**User Manual**

1. **CPU Scheduling Algorithms**

* **First come First Serve**

It is the simplest scheduling algorithm. It simply queues processes in the order that they arrive in the ready queue. In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

* **Shortest Job First**

It is the best approach to minimize waiting time. In this, the process that has shortest job will be executed first and next process starts only after the previous gets fully executed.

* **Round Robin**

It is an arrangement of choosing all elements in a group equally in some rational order, usually from the top to the bottom of a list and then starting again at the top of the list and so on.

* **Priority**

In this each process is assigned a priority. Process with the highest priority is to be executed first and so on. Processes with the same priority are executed on first come first serve basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

1. **Process Synchronization**

* **Producer**

Who produces the data items when buffer is empty is called Producer

* **Consumer**

Who consumes the data items when buffer has at least one element is called Producer

* **Producer Consumer Problem**

The problem describes two processes, the producer and the consumer, who share a common, fixed-size [buffer](https://en.wikipedia.org/wiki/Buffer_(computer_science)) used as a [queue](https://en.wikipedia.org/wiki/Queue_(data_structure)). The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

* **Solution**

The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

The solution can be reached by means of inter process communication using semaphore.

1. **Deadlock Avoidance**

* **Deadlock**

A deadlock is a state in which each member of a group is waiting for another member.

**Requirements of a deadlock:**

1. Mutual exclusion

2. Hold and wait

3. No pre-emption

4. Circular wait

**Handling a deadlock:**

1. Deadlock prevention and avoidance
2. Deadlock detection and recovery
3. Ignore the problem all together

* **Banker’s Algorithm**

It is a resource allocation and deadlock avoidance algorithm that tests for the safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes a ***‘s-state’*** check to test for allocation should be allowed to continue or not.

This algorithm works by taking the number of processes and the number of resources in a matrix. It also needs a matrix which represents the maximum resource and also a matrix of size 1Xn (where 1 = number of row and n = number of columns) which represents the available matrix.

1. **Page replacement Algorithms**

The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).

* **Optimal Page Replacement algorithm**

 This algorithm replaces the page which will not be referred for so long in future. Although, it cannot be practically implementable but it can be used as a benchmark. Other algorithms are compared to this in terms of optimality.

* **Least recent used (LRU) page replacement algorithm**

 This algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future.

* **FIFO**

 In this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rare end of the queue will be replaced on the every page fault.

1. **Process Creation**

* **PROCESS:-**

A process can be thought of as a program in execution

A process needs certain resources – such as CPU time, memory, files, I/O devices to accomplish its task. These resources are allocated to the process either when it is created or while it is executing.

* **Process State:-**

As a process executes, it changes State. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:-

New – The process is being created.

Running – Instructions are being executed.

Waiting – The process is waiting for some event to occur.

Ready – The process is waiting to be assigned to a processor.

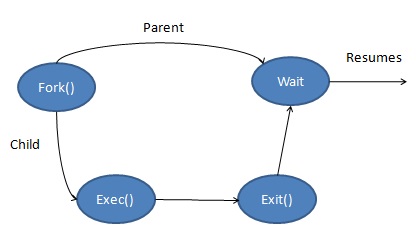
Terminated – The process has finished execution.

* **Process Creation**

A process can creates new several process via a create system call.

Create process system calls are-

Fork (), exec ()



* **Process Termination**

A process terminates when it finishes executing its last statement. Its resources are returned to the system, it is purged from any system lists or tables, and its process control block (PCB) is erased i.e., the PCB's memory space is returned to a free memory pool. The new process terminates the existing process, usually due to following reasons:

* **Normal Exist**    Most processes terminates because they have done their job. This call is exist in UNIX.
* **Error Exist**    When process discovers a fatal error. For example, a user tries to compile a program that does not exist.
* **Fatal Error**    An error caused by process due to a bug in program for example, executing an illegal instruction, referring non-existing memory or dividing by zero.
* **Killed by another Process**    A process executes a system call telling the Operating Systems to terminate some other process. In UNIX, this call is kill. In some systems when a process kills all processes it created are killed as well (UNIX does not work this way).