e-PG Pathshala
Subject: Computer Science
Paper: Embedded System
Module: RTOS: Scheduling policies-2
Module No: CS/ES/26
Quadrant 1 – e-text

In this lecture concepts of priority based scheduling and its types will be discussed. Then EDF and RMS scheduling algorithms will be explained in detail in this lecture. Inter process communication will also be discussed in detail.

1.1 Priorities in scheduling

In priority scheduling, an OS kernel determines which process is to be executed next. For each task a numerical priority is assigned. The kernel can simply look at the processes and their priorities. It sees which one actually wants to execute and selects the highest priority process that is ready to run.This mechanism is flexible and fast.

1.2. Priority-driven scheduling

Priority-driven scheduling is easy to implement. It does not require the prior information on the release times and execution times of the jobs. Each process has a fixed priority that does not vary during the course of execution.The ready process with the highest priority is selected for execution. A process continues execution until it completes or it is pre-empted by a higher-priority process. Figure 1 shows an example for priority process. Consider there are three processes P1 , P2 and P3 with the execution time of 10,20 and 30 time units respectively. In this example, P1 is the highest priority; P2 is having the middle priority and P3 is the lower priority process in order.

Process	Priority	Execution time
P1	1	10
P2	2	30
P3	3	20

Figure 1. Priority of Processes

An example of Priority scheduling for this set of processes is shown in Figure 2. Assume that P2 is ready to run. When the system is started, P1 is released at time 15, and P3 is released at time 18. P2 is the only ready process, so it is selected for execution. At time 15, P1 becomes ready; it pre-empts P2 and begins execution since it has a higher priority. Since P1 is the highest-priority process in the system, it is guaranteed to execute until it finishes. P3's data arrives at time 18, but it cannot pre-empt P1. Even when P1 finishes, P3 is not allowed to run. P2 is still ready and has higher priority than P3. P3 starts only when both the process P1 and P2 are finished. Priority inversion problem may occur if priority scheduling is used.

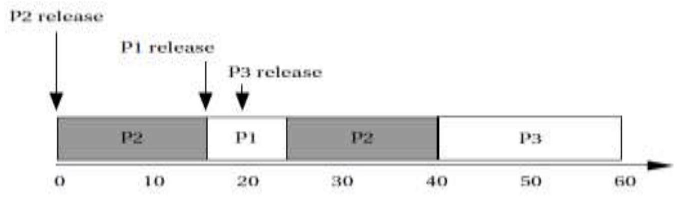


Figure 2. Priority Scheduling

1.3 Priority inversion problem

Scheduling the processes without considering the resources those processes require, can cause priority inversion, in which a low-priority process blocks execution of a higher priority process by keeping hold of its resources. Priority inversion problem occurs commonly in real time kernels. Example : Consider task 1 has a higher priority than task 2 and task 2 has a higher priority than task 3. Assume task 1 and task 3 share a resource through mutual exclusion. While task 3 is executing and holding the resource if task 2 is ready, it is scheduled because it has higher priority. At this time, even though task 1 has higher priority it cannot execute because the blocked task 3 is holding the shared resource. That is, a lower priority process is blocking a higher priority process. This is the priority inversion problem.

1.3.1 A Solution to priority inversion problem

We can correct the problem by raising the priority of task 3, just for the time when it accesses the shared resource. After that, the task 3 returns to its original priority. If task 3 finishes the access before being preempted by task 1 then it incurs overhead for nothing. A better solution to priority inversion problem is priority inheritance.

Priority Inheritance

It automatically changes the task priority when needed. That is, the task that holds the resource will inherit the priority of the task that waits for that resource until it releases the resources. Once the priorities are assigned, the OS takes care of the rest by choosing the highest-priority ready process.

2. Assigning Task priorities

There are two major ways to assign priorities as explained below.

- **Static priorities-** The priorities of the task that do not change during execution are called as static priorities. Once the Priority of the task is assigned, its value is retained till the end or completion of task.
  Example: Rate Monotonic Scheduling (RMS)
- **Dynamic priorities**- The priorities of the task that are dynamically changing during the execution are called as dynamic priorities. These priorities will change at each and every instant of time based on the current scenario.
  Example: Earliest Deadline First (EDF)

Let us discuss the RMS and EDF scheduling in detail.

2.1 Rate-Monotonic Scheduling (RMS)

RMS is one of the first scheduling policies developed for real-time systems and is still widely used. It is a static scheduling policy where only the fixed priorities are sufficient to efficiently schedule the processes in many situations.

Theory:

The theory underlying RMS is known as rate-monotonic analysis(RMA). This theory summarized below uses a relatively simple model of the system.

- All processes run periodically on a single CPU.

- Context switching time is ignored.
- There are no data dependencies between processes.
- The execution time for a process is constant.
- All deadlines are at the ends of their periods.
- The highest-priority ready process is always selected for execution.

The major result of RMA is that a relatively simple scheduling policy is optimal under certain conditions. Priorities are assigned by rank order of period, i.e. process with the shortest period is assigned the highest priority. It provides the highest CPU utilization while ensuring that all processes meet their deadlines.

2.1.1 Example of Rate Monotonic Scheduling

Suppose P1 has the highest priority, P2 the middle priority, and P3 the lowest priority. Then all periods start at time zero, as per RMS execution. This is explained in Figure 4 shown below.

Process	Execution time	Period
P1	1	4
P2	2	6
P3	3	12

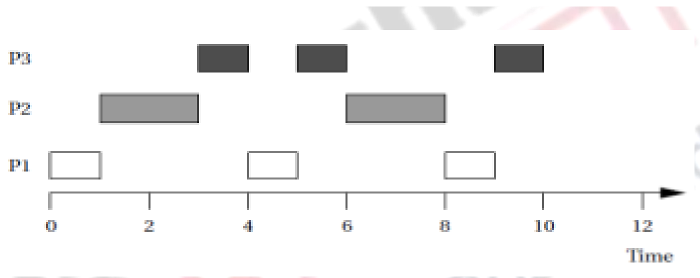
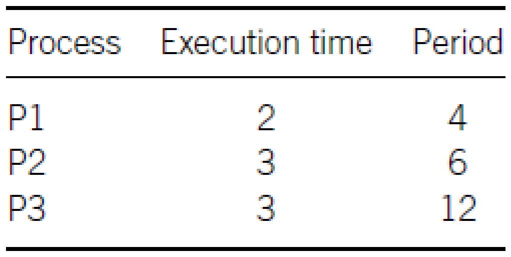


Figure 4: Rate Monotonic Scheduling

All three periods starts at time zero. P1's data arrives first. Since P1 is the highest-priority process, it can start to execute immediately. After one-time unit, P1 finishes and goes out of the ready state until the start of its next period. At time 1, P2 starts executing as the highest-priority ready process. At time 3, P2 finishes and P3 starts executing. P1's next iteration starts at time 4, at which point it interrupts P3. P3 gets one more time unit of execution between the second iterations of P1 and P2, but P3 does not get to finish until after the third iteration of P1.

Consider a different set of execution times for the three processes, keeping the same deadlines as shown in figure 5. Each process alone has an execution time significantly less than its period. The combinations of processes can require more than 100% of the available CPU cycles. For example, during one 12 time-unit interval, we must execute P1 three times, requiring 6 units of CPU time; P2 twice, costing 6 units of CPU time; and P3 one time, requiring 3 units of CPU time. The total of 6 + 6 + 3 = 15 units of CPU time which is more than the 12 time units available, clearly exceeding the available CPU capacity.



Process	Execution time	Period
P1	2	4
P2	3	6
P3	3	12

Figure 5: RMS with different execution times

2.1.2 RM – Utilization Bound:

Real-time system is schedulable under RM if

$\sum U_i \leq n (2^{1/n}-1)$, where U is the utilization and n refers to the total number of tasks.

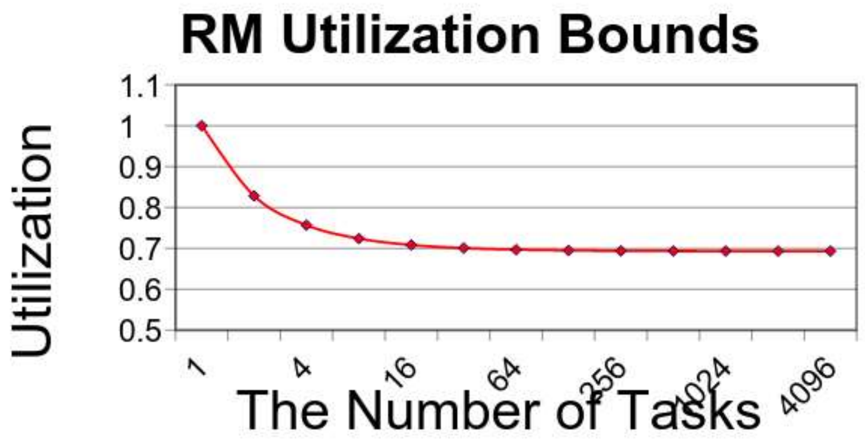


Figure 6 : Calculation of RM Utilization Bounds.

Example: Consider there are three tasks $T_1(1,4)$, $T_2(1,5)$, $T_3(1,10)$. The calculation of RM Utilization is as follows:

. The task is represented by T (e,p) where p is the inter release time and e is the maximum execution time.

The utilization is given by U = e/p.

$\sum U_i = 1/4 + 1/5 + 1/10 = 0.55$

Real-time system is schedulable under RM if

$\sum U_i <= n(2^{1/n} -1)$, where n is the number of task.

$3 (2^{1/3}-1) \approx 0.78$

Thus, $\{T_1, T_2, T_3\}$ is schedulable under RM. Example of RM utilization bound is shown in Figure 6.

Deadline miss with RM

Consider scheduling the following task set T1 = (1,4) , T2 = (2,6) and T4 = (3,8) (figure 7).

Utilization bound (U) = ¼ + 2/6 + ⅜ = 23/24.

The utilization is greater than the bound. Hence there is a deadline miss. Observe that at time 6, even if the deadline of task T3 is very close, the scheduler decides to schedule task T2. This is the main reason why T3 misses its deadline.

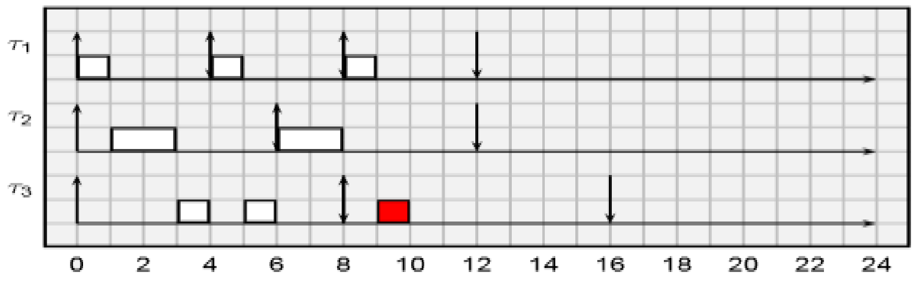


Figure 7 : Deadline miss with RM

EDF Scheduling is preferred to avoid this kind of situation. Let us discuss EDF in detail.

2.2 Earliest-Deadline-First(EDF) Scheduling

An important class of scheduling algorithms is the class of dynamic priority algorithms. In dynamic priority algorithms, the priority of a task can change during its execution based on the initiation times. Fixed priority algorithms are a subclass of the more general class of dynamic priority algorithms, the priority of a task does not change.The most important (and analyzed) dynamic priority algorithm is Earliest Deadline First (EDF). The priority of a job (instance) is inversely proportional to its absolute deadline. In other words, the highest priority job is the one with the earliest deadline. If two tasks have the same absolute deadlines, chose 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.It achieves higher CPU utilizations than RMS. The highest-priority process is the one whose deadline is nearest in time, and the lowest priority process is the one whose deadline is farthest away. Priorities must be recalculated at the completion of every process. Final procedure is same as RM.

2.2.1 Example 1: Scheduling with EDF

Now we schedule the same task (shown in Fig. 7) with EDF.

T1 = (1,4) , T2 = (2,6) and T3 = (3,8)

U = ¼ + 2/6 + ⅜ =23/24

Again the Utilization is very high. However, there is no deadline miss in the hyperperiod. (The hyperperiod is the smallest interval of time after which the periodic patterns of all the tasks is repeated).

Observe that at time 6, the problem does not appear, as the earliest deadline job (the one of T3) is executed as shown in figure 8. It demonstrates how the deadline miss is avoided in EDF scheduling in the above example.

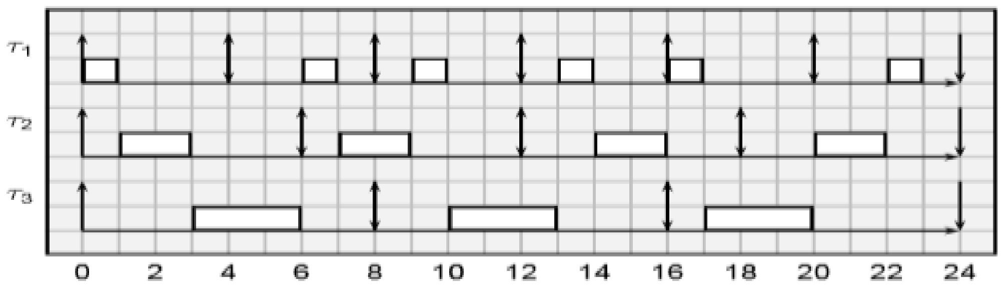


Figure 8. EDF Scheduling with no deadline miss

2.2.2. Example 2

Optimal dynamic priority scheduling. A task with a shorter deadline has a higher priority, i.e., it executes a job with the earliest deadline. It is an Optimal scheduling algorithm. If there is a schedule for a set of real-time tasks, EDF can schedule it. This is explained in Figure 9 shown below. It demonstrates how three tasks T1(1,4) ,T2(2,5) and T3(2,7) scheduled for 15 periods efficiently using earliest deadline first scheduling. There is no deadline miss and it effectively utilizes the time periods.

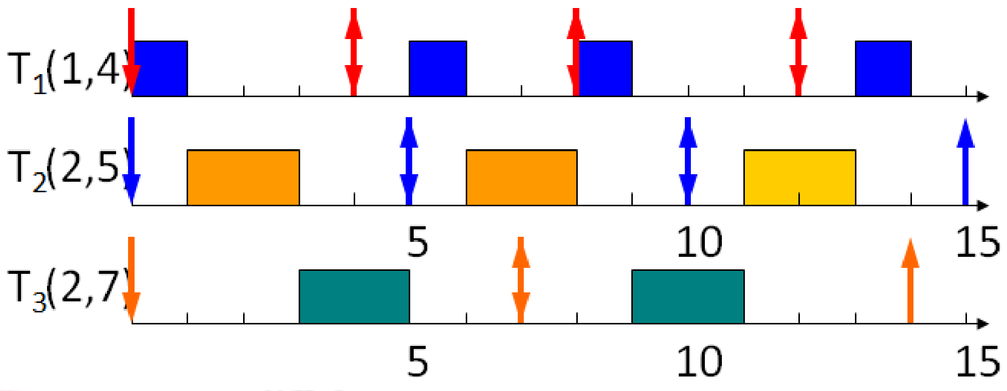


Figure 9 : Earliest Deadline First Scheduling

2.2.3 EDF-Overload conditions

A disadvantage of EDF is that Domino effect occurs during overload conditions. This is explained below in the figure 10 with the example: $T_1(3,4)$, $T_2(3,5)$, $T_3(3,6)$, $T_4(3,7)$. The hyperperiod given is 7, but actually 12 (3+3+3+3) time slots are needed to complete all processes at least once. Hence deadline miss occurs. This overload condition is known as the domino effect. In Figure 10, first the three tasks T2,T3 and T4 are having deadline miss. If we schedule T3 after T1 or T4 after T1, only 2 tasks will get deadline miss. When compared to the first schedule, these two are better schedules even though they have deadline miss.

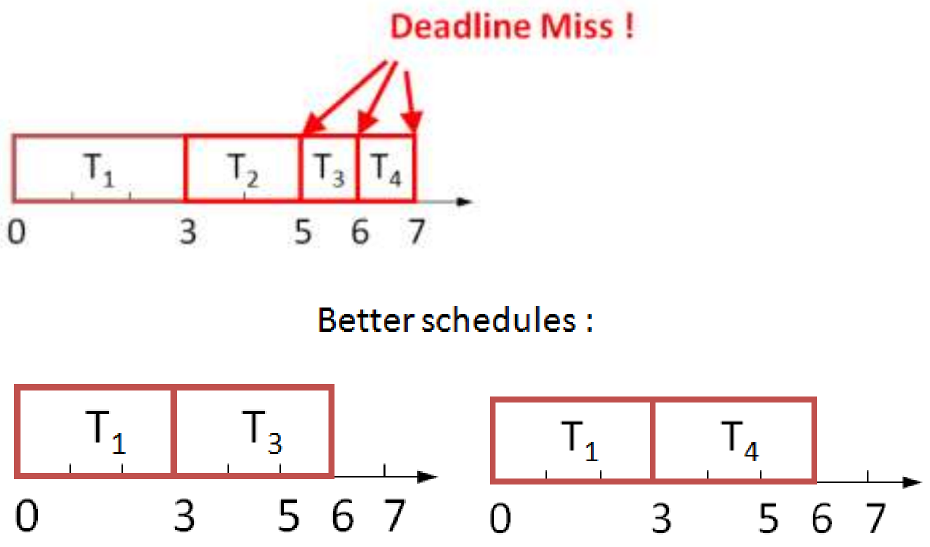
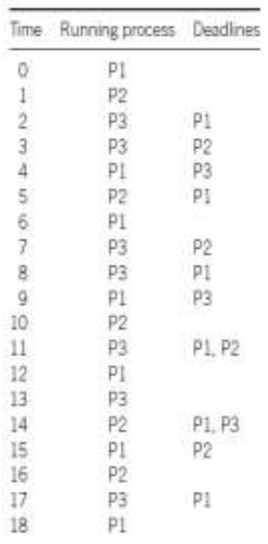


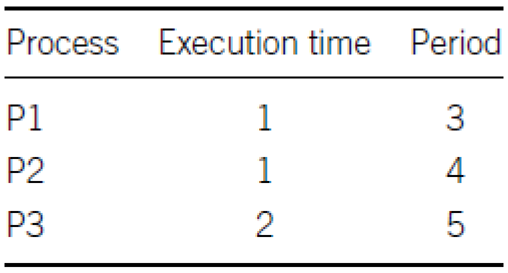
Figure 10 : EDF overloading

2.2.4 Example 3

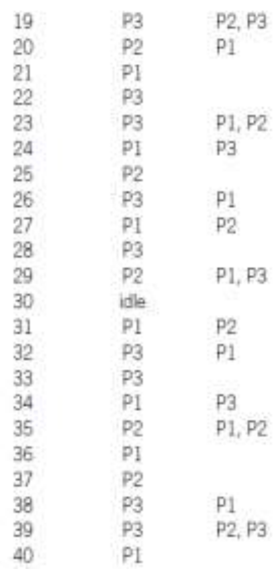
In EDF, priorities are assigned in order of deadline. The highest priority process has deadline nearest in time and the lowest priority process is having deadline which is farthest away. The priorities are recalculated during every completion of process and the highest priority ready process is chosen for execution. This is another example to demonstrate EDF Scheduling for 60 time slots, which is shown in Table 1.



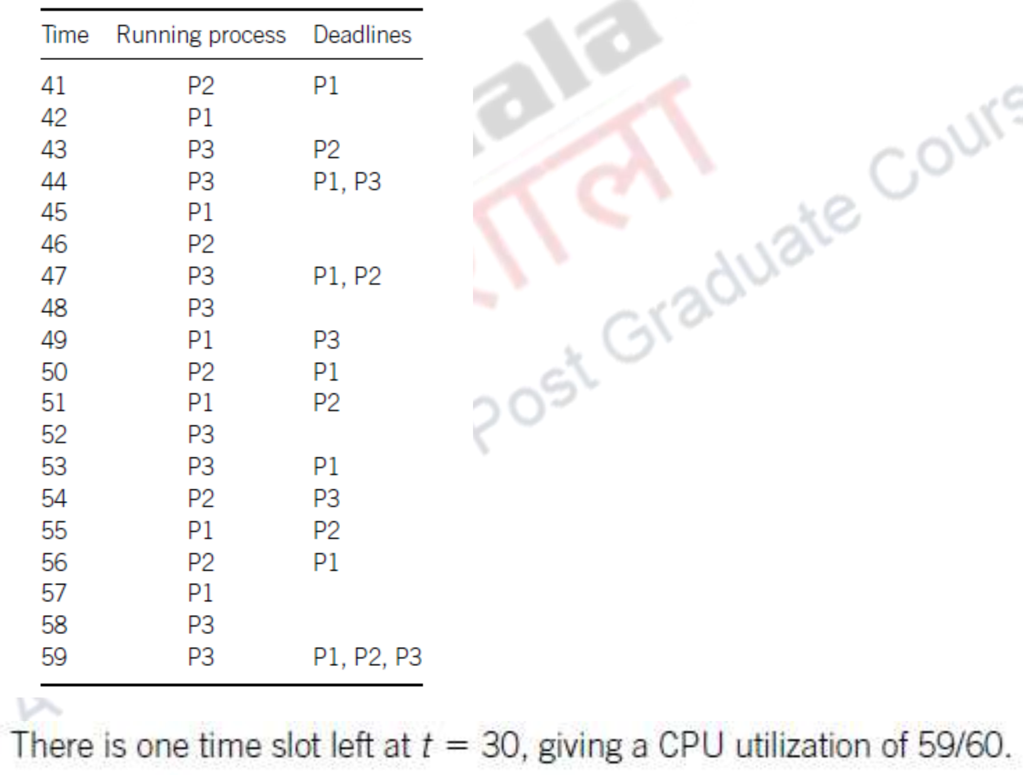
Time	Running process	Deadlines
0	P1	
1	P2	
2	P3	P1
3	P3	P2
4	P1	P3
5	P2	P1
6	P1	
7	P3	P2
8	P3	P1
9	P1	P3
10	P2	
11	P3	P1, P2
12	P1	
13	P3	
14	P2	P1, P3
15	P1	P2
16	P2	
17	P3	P1
18	P1	



Process	Execution time	Period
P1	1	3
P2	1	4
P3	2	5



19	P3	P2, P3
20	P2	P1
21	P1	
22	P3	
23	P3	P1, P2
24	P1	P3
25	P2	
26	P3	P1
27	P1	P2
28	P3	
29	P2	P1, P3
30	idle	
31	P1	P2
32	P3	P1
33	P3	
34	P1	P3
35	P2	P1, P2
36	P1	
37	P2	
38	P3	P1
39	P3	P2, P3
40	P1	



Time	Running process	Deadlines
41	P2	P1
42	P1	
43	P3	P2
44	P3	P1, P3
45	P1	
46	P2	
47	P3	P1, P2
48	P3	
49	P1	P3
50	P2	P1
51	P1	P2
52	P3	
53	P3	P1
54	P2	P3
55	P1	P2
56	P2	P1
57	P1	
58	P3	
59	P3	P1, P2, P3

There is one time slot left at $t = 30$, giving a CPU utilization of 59/60.

Table 1. Hyper Period calculation using EDF algorithm

2.3 RMS vs. EDF

The following are some of the differences between RMS and EDF priority scheduling algorithms.

Rate Monotonic

- Simpler implementation, even in systems without explicit support for timing constraints (periods, deadlines).
- Predictability for the highest priority tasks.

EDF

- Full processor utilization.
- Misbehavior during overload conditions.

3. Interprocess Communication Mechanisms

Processes need communication, this is done by using Inter-process communication.This mechanism is provided by the operating system. The processes can communicate in two ways. They are blocking and non-blocking methods and are explained as follows.

Blocking: Process enters waiting state until it gets response for a communication it has sent.

Non-blocking: Continues execution after sending communication.

Two major styles of Interprocess communications are shared memory and message passing which are discussed in detail below.

Shared memory: Two components, such as a CPU and an I/O device, communicate through a shared memory location. The software on the CPU is designed to know the address of the shared location; the shared location is also loaded into the proper register of the I/O device. This is shown in figure 11.

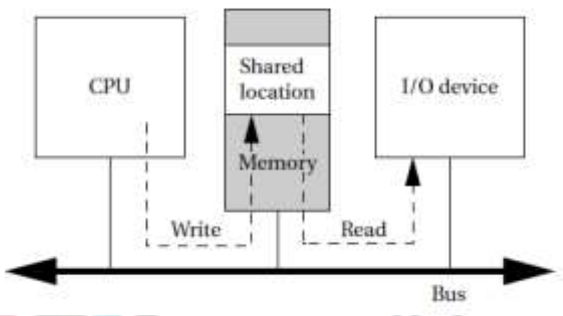


Figure 11 : Shared memory Communication

If the CPU wants to send data to the device, it writes to the shared location.The I/O device then reads the data from that location.

3.1 Message Passing:

Each communicating entity has its own message send/receive unit.The message is not stored on the communications link, but rather at the senders/ receivers at the end points. In contrast, shared memory communication can be seen as a memory block used as a communication device, in which all the data are stored in the communication link/memory. A message passing example is shown in Figure 12.

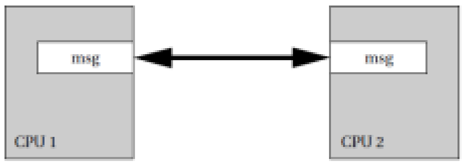


Figure 12 : Message Passing

3.2 Signalling

An UNIX inter-process communication is done through the signal. A signal is a one bit output from a process for Inter Process Communication. It is generated by a process and transmitted to another process by the OS. An advantage of using signal is that it uses the shortest possible CPU time. It is the software equivalent of the flag at a register that is set on a hardware interrupt.

4. Summary

In this module priority based scheduling is discussed. Then RMS and EDF priority based scheduling algorithms are discussed along with their advantages and limitations. After that Inter process communication is also discussed.

5. References

1. Wayne Wolf, “Computers as Components: Principles of Embedded Computing System Design”, 2008.
2. Insup Lee, “Real-Time Scheduling”, Lecture7-2005.
3. http://feanor.sssup.it/~lipari.
4. Buttazzo, “Rate monotonic vs. EDF: Judgement Day”, EMSOFT 2003.
5. Liu & Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment”, Journal of ACM, 1973.