An Approach to Process Management using Process Synchronization

Deepti Sindhu[#1]
Assistant Professor
NCIT
Israna

Anupma Sangwan[*2]
Assistant Professor
GJUS&T
Hisar

Kulbir Singh[**3]
Senior S/w Engineer
bebo Technologies
Chandigarh

ABSTRACT
The main goal of the study is to know how multiple processes [1] of a computer system is managed. Early computers used to work on uniprocessing while currently all computers work on multiprocessing. Many programs could be executed at a time concurrently, while earlier computers used to allow execution of only one program at a time. Computers now can load and execute multiple programs from different processes. These tasks could be performed concurrently and managed properly. A system consists of number of jobs and processes related to operating system and user has to execute system code and user code. Now the query is this- How all these processes are managed along with management of Operating system? There are numerous such queries which rise in one's mind. A CPU can be made more productive by any operating system if CPU switches properly between processes. It is possible if CPU is synchronized by synchronization of processes.

General Terms
Process management and synchronization are the general terms for this paper.

Keywords
Process, Process States, Process Scheduling, Interprocess Communication, Process Synchronization, Critical Section.

1. INTRODUCTION
In A process is simply a program to execute system related operations. An operating system contains number of codes to perform various operations. Some necessary resources are required to execute any operating system's operations and jobs. These mandatory resources are: Memory, CPU time, some related files and input output devices, which are used for accomplishing OS tasks. The resources can be allocated either at the time of process creation or while process execution. A process is a unit of task. Any operating system consists of number of processes.Operating System [11] processes and user processes could be differentiated on the basis of their tasks; i.e. former are used to execute system codes while later are for user codes. These processes could be executed individually and concurrently. All traditional operating system's processes used to contain single thread while all modern OS processes contain multiple threads.

Operating system is managed properly only by process management. Before studying how to manage process it is necessary to study process and its states. Here process synchronization [8] algorithms are implemented for process management.

2. BASIC CONCEPTS OF PROCESS
Process if simply defined is an instance of a program to be executed [4]. But a process is different from a program. From operating system point of view, both are hard to differentiate, once created, a process is activated to perform some tasks for system and system's resources. So it can be considered that program is not a process itself, but is a passive entity of a process. All programs are not processes but mostly processes can be considered as programs. A process is an active concept or entity while program is a passive concept or entity. One or more than one process may be associated with a single program which can be considered to different execution sequences.

But question is that what all CPU activities should be called? There are numerous batch process and user programs. A single user system can run multiple program at a time, for example: a user can run multiple programs using a single user system such as Microsoft Windows. User can use a word processor, e-mail and listen to music simultaneously. It seems to execute many programs at a time, but operating system executes programs in queues using memory management. A process is a program in execution that is written in text code. It is more than a program. After passing through text section a process is represented by program counter value and registers. A process also has a stack containing some temporary data like functional parameters, variable and return addresses. Global variable are contained in data section.

3. PROCESS STATES
A process before execution passes through various states as shown in Figure 1. While passing these states if any interrupt occurs, a process can terminate immediately instead of running or execution.While execution, a process change its state. States of a process are divided in parts. Each state, from which a process passes, becomes its current state. A process can be in one of following states. The state of a process may be new, ready, running, waiting (suspend) and terminate.

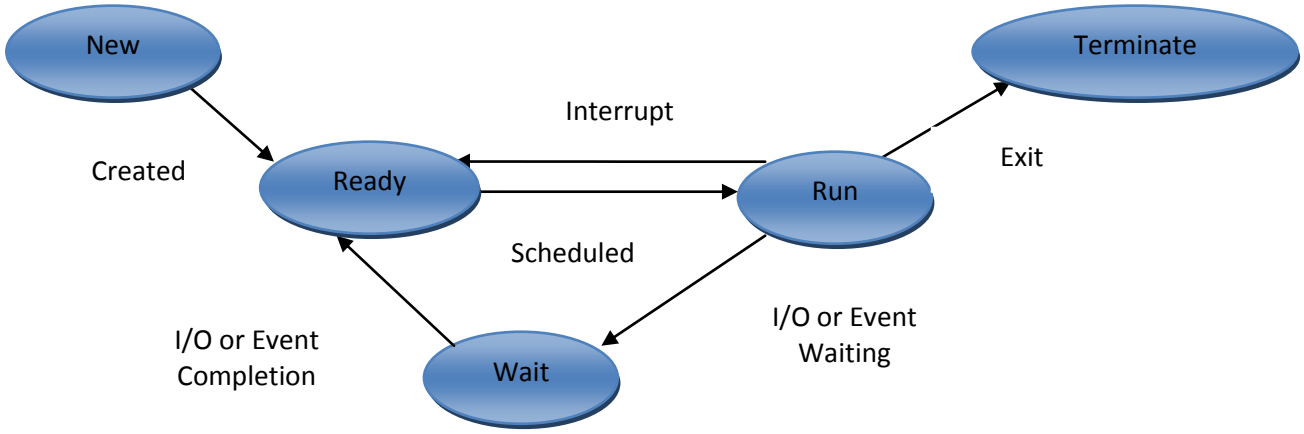

Fig 1: States of Processes during execution

- **New:** The process and programs that have not yet been created or entered into operating system.
- **Ready:** The process as soon as created, enter into ready state. All ready process are listed and CPU selects one for execution as per scheduling policy.
- **Running:** Only one process is executed by CPU at a point of time. The process possesses all necessary resources for its execution. In single processing system, only one process is executed at a time, but in multiprogramming system there may be multiple CPUs with multiple running processes.
- **Waiting:** When a process is waiting for I/O operating or some other event. This state is also called suspend state.
- **Terminate:** When the process finish execution or exits from final state.

3.1 Process Control Block

Process Control Block that abbreviated as PCB represents each process in any operating system [4][11]. It is also called Task Control Block. See Figure 2 for PCB.The main objective of using multiprogramming OS is to run multiple processes simultaneously. A process is represented physically in the memory of a computer in a proper data structure that includes some elements:

Pointer	States (Process)
Number (Process)	
Program Counter	
CPU Registers	
Memory Limits	
List of Open Files	
:	

Fig 2: Process Control Block

This system maximizes the CPU utilization by frequently switching the CPU among the processors so that user can interact with all running programs [2]. CPU executes the jobs on some scheduling criteria. Before uniprocessor OS could run only one job at a time, but by multiprocessing operating system CPU can run many processes concurrently. Scheduling process is decided by scheduling policies and methods.

4. PROCESS SCHEDULING

Each process is put into job queue after entering into the system. All processes are queued which enter the system. These processes pass through process execution cycle one by one according to their performance and scheduling criteria. Each process first enters into ready queue and wait for execution. This task is performed in main memory. The job that needs to be executed is swapped into main memory in form of stack and pointer points to the process to be executed currently. After execution process quits the queue and next process enter that was in waiting state.

4.1 Schedulers

An operating system will select the scheduler or scheduling criteria. There can be many processors that need to be executed immediately. This can be performed by proper schedulers. There are mainly two types of schedulers:

- Long Term Scheduler or Job Scheduler
- Short Term Scheduler or CPU Scheduler

The main difference between two schedulers is the execution frequency. A short term scheduler or CPU scheduler selects a process for the CPU very frequently while long term scheduler or job scheduler executes less frequently.The scheduling or performance depends upon some criteria:

- CPU Utilization
- Throughput
- Waiting Time
- Response Time
- Turnaround Time

4.2 Process Operations:

Multiple processes run concurrently in most modern computer system and create and terminate process frequently [6]. As soon as a process is executed successfully is deleted from queue from main memory and the next process is created by CPU. A process can create many other processes.

Here the creator is called **Parent process** and new processes are called **Children processes.**

Some necessary resources, as discussed earlier section are required for accomplishing its job. Children processes are sub processes. There are some possibilities related to these processes.

- The parent process waits until children processes terminate.
- Parent process also execute simultaneous as child process executes.

After completion of execution, the process terminates using **exit system call**.

5. INTERPROCESS COMMUNICATION

Interprocess communication or IPC is the facility to exchange information or data among threads in one or more processes. IPC also provides the facility to synchronize the actions of processes during communication without sharing same address space.Cooperating process communicate in shared memory environment. On the other hand interprocess communication works in distributed memory environment where the processes can be on different system. Process synchronization is useful to maintain consistency among processes. Methods of IPC are:

- Message Passing System
- Process Synchronization
- Buffering

5.1 Message Passing System

This system provides the facility of message passing to communicating processes without sharing data and variable. Communication is performed by passing of message among various communicating process. Message of communication can be fixed or variable size. System level implementation of fixed size message is simple and on the other hand system level implementation of various size messages is more complex.

Suppose P and Q are two processes to communicate. They will send and receive messages to and from each other via a communication link.

Various methods for implementation of Send/Receive operations are:

- Direct and Indirect Communication Method
- Symmetric and Asymmetric Communication Method
- Send by Copy and Send by Reference Method
- Fixed Sized and Variable Sized Communication Method

5.2 Synchronization

Process synchronization is a method to provide consistency. While sharing data with other processes may result inconsistency. Race condition occurs while two or more process read or writes some share data. Result depends who run last precisely.

There are different design options of message passing i.e. blocking or non-blocking sending and receiving or their different combinations.

5.3 Buffering

Messages are exchanged while communication of processes, reside temporarily in some queues. These queues can be implemented according to their queue length capacity in three ways, which are:

- Zero Capacity- Queue Max length 0.
- Bounded Capacity- Queue has finite length *n.*
- Unbounded Capacity- Queue has infinite length.

Counter is incremented each time an item is added to the buffer and decremented when removed from the buffer. Data mining is considered as container of business

6. SOME ISSUES IN PROCESS SYNCHRONIZATION

6.1 Critical Section Problem

A system can have number of processes. Suppose a system with n processes, Critical Section is the segment of code where shared memory is accessed, exchange common variable, write file etc. At most one process can execute in critical section of a system. No other process can enter in critical section until a program execute. Process is based on mutual exclusion [10][12] and used for cooperation of communicating processes. Each system has an ***entry*** section and an ***exit*** section.

The process need to be executed request the entry section (Critical Section) after execution exits through exit section. Other processes reside in remainder section.

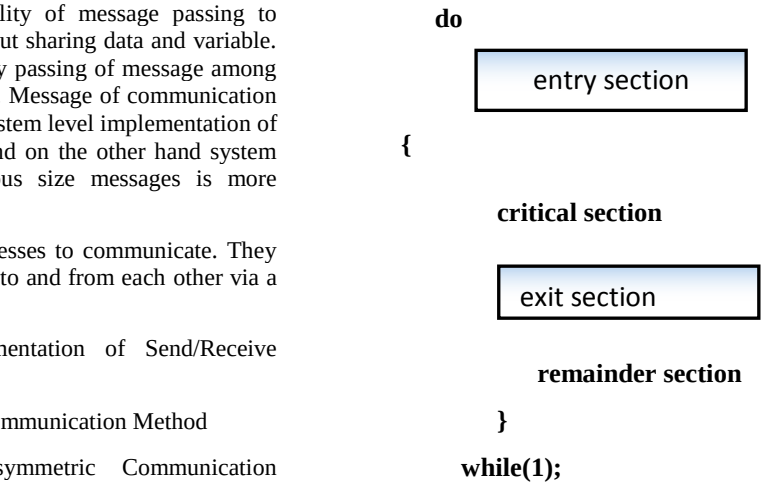

Fig 3: General Structure of Process

There are some algorithms to solve Critical Section problems. Algorithm 1, 2 and 3 give solution for Two Processes. Bakery algorithm gives solutions to Multiple Process. For solving the critical section problem some requirements must be fulfilled:

- Mutual Exclusion: Is a process is being executed in CS then no other process will be executed in that CS.
- Bounded Waiting: Other processes can enter into their CS after a process has made a request enter to into CS and before request is guaranteed.

- Progress: If no process is there in CS for execution then some other requesting process that is not executing in remainder section can enter into CS for execution after get selected to be entered in CS.

Here the purpose of solution algorithms is synchronization.

7. CLASSICAL PROBLEMS OF PROCESS SYNCHRONIZATION

There are some synchronization problems which are used for testing newly proposed synchronization.

7.1 Bounded Buffer Problem

It is also called Producer-Consumer problem. It is called so because of dealing with two processes; one is producer and other is consumer. Both share a common buffer and run concurrently. Bounded buffer means each buffer contains one record at a time.A producer process generates a single record at a time and sends to the buffer. Consumer process also consumes a single record at a point of time. More record can't be written in buffer if buffer is full and record can't be copied out if buffer is empty means no record in buffer. Some necessary constraints for solution of producer consumer are as below:

- A producer process cannot write if a buffer is full.
- A consumer cannot consume a record if buffer is empty.
- Producer and consumer follow mutual exclusion.
- The record entered or produced first, will be consumed first; means FIFO manner.

Producer consumer process performs in critical section. A producer process produces after entering in CS if buffer is empty else exits from critical section. Example of this problem is message inbox. No message can enter into inbox if inbox is full and no message can be read if message inbox is empty. It is called Bounded buffer because of its bounded or fixed size.

Structure of Producer

```
while produced = false
{
    if buffer is empty
    then
    {
        produce in buffer
    }
    produced = true;
}
```

Structure of Consumer

```
while consumed = false
{
    if there exists a buffer
    then
    {
        consume from buffer
    }
    consumed = true;
}
```

The mutual exclusion semaphores will not allow two processes to use a buffer concurrently.

7.2 Readers Writers Problem

Reader-writer is a problem in which number of concurrently executing processes want to access some particular object [9]. The process that extracts or reads shared information is reader process and another process that inserts or edits some data is writer information. Some rules on reader-writer problem are:

- No of readers can read concurrently but at most one writer can write at a point of time.
- Reading and writing process cannot perform simultaneously. Reader has to wait until writing.
- Process is accessing any data due to mutual exclusion. On the other hand if reader is performing on any object and writer wants to access that object, he will get the permission. Priority is given to writer process.

Structure of Reader Process

```
begin
 if writer is processing
 then
   {wait};
     {read};
 if writer is waiting
 then
   {write};
end;
```

Structure of Reader Process:

```
writer process
 while(true)
 {
  wait(writing)
    when writer is waiting
  signal(writing)
 }
```

7.3 Dining Philosophers Problem

Consider 5 philosophers are sitting on a dining with chairs, who want to eat a tasty dish kept in the centre of the table. Five plates are arranged in front of each chair. But there are only 5 chopsticks. Each person wants to eat with two chopsticks so everyone can't eat simultaneously. So each person has to wait for his left and right closest chopstick to get free. So each person eats for some time and sits idle and start thinking. Others wait for the chopsticks to become free that is already in his left and right neighbor's hands. If a philosopher does not release chopstick and keeps eating continuously for long time then deadlock like situation occurs. If he does not release chopstick until his dish

finished others have to wait for infinite time and other can get starved.

Dining philosopher problem is like classical synchronization problem that is equal to concurrency control [8] problem. Some simple solutions for such problems can be:

- By allowing four persons to sit on dining
- Philosophers should be allowed to pick two chopsticks if both of his sides are free.
- Use even or odd formula, means at a time either all even philosophers should be allowed to pick chopsticks or odd one.

8. VARIOUS APPROACHES OR SOLUTION TO CRITICAL SECTION PROBLEM

8.1 Semaphores

Semaphore [7] is a tool for solving the difficulties of critical section problem which are very complex and not easy to generalize. ***Sem*** is a variable that can be accessed through two atomic operations which are **wait** and **signal.** These operations originally were termed as P for **wait** (from the **Dutch** Problem to **test**) and **V** for **signal** (from **Verhogen**, to **increment**).

The pseudo-code for semaphores:

```
wait operation (Sem)
{
   while(Sem ≤ 0)
   Sem- -;  //Decrement Semaphore
 }
signal operation (Sem)
{
   Sem++;  //Increment Semaphore
}
```

Both operations will synchronize their execution and will modify according o each other. No two processes can modify simultaneously same Semaphore.

8.2 Mutex Locks

Mutex Locks [3] is software that is an approach to synchronize hardware. A lock is acquired in critical section where resources are acquired and lock is released at the exit. The lock is used to proper execution of a process in critical section and when the process is executed properly the lock is released.

9. CONCLUSION

Hence from the above study it is concluded that no OS can be managed without proper management of processor and processes. Different algorithms and approaches have been used to solve difficulties of critical section where all the processes are executed. Some approaches which are used for synchronization problems are Semaphores, Mutex Locks etc [14]. These approaches make the implementation of critical section efficient and hardware support for atomic operations but if these approaches are not used properly can create problem. Implementation of mutual exclusion is also necessary to avoid deadlock like situations. Process synchronization makes multiprogramming and multiprocessing very efficient.

10. ACKNOWLEDGMENTS

Our thanks to all the experts who have contributed towards development of this paper.

11. REFERENCES

[1] Ben M. and Ari 1990, Principles of Concurrent and Distributed Program. Prentice Hall.

[2] Jahorjan J. and McCann C. 1990 Processor Scheduling in Shared Memory Multiprocessors. Proceedings of the Conference on the Measurement and Modeling of Computer Systems.

[3] Keag M. and Wilson R. 1976 Studies in Operating System. Academic Press.

[4] Knuth E. 1966 Additional Comments on a Problem in Concurrent Program Control. Comm. of ACM, Vol. 9(5): 321-322.

[5] Kosaraju S. 1973 Limitation of Dijkstra's Semaphore Primitives and Petri Nets. Operating System Rev., Vol.7(4): 122-126.

[6] Lamport L. 1976 Synchronization of Independent Processes. Acta Informatica, Vol. 7(1): 15-34.

[7] Lamport L. 1977 Concurrent Reading and Writing. Comm. of ACM, Vol. 20(11): 806-811.

[8] Lamport L. 1986 The Mutual Exclusion Problem. Comm. of ACM, Vol. 33(2): 313-348.

[9] Lamport L. 1991 The Mutual Exclusion Problem has been Solved. Comm. of ACM, Vol. 34(1): 110.

[10] Raynal M. 1986 Algo. for Mutual Exclusion. MIT Press.

[11] Silberscartz, A. Galvin, B. and Gagne 2003 Operating System Concepts. 6th Edition, John Wiley.

[12] Tanenbaum, A. 2001 Modern Operating System. 2nd Edition, Prentice Hall.

[13] Tucker A. and Gupta A. 1989 Process Control and Scheduling Issues for Multiprogrammed Shared Memory Multiprocessors. Proc. of ACM Symposium on OS Principles.

[14] William Stallings 2000 Operating Systems. 4th Edition, Prentice Hall.