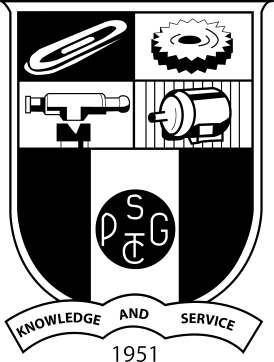
**BANKER’S ALGORITHM**

A report submitted by:

Bhargavi M (18PD06)

Harish S N (18PD10)

Subject : Operating Systems (18XD44)



Department Of Applied Mathematics and Computational Sciences

PSG College Of Technology

Coimbatore - 641004

**Abstract:**

This project on Banker’s Algorithm includes the need for this algorithm, how the algorithm works with a few examples and the implementation of the same using Pthreads in C.

**1.1 Introduction**

It is well known that operating systems may sometimes face a situation of being deadlocked. Such situations are often difficult to handle. However, operating systems use the concept of ‘Deadlock Avoidance’, in order to prevent deadlock, for the implementation of which, the Banker’s Algorithm is used.

Banker’s Algorithm is a resource allocation and deadlock avoidance algorithm designed to help operating systems check the state of a system (safe or unsafe) given a few processes and their respective resource needs and allocation, as well as the availability of resources in the system. It tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue or not.

The Banker's Algorithm derives its name from the fact that this algorithm could be used in a banking system to ensure that the bank does not run out of resources, because the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers . By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.

**1.2 Description**

**1.2.1 Data Structures**

Let ‘n’ be the number of processes in the system and ‘m’ be the number of resources types. Then the data structures used are defined as follows:

Available :

* It is a 1-d array of size ‘m’ indicating the number of available resources of each type.
* Available[ j ] = k means there are ‘k’ instances of resource type Rj

Max :

* It is a 2-d array of size ‘n\*m’ that defines the maximum demand of each process in a system.
* Max[ i, j ] = k means process Pi may request at most ‘k’ instances of resource type Rj

Allocation :

* It is a 2-d array of size ‘n\*m’ that defines the number of resources of each type currently allocated to each process.
* Allocation[ i, j ] = k means process Pi is currently allocated ‘k’ instances of resource type Rj

Need :

* It is a 2-d array of size ‘n\*m’ that indicates the remaining resource need of each process.
* Need [ i, j ] = k means process Pi currently allocated ‘k’ instances of resource type Rj
* Need [ i, j ] = Max [ i, j ] – Allocation [ i, j ]

**1.2.2 Pthreads**

A thread is a single sequence stream within a process. The Pthreads library is a POSIX C API thread library that has standardized functions for using threads across different platforms, which are defined as a set of C language programming types and procedure calls. It allows us to create multiple threads for concurrent process flow.

**1.2.3 Safety Algorithm**

This is an algorithm that is used to check if the given state of a system is safe or not by making use of the data structures mentioned above. The pseudo code is as follows:

Step 1. Initialize: Work= Available

Finish [i] = false; for i=1,2,……,n

Step 2. Find an i such that both

a) Finish [i] = false

b) Need[i] <= work if no such i exists goto step (4)

Step 3. Work = Work + Allocation[i]

Finish[i]= true goto step(2)

Step 4. If Finish[i]=true for all i,

then the system is in a safe state.

**1.2.4 Banker’s Algorithm**

Before the safety algorithm is run, the operating system is required to check for the conditions which on satisfaction calls for the Safety algorithm to complete the Banker’s algorithm. The conditions are as follows:

* Process cannot request more than its need
* Process request should be less than current available

**1.3 System Calls Used**

* **sleep()**: C programming language provides sleep() function in order to wait for a current thread for a specified time(in . sleep() function will take an integer value as its parameter that sleeps given thread, currently under execution for the specified time. Of course, the CPU and other processes will run without a problem.

**1.4 APIs Used**

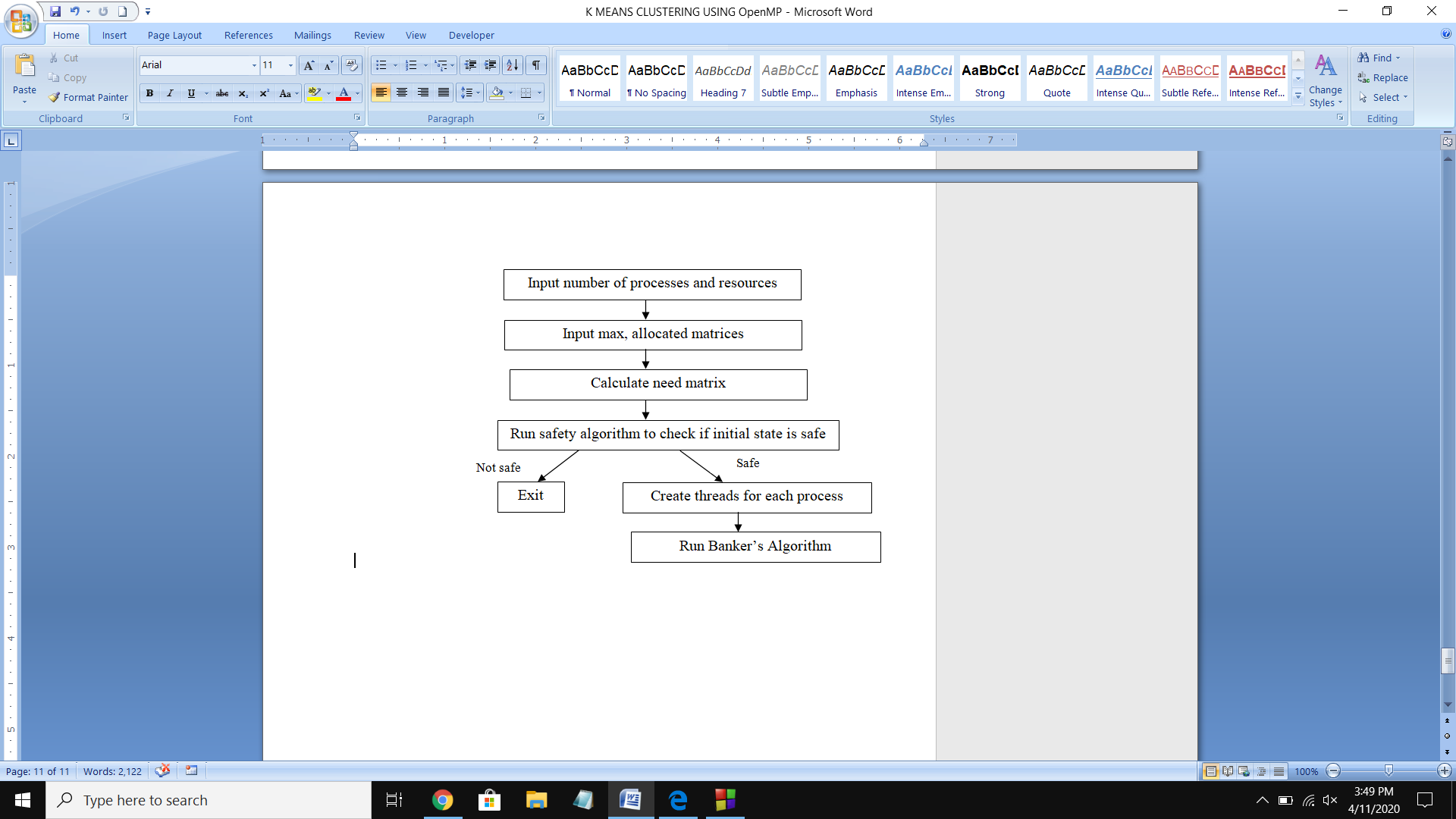
* **pthread\_mutex\_init():** The *pthread\_mutex\_init()* function initialises the mutex referenced by *mutex* with attributes specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect is the same as passing the address of a default mutex attributes object. Upon successful initialisation, the state of the mutex becomes initialised and unlocked.
* **pthread\_attr\_init(&attrDefault):** The pthread\_attr\_init() function initializes the thread attributes in the thread attribute object attr to their default values
* **pthread\_create():** The pthread\_create() function starts a new thread in the calling process.
* **pthread\_join():** The pthread\_join() function waits for the thread specified by *thread* to terminate. If that thread has already terminated, then pthread\_join() returns immediately. The thread specified by *thread* must be joinable.
* **pthread\_mutex\_lock():** The mutex object referenced by *mutex* shall be locked by calling *pthread\_mutex\_lock*(). If the mutex is already locked, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner.
* **pthread\_mutex\_unlock():** The mutex object referenced by *mutex* shall be locked by calling *pthread\_mutex\_lock*(). If the mutex is already locked, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner.

**1.5 Tools and Technology**

We have used POSIX (Portable Operating System Interface) threads for implementing multithreading in our program. POSIX is an IEEE standard designed to facilitate application portability. The main use of POSIX is that we can port them easily among a large family of Unix derivatives.

**1.6 Workflow**

The flowchart below depicts how the implementation of this project has been done in C using pThreads.



**1.7 Results and Discussion**

Once the operating system has got the input required, it creates threads and executes them. The threads when in execution request for resources. The operating system then runs the Banker’s Algorithm to check if resource allocation would lead to safe or unsafe state based on which it proceeds in resource allocation.

**1.8 Conclusion**

Given the number of processes, resources and the max and allocated matrix, the operating system can decide if the system would be safe or not using the Banker’s algorithm.

**1.9 Bibliography**

**1.9.1 Books**

* Abraham Silberchatz, Peter B Galvin, Gerg Gange, “Operating System Concepts 9th edition”, Wiley, 2015

**1.9.2 Websites**

* <https://www.geeksforgeeks.org/multithreading-c-2/>
* <https://www.wikitechy.com/technology/bankers-algorithm/>
* <https://www.poftut.com/what-is-sleep-function-and-how-to-use-it-in-c-program/>
* <https://linux.die.net/>