CS 255 - OPERATING SYSTEMS LAB

Mini Project Simulator Report

Team Members :

191CS121 - Harshwardhan Singh Rathore
191CS122 - Himanshu Kumar
191CS123 - Ikjot Singh Dhody
191CS124 - Ishaan Singh
191CS125 - Bhavya Geetanjali Jorige
191CS126 - Joshitha Reddy Dongala
191CS127 - Karaka Jagadeeswara Sai
191CS128 - Katravulapalli Dheemanth
191CS129 - Kintali Praveen
191CS130 - Korada Srinivas Kalyan

Contents

Introduction……02
CPU Scheduling……02
Producer Consumer Problem……06
Reader Writer Problem……06
Dining Philosophers Problem…..……07
Sleeping Barber Problem……08
Banker's Algorithm……..…09
Memory Management……...10
Partitioning Algorithms……...11
Paging……..13
Page Replacement Algorithms……..……14
Disc Scheduling Algorithms……..15
Conclusion……..17
Contribution of code……18

Acknowledgement

The purpose of this report is to understand the concepts of operating system and how to implement them using a programming language. We are thankful to our professor Shashidhar G Koolagudi Sir and Ankit R Kurup Sir for providing us with this opportunity to learn and explore the field in this manner. We would also like to thank the Department of Computer Science and Engineering of the National Institute of Technology for the same.

Introduction

The purpose of this project is to design an Operating system simulator showing different aspects of modern-day computing. Modern day systems need to manage computer resources in efficient ways, maximising CPU utilisation, synchronize concurrent processes, deal with deadlocks, etc.

Based on our observation, we have split a computer into 5 sectors:

Process Synchronisation

CPU scheduling

Disc Scheduling

Paging

Memory Management

CPU Scheduling

CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast, and fair.
Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute and allocates the CPU to one of them.

FCFS - First come first serve

It is a non preemptive scheduling algorithm that simply executes processes in order of their arrival time. One main disadvantage is if the process with high burst time arrives before the process with low burst time then the low Burst time process has to wait really long to get CPU, this is called convoy effect.

Algorithm - We sort the processes according to arrival time and simply calculate the completion time, waiting time and turnaround time.

SJF – Shortest Job first

It is a non preemptive scheduling algorithm that executes the process with the lowest burst time among the list of available processes in the ready queue. In this algorithm, once a process starts its execution then it cannot be interrupted in between its processing. Any other process can be executed only after the assigned process has completed its processing and has been terminated.

Algorithm - We sort the processes according to arrival time and then find the process with least burst time among which have already arrived if the process is found we calculate the completion time, waiting time and turnaround time else we increment the current time and repeat the algorithm

RR - Round Robin

It is a preemptive algorithm. The CPU is shifted to the next process in the ready queue after a fixed interval of time (called time quantum/time slice) or after it has been terminated whichever happens first. The process that is preempted is added to the end of the ready queue. This algorithm also offers starvation free execution of processes.

Algorithm - We sort the processes according to arrival time and insert them in a ready queue. The process is selected from the front to ready queue for execution. It will execute for a fixed interval of time (time slice) and its remaining burst time is maintained, if the process still has some burst time remaining it will again be pushed to the rear end of the queue. But if the burst time becomes zero means the process is completed and we calculate the completion time, waiting time and turnaround time

SRTF - Shortest Remaining Time first

This Algorithm is the preemptive version of SJF scheduling. In SRTF, the execution of the process can be stopped after a certain amount of time. At the arrival of every process, the short-term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Algorithm - We sort the processes according to arrival time and then find the process with minimum burst time among the processes which have already arrived. . If the process is found we give one time unit to the process and decrease its burst time (update remaining burst time) else we increment the current time. We repeatedly find the processes with maximum burst time as new processes might have greater burst time than the processes under execution.

If the burst time becomes zero then we calculate the completion time, waiting time and turnaround time.

LJF – Longest Job First

It is a non-preemptive scheduling algorithm. This algorithm mainly keeps the track of Burst time of all processes that are available at the arrival time itself and then it will assign the processor to the process having the longest burst time. In this algorithm, once a process starts its execution then it cannot be interrupted in between its processing. Any other process can be executed only after the assigned process has completed its processing and has been terminated.

Algorithm - We sort the processes according to arrival time and then find the process with maximum burst time among which have already arrived if the process is found we calculate the completion time, waiting time and turnaround time else we increment the current time and repeat the algorithm

LRTF – Longest Remaining Time First

This algorithm is the preemptive version of LJF scheduling. In this, the process having the maximum remaining time is processed first. Here, we will check for the maximum remaining time after an interval of time(say 1 unit) to see if there is another process having more Burst Time arrived up to that time.

Algorithm - We sort the processes according to arrival time and then find the process with maximum burst time among which have already arrived. . If the process is found we give one time unit to the process and decrease its burst time (update remaining burst time) else we increment the current time. We repeatedly find the processes with maximum burst time as new processes might have greater burst time than the processes under execution. If the burst time becomes zero then we calculate the completion time, waiting time and turnaround time.

Priority – Preemptive

Priority Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a FCFS basis. Sometimes it is important to run a task with a higher priority before another lower priority task,

even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

Algorithm - In Preemptive Scheduling the processes are inputted with arrival time, burst time and priority. We sort the processes according to arrival time and if arrival time is the same then sorted according to priority. We then find the process with maximum priority among which have already arrived. If the process is found we give one time unit to the process and decrease its burst time (update remaining burst time) else we increment the current time. We repeatedly find the processes with highest priority as new processes might have higher priority than the processes under execution. If the burst time becomes zero then we calculate the completion time, waiting time and turnaround time.

Priority – Non Preemptive

Priority Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a FCFS basis. In this algorithm, once a process starts its execution then it cannot be interrupted in between its processing. Any other process can be executed only after the assigned process has completed its processing and has been terminated.

Algorithm – We sort the processes according to arrival time and if arrival time is the same then sorted according to priority. We then find the process with maximum priority among which have already arrived. If the process is found we calculate the completion time, waiting time and turnaround time else we increment the current time and repeat the algorithm

HRRN – Highest Response Ratio First

It is one of the most optimal scheduling algorithms. This is a non-preemptive algorithm in which the scheduling is done on the basis of an extra parameter called Response Ratio. Response Ratio is calculated for each of the available jobs and the job with the highest response ratio is given priority over the others. This algorithm not only favours shorter jobs but it also concerns the waiting time of the longer jobs.

Response Ratio is calculated by the given formula,

Response Ratio = (W+S)/S ; where, W = Waiting Time, S = Service Time or Burst Time

Algorithm - We sort the processes according to arrival time and then find the process with the highest response ratio among which have already arrived using the above formula. If the process is found we calculate the completion time, waiting time and turnaround time else we increment the current time and repeat the algorithm.

Producer Consumer Problem

In computing, the producer–consumer problem is a classic example of a multi-process synchronization problem. There are two cyclic processes, a producer and a consumer, which share a common, fixed-size buffer used as a queue. The producer repeatedly generates data and writes it into the buffer. The consumer repeatedly reads the data in the buffer, removing it in the course of reading it, and using that data in some way. Multiple producers and consumers sharing a finite buffer, adds the additional problem of preventing producers from trying to write into buffers when all were full, and trying to prevent consumers from reading a buffer when all were empty. Apart from this, consumers and producers cannot write and read at the same time. In this problem:

1. Producers add items to the buffer. The buffer size grows.
2. Consumers extract items from the buffer. The buffer size reduces.
3. Producers and Consumers have the same priority and multiple producers or multiple consumers should not access the buffer at the same time.

Therefore, we make use of a counting semaphore with mutual exclusion on the use of buffer to allow the producer and the consumer to access the buffer in the manner above.

While the semaphore maintains the state of the buffer at all times, the mutex lock ensures only 1 worker can operate on the buffer at a time.

Reader Writer Problem

It is a process synchronization problem. In an OS there are a number of processes which perform read/write operations on a file (called the critical section) . In such a multiprocessor system following problems may arise :

- Two processes trying to write on the file at the same time will lead to data inconsistency.

- One process writing and another process reading the file at the same time will lead to dirty read as the file is getting changed but the reading process will read only the old data.

To solve this problem semaphores are used. A semaphore is a variable that controls the number of resources that are available in the system and it is used for process synchronization.

The solution is as follows:

- If a process is performing write operation, then no other processes will be allowed to do read/write operations.
- If a process is doing only read operation, then other processes which perform only read operations can be allowed to access the critical section. But processes which perform write operations will not be allowed to access the critical section.

So, in our implementation, we are given the number of readers and writers along with their arrival times and the time they will need to do the read/write operation. We have stored the data for readers and writers in an array of structures. We firstly sort the array according to their arrival times and give priority to the writer in case of a tie. We have a variable which keeps track of the current time. We iterate through the array and call the helper function for the present reader/writer.

- The function for the reader will allow all the readers which arrive before this reader finishes, to access the critical section. We print all the readers along with the time interval . We keep track of the current time and update it with the finishing time of the last reader. Then we call another function which will print all the writers who were waiting during the read operation.
- The function for the writer will print the writer along with the time interval. We update the current time with the finishing time of the writer. Then we call another function which will print all the readers/writers which were waiting during the write operation.

Dining Philosophers Problem

Dining Philosophers is a process synchronization problem. The problem goes like there are N philosophers sitting around a round table and there is one chopstick between each philosopher. A philosopher can only eat if he has two chopsticks. One chopstick can only be taken by one philosopher at a time who is adjacent to the chopstick. Every philosopher either thinks or eats. How can

this be done such that no philosopher starves? It is not known when a philosopher may want to eat or think.
Let's say each philosopher takes the chopstick on their left. So, each philosopher will have one chopstick and will be waiting for the other but there will never be any available chopsticks as every philosopher will be holding one. In this case all of them will starve, that is, a deadlock will occur. The same would happen if each took a chopstick on their right.
One proposed solution for this would be if the first person takes the chopstick on the right and the others would take the chopstick on their left. But the second person would not have a chopstick on their left as the first person had taken the chopstick on their right. Now after the remaining philosophers take the chopstick on their left, there will still be one chopstick free on the left of the first person and so he takes it and eats. Then he puts down both the chopsticks. This makes the chopstick on the right of the last person to be free and so the last person takes it and eats and then puts down the chopsticks. In this manner all the philosophers get to eat and don't starve.

Sleeping Barber Problem

This is another process synchronization problem. There is a barber shop with one barber, one barber chair and a waiting room with many chairs. Once the barber cuts a person's hair he goes and checks the waiting room to see if any customers are present. If they are present, he brings one back with him to the barber chair and cuts their hair. If there are no customers present in the waiting room, he goes back to the barber chair and sleeps.
Once a customer comes, they check what the barber is doing. If he is sleeping, they wake him up and get their hair cut. But if he is tending to a customer, they check the waiting room to see if there is a free seat. If there is a free seat, they take it otherwise they leave the barber shop.
This may seem like a good system but there are a few complications that may occur. Mostly these complications occur as the time taken for each action is not known. For example, one issue that may occur is if two customers arrive at the same time and the barber is cutting someone's hair and so when they go to the waiting room there is only one chair left. Both would attempt to occupy that chair. Another problem that may occur is that the customer may wait in the waiting area indefinitely while the barber sleeps in his chair.
The key would be to keep track of the free waiting room seats (the buffer), the status of the barber and enforce mutex locks.

Banker's Algorithm

This algorithm is a deadlock avoidance and a resource allocation algorithm. It checks whether a certain allocation is safe or not. It's called Banker's Algorithm because it is used in the banking system to see whether a loan can be given to a person or not depending on the resources, that is money, available.

Example:
Let's say there are three processes P1, P2, P3 with three resources A, B, C. Maximum number of resources for A is 9, B is 7 and C is 10.

Process	Required			Allocated		
	A	B	C	A	B	C
P1	4	3	5	2	2	2
P2	3	6	4	2	3	4
P3	5	1	1	2	0	1

This is a safe configuration and the safe order can be: P1 -> P2 -> P3

Process	Required			Allocated		
	A	B	C	A	B	C
P1	9	3	5	2	2	2
P2	7	6	4	2	0	4
P3	5	1	5	2	0	1

This is an unsafe configuration and none of the processes can be satisfied here.

So, in our code we input the number of processes and resources. Then we store the maximum number of each resource available. For each process we see how many of each resource is required and how many have already been allocated. Now we iterate through all the processes and see whether the requirements of the process can be satisfied with the currently available resources. If it can be satisfied, we mark the process as complete and free the resources allocated to it. We keep track of the order in which the processes are complete as that will give us the safe sequence in case this is a safe configuration. We keep iterating through the processes until all the processes are complete or if no new process

got completed during the iteration. If all the processes are completed then it means that this is a safe sequence and we output the order through which we can avoid a deadlock. If in case we break out of the loop without having all the processes complete it means that this is an unsafe situation and a deadlock cannot be avoided.
In this manner we can decide whether a configuration is safe or not and if it is safe what order are the processes completed to avoid deadlock.

Memory Management Techniques

In operating systems, memory management is the function responsible for managing the computer's primary memory.

The memory management function keeps track of the status of each memory location, either allocated or free. It determines how memory is allocated among competing processes, deciding which gets memory, when they receive it, and how much they are allowed. When memory is allocated it determines which memory locations will be assigned. It tracks when memory is freed or unallocated and updates the status. Here are few memory management techniques,

MFT or fixed partitioning scheme

MFT is one of the old memory management techniques in which the memory is partitioned into fixed size partitions and each job is assigned to a partition. The memory assigned to a partition does not change. MFT suffers with the problem of internal fragmentation.

1. The OS is partitioned into fixed sized blocks at the time of installation.

2. It is possible to bind addresses at the time of compilation.

3. It is not flexible because the number of blocks cannot be changed.

4. There can be memory wastage due to fragmentation.

MVT or variable partitioning scheme

MVT is the memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning of memory is dynamic and changes as jobs enter and leave the system.

1. No partitioning is done at the beginning.

2. Memory is given to the processes as they come.

3. This method is more flexible.

4. Variable size of memory can be given as there is no size limitation.

5. There is no internal fragmentation. But there can be external fragmentation.

6. Compile time address binding cannot be done.

We have implemented both partition schemes and we conclude MVT is a more efficient user of resources. The external fragmentation caused by MVT is removed by paging which will be discussed later. We have different partitioning algorithms to implement MVT and MFT.

Partitioning Algorithms

There are many algorithms which are implemented by the operating system to find holes in the list and allocate them to the processes

First Fit

The first fit approach is to allocate the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.

Algorithm-

1. Get no. of Processes and no. of blocks.
2. After that get the size of each block and process requests.
3. Now allocate processes
   if(block size >= process size) allocate the process
   else move on to next block
4. Display the processes with the blocks that are allocated to a respective process.
5. Stop.

Advantages-
It is fast in processing. As the processor allocates the nearest available memory partition to the job, it is very fast in execution.

Disadvantages-
It wastes a lot of memory. The processor ignores if the size of partition allocated to the job is very large as compared to the size of job or not. It just allocates the memory. As a result, a lot of memory is wasted and many jobs may not get space in the memory, and would have to wait for another job to complete.

Best Fit

The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to the actual process size needed.

Algorithm-

1. Get no. of Processes and no. of blocks.
2. After that get the size of each block and process requests.
3. Then select the best memory block that can be allocated using the above definition.
4. Display the processes with the blocks that are allocated to a respective process.
5. Value of Fragmentation is optional to display to keep track of wasted memory.
6. Stop.

Advantages-
Memory Efficient. The operating system allocates the job minimum possible space in the memory, making memory management very efficient. To save memory from getting wasted, it is the best method.

Disadvantages-
It is a Slow Process. Checking the whole memory for each job makes the working of the operating system very slow. It takes a lot of time to complete the work.

Worst fit

The worst fit approach is to locate the largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

Algorithm-

1. Get no. of Processes and no. of blocks.
2. After that get the size of each block and process requests.
3. Then select the highest memory block that can be allocated using the above definition.
4. Display the processes with the blocks that are allocated to a respective process.
5. Value of Fragmentation is optional to display to keep track of wasted memory.
6. Stop.

Advantages-
Since this process chooses the largest hole/partition, therefore there will be large internal fragmentation. Now, this internal fragmentation will be quite big so that other small processes can also be placed in that leftover partition.

Disadvantages-
It is a slow process because it traverses all the partitions in the memory and then selects the largest partition among all the partitions, which is a time-consuming process.

Paging

Page replacement algorithms are used in operating systems that uses paging for memory management. Whenever a new page comes in, this algorithm decides which page needs to be replaced. Page replacement is done when a page fault occurs which means that when the requested page is not found in the main memory, it has to be brought from the secondary memory.

Algorithm - We take input the virtual address space, physical address space and page size. Then ask the user to insert the page specifying the virtual address one by one and if an empty frame is available, we insert that page in the frame and update the page table. If the page already exists, we display the corresponding physical address of that page. If all the frames are filled, we replace the page using FIFO page replacement algorithm and update the page table. When the user stop adding more pages, we display the final physical memory with all frames and the final page table.

Page Replacement Algorithms

There are four main page replacement algorithms namely FIFO, LRU, MRU, optimal.

FIFO - First In First Out

It is the simplest page replacement algorithm in which the operating system keeps track of all the pages in the main memory in the form of a queue such that the oldest page remains in the front of the queue and the latest will remain at last. So, whenever a new page comes in, this page gets replaced with the page present at the front of the queue and the newly added page remains at the end of the queue as it is the latest one.

LRU - Least Recently Used

It is a greedy algorithm in which the operating system keeps track of all the pages in the main memory in the form of a queue where the pages are placed in the order in which they are used. The least recently used page will be placed at the front of the queue and the most recently used page will be placed at the end. So, when a new page is called in, the least recently used page gets replaced.

MRU - Most Recently Used

It is also a greedy algorithm in which the operating system keeps track of all the pages in the main memory in the form of a stack where the page that is used most recently will be placed at the top of the stack. When a page which is already present in the main memory is used again, it shifts to the top of the stack as it is the most recently used page. So, whenever a new page is called in, it gets replaced with the page present at the top of the stack.

Optimal Page Replacement

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.

In all the four algorithms mentioned above, page fault occurs only when a new page (page that is not present in the main memory at that point of time) is called. When a page which is already present in the memory is called again it shows ‘page hit’ which means that the page is already present in the main memory so there is no need for page replacement. So, after all the required pages are

called, the number of page faults occurred will be shown and the page fault ratio is also calculated. Page fault ratio refers to the ratio of number of page faults to the total number of pages called during the whole process.
Page fault ratio = number of page faults/total number of pages called.

Disc Scheduling Algorithms

FCFS-First Come First Serve

In FCFS, the requests are addressed in the order they arrive in the disk queue. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

Example:
Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is : 50

So, total seek time = (82-50)+(170-82)+(170-43)+(140-43)+(140-24)+
(24-16)+(190-16) = 642

Advantages:
Every request gets a fair chance
No indefinite postponement

Disadvantages:
Does not try to optimize seek time
May not provide the best possible service

Scan(Elevator)

It is also called The Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns the backend and moves in the reverse direction satisfying requests coming in its path.
It works in the way an elevator works, as an elevator moves in a direction completely till the last floor of that direction and then turns back.

Example:
Suppose the order of request is- (82,170,43,140,24,16,190)
and the current position of Read/Write head is : 50, and the disc moves towards the right end.
So, total seek time = (50-43)+(43-24)+(24-16)+(82-16)+(140-82)+
(170-40)+(190-170) =208

Advantages:
Average Response Time decreases
Throughput increases

Disadvantages:
Overhead to calculate seek time in advance
Can cause Starvation for a request if it has higher seek time as compared to incoming requests
High variance of response time as SSTF favours only some requests

C-Scan

In the SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.
These situations are avoided in the CSCAN algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to the SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).
Example:
Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50.
Seek time is calculated as =(199-50)+(199-0)+(43-0) = 391

Advantages:
Provides more uniform wait time compared to SCAN

Look

It is like SCAN scheduling Algorithm to some extent except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of the SCAN algorithm which forces the disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.

Example:
Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and the disc moves towards the right end.
So, the seek time is calculated as =(190-50)+(190-16) =314

C-Look

As LOOK is similar to SCAN algorithm, in a similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to

the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:
Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and the disc moves towards the right end.
So, the seek time is calculated as: =(190-50)+(190-16)+(43-16) = 341

SSTF-Shortest Seek Time First
In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of the system. Let us understand this with the help of an example.

Example:
Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is : 50
So, total seek time: = (50-43)+(43-24)+(24-16)+(82-16)+(140-82)+
(170-40)+(190-170) = 208

Advantages:
Average Response Time decreases
Throughput increases

Disadvantages:
Overhead to calculate seek time in advance
Can cause Starvation for a request if it has higher seek time as compared to incoming requests

Conclusion

In conclusion, this was an enlightening experience for both the students and the instructor. While it was not the original intent of the assignment, this project touched upon many separate threads discussed in previous courses. These threads were woven together into a cohesive whole -- a cloth that tied together many concepts, allowing them to see and appreciate why we previously spent all that time over "boring, useless" material. And as the threads that enter a loom are woven into patterns, so too were the concepts that the students used in designing their project woven into a fabric that gave them the opportunity to

learn about processes and to further their knowledge of the architecture, organization, and operation of a computer system

Contributions

- **Driver Program** - Ishaan Singh
- **CPU Scheduling** - Himanshu Kumar
- **Producer Consumer Problem** - Ikjot Singh Dhody
- **Reader Writer Problem** - Harshwardhan Singh Rathore
- **Dining Philosophers Problem** - Ishaan Singh
- **Sleeping Barber Problem** - Ishaan Singh
- **Banker's Algorithm** - Joshitha Reddy Dongala
- **MFT** - Bhavya Geetanjali Jorige
- **MVT** - Kintali Praveen
- **Paging** - Himanshu Kumar
- **Page Replacement** - Korada Srinivas Kalyan
- **Disk Scheduling** - Katravulapalli Dheemanth