Project OS

Implementation of

CPU Scheduling Algorithms

Disk Scheduling Algorithm

Page Replacement Algorithms

Deadlock and Concurrency Algorithms

Overview

This Project was divided into three groups. Our team was 32 and has successfully implemented all the topics covering Concurrency and Deadlock.

Specifications

The Application has a sleek blue white minimal design which you can easily get to use on your own without this guide. This guide will instruct every step to take inorder to fully utilize our application with proper theoretical explanation given as well. The Homepage of the App contains four buttons from which one can choose his/her desired task (algorithm).

Full Team Member List

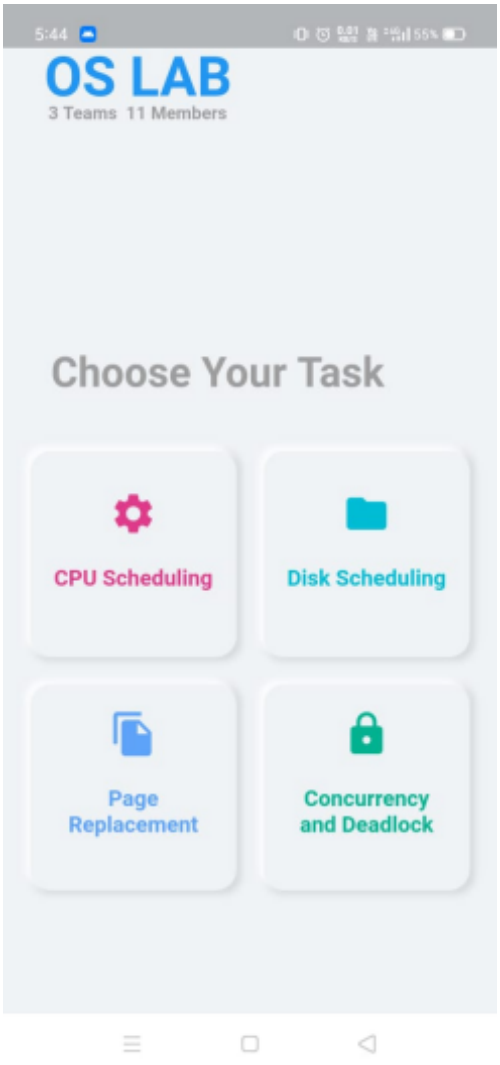
1. Team 26 (Disk Scheduling Project)
   - Devansh Shah 19BCP027
   - Dwireph Parmar 19BCP041
   - Geet Sharma 19BCP042
   - Harshal Shah 19BCP049
   - Kohav Yadav 19BCP069
2. Team 31 (Page Replacement Project)
   - Nisarg Koradia 19BCP088
   - Shashank Kumar 19BCP120
   - Aditya Kishtawal 19BCP147
3. Team 32 (Deadlock and Concurrency Project)
   - Mustafa Africawala 19BCP083
   - Parth Raval 19BCP090
   - Pathik Viramgama 19BCP093

Instructions for Homepage

The Homepage of the App contains four buttons from which one can choose his/her desired task (algorithm).

The Algorithms on the homepage are created by the following teams :

CPU Scheduling	:	Team 31
Disk Scheduling	:	Team 26
Page Replacement	:	Team 31
Concurrency and Deadlock	:	Team 32



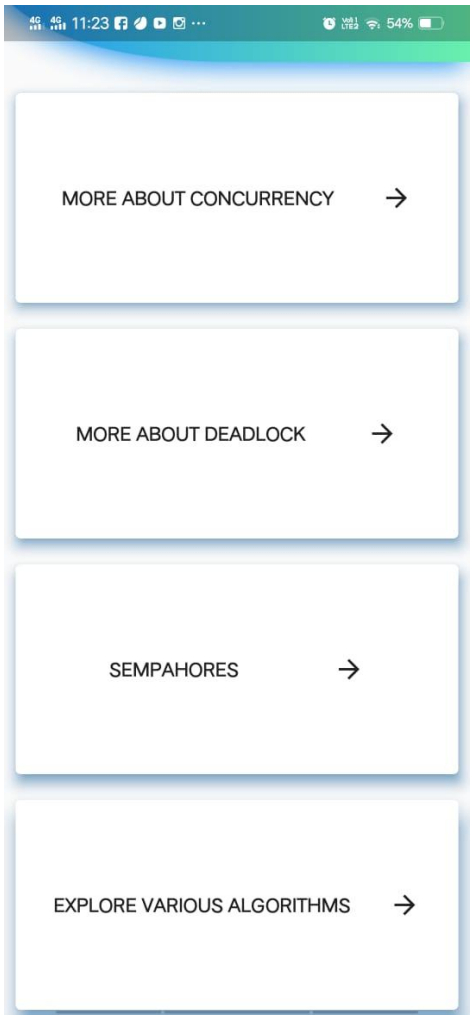
Instructions for Concurrency and Deadlock Project

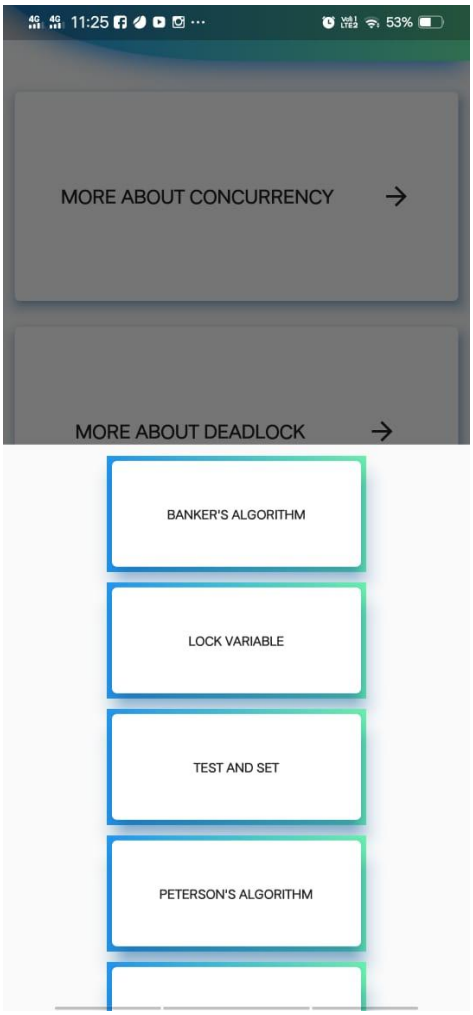
1. Guiding through main page interface

It is a simple sleek design with our application and you can surf through all buttons and arrows to the place you want to go.

We have four main screen pages. Top three buttons lead you to pages containing in-app instructions and theory regarding the whole topic itself.

The last button opens up the list of all algorithms that have been implemented. You can select your desired algorithm for implementation.

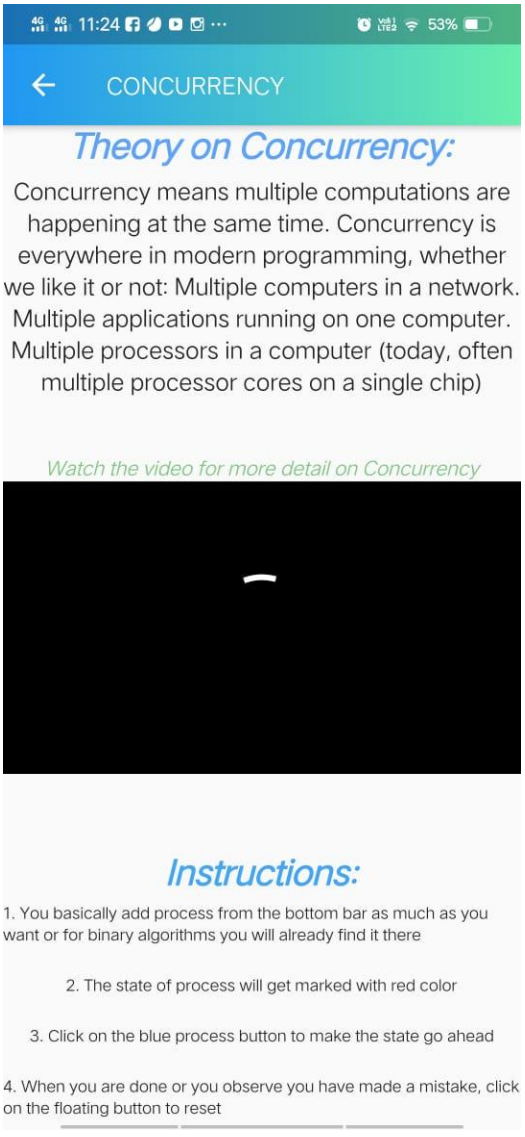




2. More About Concurrency

Concurrency means multiple computations are happening at the same time. Concurrency is everywhere in modern programming, whether we like it or not: Multiple computers in a network. Multiple applications running on one computer. Multiple processors in a computer (today, often multiple processor cores on a single chip). Simply described, it's when you are doing more than one thing at the same time. Not to be confused with parallelism, concurrency is when multiple sequences of operations are run in overlapping periods of time.

If you click on the tab in application you will find this information with a video explaining the whole concept. You will also find instructions to operate all algorithms working on concurrency.



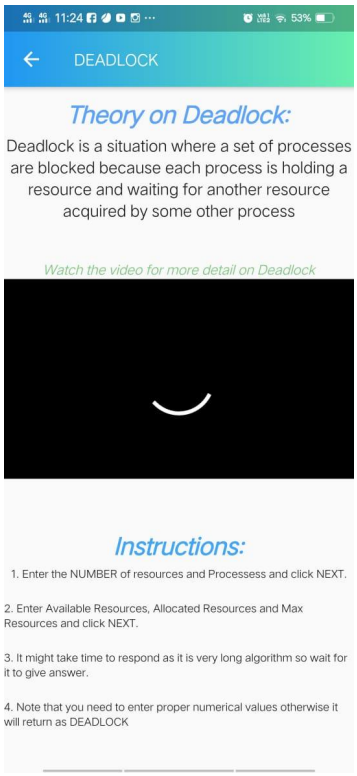
3. More About Deadlock

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. Resource Deadlock. Occurs when processes are trying to get exclusive access to devices, files, locks, servers, or other resources. In the Resource deadlock model, a process waits until it has received all the resources that it has requested. In a database, a deadlock is a situation in which two or more transactions are waiting for one another to give up locks. All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.

Four Necessary and Sufficient Conditions for Deadlock. Violate these and Deadlock is avoided.

1. Mutual exclusion.
2. Hold and wait or partial allocation.
3. No pre-emption.
4. Resource waiting or circular wait.

If you click on the tab in application you will find this information with a video explaining the whole concept. You will also find instructions to operate all algorithms working on Deadlock concept.



4. More About Semaphores

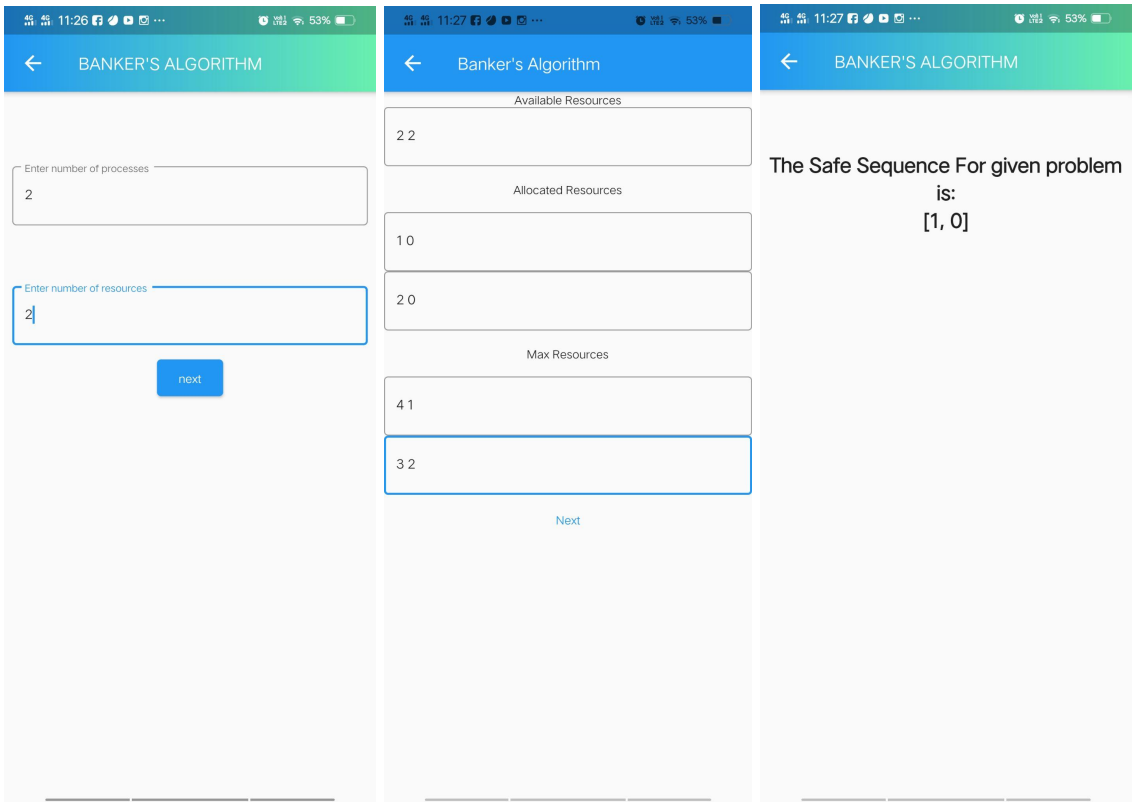
A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread. It uses two atomic operations, 1) wait, and 2) signal for the process synchronization. A semaphore either allows or disallows access to the resource, which depends on how it is set up. Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization. The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed. Semaphores which allow an arbitrary resource count are called counting semaphores, while semaphores which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called binary semaphores and are used to implement locks.

If you click on the tab in application you will find this information with a video explaining the whole concept. You will also find instructions to operate all algorithms working on Semaphores concept.

5. Banker's Algorithm

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue. Banker's Algorithm is used majorly in the banking system to avoid deadlock. It helps you to identify whether a loan will be given or not. This algorithm is used to test for safely simulating the allocation for determining the maximum amount available for all resources. Similarly, it works in an operating system. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as deadlock avoidance algorithm or deadlock detection in the operating system.

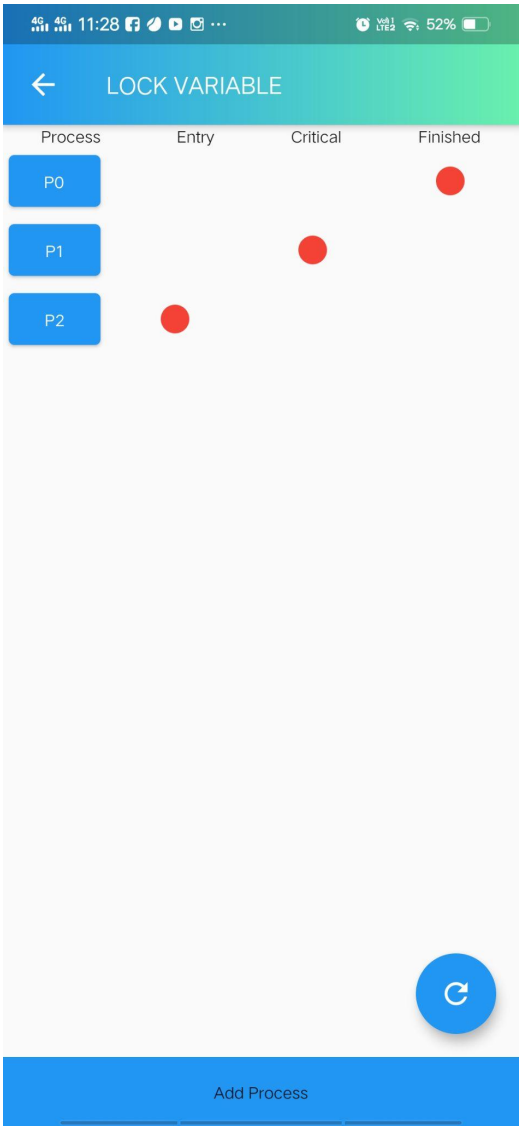
When you click on the tab of Banker's Algorithm, it leads you to ask the number of processes and resources. Enter an integer value and next. Now you enter the available array, allocated matrix and max matrix. Click on next. You will get the answer in the form of an array.



6. Lock Variable

A lock variable provides the simplest synchronization mechanism for processes. It's a software mechanism implemented in user mode, i.e. no support required from the Operating System. It's a busy waiting solution (keeps the CPU busy even when it's technically waiting). It can be used for more than two processes. When Lock = 0 implies the critical section is vacant (initial value) and Lock = 1 implies critical section occupied.

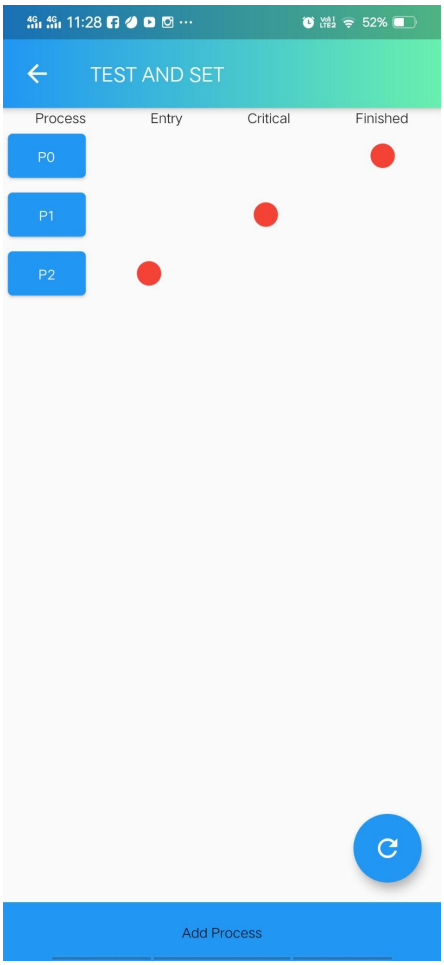
We have a simple interface. Click in add process to add process in the system. Click on the process to move it forward. Red light will turn on below the respective column. If the laws permit you will successfully complete the task. It wont move if it is not feasible. The other button on bottom right will reset the system to start over.



7. Test and Set Pointer Variable

The shared variable is lock which is initialized to false. TestAndSet(lock) algorithm works in this way – it always returns whatever value is sent to it and sets lock to true. The first process will enter the critical section at once as TestAndSet(lock) will return false and it'll break out of the while loop. The other processes cannot enter now as lock is set to true and so the while loop continues to be true. Mutual exclusion is ensured. Once the first process gets out of the critical section, lock is changed to false. So, now the other processes can enter one by one. Progress is also ensured. However, after the first process any process can go in. There is no queue maintained, so any new process that finds the lock to be false again, can enter. So bounded waiting is not ensured.

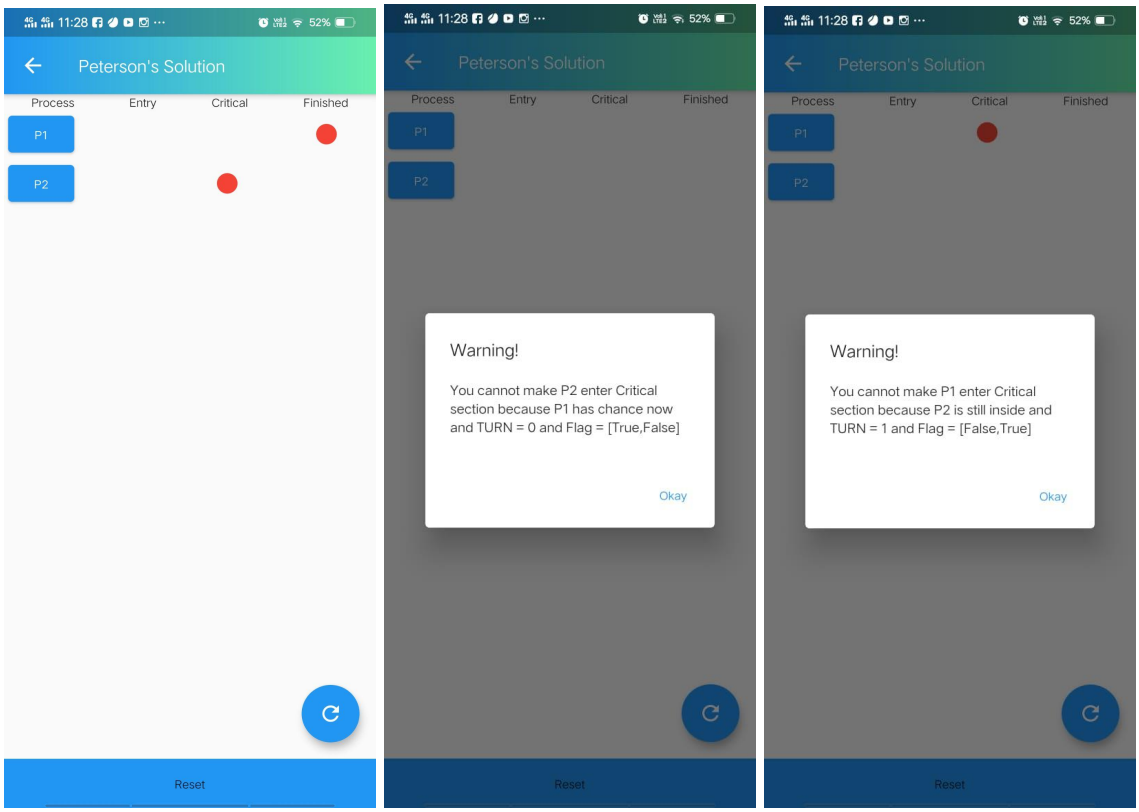
We have a simple interface. Click in add process to add process in the system. Click on the process to move it forward. Red light will turn on below the respective column. If the laws permit you will successfully complete the task. It wont move if it is not feasible. The other button on bottom right will reset the system to start over.



8. Peterson's Solution

Peterson's algorithm (or Peterson's solution) is a concurrent programming algorithm for mutual exclusion that allows two or more processes to share a single-use resource without conflict, using only shared memory for communication. It was formulated by Gary L. Peterson in 1981.Peterson's solution works for two processes, but this solution is best scheme in user mode for critical section. This solution is also a busy waiting solution so CPU time is wasted. So that "SPIN LOCK" problem can come. And this problem can come in any of the busy waiting solution.

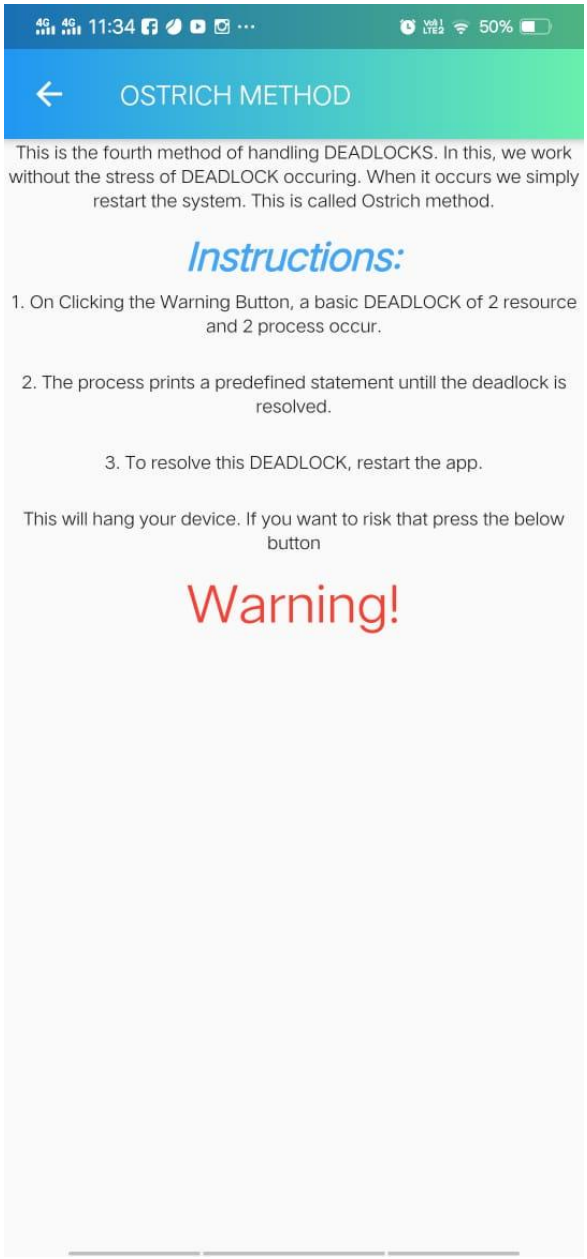
We have a simple interface. Click on the process to move it forward. Red light will turn on below the respective column. If the laws permit you will successfully complete the task. It wont move if it is not feasible and will display a message regarding why you cannot do the operation. The other button on bottom right will reset the system to start over.

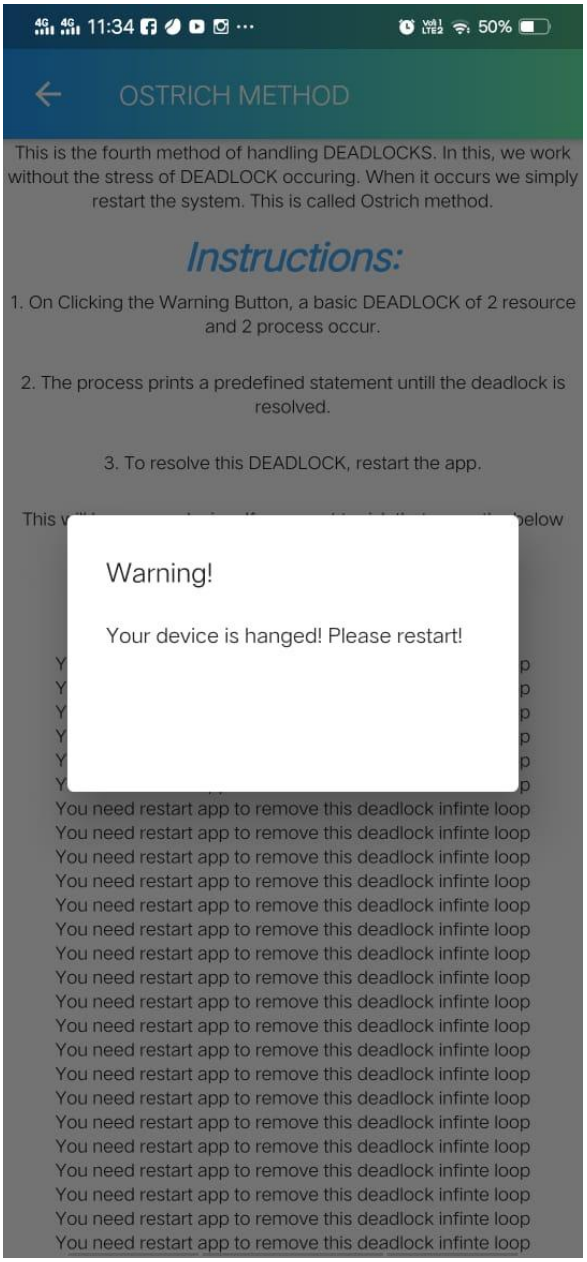


9. Ostrich Method

In computer science, the ostrich algorithm is a strategy of ignoring potential problems on the basis that they may be exceedingly rare. It is named for the ostrich effect which is defined as "to stick one's head in the sand and pretend there is no problem".

This is a very risky algorithm. Click on the warning button and your app will hang if your system is a low end one.

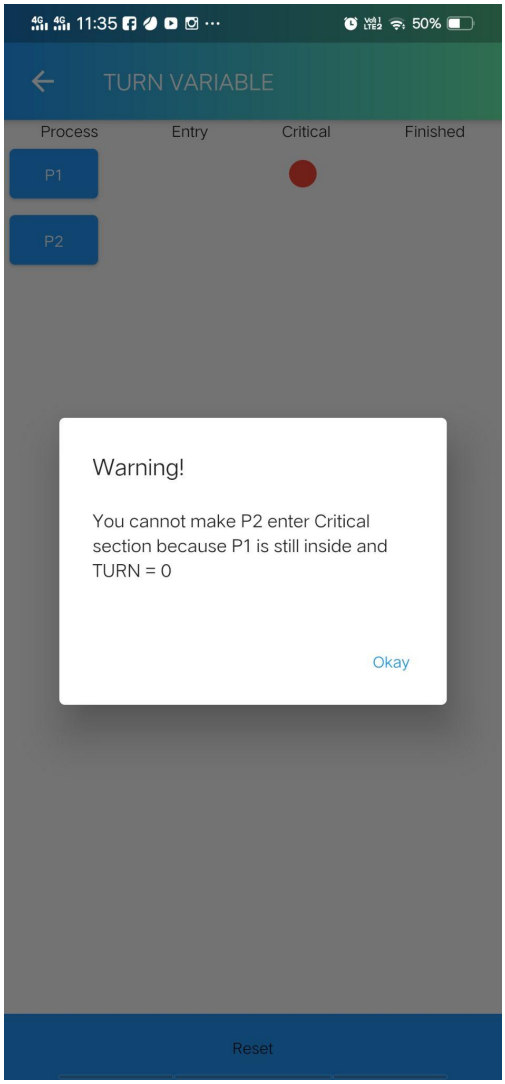


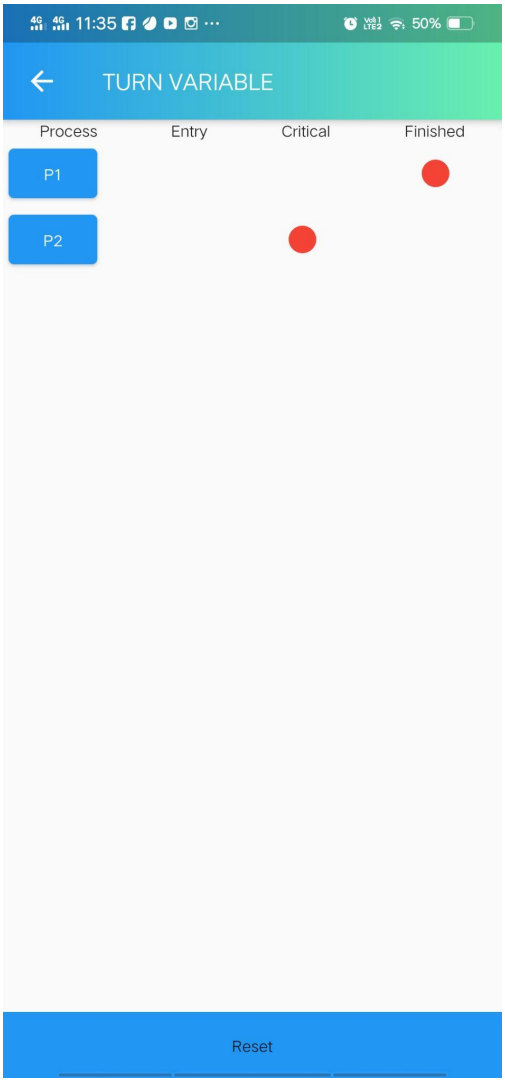


10. Turn Variable

Turn Variable or Strict Alternation Approach is the software mechanism implemented at user mode. It is a busy waiting solution which can be implemented only for two processes. In this approach, A turn variable is used which is actually a lock.

We have a simple interface. Click on the process to move it forward. Red light will turn on below the respective column. If the laws permit you will successfully complete the task. It wont move if it is not feasible and will display a message regarding why you cannot do the operation. The other button on bottom right will reset the system to start over.

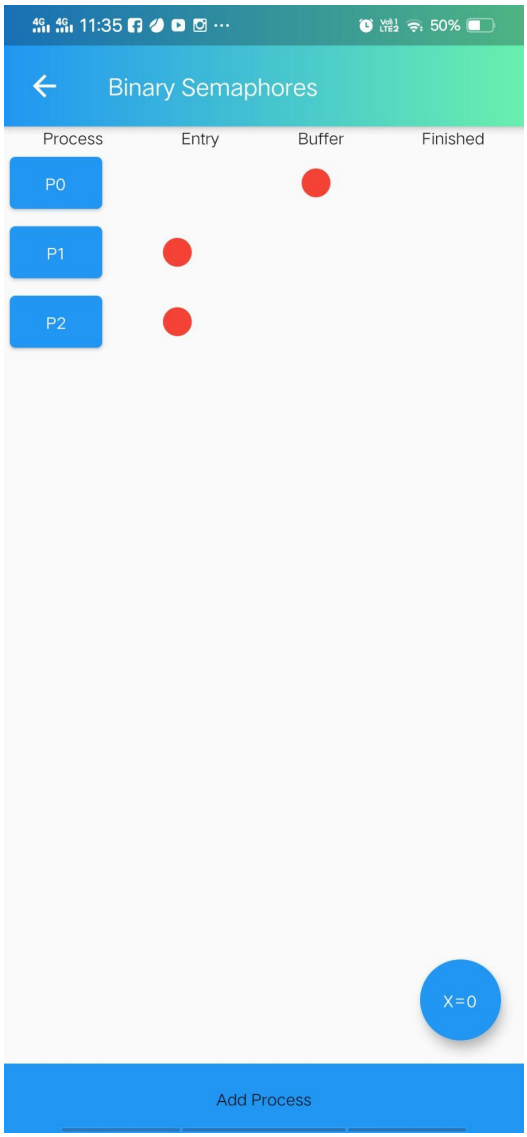


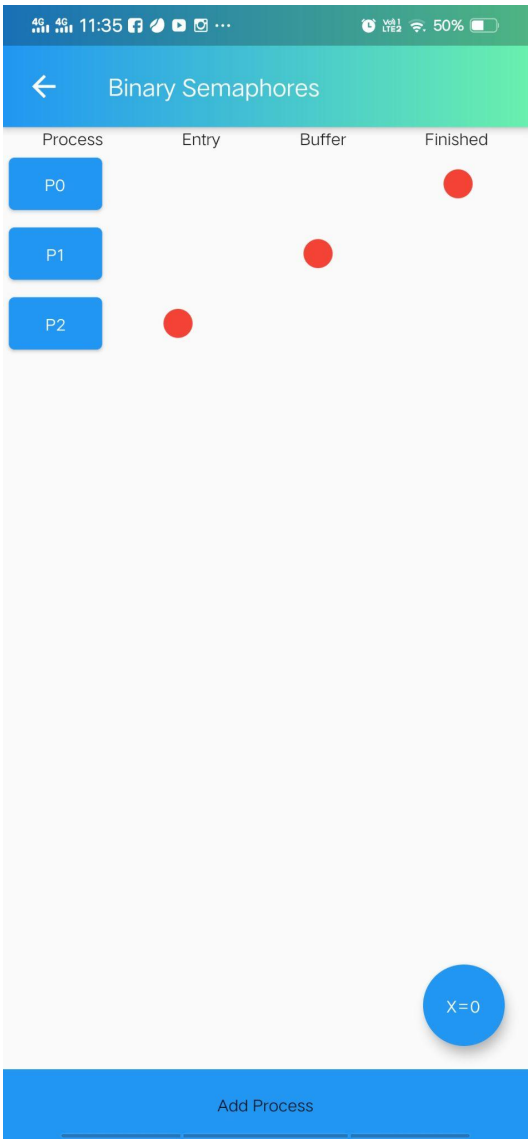


11. Binary Semaphore

The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0.

We have a simple interface. Click in add process to add process in the system. Click on the process to move it forward. Red light will turn on below the respective column. If the laws permit you will successfully complete the task. It wont move if it is not feasible. The other button on bottom right will reset the system to start over.

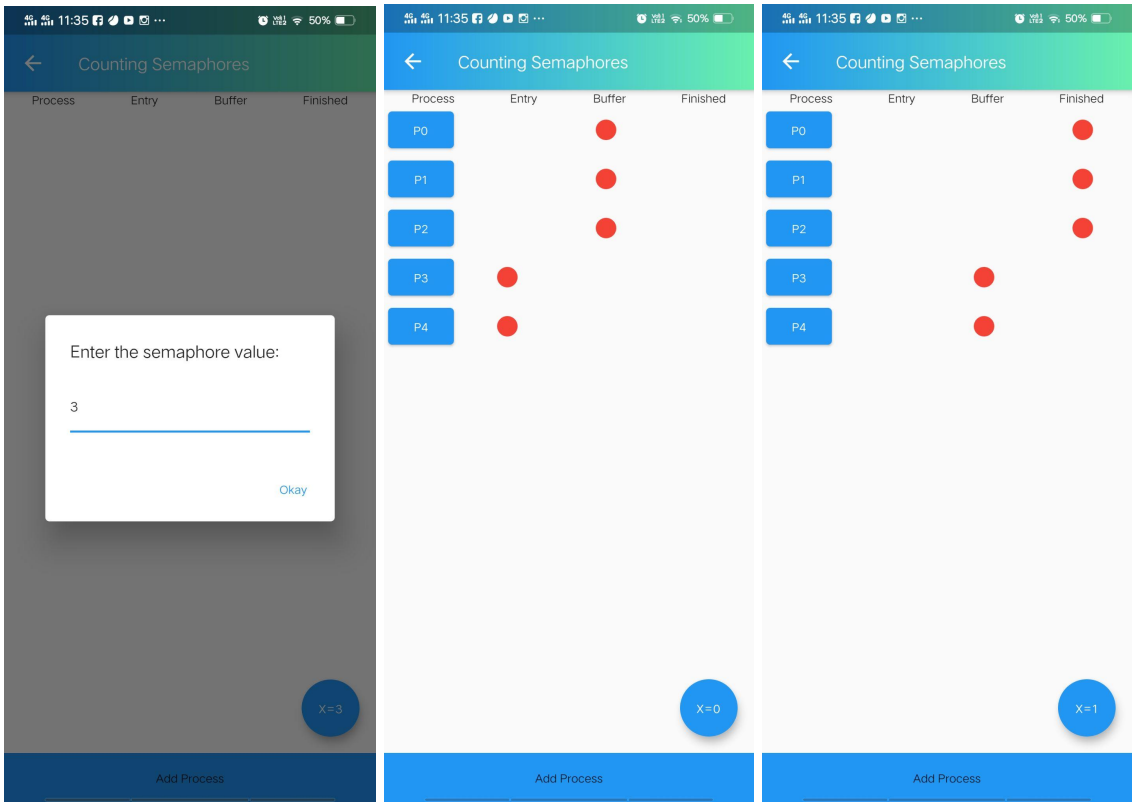




12. Counting Semaphore

The value of counting semaphore at any point of time indicates the maximum number of processes that can enter in the critical section at the same time. A process which wants to enter in the critical section first decreases the semaphore value by 1 and then checks whether it gets negative or not.

We have a simple interface. Click on the X value button to enter the value of Counting Semaphore. Same button is used to reset the system to start over. Click in add process to add process in the system. Click on the process to move it forward. Red light will turn on below the respective column. If the laws permit you will successfully complete the task. It wont move if it is not feasible.



Instructions for Page Replacement Project

Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

1. Page Replacement Algorithms

First In First Out (FIFO) :

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced, the page in the front of the queue is selected for removal.

Optimal Page Replacement :

An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.

Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Least Recently Used (LRU) :

An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.

Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Last In First Out (LIFO) :

This Page Replacement algorithm stands for "Last In First Out".This algorithm works in a similar way to the LIFO principle.

In this, the newest page is replaced which is arrived at last in the primary memory

This algorithm makes use of the stack for monitoring all the pages.

2. Belady's Anomaly

Bélády's anomaly is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern.

This phenomenon is commonly experienced in following page replacement algorithms:

1.First in first out (FIFO)

2.Second chance algorithm

3.Random page replacement algorithm

Reason of Belady's Anomaly –

The other two commonly used page replacement algorithms are Optimal and LRU, but Belady's Anomaly can never occur in these algorithms for any reference string as they belong to a class of stack based page replacement algorithms.

A stack based algorithm is one for which it can be shown that the set of pages in memory for N frames is always a subset of the set of pages that would be in memory with N + 1 frames. For LRU replacement, the set of pages in memory would be the n most recently referenced pages. If the number of frames increases then these n pages will still be the most recently referenced and so, will still be in the memory. While in FIFO, if a page named b came into physical memory before a page – a then priority of replacement of b is greater than that of a, but this is not independent of the number of page frames and hence, FIFO does not follow a stack page replacement policy and therefore suffers Belady's Anomaly.

3. Page Replacement page

This page contains the theory of page replacement algorithms and a few text fields where you can enter the required reference string and number of frames.

The Result Page :

The result page contains the following components :

Title and Info Button :

The name of the currently working algorithm is shown on the top. Along with it, an info button is there which shows the theory of the algorithm.

Fault Ratio vs Frames Graph :

This graph calculates the fault ratio for increasing number of frames. This graph is used to analyze Belady's Anomaly.

Page Hit and Page Fault Indicators :

The area below the graph shows the Page Hit number, Page Fault number and their ratios.

Compare Button :

The compare button is used to compare all the algorithms at the same time.

Table Button :

This button is used to check the reference string table which shows where the page hit and page fault is occuring.

Algorithm Switching button :

This button is used to change the working algorithm for page replacement. By changing the algorithm, it also changes the Title and Graph on the page.

Comparison Page :

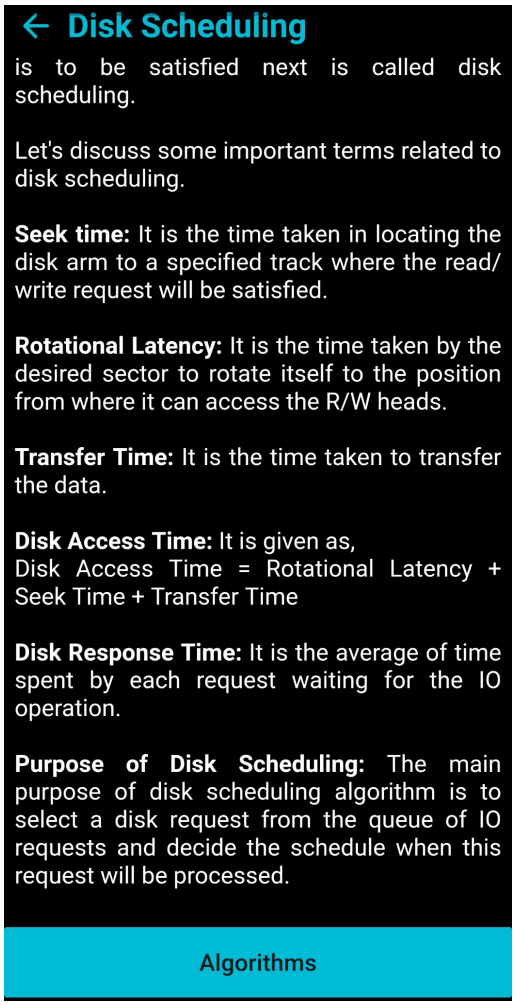
This page shows the page hit, page fault and their ratios generated by all the algorithms. It also contains a button which when clicked, shows the comparison result graphically.

Reference Table :

This page contains a table which shows the working of the algorithm. It shows which string is added when and if it is a page hit or page fault. The page hits are shown in blue color and page faults are shown in red color.

Instructions for Disk Scheduling Project

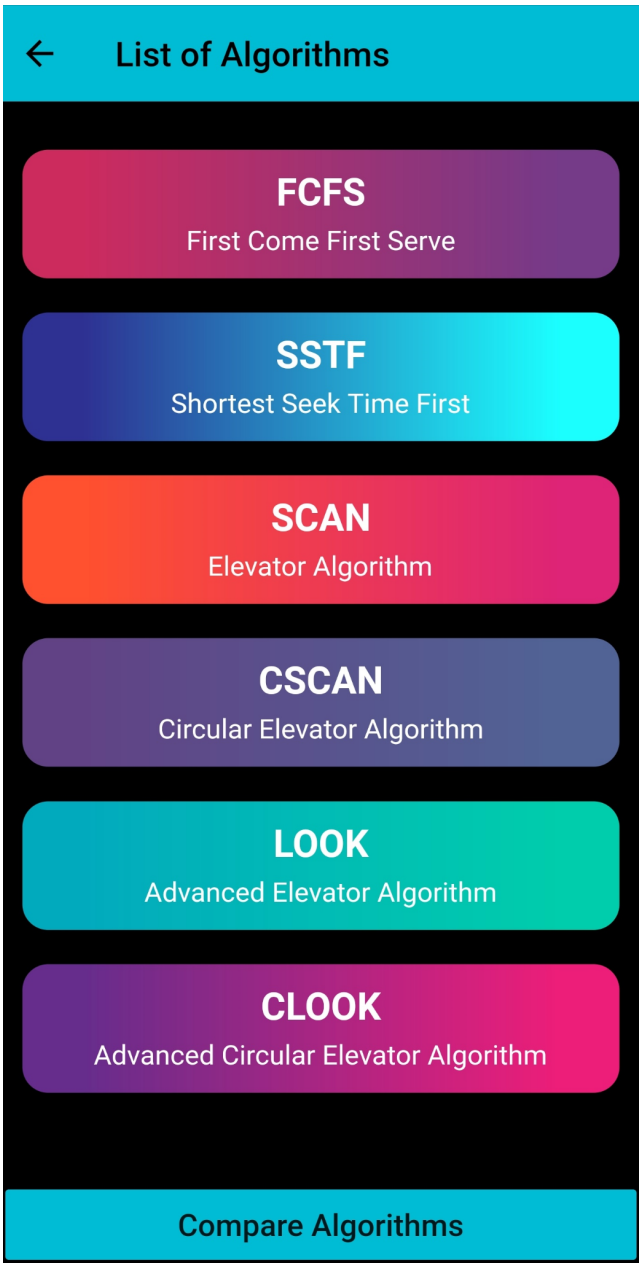
1. Introduction to Disk Scheduling Algorithms



Disk scheduling is a technique used by the operating system to schedule multiple requests for accessing the disk. On the introduction page, the basics of disk scheduling is discussed and definitions of some important terms have been mentioned, The algorithms button at the bottom of the screen takes you to the next page.

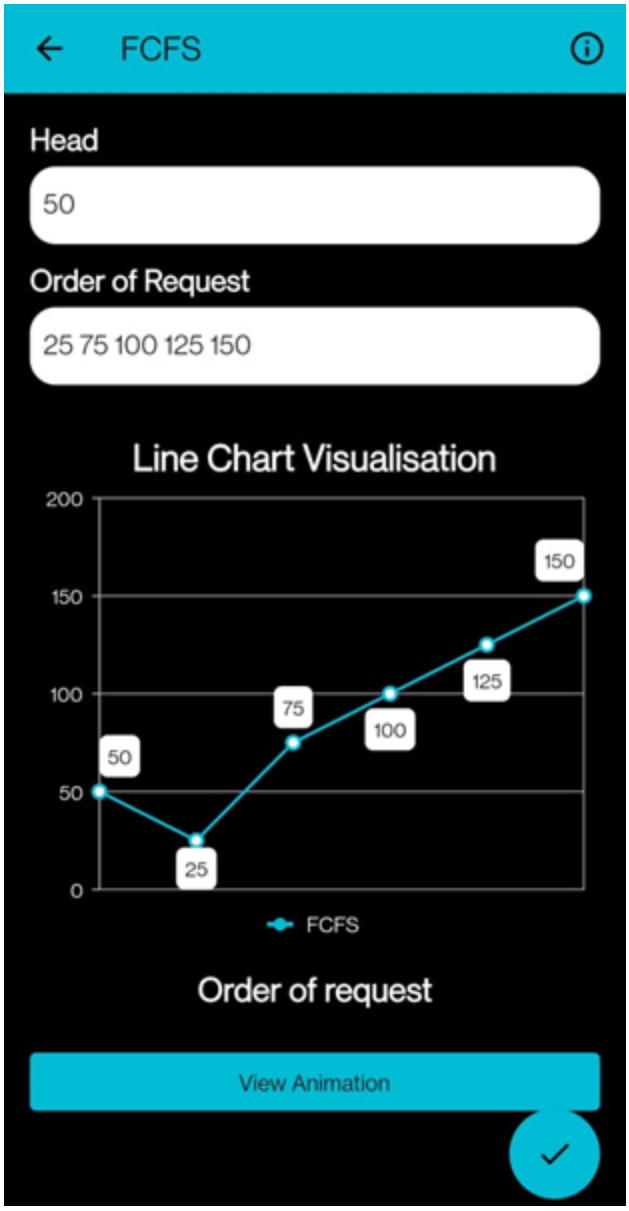
2. List of algorithms

The second page consists of the list of all the algorithms.



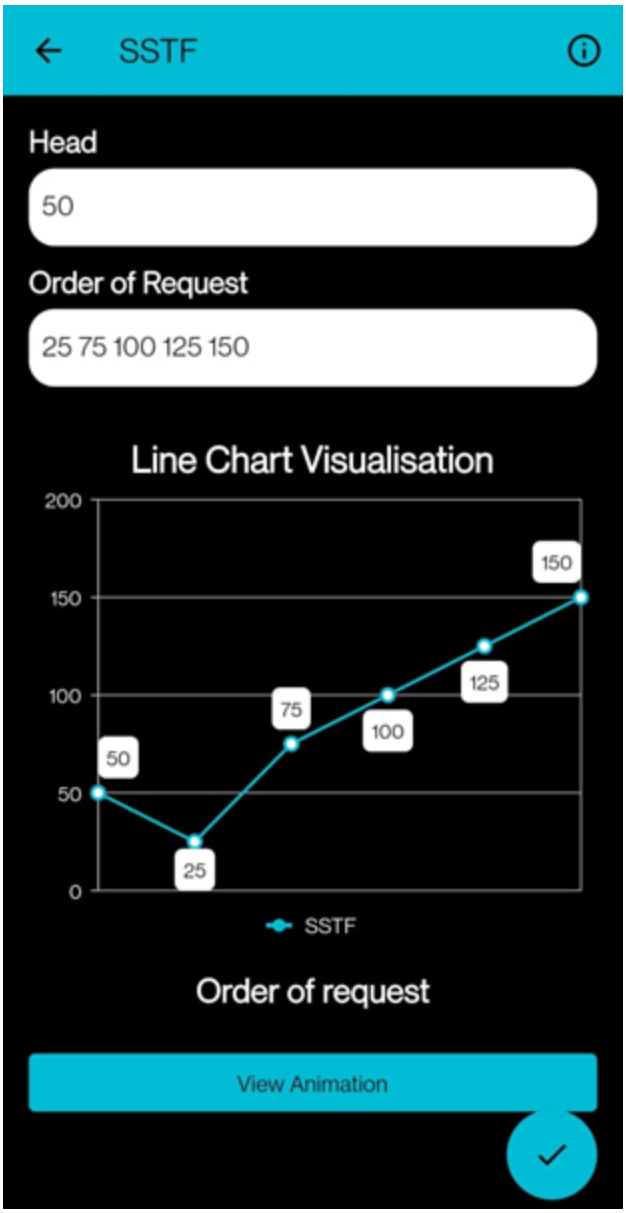
3. FCFS

First Come First Serve (**FCFS**) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which request the CPU first get the CPU allocation first. This is managed with a FIFO queue. The full form of FCFS is First Come First Serve. As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue and, when the CPU becomes free, it should be assigned to the process at the beginning of the queue.



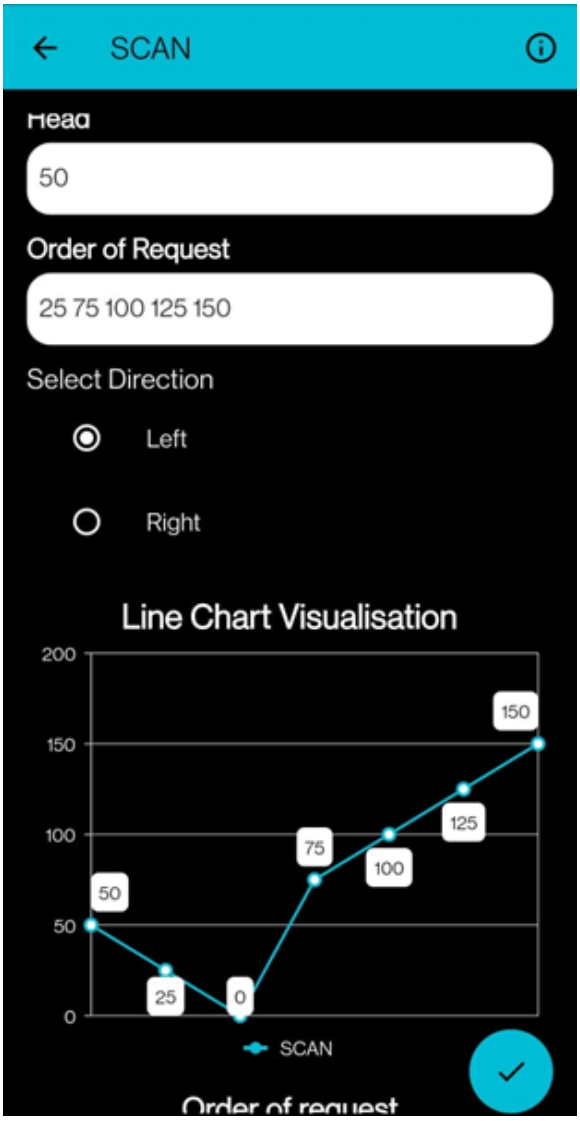
4. SSTF

Shortest seek time first (**SSTF**) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS. It allows the head to move to the closest track in the service queue.



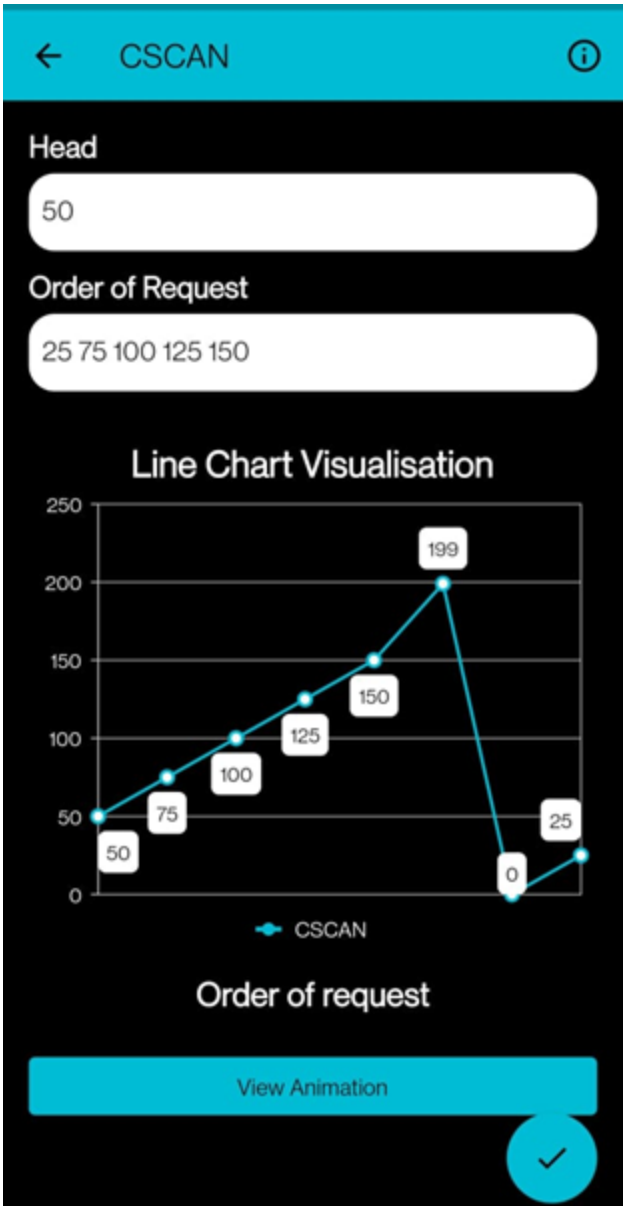
5. SCAN

It is also known as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till end, satisfying all the requests coming in its path and then it turns back and moves in the reverse direction satisfying requests coming in path.



6. CSCAN

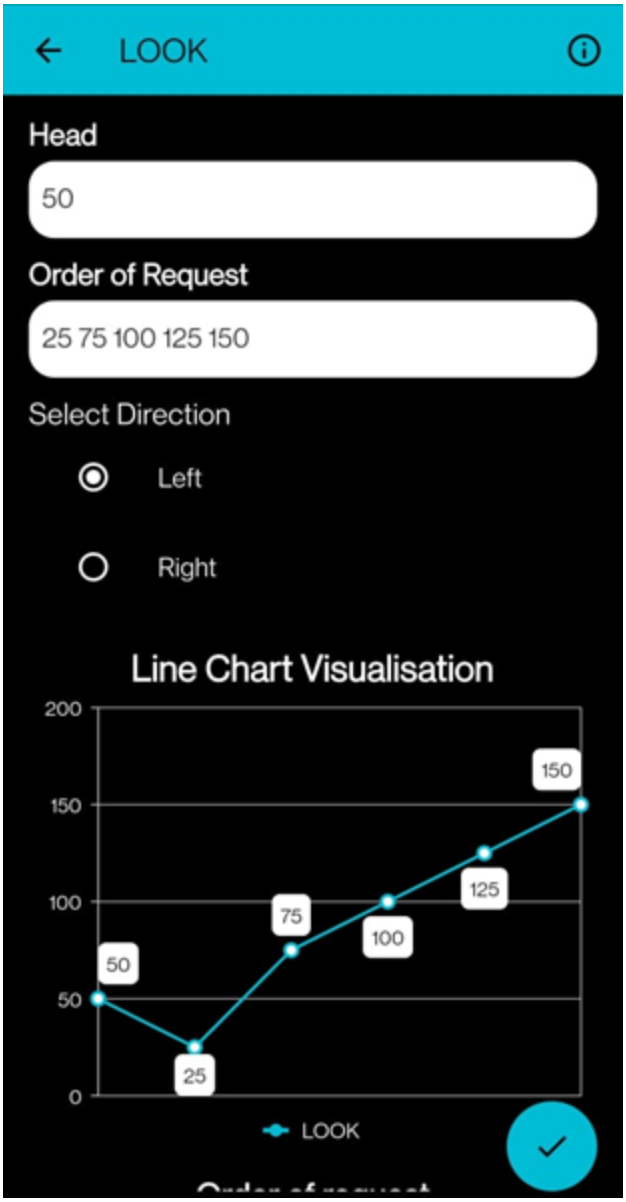
In CSAN algorithm, the arm of the disk moves in particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.



7. LOOK

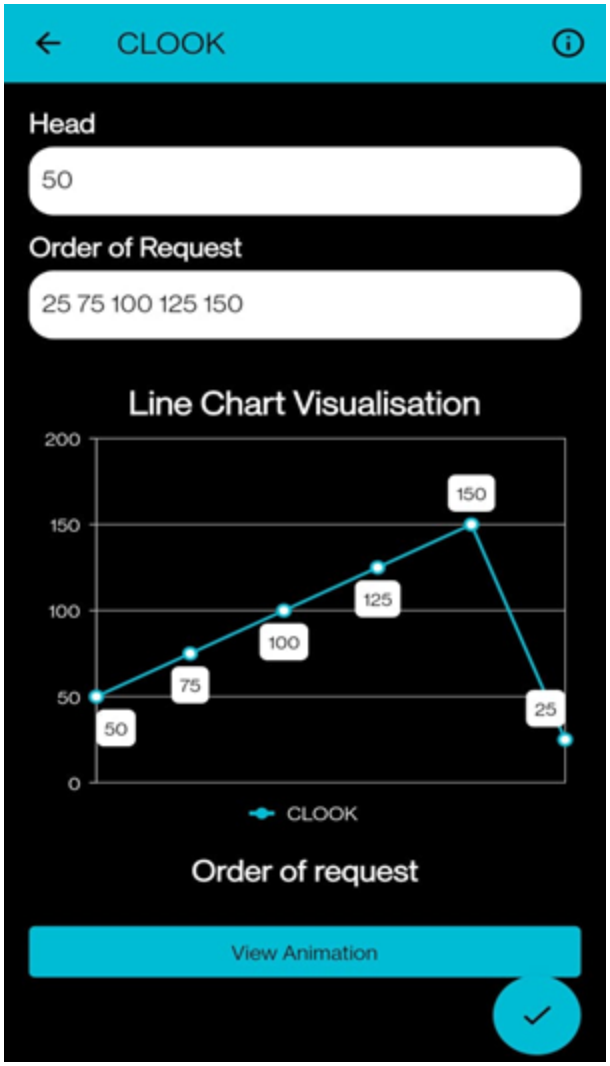
The LOOK algorithm services request similarly as SCAN algorithm meanwhile it also “looks” ahead as if there are more tracks that are needed to be serviced in the same direction. If there are no pending requests in the moving direction the head reverses the direction and start servicing requests in the opposite direction.

The main reason behind the better performance of LOOK algorithm in comparison to SCAN is because in this algorithm the head is not allowed to move till the end of the disk.



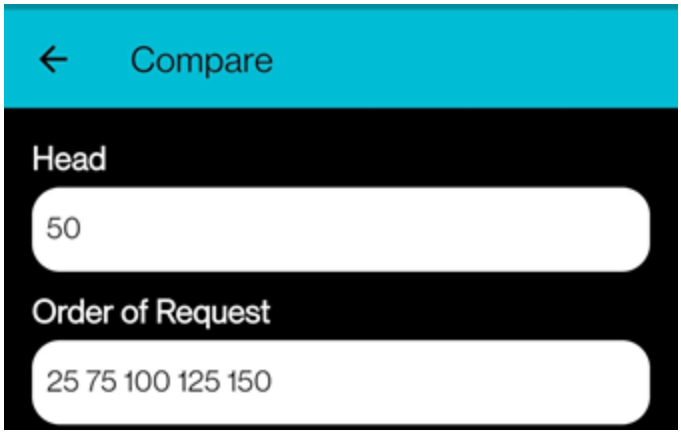
8. CLOOK

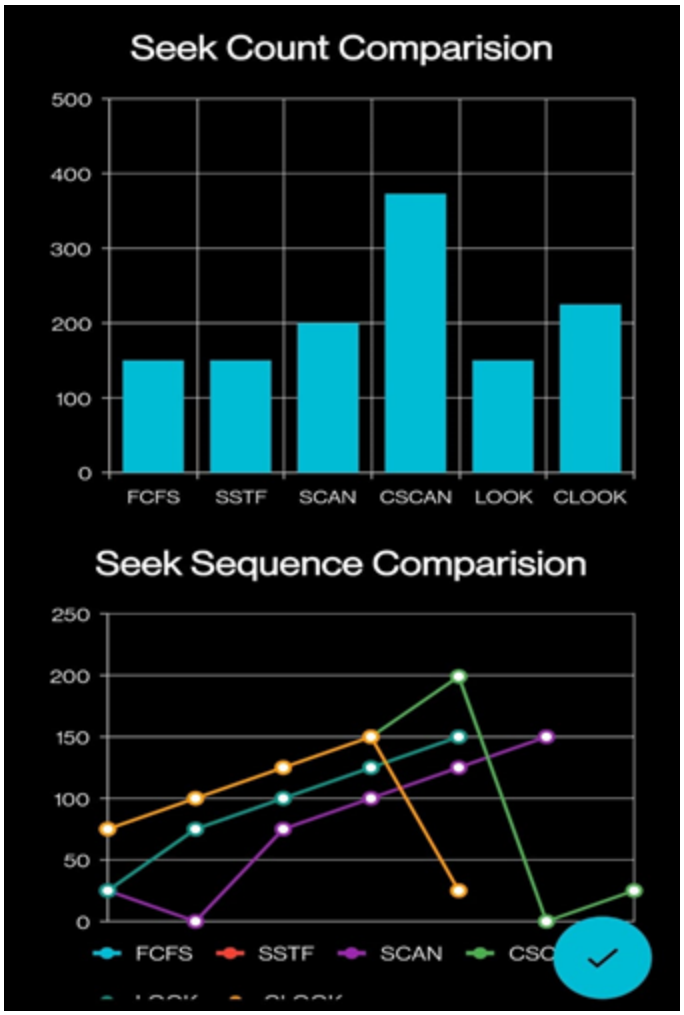
C-LOOK is an enhanced version of both SCAN as well as LOOK disk scheduling algorithms. This algorithm also uses the idea of wrapping the tracks as a circular cylinder as C-SCAN algorithm but the seek time is better than the C-SCAN algorithm.



9. Comparing Algorithms

On this page, you can compare the seek count and seek sequence of all the algorithms against a single input.



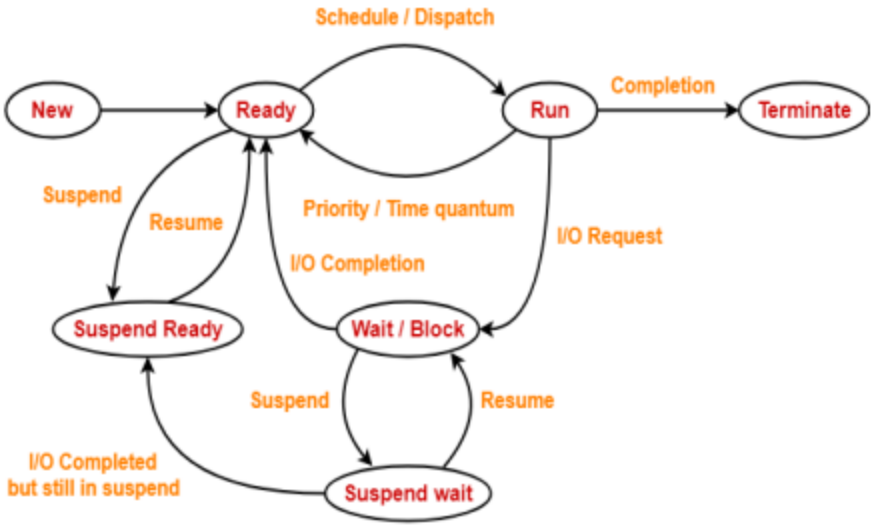


Instructions for Cpu Scheduling Project

- Introduction to CPU Scheduling:

  CPU Scheduling is the process of determining which process will own CPU for execution while another process is on hold. The main task of CPU Scheduling is to make sure that whenever the CPU remains idle, the operating system at least selects one of the process available in the ready queue for execution. The selection process in the memory that are ready for execution.

- States through which process have to pass:



Process State Diagram

- The above diagram is the process state diagram.
- A process necessarily goes through minimum 4 states- New State, Ready state, run state, Terminate state.
- If a process requires I/O time then it should must pass through 5 states of the above diagram.
- This all states are present in the memory. some of them are present in the main memory where as some are in the secondary memory.
- In the main memory: Ready state, run state, Wait state.

- Various times Related to Process Scheduling:

  - ➢ Arrival Time: Time at which process enters the ready queue.
  - ➢ Burst Time / Execution Time / Running Time: Amount of time required by a process for executing CPU.
  - ➢ Completion Time / Exit Time: Time at which a process completes its execution and exit from the system.
  - ➢ Turn Around Time: Total amount of the spent by the process in the system. (I.e. TAT=Completion time – Arrival time or TAT= BT + WT)
  - ➢ Waiting Time: the amount of time spent by the process waiting in the ready queue for getting the CPU. (i.e. WT= TAT – BT)
  - ➢ Response Time: the amount of time after which a process gets the CPU for the first time after entering the ready queue. (i.e. RT = Time at which process first gets the CPU – Arrival Time)

- Nature of process:

  Preemptive Nature: A processor can be preempted to execute the different processes in the middle of any current process execution.

  Non-Preemptive Nature: Once the processor starts its execution, it must finish it before executing the other. It can't be paused in the middle.

- List of Scheduling Algorithms implemented:

  1. First Come First Serve (FCFS)
  2. Shortest Job First (SJF)
  3. Shortest Remaining Time First (SRTF)
  4. Longest Job First (LJF)
  5. Longest Reaming Time First (LRTF)
  6. Round Robin (RR)
  7. Priority
  8. Priority with preemptive

- CPU Scheduling Algorithms:

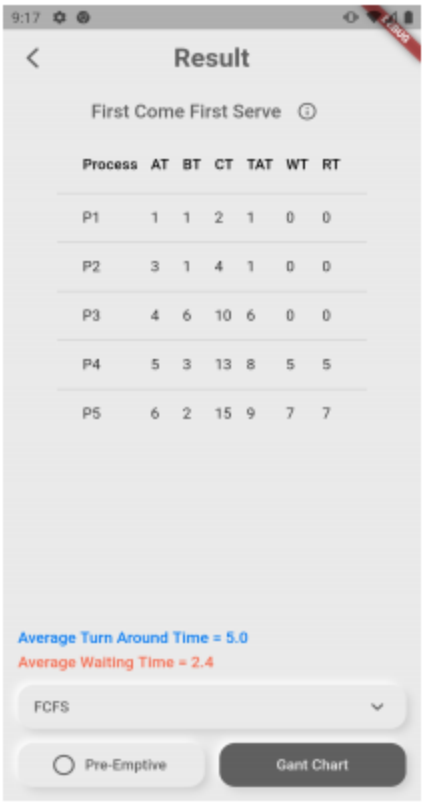
1. First Come First Serve (FCFS):
   - ➢ The process which arrives first in the ready queue is firstly assigned to the CPU.
   - ➢ When more than one process has the similar arrival time then the process having small process Id will assign to the CPU first.
   - ➢ It is always non-preemptive in nature

Advantages:

- ➢ It can be easily implemented using queue data structure.
- ➢ Does not lead to starvation.

Disadvantages:

- ➢ It does not consider the priority or burst time of the processes.
- ➢ It suffers from convoy effect. Convoy effect occurs when the process having higher burst time arrives before the process having less burst time.so the smaller process has to wait for the longer time.



2. Shortest Job First (SJF):
   - ➢ The process which is having the smallest burst time is firstly assigned to the CPU.
   - ➢ When more than one process has the similar burst time then the process having small process Id will assign to the CPU first.
   - ➢ It can be used in both preemptive and non-preemptive mode.

   Advantages:
   - ➢ It is having minimum average waiting and turnaround time.
   - ➢ Gives the maximum throughput.

   Disadvantages:
   - ➢ It suffers from starvation.
   - ➢ It is not implementable because the exact Burst time for a process can't be known in advance.

3. Shortest Reaming Time First (SRTF):

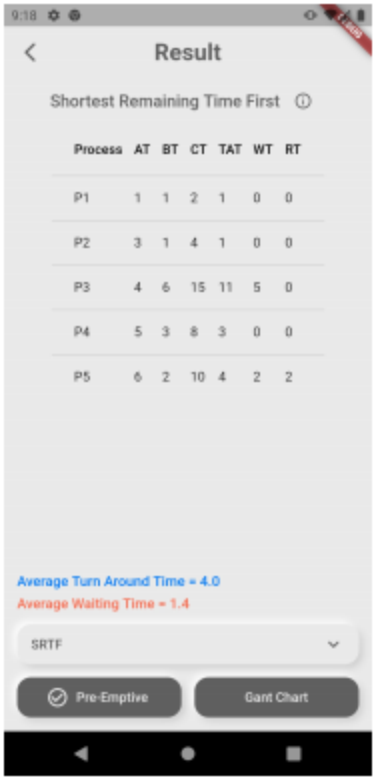
- Preemptive mode of Shortest Job First is called as Shortest Remaining Time First (SRTF)
- The process which is having the smallest burst time is firstly assigned to the CPU.
- When more than one process has the similar burst time then the process having small process Id will assign to the CPU first.

Advantages:

- It is Optimal and guarantees the minimum average waiting.
- Time it provides a standard fot other algoritms sins no other algorithms performs better than this

Disadvantages:

- Priorities cannot be set for the processes.
- Processes with larger burst time have poor response time.
- It is not implementable because the exact Burst time for a process can't be known in advance.



4. Longest Job First (LJF):
   - ➢ The process which is having the longest burst time is firstly assigned to the CPU.
   - ➢ When more than one process has the similar burst time then the process having small process Id will assign to the CPU first.
   - ➢ It can be used in both preemptive and non-preemptive mode.

Advantages:

- ➢ No process can complete until the longest job also reaches its completion.
- ➢ All the processes approximately finish at the same time.

Disadvantages:

- ➢ It suffers from convoy effect.
- ➢ It gives very high average waiting time and average turnaround time for a given set of process.
- ➢ It reduces the processing speed and thus reduces the efficiency and utilization of the system.
- ➢ It may happen that the shorter process may never get executed and the system keeps on executing the longer process.

5. Longest Reaming Time First (LRTF):

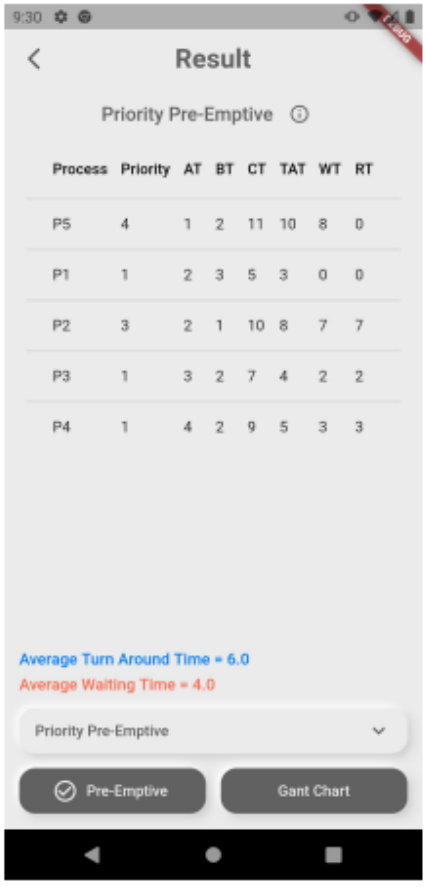
- Preemptive mode of Longest Job First is called as Longest Remaining Time First (LRTF)
- The process which is having the Largest burst time is firstly assigned to the CPU.
- When more than one process has the similar burst time then the process having small process Id will assign to the CPU first.

Advantages:

- No process can complete until the longest job also reaches its completion.
- All the processes approximately finish at the same time.

Disadvantages:

- Waiting time is very high.
- Processes with smaller burst time may starve for CPU.



6. Round Robin (RR):

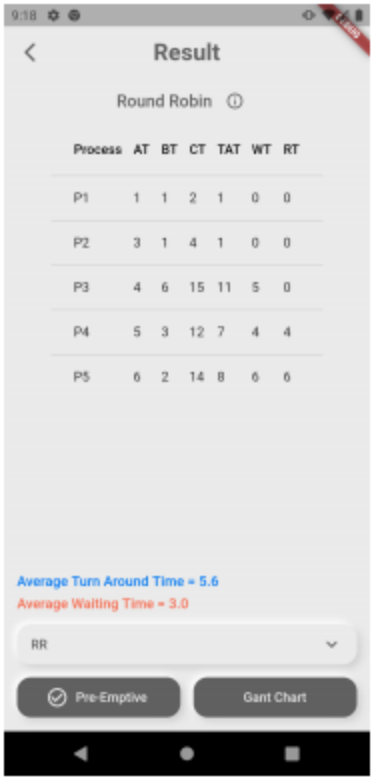
- CPU is assigned to the process on the basis of FCFS for a fixed amount of time. This fixed amount of time is called time quantum or time slice.
- After the time quantum expires, the running process is preempted and sent to the ready queue.
- Then, the processor is assigned to the next arrived process.
- It is always preemptive in nature.

Advantages:

- It gives the best performance in terms of average response time.
- It is best suited for time sharing system, client server architecture and interactive system.

Disadvantages:

- It leads to starvation for processes with larger burst time as they have to repeat the cycle many times.
- Its performance heavily depends on time quantum.
- Priorities cannot be set for the processes.



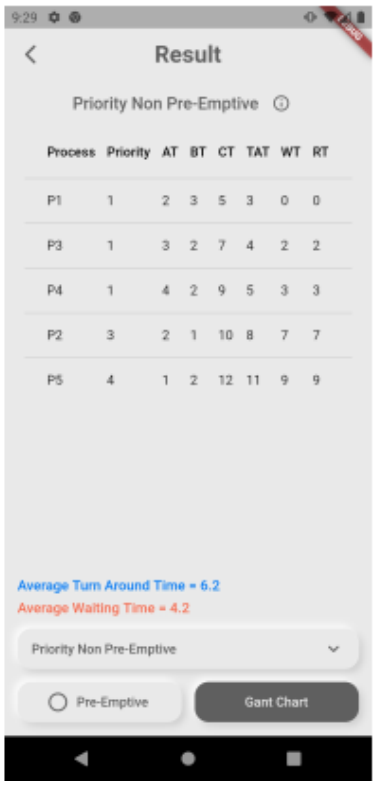
7. Priority:
   - ➢ CPU is assigned to the process having the highest priority.
   - ➢ When two process having same priority then the process having the smaller process id is assigned to CPU.
   - ➢ It can be used in both preemptive and non-preemptive mode.

Advantages:

- ➢ Processes are executed on the basis of priority so high priority does not need to wait for long which saves time
- ➢ This method provides a good mechanism where the relative important of each process may be precisely defined.

Disadvantages:

- ➢ This scheduling algorithm may leave some low priority processes waiting indefinitely.
- ➢ If a new higher priority process keeps on coming in the ready queue, then the process which is in the waiting state may need to wait for a long duration of time.
- ➢ If the system eventually crashes, all low priority processes get lost.



8. Priority with preemptive:

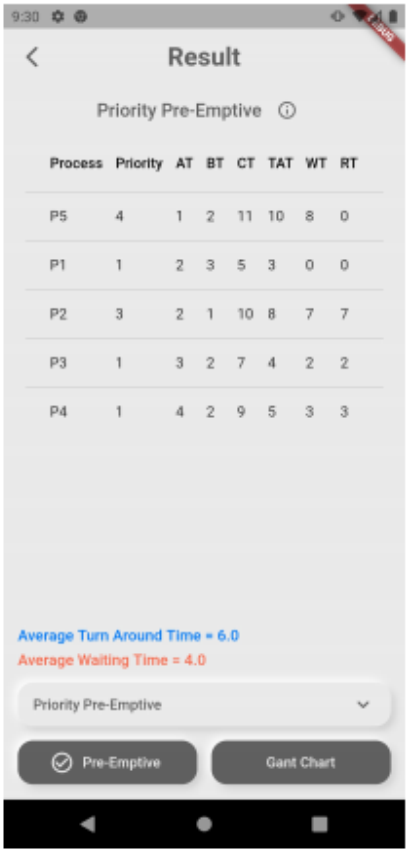
➢ CPU is assigned to the process having the highest priority. ➢ When two process having same priority then the process having the smaller process id is assigned to CPU.
➢ It is non-preemptive version of the priority Scheduling.

Advantages:

➢ It considers the priority of the processes and allows the important processes to run first.
➢ Priority scheduling in preemptive mode is best suited for real time operating system.

Disadvantages:

➢ Processes with lesser priority may starve for CPU. ➢ There is no idea of response time and waiting time.



How to use it?

We have uploaded a working video of the project on Youtube which will give you all the information on how to use the Concurrency and Deadlock algorithms and how to take benefit of all the features provided.

Links of the video:

Page Replacement Algorithm:

https://youtu.be/xw6zmtpAFDM

Deadlock And Concurrency Algorithm:

https://youtu.be/CvEuzlaQ_wg

Disk Scheduling Algorithm:

https://youtu.be/yuRvvWzmus4