Web-Based Simulator for Operating Systems

K. Prajwal, P. Navaneeth, K. Tharun, Trupti Chandak, and M. Anand Kumar

1 Introduction

In the operating system field, we come across various concepts and solution to problems that have unique ways of being solved. Some of the various such concepts are CPU scheduling, memory management, disk scheduling, file system management, semaphores and some more. All the concepts mentioned above are fundamental when considering the operating system, as these algorithms that are suggested increase the operating system's speed significantly. Problems like deadlock and loss of information have been overcome by implementing these operating system's algorithms. Some of the concepts like Dining Philosophers and Consumer Producer are put forth as problems whose solutions have significantly affected how computers work today. Therefore, in light of this, we have decided to implement these concepts as a simulation.

To make the simulation more interactive and creative, we have combined the knowledge of web technologies and operating systems to come up with the following

K. Prajwal (✉) · P. Navaneeth · K. Tharun · T. Chandak · M. A. Kumar
Department of Information Technology, National Institute of Technology Karnataka, Surathkal 575025, India
e-mail: kprajwal.191it222@nitk.edu.in

P. Navaneeth
e-mail: navaneethp.191it132@nitk.edu.in

K. Tharun
e-mail: tharunk.191it255@nitk.edu.in

T. Chandak
e-mail: al.trupti@nitk.edu.in

M. A. Kumar
e-mail: manandkumar@nitk.edu.in

B. Agarwal et al. (eds.), *Proceedings of International Conference on Intelligent Cyber-Physical Systems*, Algorithms for Intelligent Systems,
https://doi.org/10.1007/978-981-16-7136-4_10

web application. For our project, we created a web application that simulates various CPU and OS concepts such as the ones mentioned above using graphical ways.

We have deployed our app at https://os-simulator.herokuapp.com/ which can be accessed by anyone, anywhere with internet access. Building the simulator was the main objective we were trying to achieve, a tool which can be used by anyone who wants to acquire knowledge on Operating System concepts, but reading text books or reference books is hard for them. This graphical way gives a whole new way of learning to either a student or a professor who wishes to use our web application.

Keeping the objective of building an educational tool in mind, we first decided to start with some basic yet essential algorithms to simulate with further research on advanced topics such as Socket Programming, Threading and System Calls.

2 Literature Survey

Operating System simulators simulate various Operating System concepts, and they vary from basic to complex real-time simulators. These simulators are used as educational tools at many top-tier institutions to make learning more interactive than just conventional reading from textbooks or reference books. Below, we have listed the popular and trending simulators available on the Internet with a short description. We have also tabulated this data, making it easier to see the differences and compare the available simulators.

SchedulerSim [1]: Here, a simulator of operating system was developed using various scheduling concepts. There was no simulation of the CPU processes. One of the main concepts here that was implemented in this project was the scheduling concept [1].

Sim. + assembler [2]: This project has a well-implemented CPU simulation. There is no OS simulation. Some of the important concepts that were implemented here were IO processing and interrupt handling [2].

MKit [3]: This project consisted of a very well-maintained OS simulator. No CPU simulation was implemented. Some of the concepts implemented were data path simulation as well as control unit management systems [3].

SOsim [4]: A simulator for an operating system was deployed in this project. There was no simulation on any CPU-based concepts. Some of the concepts implemented here are process management and memory management systems [4].

PsimJ sim [5]: It has an OS simulator set in place with various isolated OS component simulations that are present [5].

We have looked into each of these applications and felt like they could have included a few other concepts such as Memory Allocation Techniques, Page Replacement, Disk Scheduling which are also considered important in the field of Operating Systems. Hence, we decided to provide a graphical and interactive way of implementing the concepts including the other topics which were touched upon by the previous works such as CPU Scheduling algorithms. Using the concepts of operating system, to build an educational web app that simulates these concepts in an efficient and graphical way.

3 Proposed Work

We have implemented a simulation for six Operating System concepts.

1. Process Scheduling
2. Semaphores
3. Memory Allocation
4. File Systems
5. Disk Scheduling
6. Page Replacement.

We have built a web application using the following technologies which are available.

- Python3—used for scripting the algorithms
- JavaScript—used for scripting the algorithms
- Django—used as back-end for our web application
- HTML/CSS—used as frontend for our web application.

We have used python as to implement all the algorithms and concepts mentioned above and we used HTML, CSS and Javascript to implement, style and simulate them in a web environment.

We used Django framework to inter-link all these files into a web server which can be hosted on a server to make it available.

3.1 Process Scheduling

Pseudo Code for some of the scheduling concepts implemented have been written below. We have implemented the following algorithms.

- First Come First Serve (FCFS)
- Shortest Job First (SJA)
- Round Robin (RR)
- Priority Scheduling (PS)
- Shortest Run Time First (SRTF)
- Multi-level Queue Scheduling

$$\text{Turnaround Time} = \text{Completion Time} - \text{Arrival Time} \tag{1}$$

First Come First Serve: Here, we implement the processes in the increasing order of their arrival time. First come, first serve is a non-preemptive algorithm and therefore will only take the next job after it completes the implementation of the process that it is already implementing.

Shortest Job First: Here, the processes are implemented in ascending order of their burst time while keeping in mind their arrival time. Shortest Job First algorithm is a non-preemptive and therefore cannot implement any other process when a certain process is being implemented.

Round Robin: In the round robin algorithm, we take a quantum and continually keep doing different processes until all the processes are finished.

Priority Scheduling: In this algorithm, we implement the processes in the increasing order of their priorities keeping in mind the arrival time of each process. Priority scheduling can be made both preemptive and non-preemptive. Here, the pseudo code discussed below shows how to implement a non-preemptive priority scheduling algorithm.

Shortest Run Time First: Shortest run time first algorithm is inspired from the shortest job first algorithm that we have implemented above. But shortest run time first algorithm is a preemptive algorithm, implying that at any point of time if there is a process that has lesser burst time than that process that is being implemented, then, that process is implemented. Shortest run time first algorithm takes into consideration the burst time of all the processes along with their arrival time.

Multi-level Queue Scheduling: The modern CPU has processes in a queue known as Ready Queue. This ready queue is further partitioned or sub-divided into separate queues, namely, and not limited to:

– Foreground
– Background

Each queue has its own scheduling algorithm, you can assign any type of algorithm, hence the name, Multi-level Queue Scheduling.

Scheduling must be done between the queues

– The foreground queue will be run first and given higher priority over the background queue; the background queue will be run once the foreground has been exhausted.
– This scheduling method also requires a Time Slice or Time Quanta similar to that of Round Robin, which is the time after which the next process will be attended to by the CPU during process scheduling.

3.2 Semaphores

Semaphores are functions that can be used to solve various problems that are used to solve various critical section problems using wait and *signal* operations. Some of the concepts that use semaphores, that we implemented were producer consumer problem and dining philosophers.

Producer Consumer Problem: The following problem is a synchronization problem. There is a fixed size buffer that the producer can put his products into. The consumer takes the products from the buffer and consumes them. The problem occurs either when the buffer is empty and the consumer tries to take the product or when the buffer is full and the consumer tries to fill the buffer. This problem can be solved using semaphores.

Dining Philosopher: Dining Philosopher is a problem that is often used to display the synchronization problem that occurs in operating systems. It also shows us a technique for solving them. In our simulation, we have five philosophers that decide to dine together in round table. Each philosopher can either eat, think or wait. To eat, the philosopher needs two chopsticks, meaning that both the chopsticks beside him must be free. Since there are only five chopsticks on the table, two beside each philosopher, we need to time which philosopher eats when perfectly, so as to avoid deadlock or a situation where a philosopher starves forever. This too can be solved using semaphores.

```
Algorithm for Dining Philosopher (Output)
1.  The array Chopsticks contain all the relation between
    the chopstick that each philosopher uses.
2.  The number of philosophers is given by the number
    n.
3.  fori in Philosophersdo
4.  while true do
5.  Think
6.  Callpickup (chopsticks[i], chopsticks[(i + 1)%n])
7.  Eat
8.  Callputdown (chopsticks[i], chopsticks [(i + 1)%n])
9.  end while
10. end for
```

3.3 Disk Scheduling

Disk Scheduling concept is a process that is implemented by the operating system to schedule and process the input output requests that it receives. It is similar to process scheduling. There are many requests that come at different times and have different execution times. Some of the algorithms that handle these kinds of requests are the following:

First Come First Serve: Here, the first request that arrives into the operating system is executed first.

Shortest Seek Time First: Here, we take into consideration all the request's seek time and their arrival time. The operating system then implements the requests in the increasing order of their seek time while taking into consideration their arrival time.

Scan Algorithm: In this algorithm, there is a head that traverses through the requests from the start to the end and executes them. Once the head reaches the end of the disk, the direction of the head is reversed and again the processes are implemented when the head reaches that request.

Look Algorithm: The look algorithm is similar to the scan algorithm, but here instead of going to the end of the disk the head goes only to the last request to be serviced in front of the head and then reverses its direction from there itself.

Circular Scan and Circular Look Algorithm: The pseudo code and the implementation for the circular scan and the circular look algorithm is quite similar to the scan and the look algorithm. The difference being that in the Circular Scan and Circular Look Algorithms, we jump to the beginning of the disk instead of reversing the direction.

3.4 Memory Allocation

These are the algorithms that helps the operating system in allocating spaces in the memory to programs that the operating system comes across. Memory allocation algorithms are spilt into two cases. One, when the partition size is fixed and when the partition size is varying.

The Fixed Sized Partition: Here, the partition is divided into equal or different sized partitions whose size is fixed and constant. Some of the algorithms that we implement to allocate memory to the blocks of programs are:

– *FirstFit*: The first fit algorithm finds the first block that fits the process/program and assigns the block to it.
– *Best Fit*: The best fit algorithm finds the smallest block that fits the program/process and assigns the block to it.
– *Worst Fit*: The worst fit algorithm finds the largest block that fits the program/process and assigns the block to it.

3.5 Page Replacement

When a new block of memory arrives and the old block of memory has to be replaced with the new one, the operating system uses some of the below mentioned algorithms:

– *First In First Out*: The first process/page that arrives first is the first one to be replaced. The operating system keeps a queue of all the process that arrives and the first process/page to arrive is at the front of the queue.
– *Optimum*: The page/program that is most likely to be not used in the future for a long time is replaced with the page that arrives. These page replacement algorithms are very hard to implement as the OS has no way of knowing whether a particular page/process may be used in the future.
– *Least Recently Used*: The page/program that is least recently used will be replaced with that of the incoming program or page.

3.6 File Structure

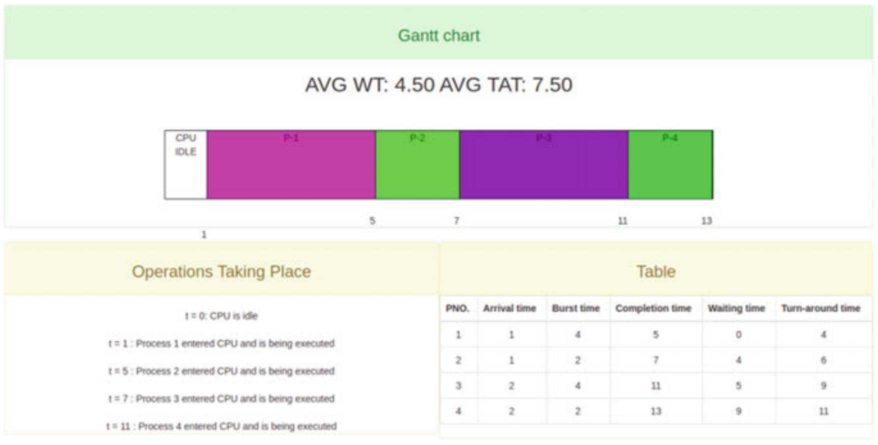
File structures are different ways of structuring the files in secondary memory that helps reduce the access time.

We have implemented a terminal style output of Linux commands which create, delete the files and folders. Hence, the user can also learn how files and folders can be created and deleted on a Linux terminal.

– *Single Level*: Here, there is only a big list of all the files that are present in the directory. The user can have only one directory and all the files must be present inside that. The implementation of this is very simple and the deletion and insertion is very fast and easy, but we cannot have two files with the same name and the protection of files are not guaranteed for multiple users.
– *Two Level*: In this File Structure algorithm, the user has two levels of files; the first level, being the directory and the second level being the files. Files cannot be created without being in a directory. Multiple folders having multiple files can be created.
– *Tree Structure*: Tree Structure is how modern operating systems implement file structure. A directory can have several files and sub-directories. These sub-directories can then further branch to contain additional files and sub-directories.

4 Result and Analysis

Given the process ID, arrival time and burst time, the process scheduling algorithms compute the Gantt chart, waiting time and the turnaround time. Additionally, the steps taken to arrive at the result and the time stamp in the simulation are presented at the bottom left (Fig. 1).



AVG WT: 4.50 AVG TAT: 7.50

Operations Taking Place

t = 0: CPU is idle

t = 1 : Process 1 entered CPU and is being executed

t = 5 : Process 2 entered CPU and is being executed

t = 7 : Process 3 entered CPU and is being executed

t = 11 : Process 4 entered CPU and is being executed

Table

PNO.	Arrival time	Burst time	Completion time	Waiting time	Turn-around time
1	1	4	5	0	4
2	1	2	7	4	6
3	2	4	11	5	9
4	2	2	13	9	11

Fig. 1 Graphical representation of the data, along with the Gantt chart and the tabular format of experimental data

In the Reader Writer simulation. We need to select what the seven processes want to do, i.e., read, write or stay idle. Then on submitting using the “Submit Query” button, the result shows that if the process was permitted to continue, it is shown with blue indicating the process is allowed to read and with red indicating the process is allowed to write. We see that parallel reads are allowed but only one process can write at a time.

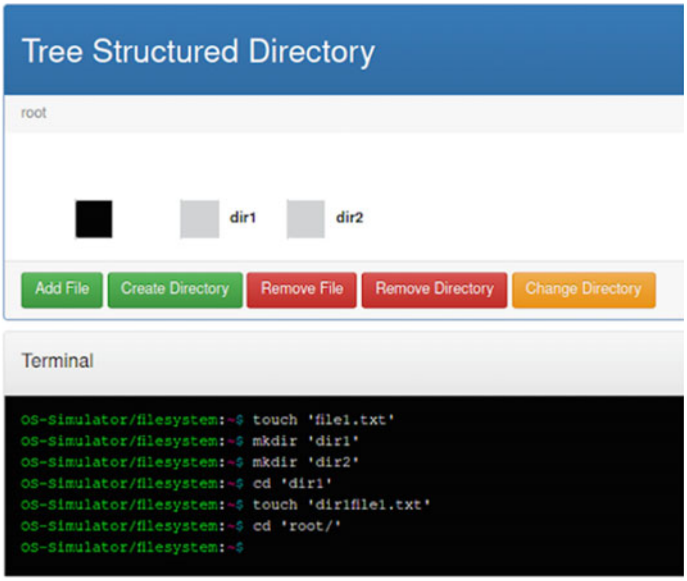
Given the cache size and a space separated list of requests, the cache hits represented by green tick and the cache miss represented by red cross can be visualized. The “Proceed” button processed the next request and displays if it is cache hit or miss. The algorithms FIFO, LRU and Optimum can be selected by the tabs provided.

Fig. 2 shows is the simulation of the file tree implemented step by step. The files and directories in the selected directory are shown at the top. The path to the current directory is shown above that. For every action, the equivalent terminal command to act is shown.

Fig. 3 shows the seek graph for the FCFS disk scheduling algorithm. The seek requests are given as a space separated list. Using the current position and the number of cylinders parameter, the order in which the seek happens is shown in the graph.

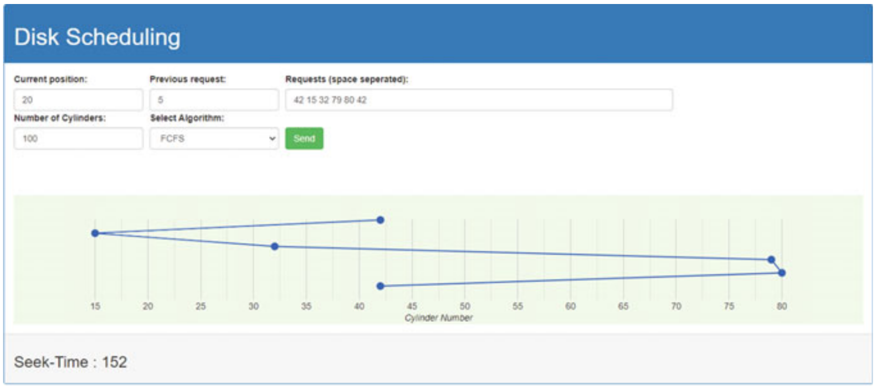
5 Conclusion and Future Scope

We successfully implemented all of the objectives, including concepts such as File Systems and Disk Scheduling, which have been entirely made into a graphical and interactive simulation, making it an immersive experience for the end-user, which helps understand how the concepts work.



```
OS-Simulator/filesystem:~$ touch 'file1.txt'
OS-Simulator/filesystem:~$ mkdir 'dir1'
OS-Simulator/filesystem:~$ mkdir 'dir2'
OS-Simulator/filesystem:~$ cd 'dir1'
OS-Simulator/filesystem:~$ touch 'dir1file1.txt'
OS-Simulator/filesystem:~$ cd 'root/'
OS-Simulator/filesystem:~$
```

Fig. 2 Output of one of the algorithms used in file systems



Current position: 20
Previous request: 5
Requests (space seperated): 42 15 32 79 80 42
Number of Cylinders: 100
Select Algorithm: FCFS

Seek-Time : 152

Fig. 3 Seek graph for the FCFS disk scheduling algorithm using experimental data

We have also deployed our app on Heroku web servers for easy access; it is deployed at https://os-simulator.herokuapp.com/.

One can open this using a device with internet access and a browser app. Just click on the link or copy-paste in the address bar to access the web page. There are no other requirements, making it very light and standalone, which is not the case with other simulators available, which requires various libraries and to run it locally on your machine.

The complete source code to our implementation can be found on the GitHub of one of the authors, using which we collaborate to put together this Operating System Simulator.

The link to the GitHub repository is https://github.com/navaneethp123/os_simulator/. Our future works are not limited to including advanced and latest Operating Systems such as Socket Programming and Parallel Programming\.

References

1. Chan, T.W.: A software tool in java for teaching CPU scheduling. JCSC **19** (2004)
2. Than, S.: Use of a simulator and an assembler in teaching input-output processing and interrupt handling. JCSC **22** (2007)
3. Nishita, S.: MKit simulator for introduction of computer architecture. In: 31st International Symposium on Computer Architecture, June 19, 2004. Munich, Germany
4. Maia, L.P., Pacheco, A.C.: A simulator supporting lectures on operating systems. In: 33rd ASEE/IEEE Frontiers in education Conference, November 5–8, 2003. Boulder, CO
5. Garrido, J.M., Schlesinger, R.: Principles of Modern Operating Systems (2008)