[[1]](#footnote-1)

Implementing multi process synchronization using Win 32 API

Meenatchi S1, Hemant Ravi2, Rishav Srivastava3, Aditya Swaroop4   
School of Information Technology and Engineering  
Vellore Institute of Technology

*Abstract*— A very common definition of a process is that it is a program in execution. It is actually correct to say like that, as every program or a software runs internally as a process. At any point of time, there are number of processes executing inside the computer system and number of processes will be waiting to acquire the CPU for their execution. So, there should be a provision that will help in choosing the next process to which CPU will be allocated. This is where process synchronization comes into picture. Synchronization of processes is also important as there can be some processes sharing data among them, and if this sharing of data is not done in a controlled environment, then the results may be varying. This will not only produce erroneous results but at the same time question the correctness of the computer system. In this paper, we will implement multi process synchronization on readers-writers problem which is a classical problem in process synchronization. We will implement it using Win 32 API which is the main set of Windows APIs for creating 32-bit applications**.**

*Index Terms*— Multi process synchronization, Win 32 API.

# INTRODUCTION

A process can be defined as the active entity of any program where program being the passive entity itself. It is a program which is running inside the system. As already stated, there are a number of processes executing at the same time. Some of these processes may be generated by the operating system itself to perform various tasks. After the completion of the task those processes are terminated so that the resources which they using can be allocated to other processes so that they can carry out their task effectively. On the other hand, there are processes which are generated by user actions which in turn are handled by the operating system. Processes can be independent process or cooperating process. Independent processes work independently and do not share any data or resource among them and also do not affect the execution of other processes. Whereas cooperating processes are those that share data with processes and can affect or get affected by other process execution. Synchronization of cooperating process is necessary as they are working or using some common data and if it is not

used in controlled way then we may get erroneous results and these results will be based on the execution order of the two process.

# Literature survey

In this paper [1] the authors proposed a new design of using a special kind of “agent” for solving the process synchronization problem. This agent-based metric is a new method to solve the problem of inter process synchronization. According to them, the agent will run as a background process as system service. The agent will also have other properties like – learning from the operating environment, the agent will have intelligence to apply existing measures kept in its database, it will also save new solutions into the knowledge database, the agent also has the ability to get triggered automatically in the case of problem, etc. This solution can also be used in networking environment. They proposed that for the above solution to be implemented, two data structures will be required. One of them will be used for storing all the necessary information about the processes executing on the system and the other data structure will keep information regarding the various resources of the system like resource id, type of resource, allocation status, process id (if the resource is allocated to any process) etc. The different task for the agent using the afore mentioned data structure will be to identify the priorities of different process, maintaining process-resource-request queue, assign tokens to those processes that get chance to allocate resource etc.

In this paper[2] author introduced a new approach for solving the multiprocess synchronization by introducing two variables w(i) and n(i) where w(i) stores the count of total number of times a process had become idle and n(i) stores the count of total number of times a process has committed to any event. The author has also used other algorithms namely, Centralized algorithm and Event manager algorithm. These algorithms use the message count to tackle the synchronization problem. For solving the exclusion problem, they used the concept of circulating token, or by using secondary resources as in the solutions for the dining philosophers’ problem. The experiments measure the response time and message complexity of each algorithm by doing certain variations in different model parameters including network topology and level of conflict in the system.

In this paper [3], the author introduced a new approach for solving the readers-writers concurrency problem. In this paper, the author presented another semaphore- based solution which will not go under starvation for the reader process and as well as for the writer process. Initially the solutions which were present allowed only one reader process to access the resource. So due to this other there was a chance that the readers may indefinitely lock out writers, or permitted writers will indefinitely lock out readers. The method was implemented as if there were numerous readers and writers in a stream, the readers and the writers will be forced to wait on a semaphore resource. Whenever a process exited, a signal must be executed. The next waiting process should be allowed to proceed, without considering the type of the process. If the process is reader process, it will be the only reader process accessing the resource at that time, even if the next waiting process is also a reader.

In this paper [4] author wrote about process synchronization in multiprocessor environment and as well as in multi-core processor environment. The author stated that there is a use of shared-memory multiprocessors as computation servers with multiple parallel applications executing concurrently. So, the operating system’s scheduling policy can affect the efficiency of parallel application. To provide synchronization in multi-core processor systems they proposed the synchronization model of Request-Store-Forward (RSF). This model basis is a data structure called as “I-structures”. An I-structure is like a normal data structure but it also supports split-phase transaction which lowers communication overheads and limits the contention problems. The RSF model splits the synchronization operations into two phases – in the first phase, the synchronization requests are stored as and when they arrive. Consequently, state of the synchronization is stored and the rest requestors are allowed to continue with other tasks; in the second phase, when the synchronization operation has finished, a notification message is sent and as well as the requested data to the requestor. This model makes use of a buffer to keep an eye on all the undergoing operations, instead of keeping track of the synchronization states by attaching a full/empty bit to each word in memory. They discussed two synchronization methods in multiprocessor environment. The first method is Barrier synchronization in which an application is creating streams for using fine-grained parallelism and also schedules the parallel execution of streams on processors. When a threshold level is reached, the processor must wait until all the other participating processors reach the level of threshold. There are two ways of introducing Barriers – one is to automatically bring them by a parallelizing compiler and the other way is that programmer will introduce them explicitly. There are three actions listed in Barrier synchronization: first is a task to posts the synchronization arrival at the barrier. The second method is Rendezvous synchronization. In this, an entry point is created which is called as function call. One process defines its entry and makes it public. Upon executing the function call, results are returned to the caller of the function.

In this paper [5], the authors have tried to present software solutions for critical section problem. They described the previously present three algorithms for solving critical section problem. First algorithm uses one single Boolean variable that can be either 1 or 0. The second algorithm uses a Boolean array flag (flag [0] and flag [1]). The third one uses both of these variables. They also described the two types of semaphore variables – counting and binary.

In this paper [6], the authors presented their research work on testing the robustness of Win 32 API. They used Ballista software testing application and tested for different operating systems like Windows 95, 98, CE, NT and 2000. It was also tested on Linux operating system. They found that Windows 95, 98 and CE were highly vulnerable to complete system crashes caused by very simple C programs for several different functions. But on other hand no system crashes were reported on Windows NT, 2000 and Linux operating system. Linux operating system handled the exceptions generated while making system calls effectively in a program-recoverable manner than Windows NT and Windows 2000, but they also concluded that for handling C library exceptions, Windows variants were more robust than Linux (with glibc).

In this paper [7], the writers have surveyed about process management using process synchronization. Any kind of Operating System can’t be managed without proper management of processor and process. While processing multiple process at a time Operating System may face some kind of difficulties which is called as Critical Section where all processes are executed. To avoid in going to Critical Section while processing multi processes at a time, different approaches and algorithms can be used to eliminate the Synchronization problem like Semaphores, Mutex Locks etc. By applying these approaches, a process can be easily avoided in going to Critical Section. To avoid going in a deadlock situation of a process, implementation of mutual exclusion can also be proved as a very helpful approach. Synchronization of a process makes the Operating System to achieve the concept of multiprogramming and multiprocessing.

In this paper [8], there is a briefing about synchronization in distributed programs. The authors said that presently there are various message passing techniques using which we can implement multiprocess synchronization. But they approached a lower level technique for the same because there are unwanted implementations and also there is no profound support for choosing one way over the other. To overcome this issue, a new technique is developed which can be used to solve the synchronization problem directly is to construct a distributed semaphore which is well suited to use in distributed programs. By using this technique which is developed in the environment of concurrent programming in distributed programs, programmers are allowed to easily deal with failure of process problem.

In this paper [9], the authors have talked about synchronization mechanisms on modern multi-core architectures. They have proposed a method of request store and forward to help in todays’ parallel computation systems. They have further elaborated the I structure and how it can be beneficial.

In this paper [10], the author did analysis on the various factors that affects process synchronization. The author mentioned about the race condition, where the outcome depends on the way in which it is accessed by the different process. In the later parts of the research work, there is a description about critical section and semaphores. Deadlock and starvation are also discussed and finally about the readers- writers problem. She has also discussed methods for executing co-operating process in a proper order. For each of the topic discussed, the author has elaborated it using proper explanation mentioned about the race condition, where the outcome depends on the way in which it is accessed by the different process. In the later parts of the research work, there is a description about critical section and semaphores. Deadlock and starvation are also discussed and finally about the readers- writers problem. She has also discussed methods for executing co-operating process in a proper order. For each of the topic discussed, the author has elaborated it using proper explanation.

# proposed architecture

We will be implementing readers – writers’ problem to show the multi process synchronization. As we know, in readers – writers’ problem, if two or more readers wants to read data they can be allowed. But when a writer is working on the data and simultaneously a reader wants to read or a writer wants to write, then this may lead to erroneous situations. We should prohibit a reader or a writer from using the shared data when a writer is working on the data previously. We have provided implementation of this problem using Win32 API. We can run multiple instances of the same code and one instance to be a reader process and the other one to be a writer process. We have also tested all the different conditions like read-read situation, read - write situation and write- write situation. We have tried to provide proper synchronization for all these events and also tried to prevent data from being damaged.

**Description of all proposed architecture modules –**

**windows.h** – It is the main header file for WinAPI. It enables us to create 32 bit and 64bit applications. It has declarations in Unicode and ANSI. This header file allows us communicate with the operating system or the under lying filesystem. This header file can also be used in creating pop-up boxes. It also contains various pre-written macros which are very useful in creating WinAPI applications. It defines a very large number of Windows specific functions that can be used in C. The Win32 API can be added to a C programming project by including the <windows.h> header file and linking to the appropriate libraries.

**tchar.h –** This header file is used for supporting Unicode dataset. Earlier the programming was done in ANSI script, but to support internationalization Unicode came into picture. So, Microsoft came up with this data set. When we use multibyte character set, like the ANSI character set, its size is of 1 byte and the data type to be used is char. When the character set is Unicode, its size is 2 byte and the data type is wchar\_t.

**HANDLE -** It is a data type that helps us in Win32 programming. It is basically a void poiner. Windows operating system keeps all the various objects that the kernel is responsible for in a form of a table. The entries are stored along with their address in table. Whenever one enquires about them, he/she needs to get in contact over that table, and Windows will return an address of that particular entity.

**LPSTR –** Long Pointer String is a data type used for referring strings created using tchar data set. These strings can be either Unicode or ANSI, which depends on the environment in which they are used.

We will use 1KB of shared for the reader and writer process. The synchronization is done using events. We are using maximum 3 reader process at a time. There are different functions for performing the various tasks such as – initialize() function takes care of initializing the memory and the variables, deinitialize() function which does just the opposite task of initialize function, and the read and write functions to perform reading on and writing to the buffer.

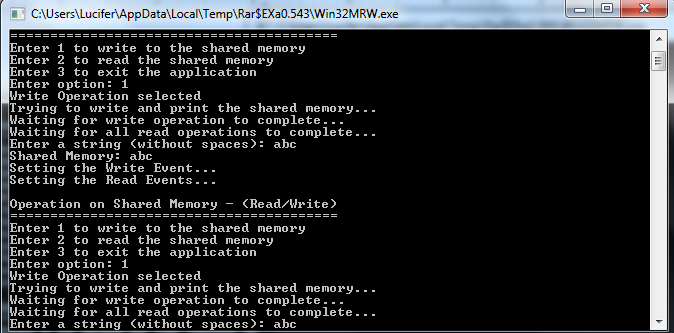


Fig. III.1 – Write-Write operation

The Fig.III.1 is taken while executing this program The first line which is printed denotes that all initialization of various variables and buffer memory was done successfully. We have provided three options for reading or writing to the shared memory buffer. We choose to write to the shared memory. If there are no process already writing or reading from the shared buffer then only, we will be allowed to perform the job or else we will have to wait.

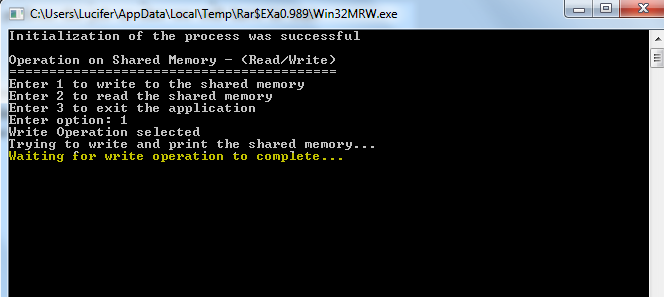


Fig.III.2 – Writing in the Buffer

The Fig.III.2 is taken when one process is already writing in the buffer. The program says – Waiting for write operation to complete, because previously a writer has been writing to the shared buffer. So, in this multi process synchronization is handled.

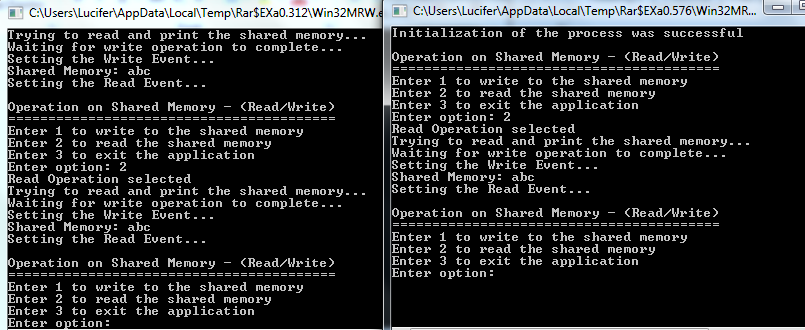


Fig.III.3- Read-Read operation

In Fig.III.3 we have depicted two reader different reader processes, which are trying to read from shared memory. The two processes are allowed to read at the same time and both read the same data.

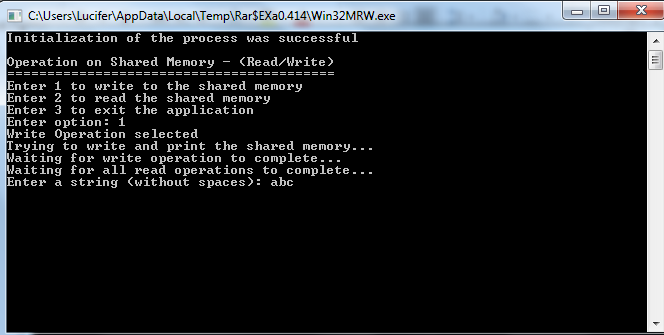


Fig.III.4 – Write-Read Operation

The Fig.III.4 depicts the first process which is writing in the shared memory.

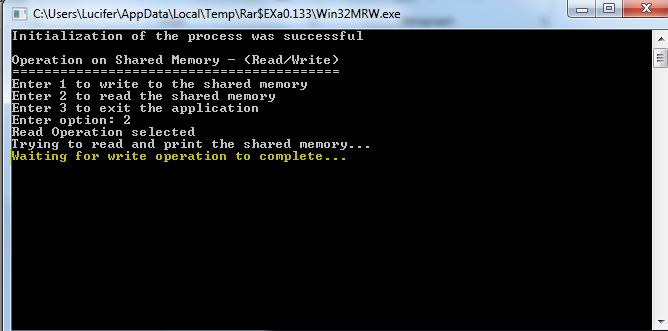


Fig.III.5 – Writing Process

The fig III.5 shows that one writer is writing, so at the same time we are not allowed to perform read operation on the data. Once the write is completed then only, we can perform this operation.

# comparison of results

To show how different languages can be used in the multi-process synchronization and how will these languages perform, it is shown with the help of a pictorial representation in the form of a graph.

Fig.IV.1 – Graphical Representation

The graph in Fig.IV.1 shows that Simple critical section with no read or write takes very low execution time. Full re-entrant lock with thread local storage and spin lock is almost 3 times slower than simple critical section. The highest execution time is taken by fully re-entrant lock with multiset.

# Conclusion

In this project, we have tried to implement multi process synchronization by using reader – writer problem and Win32 API. The various cases such as read - read, read – write and write – write conditions are also implemented in the program and tested. Only one process, if it is a writer process, is allowed to write in the shared memory at a time. Two or more reader processes may use the shared buffer at the same time and perform the read operation. If a read operation is being carried then no write operation can be performed and same is for the reverse situation. At last, we have also compared different methodologies for the same implementation. We have compared with the situation when one uses re-entrant locks provided in JAVA language with simple critical section problem.

References

[1] Deepshikha Bhargva, Dr.Madhavi Sinha (2011), “Agent based design for solving Inter Process Synchronization Problem”. International Journal of

Advanced Engineering & Applications. 0975 – 7783.

[2] Rajive Bagrodia (1989) " Process Synchronization: Design and Performance

Evaluation of Distributed Algorithms” IEEE Transactions on Software

Engineering, VOL. 15, No.Y. September 1989

[3] Jalal Kawash (2005), “Process Synchronization with Readers and Writers Revisited”. Journal of Computing and Information Technology - CIT 13, 2005, 1, 43–51

[4] Mohammed Mahmudur Rahman, "Process Synchronization in Multiprocessor and Multi-core Processor ".

978-1-4673-1154-0/12/$31.00

[5] Ankit Gupta, Arpit Gupta, Amit Mishra, “Research paper on software solution of critical section problem”.

International Journal of Computer Applications (0975 – 8887) Volume 128 – No.7, October 2015

[6] Charles P Shelton, Philip Koopman and Kobey Devale, " Robustness Testing of the Microsoft Win32 API."

[7] Deepti Sindhu, Anupma Sangwan and Kulbir Singh, " An Approach to Process Management using Process Synchronization.". International Journal of Computer Applications (0975 – 8887) Volume 128 – No.7, October

2015

[8] Fred B Schneider, " Synchronization in Distribured Pograms." ACM Transactions on Programming Languages and Systems, Vol. 4, No. 2, April 1982, Pages 179-195

[9] Shaoshan Liu and Jean-Luc Gaudiot, " Synchronization Mechanisms on Modern Multi-core Architectures ". ACSAC 2007, LNCS 4697, pp. 290 – 303, 2007.

Springer-Verlag Berlin Heidelberg 2007

[10] Arti Chhikara, “Analysis of Factors Affecting Process Synchronization”. International Journal on Computer Science and Engineering (IJCSE), Vol. 9 No.02 Feb 2017.

[11] N. Dunstan, "Synchronization Problems and UNIX System V", ACM Computing Surveys, Vol. 21, No.4, pp. 15-19, December 1989.

[12] Neil Dunstan and Ivan Fris, "Process Scheduling and. UNIX Semaphores", Software: Practice and Experience, Vol. 25, No. 10, pp. 1141- 1153, October 1995.

[13] C. A. R. Hoare, "Monitors: An Operating System Structuring Concept". Communications of the ACM, Vol. 17, No. 10, pp. 549-557, October 1974.

[14] M. H. Kelley, "Multiprocessor Aspects of the DG/UX Kernel". Proceedings of the Winter 1989 USENIX Conference, pp. 85-99, San Diego, CA, January 1989.

[15] Gadi Taubenfeld, “Synchronization Algorithms and Concurrent Programming”, Pearson/Prentice Hall, ISBN 0-13-197259-6

[16] David L. Black, "Scheduling support for concurrency and parallelism in the Mach operating system". IEEE Computer, 23(5):35-43, May 1990.

[17] Helen Davis, Stephen Goldschmidt, and John Hermessy. Tango, “A multiprocessor simulation and tracing system”. Technical Report CSL-TR-90-439, Stanford University, 1990.

[18] Tai K., Carver R., Obaid E., “Debugging concurrent ada programs by deterministic execution.”. IEEE Trans. Softw. Eng. 17(1), 45–63 (1991)

[19] Paas, S. M.; Scholtyssik, K., "Efficient Distributed Synchronization within an all-software DSM system for clustered PCs.” 1st Workshop Cluster-Computing, TU Chemnitz-Zwickau, November 6-7, 1997

[20] Itzkovitz, A., Schuster, A., Shalev, L., “Millipede: a User-Level NT-Based Distributed Shared Memory System with Thread Migration and Dynamic RunTime Optimization of Memory References”, Proc. Of.

1. [↑](#footnote-ref-1)