INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATION AND MANAGEMENT

## CONTENTS

# CALL FOR MANUSCRIPTS

We invite unpublished novel, original, empirical and high quality research work pertaining to recent developments & practices in the area of Computer, Business, Finance, Marketing, Human Resource Management, General Management, Banking, Insurance, Corporate Governance and emerging paradigms in allied subjects like Accounting Education; Accounting Information Systems; Accounting Theory & Practice; Auditing; Behavioral Accounting; Behavioral Economics; Corporate Finance; Cost Accounting; Econometrics; Economic Development; Economic History; Financial Institutions & Markets; Financial Services; Fiscal Policy; Government & Non Profit Accounting; Industrial Organization; International Economics & Trade; International Finance; Macro Economics; Micro Economics; Monetary Policy; Portfolio & Security Analysis; Public Policy Economics; Real Estate; Regional Economics; Tax Accounting; Advertising & Promotion Management; Business Education; Business Information Systems (MIS); Business Law, Public Responsibility & Ethics; Communication; Direct Marketing; E-Commerce; Global Business; Health Care Administration; Labor Relations & Human Resource Management; Marketing Research; Marketing Theory & Applications; Non-Profit Organizations; Office Administration/Management; Operations Research/Statistics; Organizational Behavior & Theory; Organizational Development; Production/Operations; Public Administration; Purchasing/Materials Management; Retailing; Sales/Selling; Services; Small Business Entrepreneurship; Strategic Management Policy; Technology/Innovation; Tourism, Hospitality & Leisure; Transportation/Physical Distribution; Algorithms; Artificial Intelligence; Compilers & Translation; Computer Aided Design (CAD); Computer Aided Manufacturing; Computer Graphics; Computer Organization & Architecture; Database Structures & Systems; Digital Logic; Discrete Structures; Internet; Management Information Systems; Modeling & Simulation; Multimedia; Neural Systems/Neural Networks; Numerical Analysis/Scientific Computing; Object Oriented Programming; Operating Systems; Programming Languages; Robotics; Symbolic & Formal Logic; Web Design. The above mentioned tracks are only indicative, and not exhaustive.

Anybody can submit the soft copy of his/her manuscript **anytime** in M.S. Word format after preparing the same as per our submission guidelines duly available on our website under the heading guidelines for submission, at the email addresses, infoijrcm@gmail.com or info@ijrcm.org.in.

# GUIDELINES FOR SUBMISSION OF MANUSCRIPT

1. **COVERING LETTER FOR SUBMISSION:**

**DATED: _____________**

***THE EDITOR***

IJRCM

Subject: **SUBMISSION OF MANUSCRIPT IN THE AREA OF ____________________________________________.**

**(e.g. Computer/IT/Finance/Marketing/HRM/General Management/other, please specify).**

**DEAR SIR/MADAM**

Please find my submission of manuscript titled '___________________________________________' for possible publication in your journal.

I hereby affirm that the contents of this manuscript are original. Furthermore it has neither been published elsewhere in any language fully or partly, nor is it under review for publication anywhere.

I affirm that all author (s) have seen and agreed to the submitted version of the manuscript and their inclusion of name (s) as co-author (s).

Also, if our/my manuscript is accepted, I/We agree to comply with the formalities as given on the website of journal & you are free to publish our contribution to any of your journals.

**NAME OF CORRESPONDING AUTHOR**:

Designation:

Affiliation with full address & Pin Code:

Residential address with Pin Code:

Mobile Number (s):

Landline Number (s):

E-mail Address:

Alternate E-mail Address:

2. **INTRODUCTION**: Manuscript must be in British English prepared on a standard A4 size paper setting. It must be prepared on a single space and single column with 1" margin set for top, bottom, left and right. It should be typed in 8 point Calibri Font with page numbers at the bottom and centre of the every page.
3. **MANUSCRIPT TITLE**: The title of the paper should be in a 12 point Calibri Font. It should be bold typed, centered and fully capitalised.
4. **AUTHOR NAME(S) & AFFILIATIONS**: The author (s) full name, designation, affiliation (s), address, mobile/landline numbers, and email/alternate email address should be in italic & 11-point Calibri Font. It must be centered underneath the title.
5. **ABSTRACT**: Abstract should be in fully italicized text, not exceeding 250 words. The abstract must be informative and explain the background, aims, methods, results & conclusion in a single para.
6. **KEYWORDS**: Abstract must be followed by list of keywords, subject to the maximum of five. These should be arranged in alphabetic order separated by commas and full stops at the end.
7. **HEADINGS**: All the headings should be in a 10 point Calibri Font. These must be bold-faced, aligned left and fully capitalised. Leave a blank line before each heading.
8. **SUB-HEADINGS**: All the sub-headings should be in a 8 point Calibri Font. These must be bold-faced, aligned left and fully capitalised.
9. **MAIN TEXT**: The main text should be in a 8 point Calibri Font, single spaced and justified.
10. **FIGURES &TABLES**: These should be simple, centered, separately numbered & self explained, and titles must be above the tables/figures. Sources of data should be mentioned below the table/figure. It should be ensured that the tables/figures are referred to from the main text.
11. **EQUATIONS**: These should be consecutively numbered in parentheses, horizontally centered with equation number placed at the right.
12. **REFERENCES**: The list of all references should be alphabetically arranged. It must be single spaced, and at the end of the manuscript. The author (s) should mention only the actually utilised references in the preparation of manuscript and they are supposed to follow **Harvard Style of Referencing**. The author (s) are supposed to follow the references as per following:

- All works cited in the text (including sources for tables and figures) should be listed alphabetically.
- Use (**ed.**) for one editor, and (**ed.s)** for multiple editors.
- When listing two or more works by one author, use --- (20xx), such as after Kohl (1997), use --- (2001), etc, in chronologically ascending order.
- Indicate (opening and closing) page numbers for articles in journals and for chapters in books.
- The title of books and journals should be in italics. Double quotation marks are used for titles of journal articles, book chapters, dissertations, reports, working papers, unpublished material, etc.
- For titles in a language other than English, provide an English translation in parentheses.
- The location of endnotes within the text should be indicated by superscript numbers.

**PLEASE USE THE FOLLOWING FOR STYLE AND PUNCTUATION IN REFERENCES:**

**BOOKS**

- Bowersox, Donald J., Closs, David J., (1996), "Logistical Management." Tata McGraw, Hill, New Delhi.
- Hunker, H.L. and A.J. Wright (1963), "Factors of Industrial Location in Ohio," Ohio State University.

**CONTRIBUTIONS TO BOOKS**

- Sharma T., Kwatra, G. (2008) Effectiveness of Social Advertising: A Study of Selected Campaigns, Corporate Social Responsibility, Edited by David Crowther & Nicholas Capaldi, Ashgate Research Companion to Corporate Social Responsibility, Chapter 15, pp 287-303.

**JOURNAL AND OTHER ARTICLES**

- Schemenner, R.W., Huber, J.C. and Cook, R.L. (1987), "Geographic Differences and the Location of New Manufacturing Facilities," Journal of Urban Economics, Vol. 21, No. 1, pp. 83-104.

**CONFERENCE PAPERS**

- Garg Sambhav (2011): "Business Ethics" Paper presented at the Annual International Conference for the All India Management Association, New Delhi, India, 19–22 June.

**UNPUBLISHED DISSERTATIONS AND THESES**

- Kumar S. (2011): "Customer Value: A Comparative Study of Rural and Urban Customers," Thesis, Kurukshetra University, Kurukshetra.

**ONLINE RESOURCES**

- Always indicate the date that the source was accessed, as online resources are frequently updated or removed.

**WEBSITE**

- Garg, Bhavet (2011): Towards a New Natural Gas Policy, Economic and Political Weekly, Viewed on July 05, 2011 http://epw.in/user/viewabstract.jsp

# MULTIPROGRAMMING AND REAL TIME SYSTEMS: FUNCTIONAL REQUIREMENTS

*DEVENDRA KUMAR TIWARY*
*ASST. PROFESSOR*
*DEPARTMENT OF COMPUTER APPLICATION*
*TECHNICAL EDUCATION & RESEARCH INSTITUTE*
*RAVINDRAPURI, GHAZIPUR - 233 001*

## ABSTRACT

*Designing a multiprogramming operating system is still a major consideration for system designers. Multiprogramming operating systems support concurrent execution of tasks, sophisticated form of memory management and file management. These are the basic requirements for a multiprogramming operating system and impose additional penalties on the system. There always exists a trade-off between plenty of capabilities vs. system complexity. Therefore, these contradicting aspects must be carried out carefully. To reduce system complexity and additional overhead, an environment specific multiprogramming system can be designed. Another solution is to design a giant system incorporating almost all capabilities of multiprogramming leaving activation of specific capability on user's choice at system startup time. In this paper all these aspects are deeply discussed in the light of POSIX (Portable Operating System Interface) standard 1003.1 which defines the basic functions of UNIX operating system, its real-time extensions POSIX 1003.1b for prioritized scheduling, enhanced signals, IPC primitives, high-resolution timer, memory locking, synchronized I/O, asynchronized I/O, contiguous files and POSIX 1003.1c for creation of threads and management of their execution. This paper begins with the requirements for a generic multiprogramming operating system and latter concentrates on an environment specific multiprogramming operating system; Real-Time operating system. The processor scheduling, process communication and synchronization mechanisms, event notification and software interrupt and memory management for a real-time operating system are discussed. The functional requirements have been contraindicated.*

## KEYWORDS

Interrupt, POSIX, Real-Time System, Synchronization, Virtual Mapping.

## INTRODUCTION

Operating systems are essential components of computer system. It is well known that in lack on software, the computer hardware will have no use. Out of these software's, the operating systems are the most important. If the computer hardware is to be made workable at least one operating system must be installed there. Operating system has greater control over the hardware and it works as an interface between hardware and user's programs. There is a proven requirement of operating system as without it nothing will happen in computer system. Whenever a user performs a task in application software, the application software issues an appropriate system calls. In response of system calls, the operating system enters into the kernel mode to instruct the CPU to execute the instructions. For the user, operating system is private secretary, for processes it traffic controller, for resources it is resource manager. The primary objective of operating system is to increase productivity of processing resources. There is a wide range of tasks carried out by the operating system: -

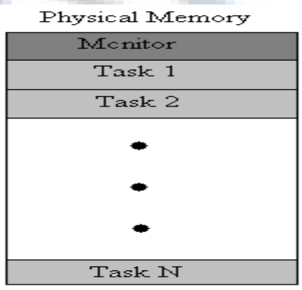
- Operating system provides an easy interface between users and physical computer.
- It controls the execution of programs.
- Operating system loads and schedules user's programs along with necessary compilers when requested by the user.
- Operating system controls the functioning of input and output devices.
- Process scheduling
- Management of physical memory.
- Managing files

## MULTIPROGRAMMING

Multiprogramming has become quite common now-a-days. As the enhancement in hardware, especially in processors, the resource utilization can not be optimized with a single task. Allocating more tasks to CPU lead to better utilization of resources and increased throughput. But multiprogramming has also imposed complexities on operating system. In early days of computing starting from sequential processing and batch processing, the operating systems were very simple. There was little or no processing overheads were associated with these systems. As the hardware upgraded rapidly, researchers moved their attention towards developing systems that can exploit the CPU and give better throughput.

Buffering and spooling improve system performance by overlapping the input, output and computation of a single job, but both have their limitations. A single user can not keep the CPU or I/O devices busy at all times. Multiprogramming offers a more efficient approach to increase system performance. In order to increase resource utilization, systems supporting multiprogramming approach allow more than one job to utilize CPU time at any moment. More number of programs competing for system resources, better will be resource utilization. Many tasks simultaneously reside in physical memory as illustrated in *figure 1*.

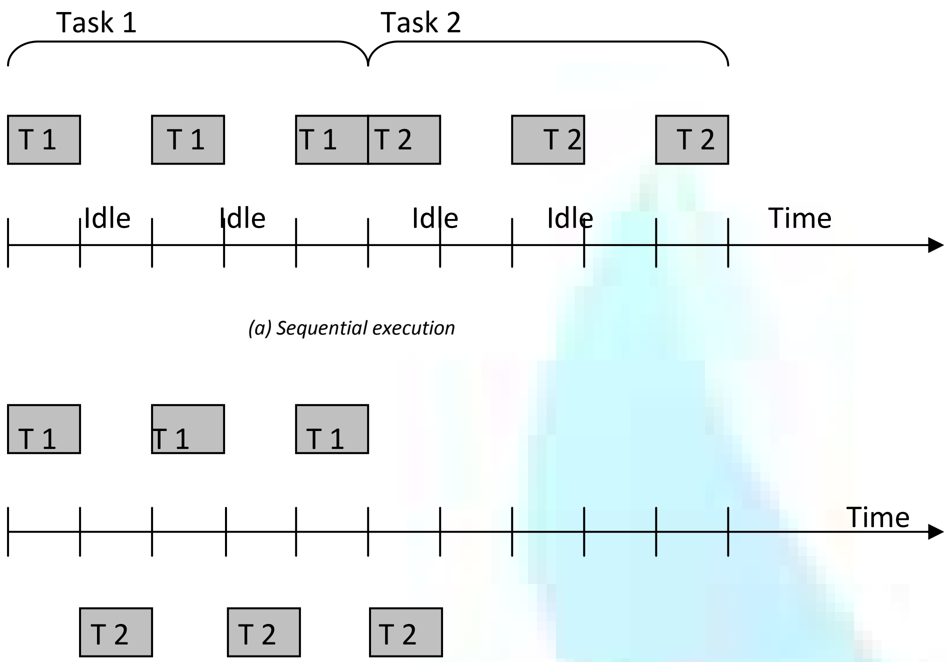
**FIGURE: 1 ALLOCATION OF PHYSICAL MEMORY IN MULTIPROGRAMMING ENVIRONMENT**




In sequential execution, a single task is executed at a time till its completion. If the task requires some I/O operation to complete, the CPU sits idle during this time leading to poor throughput. In figure two tasks are completed in 10 units of time. Out of 10 units of time CPU works only for 6 units of time giving only 60% of CPU utilization. In multiprogramming as shown in figure, the operating picks one of the tasks and starts execution. During the execution process if task 1 need some I/O operation, the operating system simply switch over to the next task. If there is no task left in physical memory, the CPU will pass its control to the previous task. This helps in improved system performance and resource utilization. However, the turnaround delay for T1 and T2 is equal in both cases (5 units of time); the better throughput is achieved in multiprogramming. As multiprogramming supports concurrent execution of programs, the multiprogramming operating system must be capable to handle four activities: -

- CPU scheduling
- Physical memory management
- Input / Output management
- File management

**FIGURE: 2 SEQUENTIAL EXECUTION AND MULTIPROGRAMMING**




*(a) Sequential execution*

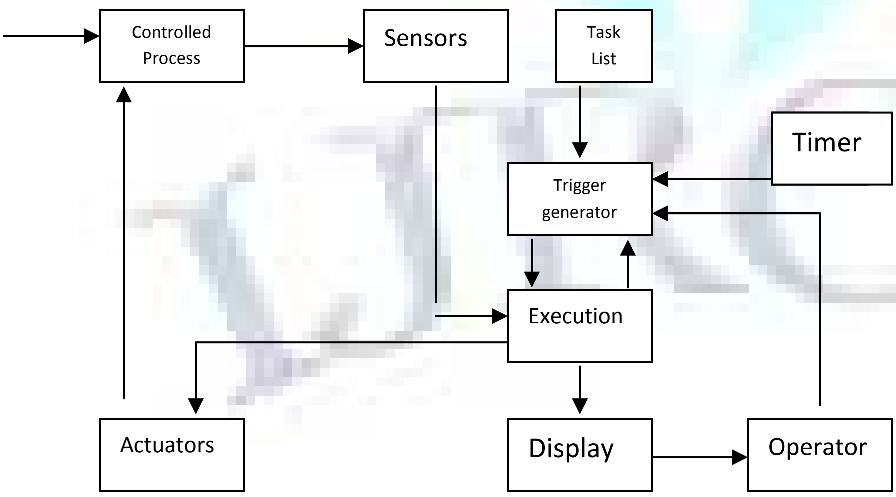
*(b) Execution in multiprogramming environment*

**REAL-TIME OPERATING SYSTEMS**

Real-time systems are most suitable in the changing environment where large number of external events must be accepted and processed in certain deadline. The primary objective of real-time systems is minimizing event response time. Processor utilization and throughput are secondary concern. In dynamic environment thousands of interrupt may occur in a second. The real-time systems must be capable to process them in short time without missing a single interrupt. The timing constraint of a task in real-time system can be hard or soft depending on whether a rigorous validation of the timing constraint is required or not. The former is hard and the latter is soft. In practice, a hard real-time system invariably has many soft real-time jobs and vice-versa

*Figure* shows the block diagram of a real-time system. Sensors acquired state of controlled process and operating environment and provide inputs for controller, the real-time computer. There is a fixed set of application tasks. The software for these tasks is preloaded into computer. If the computer has the shared main memory, then the entire software is loaded into that. If it consists of a set of private memories belonging to individual processors, the question arises as to which memories each task should be loaded into. The trigger generator represents the mechanism used to trigger individual task for execution. After execution the output is sent to display and actuators.

**FIGURE: 3 BLOCK DIAGRAM OF REAL-TIME SYSTEM**




**PROCESSOR SCHEDULING**

The scheduling of tasks to ensure that real-time deadline is met is the central to the objective of a real-time operating system. A may be *periodic* which runs at regular intervals or *sporadic* which invoked at irregular intervals. A scheduling may be offline or online. *Offline* scheduling involves scheduling in advance of the operation, with specifications of when the periodic tasks will be run and slots for the sporadic tasks in the event that they are invoked. In *online* scheduling, the tasks are scheduled as they arrive in the system. *Static-priority* algorithms (RM, for example) are used where priority of a task does not change within a mode.

Dynamic-priority algorithms (EDF, for example) assume that priority of a task can change with time. Scheduling may be *preemptive* if tasks can be suspended from execution due to arrival of higher priority task in the system or *non-preemptive*.

A truly real time operating system must offer at least 32 priority levels which is the minimum number required to be real time operating system. Most real-time operating system offer 128 or 256 priority levels. In real-time systems, one process is charged for handling one event. These processes are mostly explicit, user-defined processes. An interrupt causes the process to wakes-up on occurrence of the related event The processes are given a priority which signifies the importance of event associated with it and process having higher priority preempts the CPU from lower priority process. So the process with highest priority is executed first. When priority levels of two threads are equal, FIFO or Round-Robin scheduling can be opted. To control the custom, *priority based preemptive scheduling* is used in most of the real-time system.

A lower priority may be blocked by higher priority process. If this situation is intolerable, a technique called *aging* is applied. This enforces gradually increment of process priority when the process spends a certain amount of time in the system. This scheduling can be further refined into *earliest-deadline first* (EDF) scheduling in hard real-time systems which guarantees the execution of time-critical processes before expatriation of deadline.

**PROCESS COMMUNICATION AND SYNCHRONIZATION MECHANISMS**

In multiprogramming system, the concurrently executing processes must communicate and synchronize in order to cooperate to each other. Interprocess communication is based on the use of *shared variables* or *message passing*. Processes synchronize using mutexes, reader/writer locks, condition variables and semaphores. Processes that are working together often share some common storage that one can read and write. Each process has a segment of code called critical section that access shared memory. The key issue involving shared memory is to find way to prohibit more than one process from reading and writing the shared data at the same time. Some way of making sure that if one process is executing in its critical section, the other process will be excluded from doing the same thing. This requirement is called ***mutual exclusion***. The Dutch mathematician *Dekker* is believed to be the first to solve the mutual exclusion problem. But his original algorithm works only for two processes. It can not solve all the problems of mutual exclusion.

**MODULE MUTEX**

```
    var P1busy, P2busy: boolean;
    Process P1;
        begin
            while true do
                begin
                    P1busy:=true;
                    while P2busy do {keeptesting};
                        critical_section;
                        P1busy:=false;
                    rest_P1_processing;
            end{while}
    end; (P1)
    Process P2;
        begin
            while true do
                begin
                    P2busy:=true;
                    while P1busy do {keeptesting};
                        critical_section;
                        P2busy:=false;
                    rest_P2_processing;
            end{while}
    end; (P2)
{parent process}
begin{mutex}
    P1busy:=false;
    P2busy:=false;
    initiate P1, P2;
end{mutex}
```

**ALGORITHM: 1 MUTUAL EXCLUSION**

A synchronization tool called semaphore is used solve mutual exclusion problem. A semaphore is a variable which accepts non negative integer values and except for initialization may be accessed and manipulated through two primitive operations – *wait* and *signal* and implemented as system calls or as built-in functions. The two primitives take only arguments as the semaphore variables, and defined as:

```
wait(S)
    While S <=0 do {keeptesting} S:=S-1;
signal(S)
    S:=S+1;
```

Keeping critical section between these two primitive operations, mutual exclusion can be achieved.

```
wait(S)
{critical_section};
signal(S)
```

***Module sem_mutex***

```
var bsem: semaphore; {binary semaphore};
Process P1
    begin
        while true do
            wait(bsem)
            critical_section;
            signal(bsem)
            rest_P1_processing;
        end (while)
end; (P1)
```

```
Process P2
        begin
                while true do
                        wait(bsem)
                        critical_section;
                        signal(bsem)
                        rest_P2_processing;
                end {while};
end; (P2)
.
.
.
Process n
        begin
                while true do
                        wait(bsem)
                        critical_section;
                        signal(bsem)
                        rest_Pn_processing;
                end {while}
end; (Pn)
{parent process}
        begin {sem_mutex}
        bsem:=1 {free}
        initiate P1, P2, …,  Pn
end; {mutex}
```

**ALGORITHM 2: MUTUAL EXCLUSION WITH BINARY SEMAPHORE**

Modification to the integer value of the semaphore in the wait and signal operations are executed individually. Therefore any number of concurrently executing process can be mutually excluded from executing their critical section at the same time.

**EVENT NOTIFICATION AND SOFTWARE INTERRUPT**

Event notification, exception handling and software interrupts are the essential requirements of multiprogramming systems. Responsive mechanisms are needed to inform threads of the occurrence of timer events, the receipt of messages, and the completion of asynchronous I/O and so on. Real-time POSIX improves predictability in interprocess communication by providing the application with control over message passing. A real-time POSIX messages are prioritized just like threads and they can dequeued in priority order. Send and receive are nonblocking. Moreover, receive notification makes it unnecessary for a receiver to check for the arrival of a message to an empty queue.

In multiprogramming system, interrupt handler and kernel use signals as a means to inform threads of the occurrence of exceptions or waited for events. Signals are primarily for event notification and software interrupt. POSIX provides only two application defined signals. POSIX signals are delivered in FIFO order, and are not queued, and can not pass data. A thread signals another thread to synchronize and communicate. A Real-Time POSIX compliant system provides at least eight application defined signals. The Real-Time POSIX extensions signals are characterize by: -

- Eight signals are there in real-time POSIX extensions. These signals are numbered from SIGRTMIN to SIGRTMAX.
- Real-Time POSIX extensions signals can be queued.
- A queued real time signal can carry data.
- Queued signals are prioritized.
- POSIX real-time extensions provide a new and more responsive synchronous signal-wait function called *sigwaitinfo*.

**MEMORY MANAGEMENT**

Because main memory is much more expensive, per bit than disk memory, it is usually economical to provide most of the memory requirements of a computer system as disk memory. Disk memory is also ``permanent'' and not (very) susceptible to such things as power failure. Data, and executable programs, are brought into memory, or *swapped* as they are needed by the CPU in much the same way as instructions and data are brought into the cache. Most large systems today implement this ``memory management'' using a hardware memory controller in combination with the operating system software.

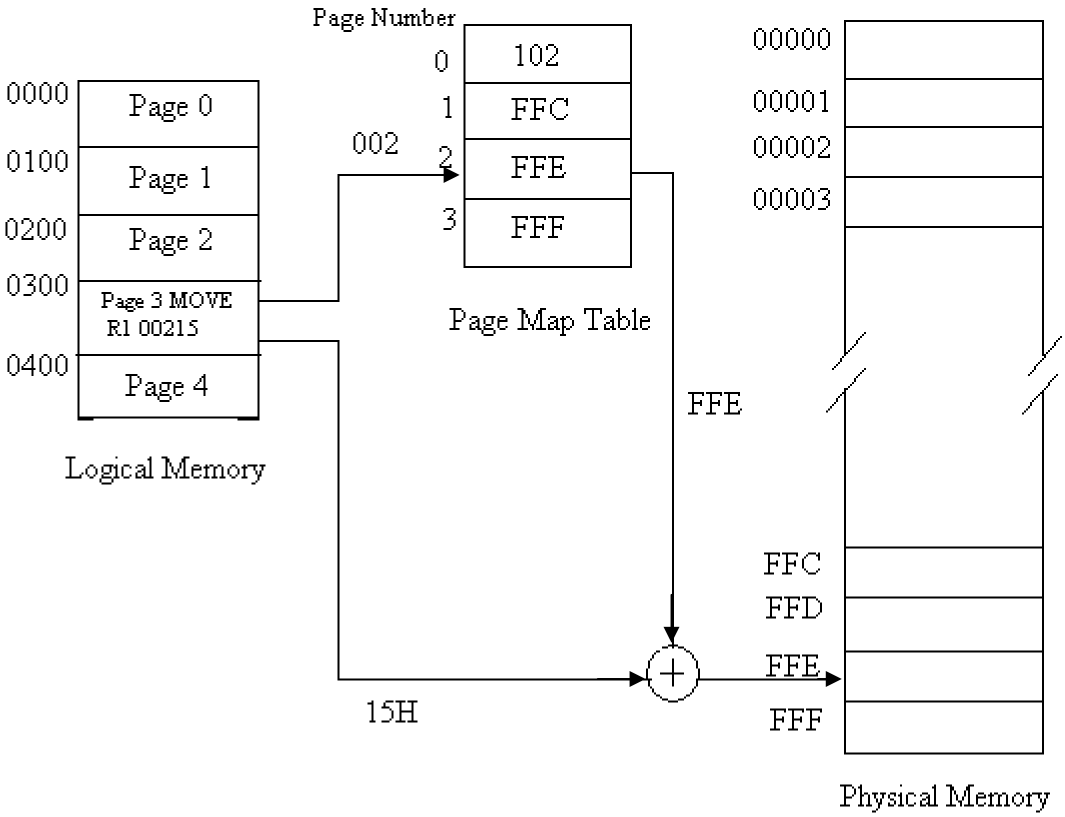
In a computer system which supports virtual memory management, the computer appears to the programmer to have its address space limited only by the addressing range of the computer, not by the amount of memory which is physically connected to the computer as main memory. In fact, *each process* appears to have available the full memory resources of the system. Processes can occupy the same virtual memory but be mapped into completely different physical memory locations. Of course, the parts of a program and data which are actually being executed must lie in main memory and there must be some way in which the ``virtual address'' is translated into the actual physical address in which the instructions and data are placed in main memory. The process of translating, or mapping, a virtual address into a physical address is called *virtual address translation*.

In real-time system most of the processes permanently reside in physical memory. Therefore memory management in real-time system is less demanding. There is little or no need to moving processes from physical memory to secondary storage. Swapping of processes is rarely done. This reduces considerable overhead of memory manger. But all multiprogramming capabilities must be considered if circumstances warrant so. A real-time system may support virtual memory mapping and paging or none of these. Real-time systems designed primarily for embedded real-time applications such as data acquisition, signal processing and monitoring may not support virtual memory mapping.

Fragmentation of memory is a potential problem for a system that does not support virtual mapping. After allocation of variable size segments, large fraction of individual blocks may be unused. The available space may not contiguous or contiguous area in memory may not enough to meet the application's buffer demand. These all cases lead to wastage of memory. Virtual address mapping impose additional penalty on the system in terms of *address translation table* which must be maintained. *Address translation table* further complicates DMA-controlled I/O.

Given a physical memory of 1 MB where virtual and physical address is 20 bits long each. The page size is assumed to be 256 bytes. The physical memory will have maximum of (1 MB / 256 bytes) 12 frames (pages) of 256 bytes each. The corresponding virtual to physical mapping is given in *figure*.

**FIGURE: 4 VIRTUAL TO PHYSICAL MAPPING**




A real-time operating system may support paging so that memory demanding application such as editors, debuggers needed during development can run together with target real-time applications. Such system must have some means to control paging otherwise the system goes to thrashing. Real-time POSIX compliant systems support shared memory in addition to file mapping. In such a system, a process can create a shared memory object that is accessible by other processes. Shared memory must be protected so that a process can not manipulate the address space of another process. For this purpose, *mlockall* and *mlock* functions are provided in real-time extension of POSIX. The former is used to lock the entire memory; the latter is used to lock the specified range of address space of a process in memory.

## CONCLUSION

To increase resource utilization and system throughput, an operating system must incorporate multiprogramming capabilities. But the fact is, adding all multiprogramming capabilities to a single operating system make it complex. Therefore environment specific multiprogramming operating systems are preferred. The most demanding features are highlighted and other things are compromised. Increasing throughput and minimizing response delay to achieve optimized performance is still a problem in most of the multiprogramming system.

To schedule varying priorities of tasks, a combination of schedulers are used to meet the real-time deadline, to minimize response delay and to maintain system throughput. Processes in multiprogramming operating system must communicate and synchronize accordingly using semaphores, mutexes and condition variables. These synchronization tools have their own limitations. In hard real-time systems, too many interrupts generate frequently and no single interrupt can be missed. To control the custom, an effective scheduling is required. Virtual mapping must be supported by multiprogramming systems to increase the degree of multiprogramming. However, virtual mapping increases run-time overhead.

## REFERENCES

[1]. Abraham Silberscharz, Peter Bear Galvin, Greg Gagne, “Operating System Principles”, 2006, *John Wiley & Sons (ASIA) Pvt. Ltd, Singapore*

[2]. Andrew S. Tanenbaum, “Operating System Design and Implementation”, 2004, *Prentice Hall of India*.

[3]. Balakrishnan, S., and F. Ozguner, “Apriority-driven flow control mechanism for real-time traffic in multiprocessor networks”, *IEEE transactions on parallel and Distributed systems, vol. 9, no. 7, July 1998.[*

[4]. Bal Gopal, Rizwan Beg, Pankaj Kumar, “Verifying memory consistency models for DSM system: An analysis”, *The IUP Journal of Computer Science, vol. IV, no. 3, 2010*

[5]. C. M. Krishna, Kang G. Shin, “Real-Time Systems”, 1997, *McGraw-Hill International Edition.*

[6]. Davis, R. I., and A. J. Wellings, “Dual priority scheduling”, *Proceeding of IEEE Real-Time Systems Symposium, December 1995.*

[7]. Jane W. S. Liu, “Real-Time Systems”, 2006, *Pearson Education*.

[8]. Milan Melenkovic, “Operating Systems Concepts and Design”, 2006, *Tata McGraw Hill, New Delhi*

[9]. Rakesh Kumar Bansal, Kawaljeet Singh, Savina Bansal, “Performance Analysis of a Real-Time 1-fault-tolerent scheduling algorithm on heterogeneous computing platform”, *IJCSSA, vol. 3, no. 2, December 2009*

[10]. www.unix.org

# REQUEST FOR FEEDBACK

**Dear Readers**

At the very outset, International Journal of Research in Computer Application and Management (IJRCM) acknowledges & appreciates your efforts in showing interest in our present issue under your kind perusal.

I would like to request you to supply your critical comments and suggestions about the material published in this issue as well as on the journal as a whole, on our E-mails i.e. **infoijrcm@gmail.com** or **info@ijrcm.org.in** for further improvements in the interest of research.

If you have any queries please feel free to contact us on our E-mail **infoijrcm@gmail.com**.

I am sure that your feedback and deliberations would make future issues better – a result of our joint effort.

Looking forward an appropriate consideration.

With sincere regards

Thanking you profoundly

**Academically yours**

Sd/-

**Co-ordinator**