which allows IPC to be programmed at a higher level of abstraction, (e.g., send and receive).

Five types of inter-process communication are as follows :

1. Shared memory permits processes to communicate by simply reading and writing to a specified memory location.
2. Mapped memory is similar to shared memory, except that it is associated with a file in the file system.
3. Pipes permit sequential communication from one process to a related process.
4. FIFOs are similar to pipes, except that unrelated processes can communicate because the pipe is given a name in the file system.
5. Sockets support communication between unrelated processes even on different computers.

Name	Description	Scope	Use
File	Data is written to and read from a typical UNIX file. Any number of processes can interoperate.	Local	Sharing large data sets.

Pipe	Data is transferred between two processes using dedicated file descriptors. Communication occurs only between a parent and child process.	Local	Simple data sharing, such as producer and consumer.
Named pipe	Data is exchanged between processes via dedicated file descriptors. Communication can occur between any two peer processes on the same host.	Local	Producer and consumer or command-and-contro 1, as demonstrated with MySQL server and its command-line query utility.
Signal	An interrupt alerts the application to a specific condition.	Local	Cannot transfer data in a signal, so mostly useful for process management.
Shared memory	Information is shared by reading and writing from a common segment of memory.	Local	Cooperative work of any kind, especially if security is required.
Socket	After special setup, data is transferred using common input/output operations.	Local or remote	Network services such as FTP, ssh, and the Apache Web Server.

Purposes of IPC

Data transfer : One process may wish to send data to another process.

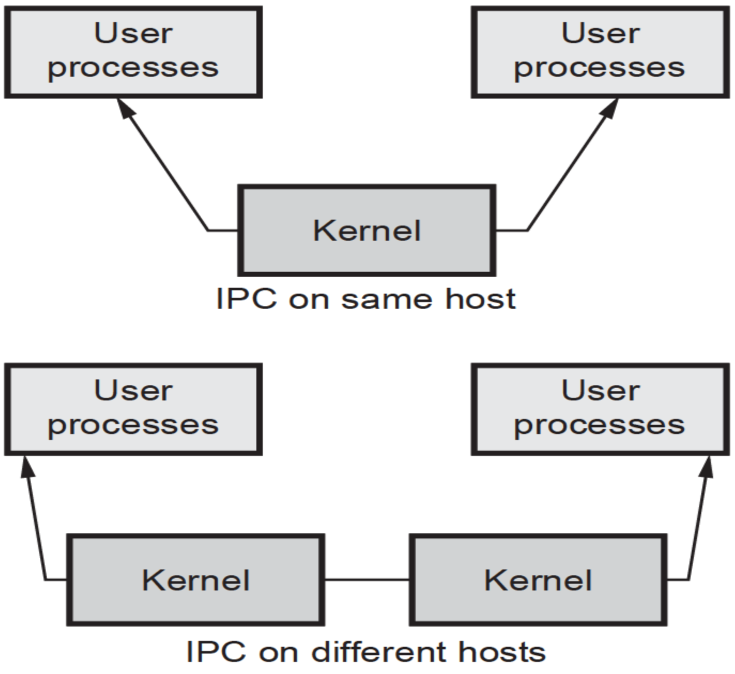
Sharing data : Multiple processes may wish to operate on shared data, such that if a process modifies the data, that change will be immediately visible to other processes sharing it

Event modification : A process may wish to notify another process or set of processes that some event has occurred.

Resource sharing : The Kernel provides default semantics for resource allocation; they are not suitable for all application.

Process control : A process such as a debugger may wish to assume complete control over the execution of another process.

IPC has two forms : IPC on same host and IPC on different hosts.



IPC on different hosts

IPC is used for 2 functions :

1. **Synchronization :** Used to coordinate access to resources among processes and also to coordinate the execution of these processes. They are record locking, Semaphores, mutexes and condition variables.
2. **Message passing :** Used when processes wish to exchange information. Message passing takes several forms such as : Pipes, FIFOs, Message queues and Shared memory.

Features of Message Passing

1. **Simplicity :** Message passing system should be simple and easy to use. It should be possible to communicate with old and new applications.
2. **Uniform semantics :** Message passing is used for two types of IPC.
    1. **Local communication :** Communicating processes are on the same node.
    2. **Remote communication :** Communicating processes are on the different nodes.

3. **Efficiency :** IPC become so expansive if message passing system is not effective. Users try avoiding to IPC for their applications. Message passing system will become more efficient if we try to avoid more message exchanges during communication process. For examples :

a. Avoiding the costs of establishing and terminating connection.
b. Minimizing the costs of maintaining the connections.
c. Piggybacking of acknowledgement.

4. **Reliability :** Distributed systems are prone to different catastrophic events such as node crashes or physical link failures. Loss of message because of communication link fails. To handle the loss messages, we required acknowledgement and retransmission policy. Duplicate message is one of the major problems. This happens because of timeouts or events of failures.
5. **Correctness :** Correctness is a feature related to IPC protocols for group communication. Issues related to correctness are as follows :

1. **Atomicity :** Every message sent to a group of receivers will be delivered to either all of them or none of them.
2. **Ordered delivery :** Messages arrive to all receivers in an order acceptable to the application.
3. **Survivability :** Messages will be correctly delivered despite partial failures of processes, machine, or communication links.

6. **Security :** Message passing system must provide a secure end to end communication.
7. **Portability :** Message passing system should itself be portable.

IPC Message Format

Message passing system requires the synchronization and communication between the two processes. Message passing used as a method of communication in microkernels. Message passing systems come in many forms. Messages sent by a process can be either fixed or variable size. The actual function of message passing is normally provided in the form of a pair of primitives.

a) Send (destination_name, message)
b) Receive (source_name, message)

Send primitive is used for sending a message to destination. Process sends information in the form of a message to another process designated by a destination. A process receives information by executing the receive primitive, which indicates the source of the sending process and the message.

Design characteristics of message system for IPC.

1. Synchronization between the process
2. Addressing
3. Format of the message
4. Queueing discipline

Issues in IPC by Message Passing

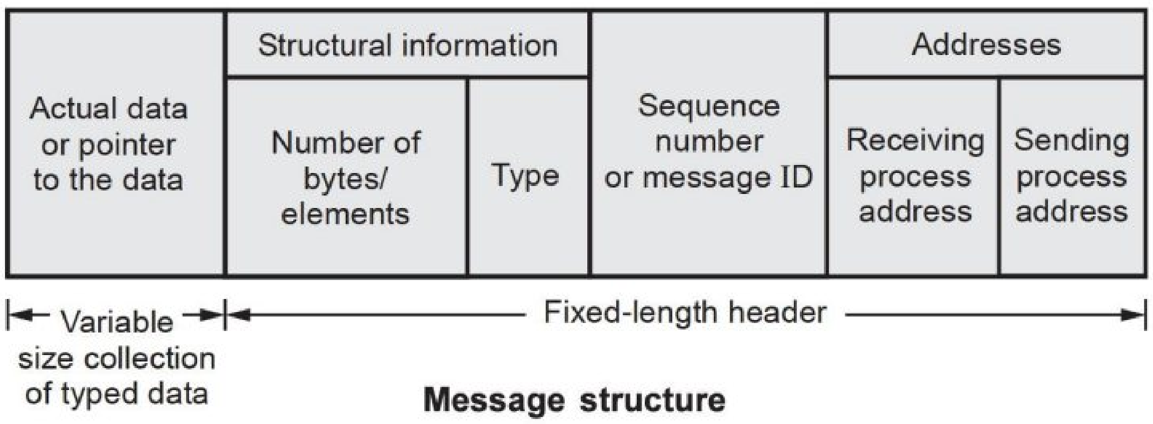
Message is a block of information.

A message is a meaningful formatted block of information sent by the sender process to the receiver process.

The message block consists of a fixed length header followed by a variable size collection of typed data objects.

The header block of a message may have the following elements :

1. **Address :** A set of characters that uniquely identify both the sender and receiver.
2. **Sequence number :** It is the message identifier to identify duplicate and lost messages in case of system failures.
3. **Structural information :** It has two parts. The type part that specifies whether the data to be sent to the receiver is included within the message or the message only contains a pointer to the data. The second part specifies length of the variable-size message.


Actual data or pointer to the data
Structural information
Number of bytes/ elements
Type
Sequence number or message ID
Addresses
Receiving process address
Sending process address
Variable size collection of typed data
Fixed-length header

Message structure

Some important issues to be considered for the design of an IPC protocol based message passing system :

I. The sender's identity
II. The receiver's identity
III. Number of receivers
IV. Guaranteed acceptance of sent messages by the receiver
V. Acknowledgement by the sender
VI. Handling system crashes or link failures
VII. Handling of buffers
VIII. Order of delivery of messages

IPC Synchronization

Send operation can be synchronous or asynchronous. Receive operation can be blocking or nonblocking.

Sender and receiver process can be blocking mode or nonblocking mode. Different possibility of sender and receivers are as follows :

1. Blocking send, blocking receive
2. Nonblocking send, block receive
3. Nonblocking send, Nonblocking receive

Blocking send, blocking receive

Blocking send must wait for the receiver to receive the message, synchronous communication is an example of blocking send. Both processes (sender and receiver) are blocked untill the message is delivered.

Rendezvous : Sending a message to another process leaving the sender process suspended until the message is received and processed.

Nonblocking send, blocking receive

Sender is free to send the messages but receiver is blocked until the requested message arrives.

Asynchronous communication is an example of nonblocking send. Asynchronous communication with nonblocking sends increases throughput by reducing the time that processes spend waiting.

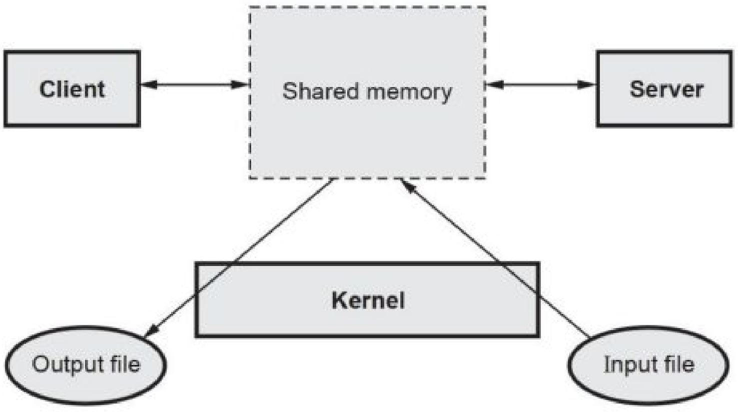
Natural concurrent programming task uses the nonblocking send. Nonblocking send is the overhead on the programmer for determine that a message has been received or not.

Shared Memory

A region of memory that is shared by co-operating processes is established. Processes can then exchange information by reading and writing data to the shared region.

Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer. Shared memory is faster than message passing, as message-passing systems are typically implemented using system calls and thus require the more time-consuming task of Kernel intervention.

In contrast, in shared-memory systems, system calls are required only to establish shared-memory regions.



Client / server with shared memory

Advantages

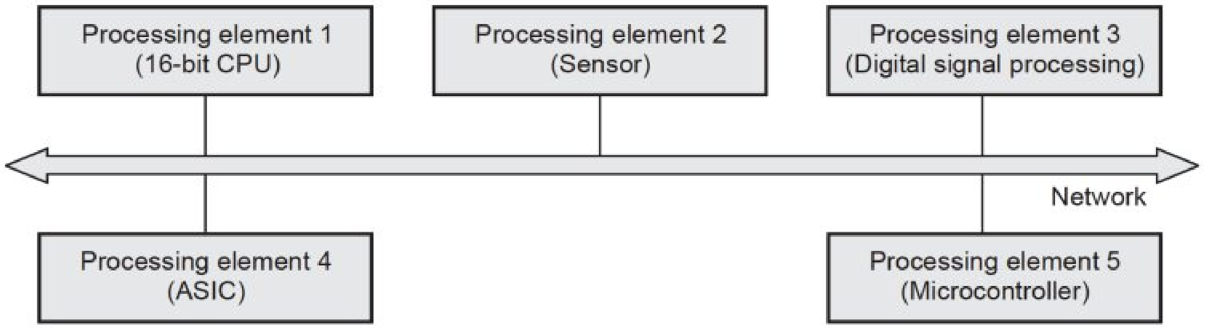
1. Good for sharing large amount of data. 2. Very fast.

Limitations

1. No synchronization provided - applications must create their own.
2. Alternative to shared memory is mmap system call, which maps file into the address space of the caller.

Distributed Embedded Systems

In a distributed embedded system, several Processing Elements (PEs) are connected by a network that allows them to communicate.



Distributed embedded system

Processing elements may includes DSP, CPU or microcontroller. Nonprogrammable unit such as the ASICs is also used to implement as PE.

By using this entire processing element, it forms bus topology. It is also possible to form other topology also. It is also possible that the system can use more than one network, such as when relatively independent functions require relatively little communication among them.

All the processing elements are connected by communication link. The system of PEs and networks forms the hardware platform on which the application runs.

Why Distributed

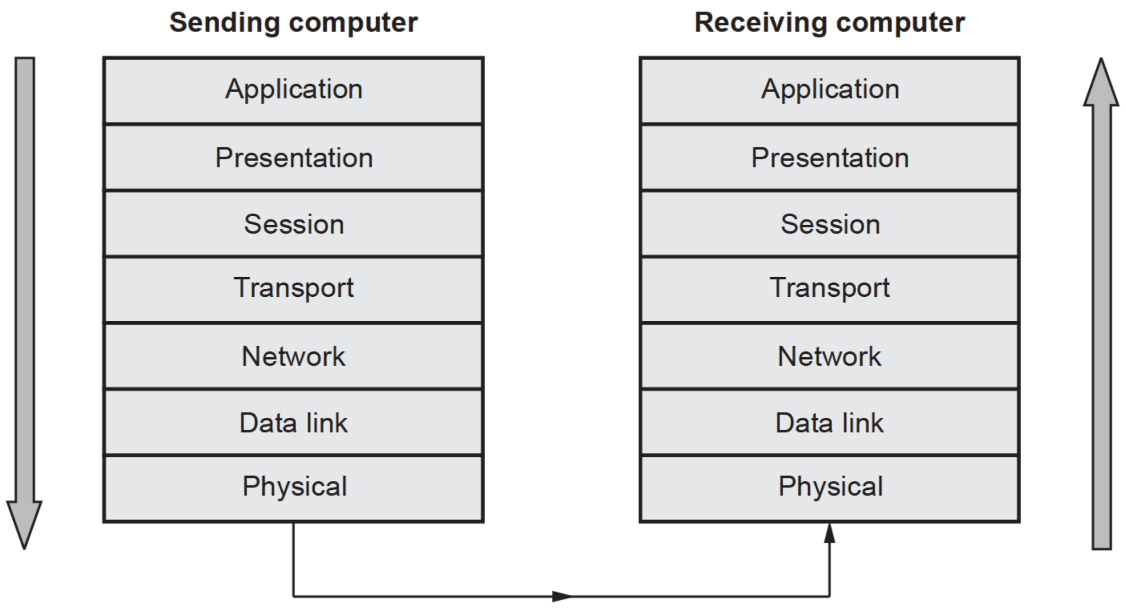
- Higher performance at lower cost.
- Physically distributed activities, i.e. time constants may not allow transmission to central site.
- Improved debugging : use one CPU in network to debug others.
- May buy subsystems that have embedded processors.
- Distributed systems are necessary because the devices that the PEs communicate with are physically separated.

Network Abstractions

- Networks are complex systems. They provide high-level services while hiding many of the details of data transmission from the other components in the system.
- To understand network design, the International Standards Organization has developed a seven-layer model for networks known as Open Systems Interconnection (OSI) models.
- It is based on a common model of network architecture and a suite of protocols used in its implementation. The International Organization for Standardization (ISO) established the Open Systems Interconnection (OSI) Reference Model.
- Each layer deals with a particular aspect of network communication. There are seven layers in the model, hence the name the 7-Layer model. The model acts as a frame of reference in the design of communications and networking products.

7. Application layer
6. Presentation layer
5. Session layer
4. Transport layer
3. Network layer
2. Data link layer
1. Physical layer
OSI layers

The OSI model describes how information or data makes its way from application programmers through a network medium to another application programmer located on another network. The OSI reference model is a hierarchical structure. Changes in one layer should not require changes in other layers.



Flow of data

1. **Physical layer :** The lowest layer of the OSI model. It is concerned with the transmission and reception of the unstructured raw bit stream over a physical medium. It describes the electrical/optical, mechanical and functional interfaces to the physical medium, and carries the signals for the entire higher layer.

2. **Data link layer :** DLL is responsible for the transfer of data over the channel. It groups zeros and ones into frames. A frame is a series of bits that forms a unit of data. The data link layer provides error-free transfer of data frames from one node to another over the physical layer. It contains two sublayers : Medium Access Control (MAC) and Logical Link Control Layer (LLC). DLL divides the bit stream of the physical layer into frames, messages containing data and control information. It handles lost, damaged and duplicate frames.

3. **Network layer :** This is responsible for addressing messages and data so they are sent to the correct destination, and for translating logical addresses and names into physical addresses. This layer is also responsible for finding a path through the network to the destination computer. Lowest layer that deals with host-to-host communication, call this end-to-end

communication. Functions of network layer : Logical addressing, Routing and Frame fragmentation

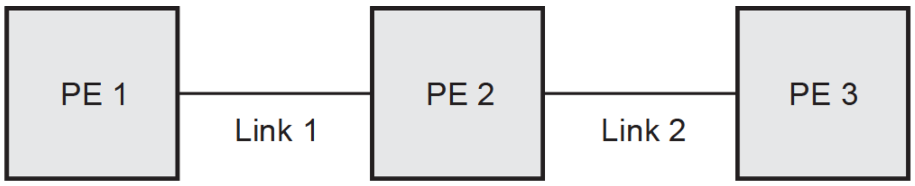
4. **The transport layer** ensures that messages are delivered error-free, in sequence, and with no losses or duplications. It relieves the higher layer protocols from any concern with the transfer of data between them and their peers. It also provides flow control, sequence numbering and message acknowledgement.Function of transport layer : Message segmentation, Message acknowledgment, Session multiplexing
5. The **session layer** adds mechanisms to establish, maintain, synchronize and manage communication between network entities. The session layer allows session establishment between processes running on different stations. Services provided by session layer : Synchronization and dialog management.
6. The **presentation layer** is responsible for data compression, data expansion, data encryption and data decryption.
7. **Application layer** : It contains all services or protocols needed by application software or operating system to communicate on the network. Typical applications include a client/server application, an e-mail and an application to transfer files using FTP or HTTP.

Hardware and Software Architectures

Distributed embedded systems can be organized in many different ways depending upon the needs of the application and cost constraints.

Point-to-point :

Point-to-point link establishes a connection between exactly two PEs. Point-to-point links are simple to design precisely because they deal with only two components.



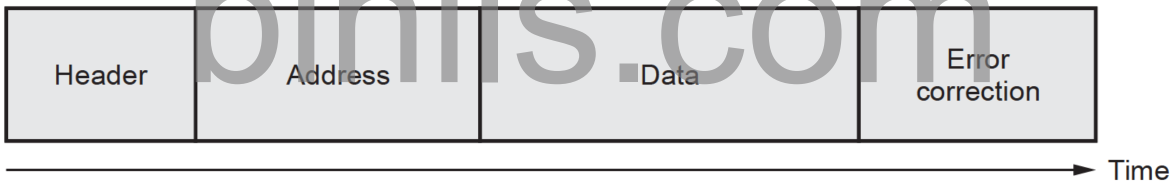
Signal processing system built from print-to-point links

Input device sampled the signal and passed to the first digital filter (Fl) by using point-to-point link. The results of that filter are sent through a second point-to-point link to filter (F2). The results in turn are sent to the output device over a third point-to-point link.

Filters must process their inputs in a timely fashion. It is possible to build full duplex point-to-point distributed system.

A bus is a more general form of network since it allows multiple devices to be connected to it. Like a microprocessor bus, PEs connected to the bus have addresses.

Communication between processing elements takes place by using packets.



Format of packet

Packet contains destination address, user data and error correction codes. Sending data size is not exactly fit into packet but processing elements must take care of packet

The data to be transmitted from one PE to another may not fit exactly into the size of the data payload on the packet. It is the responsibility of the transmitting PE to divide its data into packets; the receiving PE must of course reassemble the complete data message from the packets.

Arbitration scheme

The device that is allowed to initiate transfers on the bus at any given time is called the bus master

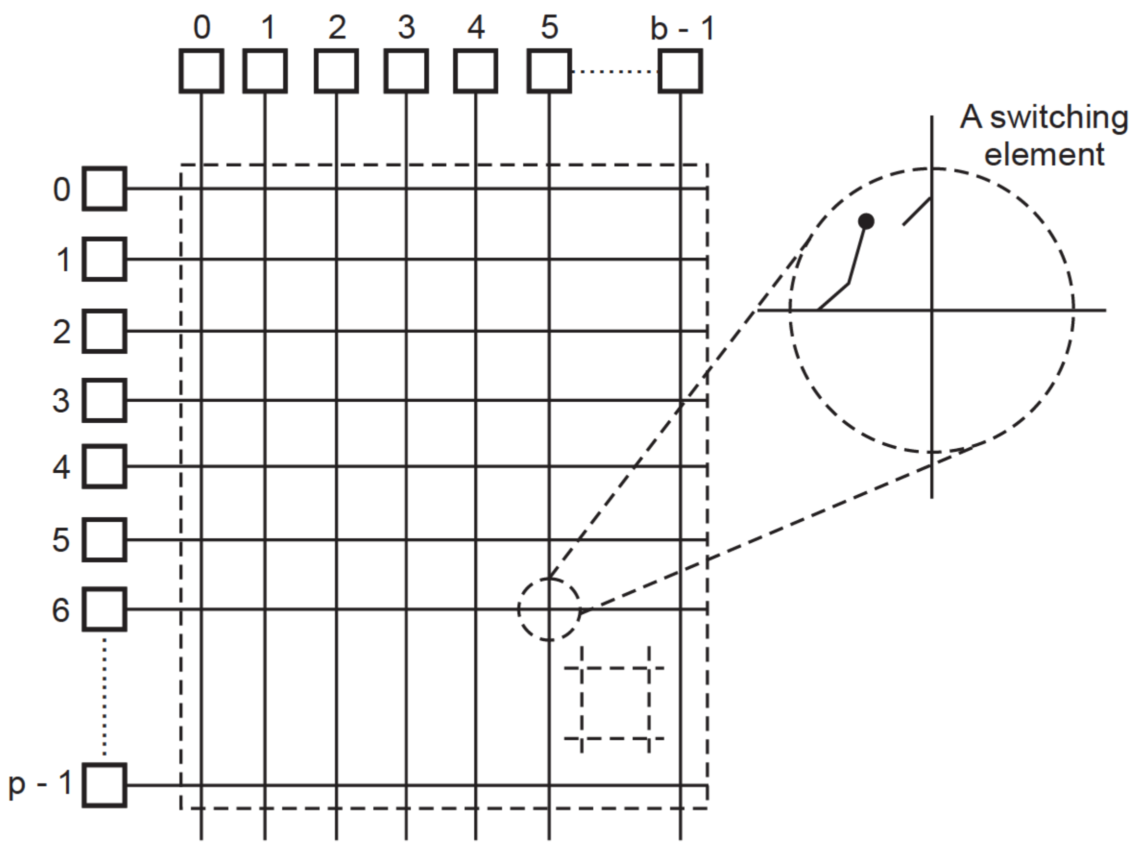
1. **Fixed-priority arbitration :** High priority devices always get chance to transmit data. If low priority device and high priority device are ready to transmit data, then high priority device will transmit data then low priority device will transmit data.
2. **Fair arbitration schemes :** This scheme take care of starvation. Round-robin arbitration is the most commonly used of the fair arbitration schemes. The PCI bus requires that the arbitration scheme used on the bus must be fair, although it does not specify a particular arbitration scheme. Most implementations of PCI use round-robin arbitration.

Crossbar network

A bus topology provides limited available bandwidth. Since all devices connect to the bus, communications can interfere with each other. For reducing communication conflicts, other network topology can be used.

A crossbar network uses an p x m grid of switches to connect p inputs to m outputs in a non-blocking manner. A connection of one processor to a given memory bank does not block a connection of another processor to a different memory bank. There must be p x b switches. It is reasonable to assume that b>p.

A link carries one message; a switch can process up to two messages at the same time. Crossbar is not fault tolerant, failure of any switchbox will disconnect certain pairs.



Direct network crossbar

Message Passing Programming

A message-passing system is a subsystem of distributed operating system that provides a set of message-based IPC protocols, and does so by shielding the details of complex network protocols and multiple heterogeneous platforms from programmers. It enables processes to communicate by exchanging messages and allows programs to be written by using simple communication primitives, such as send and receive.

Transport layer provides message-based programming interface : send_msg (adrs, datal);

Data must be broken into packets at source, reassembled at destination.

Data-push programming : receivers respond to new data.

MPSoC's and Shared Memory Multiprocessors

Single processors may be sufficient for low-performance applications that are typical of early microcontrollers, but an increasing number of applications require multiprocessors to meet their performance goals.

Multiprocessor Systems-on-Chips (MPSoC) are one of the key applications of today. MPSoC are increasingly used to build complex integrated system. A MPSoC is more than just a rack of processors shrunk down to a single chip.

Definition : Multiprocessor is parallel processors with a single shared address.

Microprocessor is now the most cost-effective processor. Multiprocessors have the highest absolute performance-faster than the fastest uniprocessor.

Parallel processing program is a single program that runs on multiple processors simultaneously.

Cluster is a set of computers connected over a Local Area Network (LAN) that function as a single large multiprocessor.

Shared memory is a memory for a parallel processor with a single address space, implying implicit communication with loads and stores.

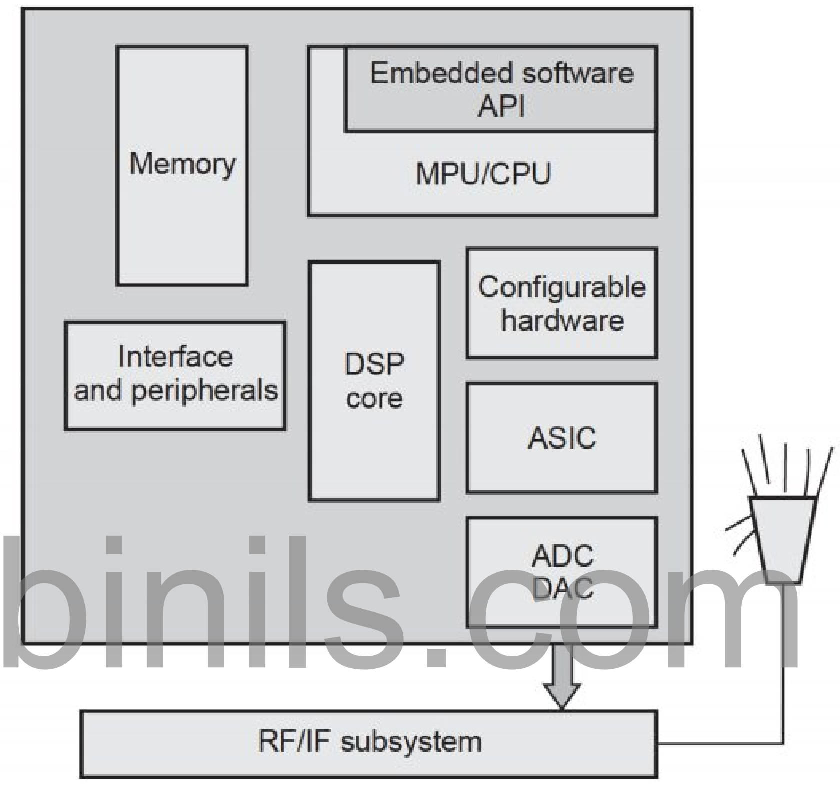
The typical MPSoC is a heterogeneous multiprocessor : there may be several different types of PEs, the memory system may be heterogeneously distributed around the machine, and the interconnection network between the PEs and the memory may also be heterogeneous

MPSoCs often require large amounts of memory. The device may have embedded memory on-chip as well as relying on off-chip commodity memory.

System-on-Chip (SoC) designs increasingly become the driving force of a number of modem electronics systems. Conceptually System on Chip refers to integrating the components of a board onto a single chip. Looks straightforward but productivity levels are too low to make it a reality.

The SoC chip includes : Embedded processor, ASIC logics and analog circuitry and Embedded memory.

The SoC software includes : OS, compiler, simulator, firmware, driver, protocol stack Integrated development environment (debugger, linker, ICE) application interface (C/C++, assembly).

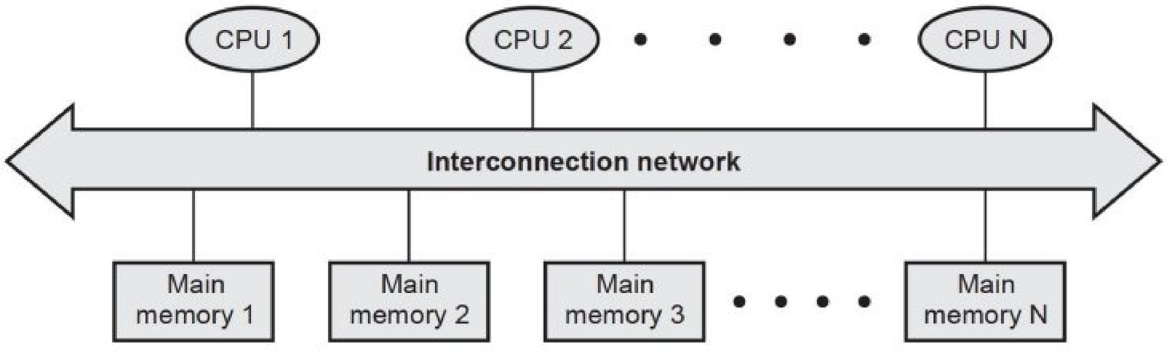


System on chip

1. Memory controller : Interfaces with the onboard RAM.
2. DMA : Handles the automated transfers of data between the RAM and memory-mapped hardware.
3. USB controller : Manages the hardware side of the device's USB connections.
4. DSP core : It provides hardware acceleration for some signal processing, such as JPEG encoding.
5. Display : Enables the SoC to drive various display types.
6. Camera : Allows the SoC to interface with a camera.

7. Storage : Manages I/O with the various types of storage that can be used with the SoC.
8. Debug : Enables the SoC to be connected to hardware debugging tools through various mechanisms, such as JTAG.

It consists of a pool of processors and a pool of memory are connected by an interconnection network.



Shared memory

A shared-memory model is often preferred because it makes life simpler for the programmer. The Raw architecture is a recent example of a regular architecture designed for high-performance computation

Signal address : Offer the programmer a single memory address space that all processors share. Processors communicate through shared variables in memory, with all processors capable of accessing any memory location via loads and stores.

Message passing : Communicating between multiple processors by explicitly sending and receiving information.

Heterogeneous memory systems : Some blocks of memory may be accessible by only one or a few processors. Heterogenous memory systems are harder to program because the programmer must keep in mind what processors can access what memory blocks

Irregular memory structures are often necessary in MPSoCs. One reason that designers resort to specialized memory is to support real-time performance.

Challenges and Opportunities

MPSoCs combine the difficulties of building complex hardware systems and complex software systems.

Methodology is critical to MPSoC design. Methodologies that work offer many advantages. They decrease the time it takes to design a system; they also make it easier to predict how long the design will take and how many resources it will require. Methodology also modify techniques for improving performance and power consumption that developers can apply to many different designs.

Methodology will necessarily be a moving target for the next decade.

MPSoC hardware architectures present challenges in all aspects of the multiprocessor : processing elements, memory and interconnects.

Configurable processors with customized instruction sets are one way to improve the characteristics of processing elements; hardware/software code sign of accelerators is another technique.