**A HUGE SEMAPHORE BASED ON OS**

**ETHEREAL**

Dolan White (1902140011)

Mohamed Rahmid Sufin (1906210003)

Mumtahina Arefin Mim (1902260016)

Fardina Kabir (1910080005)

**Course Instructor: Dr Surekha Lanka**

**Department: Information Technology**

**UNDERTAKING**

I declare that the work presented in this project titled “*A Huge Semaphore based on OS*”, submitted to the Department, Faculty of Business & Technology, Stamford International University, This is my original work. I have not plagiarized.

September 2020

Stamford International University

Fardina Kabir

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fardina Kabir 1910080005



**CERTIFICATE**

Certified that the work contained in the project titled “*A Huge Semaphore Based On OS*”, by Fardina Kabir, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Surekha Lanka

Information Technology

**INDEX SHEET**

[**ABSTRACT** 4](#_Toc51926268)

[**INTRODUCTION** 5](#_Toc51926269)

[**DESCRIPTION** 6](#_Toc51926270)

[**METHODOLOGY** 9](#_Toc51926271)

[**IMPLEMENTATION** 15](#_Toc51926272)

[**CONCLUSION** 19](#_Toc51926273)

[**REFERENCES** 20](#_Toc51926274)

# **ABSTRACT**

In this paper we will discuss the basic synchronization as well as current issues of the semaphores. We will find the root cause of the problem of the critical section in the operating systems. In this project, we will emphasize how current multi processors are intensifying the issues on coordinating and accesses the resources. Parallelly, this paper will concentrate on when to use a mutex, a category of binary semaphore, versus using a standard semaphore, as well as comparing those advantages and disadvantages of both as well. Lastly, another goal is to dictate when it is best to use binary and counting semaphores. Regardless of its imperfections, this project will show that a semaphore is boon to any operating system.

# **INTRODUCTION**

In 1965, by using the meaning of a single integer variable to synchronize the progress of overlapping systems, Dijkstra introduced a modern and quite important strategy for handling concurrent processes. This integer variable is known as semaphore. It is therefore effectively a synchronizing device and can only be reached by two low-standard atomic procedures, wait and signal identified by P(S) and V(S) respectively. Semaphore is, in very simple terms, a variable that can carry only a non-negative integer value, distributed among all threads, with waiting and signal operations that operate as follows(Coding): P(S): if S ≥ 1 then S := S - 1else <block and enqueue the process>; V(S ): if <some process is blocked on the queue>then <unblock a process>else S := S + 1; Wait is defined as : Decrees the statement S meaning as long as it is non-negative (greater than or equal to 1). Signal is defined as: Increases the worth of its statement S, as the queue is no longer interrupted by a process.

Semaphores are used for solving the critical section issues and accomplish multiprocessing process synchronization. They are machine-independent; they function in the micro processor's machine-independent code. There are also two types of semaphores, Binary semaphores and counting semaphores. Binary semaphores are used for implementing mutual exclusion also known as mutex. It is a program object that avoids many occurring access to a distributed resource. Counting semaphores are used to implement bounded concurrency also known as bounded waiting. It eliminates deadlock as well as busy waiting.

There could be a circumstance of priority inversion in which low-priority processes get access to the critical section than higher-priority processes. Semaphore programming is complicated and difficult, odds are that mutual exclusion would not be executed. Their use is not enforced, it is rather instead by convention.

# **DESCRIPTION**

Semaphore can be a mechanism that can be utilized to supply synchronization of errands. Two common semaphore are binary and counting. The Counting semaphore employments a check that makes a difference errand to be procured or discharged various times. The binary semaphores are very comparative to counting semaphores, but their esteem is confined to 0 and 1. In this sort of semaphore, the wait operation works as it were in the event that semaphore = 1, and the signal operation succeeds when semaphore= 0. The binary semaphore is simple to execute than counting semaphores.

Semaphore is a low-level synchronization component. And It is an integer variable that is used in a mutually exclusive manner by various Concurrent Cooperative Processes to achieve synchronization. Integer variables are used to the decipher censorious situation by using two atomic operations. Process synchronization is achieved using Wait and Signal. If it is positive, wait decrements the value of its argument: S. If S is negative or zero, then no action is performed. The Cooperating process is the only one that offers information with another process. If two processes are using the same file, the cooperating is meant to be processing. Concurrent processing may be a computing show in which multiple processors execute enlightening at the same time for superior execution.

While executing many concurrent processes, process synchronization assists to sustain shared data consistency and cooperating process execution. Process cooperation can be broken down into four categories: Information sharing, Computation speed-up, Modularity, and Convenience. To prevent inconsistencies, processes need to be scheduled. Process Synchronization implies sharing system resources by processes in such a way that, concurrent gets to shared information is taken care of subsequently minimizing the chance of conflicting information. The requirement for synchronization starts when processes got to execute concurrently. The most reason for synchronization is the sharing of resources without impedances utilizing mutual exclusion. Maintaining information consistency requests mechanisms to guarantee synchronized execution of collaborating processes.

**CASE QUESTIONS**

1.**What are the main causes or reasons that leads to the problem of critical section?**

Critical portion is part of a service that seeks access to available services. This resource may be a computing resource such as a memory location, data structure, Kernel, or an IO unit. The critical section cannot be done concurrently through many processes; the operating system faces difficulties in approving and rejecting the entering of the critical section of the operation. The critical segment issue is used to design a variety of protocols to ensure that the race condition never exists between the systems. If more than one process runs the same code or accesses the same memory or other shared variable in that situation, there is a chance that the output or value of the shared variable is false, meaning that all the processes running the race will claim that output is right for this condition known as shared variable. This process is known as the race condition. Several processes access and process the modification of the same data at the same time, but the result depends on the precise order in which the access takes place. To synchronize mutual operations, the key challenge is to solve the issue of the critical segment.

2.**When should binary and counting semaphores be used?**

To handle access to a singular resource, a binary semaphore could be used. In general, for a critical section of the user code it can be utilized to promote mutual exclusion. In this case, the semaphore will be generated with an original number to signify that a critical code section is not executed. A task must identify the method for obtaining the semaphores when entering the critical section, to prevent further tasks being entered. When leaving the critical area, the job must specify the release method for the semaphore to allow a different task to perform the critical section.

A counting semaphore is often used to control access to multiple or perhaps more resources in a pool. For example, a semaphore with an early count of three could administer access to three printers. If a project needs access to one printer, the method for obtaining semaphore to access a printer is determined. If a printer's not obtainable at this period, the task may wait or return immediately for a printer. After printing is complete, the procedure for release semaphore should be issued to allow access to the printer for other tasks.

3. **What are the advantages, disadvantages of semaphore and its comparisons to mutex (a category of binary semaphore)?**

Although a Mutex (Mutual Exclusion Object) is a special type of semaphore (binary), there are a lot of key differences that separate the two, such as a semaphore being an integer and a mutex an object as well as a semaphore utilizing a signaling mechanism and a mutex using a locking mechanism.

The core advantages of a semaphore Is not allowing more than one process to use the critical section, allows more than one thread to enter the critical section, is machine independent, and finally, because of busy waiting, resource as well as process time are never wasted. A mutexes core advantage is that, because of its locking system only allowing one thread to enter the critical section, this leads to there being no race conditions as well as data always being continuous.

Semaphores also have disadvantages such as priority inversion (occurs when a low priority task takes priority over a higher priority task), use on a large scale can cost it it’s modularity, and is also prone to programmer error. Most of a mutexes disadvantages stem from its core feature, the locking mechanism. Because of the locking mechanism, only one thread is allowed in the critical section at any given time and that lock must first be released by the same process that locked it. Secondly, starvation can occur when a thread obtains a lock and is then forestalled or put to sleep.

# **METHODOLOGY**

Process Synchronization is the job of organizing process operation in such a manner that neither two mechanisms will use the same resources and applications that they share.

When several processes are running together, it is particularly important in a multi-process environment, because at the same time, more than one process attempts to obtain access to the same shared resource or data.

That may lead to shared data becoming inconsistent. Thus, the update that one process made did not actually indicate if other operations used the same shared data. To avoid this kind of data inconsistency i.e. the root cause of the critical section problem, the processes must be integrated with each other.

(Research problem)

Specifically, a single process may enter critical section (or CS) that consists of code at a time. The section has shared data resources which need to be utilized via other processes.

* The wait () role deals with the escape from a critical section, which is symbolized by P ().
* The critical section is responsible for the exit via the signal () function V ().

Specifically, it may execute a single process within the CS. Most processes, pending for their critical section to be completed, must wait for the existing system to finish the operation. (Approach to problem)

Critical section performs a vital role in process management, so that the problem needs to be addressed.

On this are a few frequently used techniques for solving the issue of critical section: (Analysis)

**Peterson’s solution:**

This is a user mode software mechanism. It can only be found in two phases as a busy waiting approach. It uses two variables which are variable turn and variable of interest.

A screenshot of a cell phone

Description automatically generated

Figure 1: Code of the solution ("OS Paterson Solution - javatpoint", 2020).

This is a solution for two processes. Two cooperative processes P1 and P2 should be considered. Below are the input and output sections. The value of the variables and variable turn is initially 0. Initially P1 comes first and wants to enter the CS. It keeps the variable we want to True (Line 3) to turn to 1 as well (Line 4). Since P1 fully satisfies the requirement of line number 5, it will be entered in the critical section.



Figure 2: Code of the solution ("OS Paterson Solution - javatpoint", 2020)

In the meantime, the process P1 has been preempted and P2 has been planned. In addition, P2 wishes to enter the critical section and to execute the input section instructions 1 , 2 , 3 and 4. In instruction 5, it remains stuck because the condition does not fulfill (the value of the other variable concerned is still true). So, it is in the busy waiting.



Figure3:Code of the solution ("OS Paterson Solution - javatpoint", 2020)

P1 was again scheduled and completed the critical section with the execution of instruction no. 6. Now when P2 checks, it will meet the requirement since the interested variable of other processes is false. In the critical section, P2 will also be entered.



Figure 4: Code of the solution ("OS Paterson Solution - javatpoint", 2020)

Every method may participate the critical section several times. For this reason, the operation takes place in the recessionary order.

The following is the code of the solution:

**Mutual Exclusion**-Mutual exclusion is certainly the method. For this reason, the entry section includes the two-variable criteria so that a process cannot be entered by the critical section until the next process and the last to update the turn variable is interested.

**Progress**-The other interested process will never prevent the critical section from entering. If it is also interesting for the other process, the process is waiting.

**Bounded waiting**-The variable mechanism concerned failed because it provided no limited waiting. In Peterson 's solution, however, there can be no stalemate as the first turning variable process will surely be entered in the critical section. If a process is preempted after line 4 of the entry section, it is entered in the critical section next time.

**SYNCHRONIZATION HARDWARE**

Often hardware fixes the problems in the critical area. Some operating systems have a lock mechanism where a method gets a lock in the crucial section and unlocks the lock after it is removed.

And it would not be able to access since it is locked if another phase attempts to access the vital area. One can do this only if it gets the lock free of charge.

**MUTEX LOCKS**

Not a simple way of applying synchronization hardware for all has since been implemented, a rigid software process known as Mutex Locks was proposed.

A LOCK is achieved by the vital tools used during the critical section in the code section of this method. This lock is unlocked in the escape area.

**SEMAPHORE SOLUTION**

Semaphore is just a non-negative variable that is communicated between threads. The issue of a CS is an alternate algorithm or solution. It is a signaling pathways mechanism and a loop that wait on a semaphore, and another thread can signal it.

Two nuclear activities are used, 1) wait and 2) signal for process synchronization signal**.**

**A screenshot of a cell phone

Description automatically generated**

Figure 5: Wait and Signal code("OS Paterson Solution - javatpoint", 2020).

**Methods that have been used to solve all the issues are:**

Peterson’s solution, Mutex lock and Wait and Signal.

**Peterson’s solution**- The solution given by Peterson provides a good algorithm for solving a critical issue and demonstrates some of the software difficulties which addresses the criteria for mutual exclusion, progress, and bounded waiting.

**Mutex lock**- Mutex lock is basically a binary variable that gives code wise exclusion features. Often several threads can try to access the same resource as your memory or I / O, etc. To guarantee that no interference takes place. A locking mechanism is provided by Mutex.

**Wait and Signal**- The atomic two low standard operations, P(S) and V(S) are respectively waiting and signal, which are possible only via the Semaphore synchronization method.

**Emphasizing on mutex:** Mutex is an entity of mutual exclusion synchronizing resource usage. At the outset of a program, it is generated with a single word. The Mutex is a locking mechanism in which a single thread can pass the Mutex and access the critical section concurrently. This thread unlocks Mutex only after exiting the critical section.

The following example illustrates this −.

(mutex) wait.

.....

Critical Section

.....

(mutex) signal.

Figure 6: Semaphore in OS (Operating System).

A Mutex is different from a semaphore, as a locking mechanism is a semaphore. A Mutex may be utilized as a binary semaphore, however it may not be applied as semaphore.

# **IMPLEMENTATION**

**1.What are the main causes or reasons that leads to the problem of critical section in OS?**

Critical parts play the key role of Process Synchronization of solving the issue.

Any approaches to solve the critical element problem are commonly used here.

The solution from Peterson is commonly used for critical problems in the section. Therefore, this algorithm has been created by a Peterson computer scientist.

In this solution, once a crucial procedure is performed, the other procedure just executes the remainder of the code and the contrary will occur. It also helps ensure that only one process is running at a given point in the crucial portion.

A screenshot of a cell phone

Description automatically generated

Figure 1: Peterson’s solution (Introduction of Process Synchronization - GeeksforGeeks, 2020)

Assume that N processes exist (P1, P2, ... PN) and that any process needs to reach the critical component at some point in time.

A FLAG [] size N list that is by default false is retained. Therefore, if a process decides to reach the critical component, its flag needs to be raised. For instance, FLAG[i]=TRUE is set if Pi wishes to join this.

The method number that is currently entering in the CS is indicated by another vector called TURN.

The mechanism that joins the crucial segment during departure transforms the TURN from the list of ready processes into a different number.

Example: turn is 2 and P2 joins the Crucial segment and when you exit turn=3 P3 breaks off the waiting loop.

2.**When should binary and counting semaphores be used?**

Two kinds of semaphores exist: binary and counting semaphores.

A screenshot of a cell phone

Description automatically generatedBinary Semaphores: can be either 0 or 1. They are also known as mutex locks, as the locks will remove one another. Any process will share the same mutex semaphore initialized at 1. A method must then wait until the lock is 0. The method will then start and start the vital portion of the mutex semaphore 1. When the critical section is over, the value of mutex semaphore can be reset to 0, and any other mechanism can join its critical section.

Figure 1: Binary Semaphore ("Semaphores in Process Synchronization - GeeksforGeeks", 2020).

Counting semaphores: They are valued and do not have a certain area to be limited. They can be used to manage access to a resource which restricts the number of accesses simultaneously. The semaphore can be initialized to the number of resource instances. If this resource is used by a process, it tests if the number of such cases is greater than zero, i.e. the process has an instance available. The mechanism will then reach its critical segment and hence decrease the counting semaphore value by 1.

A picture containing drawing

Description automatically generated

Figure 2: Counting Semaphore ("Semaphores in Process Synchronization - GeeksforGeeks", 2020).

**3.** **What are the advantages, disadvantages of semaphore and its comparisons to mutex (a category of binary semaphore)?**

The key difference between a semaphore and mutex is that a semaphore utilizes signal and mutex’s utilizes a locking mechanism.  Figure 1: Semaphore and Mutex (“Semaphore Vs. Mutex - A Clear Understanding”,2020) .

Semaphores operate using wait and signal and can be an integer. In the case of a binary semaphore, 0 and 1 will be used. For counting semaphores, finite number will be used (1,2,3,4,5…n).

A mutex falls into the binary semaphore category. It uses a 0 and 1 as well, however, it acts as an object rather than an integer. If a resource must be acquired, the process will utilize a lock on the mutex object. That lock can only be released by the process using the lock.

**When do you use a semaphore vs a mutex?**

To put it simply, it is best to use a semaphore when there are multiple resources available. Likewise, it is best to use a mutex when there is only one shared resource.

# **CONCLUSION**

In this essay, we discussed the basic synchronization of the semaphores as well as current problems. We identified the root problem in the OS of the critical segment. In this project, we demonstrated how current multi-processors worsen the problems of resource alignment and access. In parallel, this essay was focused on when to use a mutual exclusion, a binary semaphore class, against using a regular semaphore, as well as comparing each of these pros and cons. Furthermore, another aim we determined in this project is when it is ideal to use binary and counting semaphores. From this project our team members learnt that how we illustrated, regardless of its flaws, that a semaphore is very helpful to any OS.

# **REFERENCES**

Ahlawat, A. (n.d.). *Process Synchronization*. Study Tonight. Retrieved September 2, 2020, from https://www.studytonight.com/operating- system/process-synchronization

A.T. (n.d.). *Process Cooperation in Operating Systems: Definition & Examples*. Study. Retrieved September 10, 2020, from https://study.com/academy/lesson/process-cooperation-in-operating- systems-definition-examples.html

*Introduction of Process Synchronization*. (2019, September 11). GeeksforGeeks. https://www.geeksforgeeks.org/introduction-of- process-synchronization/

Jain, P. (2018, August 13). *Cooperating processes in the Operating System*. Include help. https://www.includehelp.com/operating- systems/cooperating-processes-in-the-operating-system.aspx

Meador, D. (2018, October 10). *Semaphores in Operating System*. Tutorial Points. https://www.tutorialspoint.com/semaphores-in-operating- system

*OS Paterson Solution - javatpoint*. (n.d.). JavaTpoint. Retrieved September 3, 2020, from https://www.javatpoint.com/os-paterson-solution

*Process Synchronization*. (2002, September 23). SpringerLink. https://link.springer.com/chapter/10.1007/0-306-46976-6\_7

rajkumarupadhyay515. (2020, April 28). *Concurrent Processes in Operating System*. GeeksforGeeks. https://www.geeksforgeeks.org/concurrent- processes-in-operating-system/

Rungta, K. (2020a, August 8). *What is Semaphore? Binary, Counting Types with Example*. Guru99. https://www.guru99.com/semaphore-in- operating- system.html#: %7E:text=Summary%3A, acquired%20or%20released% 20numerous%20times.

Rungta, K. (2020b, September 16). *Mutex vs Semaphore: What’s the Difference?* Guru99. https://www.guru99.com/mutex-vs- semaphore.html

*Semaphore in OS (Operating System)*. (2019, March 20). STechies. https://www.stechies.com/semaphore-os\_1/

*Semaphore Vs Mutex: What Is The Difference?* (n.d.). Viva Differences. Retrieved September 1, 2020, from https://vivadifferences.com/13- difference-between-semaphore-and-mutex/

*Semaphores in Process Synchronization*. (2019, November 20). GeeksforGeeks. https://www.geeksforgeeks.org/semaphores-in- process-synchronization/

Shriram Vasudevan. (2020, January 16). *Semaphore Vs. Mutex - A Clear Understanding*. YouTube. https://www.youtube.com/watch?v=8wcuLCvMmF8