FINALS PROJECT : DOCUMENTATION

CS301 | November 18, 2023

Buduan, Mark Joseph
Keele, Regine Ann
Nacpil, Marc Jayson
Salangsang, John Patrick

I. INTRODUCTION

The SKIMATA Application Project is designed to illustrate the behavior of various memory and CPU management algorithms. Its primary goal is to provide users with a visual representation of how various algorithms perform under different configurations. This application allows users by offering them flexibility in choosing a range of memory and CPU management algorithms, allowing the users to have control or customization in terms of configurations of processes while providing real-time visualizations of algorithmic operations along with a comprehensive summary of performance metrics. This project provides significant value in enhancing comprehension of these algorithms across different scenarios. The interactive and user-friendly interface and visualizations, enables users to actively engage with and analyze how different algorithms manage and allocate its processes. Hence, the application becomes a valuable tool for gaining insights into the complexities of memory and CPU management algorithms.

Fixed Partition (BF, FF, WF)

- Fixed partitions divide the main memory used by user processes during system startup in order to support multiprogramming. The partition sizes don't change, so the system must be shut down in order to be reconfigured. On a first-come, first-served basis, one contiguous partition is allotted to each job. Three techniques are used in free space allocation: worst-fit (assigning the largest free block to a new project), best-fit (allocating the smallest partition fitting requirements), and first-fit (assigning the first partition that fits requirements). The preservation of a task's memory space is a key feature added by this approach, highlighting the necessity of matching work size with partition size for effective memory utilization.

Page Replacement Policy (FIFO)

- Page Replacement Policy is used for memory management that uses paging in deciding which page should be replaced when a new page is about to be allocated. One of the algorithms used for this policy is the First in First Out (FIFO) Algorithm which is known for its simplicity. Basically, all pages are stored in a queue which is being handled by the operating system. It decides which existing page needs to be replaced in the queue, it replaces the oldest page (first page) residing for the longest time in the memory.

Non-Preemptive CPU Scheduling (Priority)

- A non-preemptive priority scheduling algorithm is a fundamental approach in batch systems that aims to control the order of process execution. Within this structure, each incoming process is assigned depending on its initial arrival time, with those arriving earlier being prioritized for processing. When numerous processes arrive at the same time, the algorithm explores deeper into their respective priorities, favoring the highest-priority process for execution first. Furthermore, when many processes have the same priority level, the algorithm evaluates their process numbers and chooses the one with the lower process number to go first. This iterative process is repeated until all procedures are completed, ensuring a systematic and logical approach to task management and execution within the system.

Preemptive CPU Scheduling (SRTF)

- SRTF stands for Shortest Remaining Time First. It is one of the various algorithms used for process management, and it falls under the preemptive scheduling category. The primary queue used in SRTF is known as the ready queue; this queue contains all the processes ready to execute. The behavior of its scheduling algorithm in allocating a process to the processor is by selecting a process with the least amount of remaining time until completion – the scheduler will always choose the shortest remaining time in the ready queue. The scheduler may interrupt the current process in the processor when a new process arrives with a shorter burst time. Furthermore, a Gantt chart can be utilized to visualize the execution of the processes as they are allocated and scheduled over time.

II. SYSTEM DESIGN

Fixed Partition (BF, FF, WF)

The simulation application has a JavaScript-based architecture and structure, primarily utilizing HTML for the user interface. The application consists of functions and event listeners that respond to user input and dynamically update the webpage. The user interface is designed to collect input for memory partitions, processes, allocation type, and other parameters. The application employs algorithms for fixed partitioning schemes, specifically First Fit, Best Fit, and Worst Fit. Additionally, the data structures include arrays (memoryPartitions and processes) to store and manipulate partition sizes and process sizes. The simulation dynamically updates the display to showcase the allocation results, utilizing CSS animations for visual appeal. The application also incorporates input validation functions to ensure accurate and reliable user input. Overall, the structure follows a modular approach with separate functions for each aspect of the simulation, enhancing maintainability and readability.

Utility and Validation Functions	clear(): Clears various HTML elements to prepare for new simulation results. hasNegativeValue(arr): Checks whether an array contains negative values. validateCount(): Checks whether an input on the partition count is negative or null. validateSize(): Validation checks for the input on partition size (null, size!= partition count, input has spaces, negative input) validateProcess():Validation checks for the input on process size (null, size!= partition count, input has spaces, negative input)
Button Event Listeners	submitBtn: this button submits all the values inside the input fields and calls the function run() which also call for the validation functions previously mentioned and if valid, it will call the function runAlgo() which performs the simulation logic.
Simulation Logic	runAlgo(): This is the main function that will be called when a submission is given.It gets all the value inside the input fields and uses them to perform the allocation method chosen by the user. It will also display the visualization of the allocation method.
Allocation Techniques	firstFit(memoryPartitions, processes): This is a function that is callable by runAlgo() when a user chooses to perform a First Fit allocation technique. bestFit(memoryPartitions, processes): This is a function that is callable by runAlgo() when a user chooses to perform a Best Fit allocation technique.

	worstFit(memoryPartitions, processes): This is a function that is callable by runAlgo() when a user chooses to perform a Worst Fit allocation technique.

Data Structures:

Arrays	The program made use of arrays such as memoryPartitions and processes which are used heavily in processing the values in order to perform the algorithms that are present in the program.

Page Replacement Policy (FIFO)

Architecture :

Declared Global Variables :

pageFrames	This variable serves as the frame, where pages are allocated and unallocated. Specifically, it is set to an array for the startSimulation() function.
fifoQueue	This variable serves as a queue where pages are stored before being allocated in the frames.
pageFaults	This variable keeps track of the page faults in the whole process of the algorithm
totalRequest	This variable holds the length of page requests.

Declared Local Variables under startSimulation() function :

numFrames	This variable retrieves the value from the text field where the user enters the number of desired frames
pageRequest	This variable retrieves and splits the user input value of pages and stores them in an array.
framesContainer	This variable has to do with the frames that are presented during the simulation.

2 Main Functions of the Program :

startSimulation()

- This function serves as the heart of the program where the process of the algorithm is executed. Local and Global variables are used in this function. Inside the main function, there are 2 functions which are **updateFrames()** and **animateSimulation().** The updateFrames() is used to update the displayed frames in the application, this function is dependent on the numFrames which is a user input field. Lastly, animateSimulation() is where the execution of first in first out algorithm is done, consisting of nested if statements such as checking if there are remaining page requests, keeping track of the page faults and handling the page replacement using FIFO. This is also where the simulation is executed and displaying the total page faults, the success and failure rate.

resetSimulation

- This function basically, reset the simulation. Clearing all the results and inputs for every execution.

Data Structures :

- First In First Out was the main algorithm implemented in the program. It was done by utilizing the use of arrays and lists where the pages are stored. Furthermore, the use of function, loops and conditional statements are heavily used in this program specifically, in swapping pages between the queue and frames.

Non-Preemptive CPU Scheduling (Priority)

Architecture :

DOM Interaction	The code begins by declaring and initializing variables that reference HTML elements using document.getElementById. These elements include error message placeholders and input fields.
Utility Functions	clearTable: Clears various HTML elements to prepare for new simulation results. hasNegativeValue: Checks whether an array contains negative values.
Event Listeners	DOMContentLoaded: Listens for the DOM content to be loaded and defines the primary functionality of the simulation within the NonPreemptivePriority class. NonPreemptivePriority Class: This class encapsulates the logic for the Non-Preemptive Priority Scheduling algorithm.

	Constructor: Initializes data structures (processes, arrivalTime, burstTime, priority) and calculates essential parameters like numberOfProcesses.
Methods	sortAccordingArrivalTimeAndPriority: Sorts the input arrays based on arrival time and priority, maintaining process associations. priorityNonPreemptiveAlgo: Performs the priority scheduling algorithm, calculating end times, turnaround times, and waiting times for each process. buildTable: Generates an HTML table to display simulation results (process details, times). displayTimeline: Creates a visual timeline representation based on process execution times. init: Initializes the simulation by calling the necessary methods to perform the scheduling algorithm and display results.
Button Event Listeners	resetButton: Reloads the page and resets all simulation-related HTML elements. submitButton: Listens for a button click to validate input, run the simulation, and display the results.

Data Structures :

Non-Preemptive Priority Scheduling Algorithm	The program implements the non-preemptive version of the priority-based scheduling algorithm. Processes are sorted based on arrival time and priority to decide the order of execution.
Arrays	They are heavily used to store process-related data: processes, arrivalTime, burstTime, priority, endTime, turnAroundTime, waitingTime hold relevant information for each process
Bubble Sort	Utilizes nested loops to iterate through the processes and compare their arrival times and priorities. Swapping is performed based on the comparison results to ensure that the processes are arranged in ascending order of arrival time. In cases where arrival times are equal, the algorithm sorts based on priorities.
Table	The buildTable() method generates an HTML table where each row represents a process, and columns display various process details such as ID, arrival time, burst time, priority, end time, turnaround time, and waiting time. Data from the arrays holding process information is used to populate the table cells dynamically.
Gantt Chart	It is generated and displayed using HTML elements (<ul> and <li>) via the displayTimeline() method in the NonPreemptivePriority class. This representation visualizes the execution timeline of

	processes. These are created dynamically based on the calculated execution times.

Preemptive CPU Scheduling (SRTF)

Architecture:

Simulation Logic	The runScheduler() function is the core of the code. It contains the logic for simulating the process scheduling algorithm SRTF. It uses data structures and loops to simulate the execution of processes and calculate the required metrics such as completion time, turnaround time, and waiting time. ● Local Variables - numProcesses: a variable that stores the number of processes inputted. - currentTime : a count variable, this variable will continuously increment until the completion of all processes.
Displaying of Result	Three methods are created to display that various display results namely: 1. displayGanttChart(ganttChart): the method is used to generate and display a gantt chart or timeline for the process upon simulation. ● Local Variables - ganttContainer: it holds a reference to the HTML element with the ID 'gantt-container'. After obtaining the reference to this container, related data is dynamically added inside it. - table: for creating an HTML ‘table’ element. It is a part of DOM API, allowing the dynamic creation of a table for gantt chart. - processesRow: adds a new row for the table element, processes are to be placed here. - timeLabelsRow: adds a new row for the table element, current time is to be placed here. - processCell: add a new cell for every process executed during the current time. - timeLabelsCell: add a new cell to label the timeline of the processes. 2. displayTable(completedProcesses): the method is used to display the table summary which contains all of the inputted arrival time and burst time along with their respective

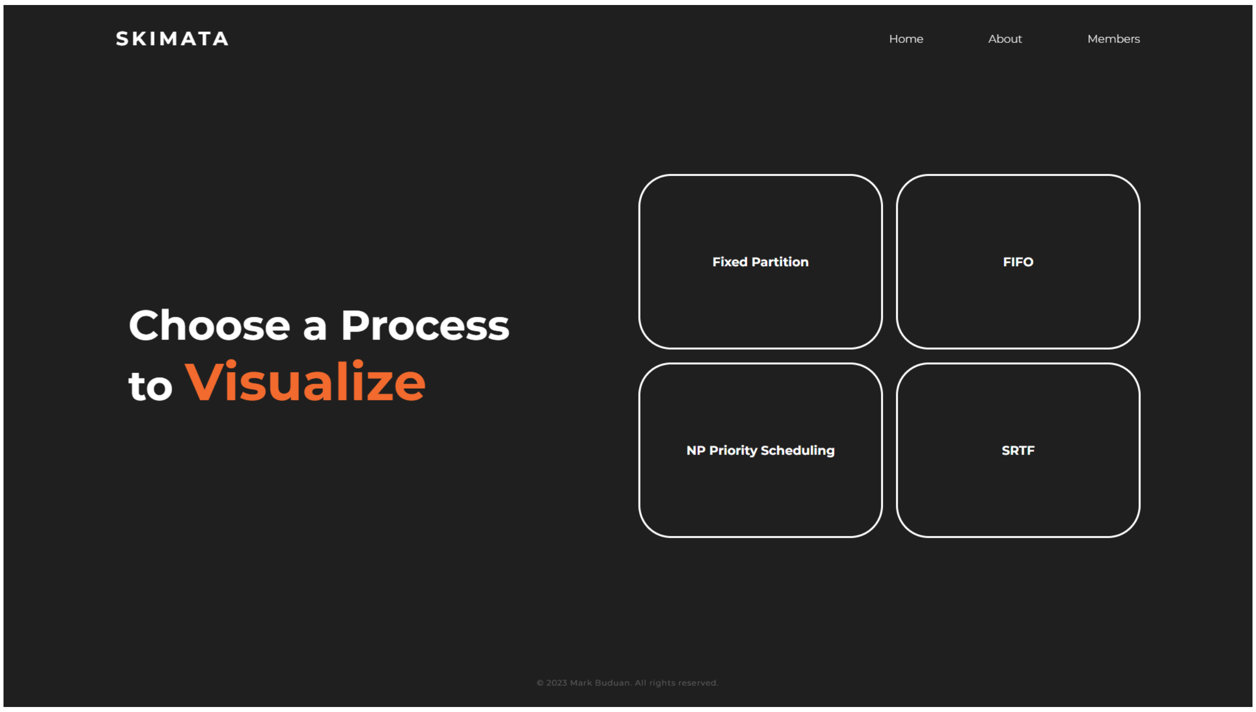
	process id and the resulting completion time, turnaround time, and waiting time. ● Local Variable - resultContainer: it contains a reference to the HTML element with the ID 'result-container'. This element can be manipulated, such as dynamically appending data to it. - table: for creating an HTML ‘table’ element. It is a part of DOM API, allowing the dynamic creation of a table for the table summary. - headerRow: adds a header row and each cell contains a label which are Process, Arrival Time, Burst Time, Completion Time, Turnaround Time, and Waiting Time. - row: used to dynamically insert a row for each process along with the creation of a new cell that fills up the data of the respective labels. 3. displayAverages(completedProcesses): the method is used to display the resulting average Turnaround Time assigned to the variable averageTurnaroundTime and the average waiting time assigned to the variable averageWaitingTime ● Local Variable - averageContainer: it holds a reference to the HTML element with the ID 'average-container', allowing dynamically appending the average result to display the average result. - averageWaitingTime: a variable that holds the result for the computed average waiting time. - averageTurnaroundTime: a variable that holds the result for the computed average turnaround time.
EventListen	These are set up to handle and respond to the user interactions including the buttons and page transitions, such as running the scheduler (Run Schedule button), resetting the input fields (Reset Button), and clicking on a page link.

Data Structures:

Arrays	● allArrivalTimes: an array variable that holds a list of arrival times inputted by the user. ● allBurstTIme: an array variable that holds a list of burst times inputted by the user. ● ganttChart: an array is used to represent the Gantt chart. Each entry in the array corresponds to the current process in the Gantt chart. ● completedProcesses: an array is used to store information about all of the completed processes.
Objects	● processes: this array stores the processes. The dictionary type object uses the processes list to represent individual processes and their attributes. Each process in the processes array is an object with properties such as process_id, arrival_time, burst_time, etc. The actual objects may vary depending on the specific values provided during the simulation.

III. USER GUIDE

Landing Page


SKIMATA
Home
About
Members
Choose a Process to Visualize
Fixed Partition
FIFO
NP Priority Scheduling
SRTF
© 2023 Mark Buduan. All rights reserved.

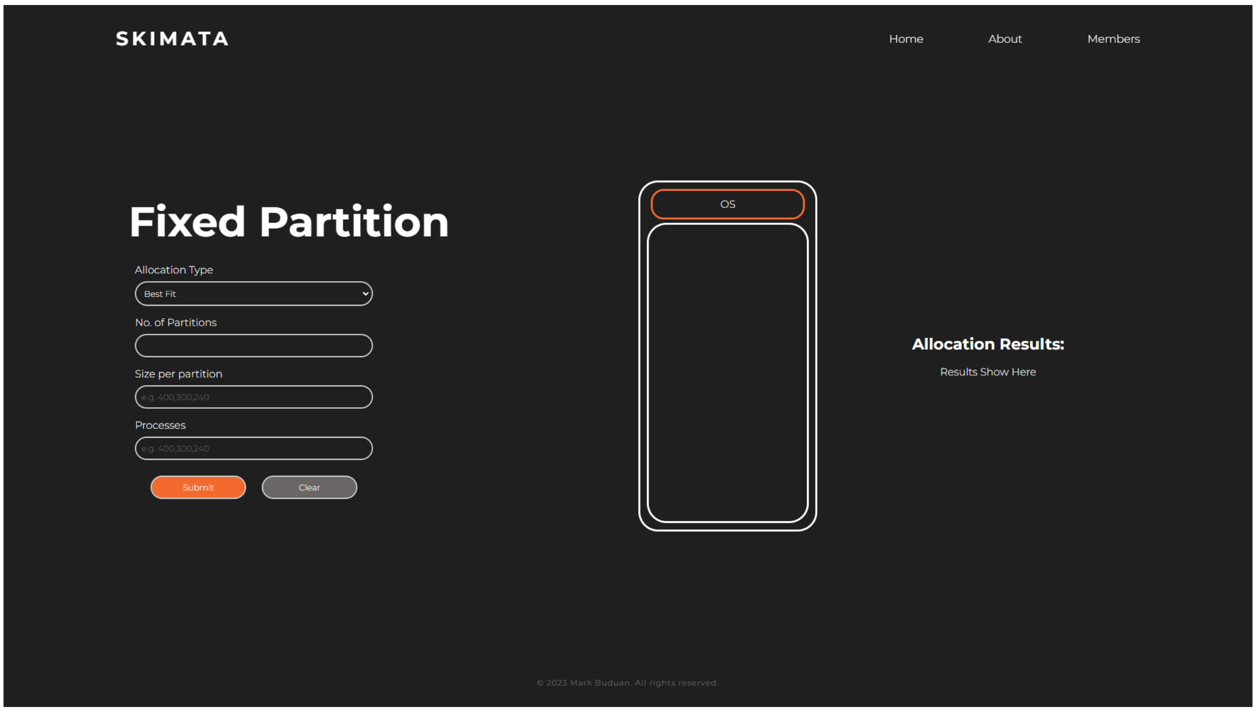
- Upon opening the simulation application, the user is greeted by a welcoming landing page.
- The landing page presents the user with four options:
  - Fixed Partition, FIFO, Priority Scheduling, SRTF
- Users can select their desired algorithm by clicking on the respective buttons.

Navigation Bar

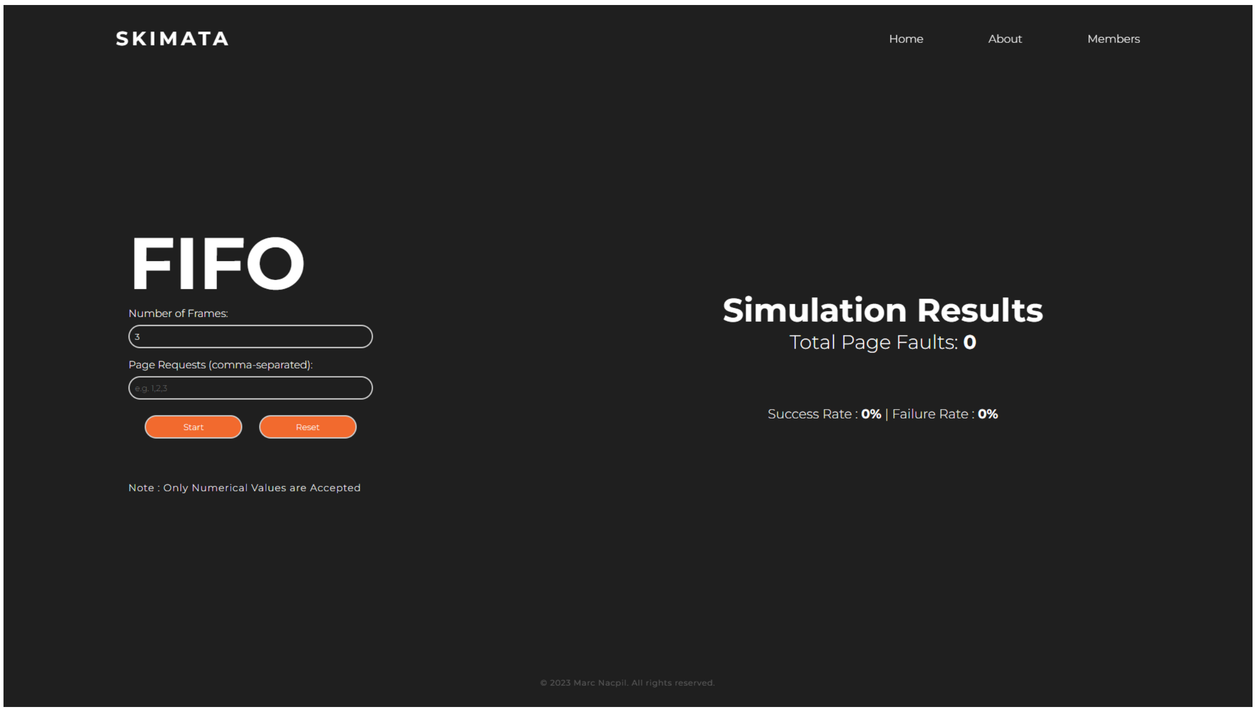
- Located at the top of the screen, the navigation bar provides quick access to essential sections of the application.
- The navigation bar consists of three buttons:
  - Home Button: Directs the user back to the landing page.
  - About Button: Provides information about the application.
  - Members Button: Navigates to a page that lists the group members.

Algorithm Selection and Navigation :

- Upon clicking on an algorithm option from the landing page, the user is directed to a dedicated page for a specific algorithm.
- After the user has been redirected to the chosen algorithm, each algorithm has their own field which the users interacts with. Each of the parameters / fields has their own implementation of error handling used to validate user inputs.



  - Fixed Partition : Allocation Type, No. of Partitions, Sizes per Partition, Processes
    - Allocation Type: The user can choose between First Fit, Worst Fit, or Best Fit.
    - No.of Partitions: The user can indicate how many partitions are inside the memory.
    - Sizes per Partitions: The user can indicate how large each partition is inside the memory.
    - Processes: The user can indicate incoming processes which are going to be allocated inside the memory.



FIFO

Number of Frames: 3

Page Requests (comma-separated): e.g. 1,2,3

Note : Only Numerical Values are Accepted

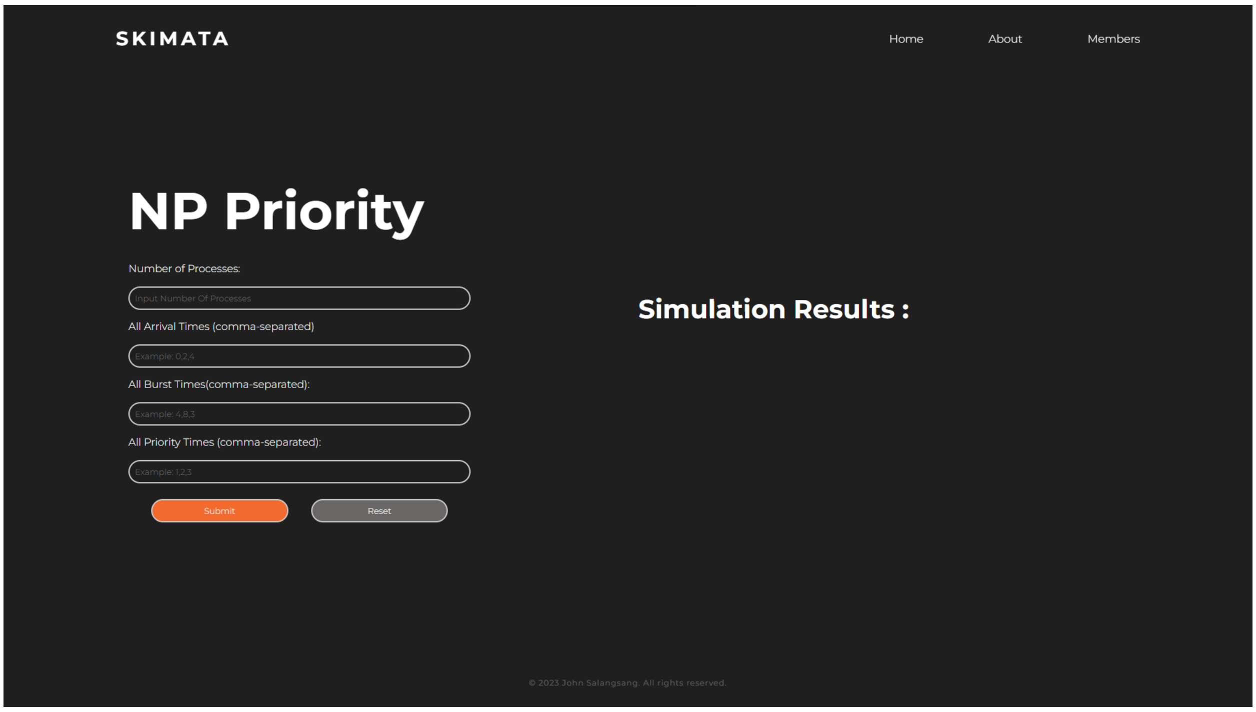
Simulation Results

Total Page Faults: **0**

Success Rate : **0%** | Failure Rate : **0%**

© 2023 Marc Nacpil. All rights reserved.

- FIFO : Number of Frames, Page Request
  - Number of Frames: The user can indicate how many frames are going to be utilized.
  - Page Requests: The user can indicate the requests that will be simulated.



NP Priority

Number of Processes: Input Number Of Processes

All Arrival Times (comma-separated): Example: 0,2,4

All Burst Times(comma-separated): Example: 4,8,3

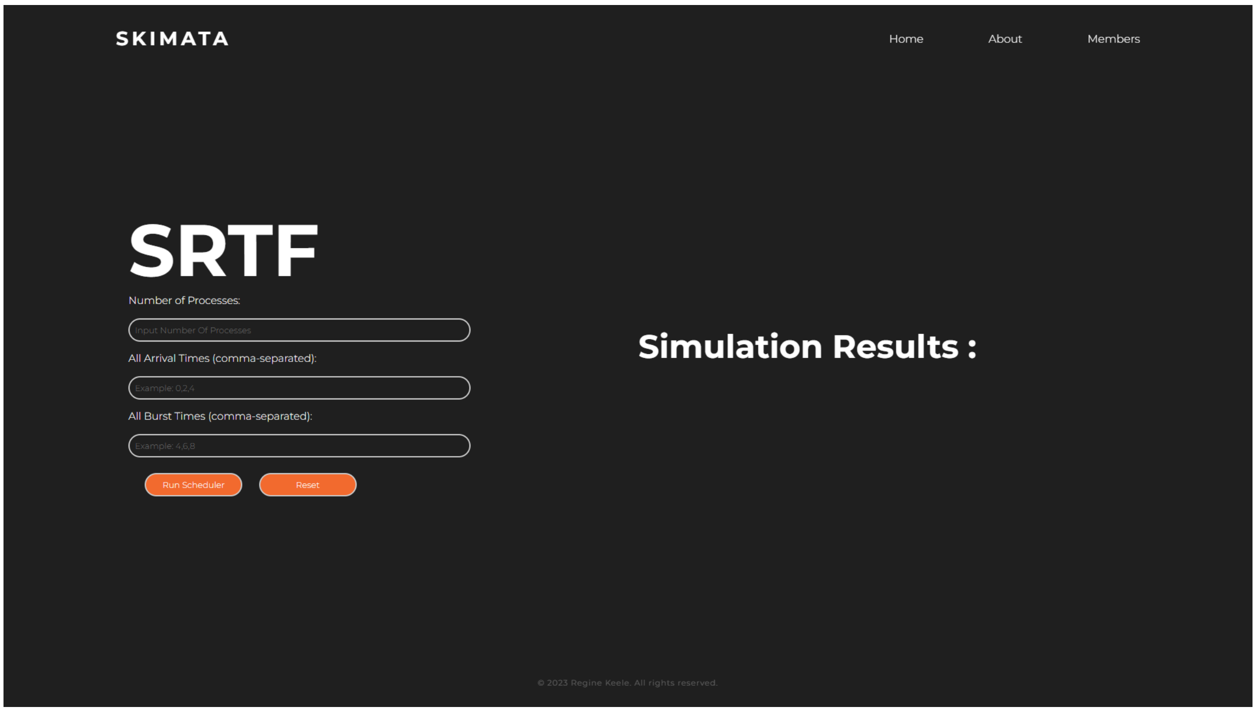
All Priority Times (comma-separated): Example: 1,2,3

Simulation Results :

© 2023 John Salangsang. All rights reserved.

- Priority : Processes, Arrival Time, Burst Time, Priority
  - Processes: The user can input how many processes they want to include in the simulation.
  - Arrival Times: The user can input the arrival time of each process.

        - Burst Times: The user can input the burst time of each process.
        - Priority: The user can input the priority number of each process.


SKIMATA
Home
About
Members
SRTF
Number of Processes:
Input Number Of Processes
All Arrival Times (comma-separated):
Example: 0,2,4
All Burst Times (comma-separated):
Example: 4,6,8
Run Scheduler
Reset
Simulation Results :
© 2023 Regine Keele. All rights reserved.

    - SRTF : No. of Processes, Arrival Time, Burst Time
        - Processes: The user can input how many processes they want to include in the simulation.
        - Arrival Times: The user can input the arrival time of each process.
        - Burst Times: The user can input the burst time of each process.

Utility Buttons

- A dedicated Run / Run Scheduler Button to execute the algorithm and a reset button to reset all fields.
- As the algorithm is executed, interactive elements such as simulations or visualizations are displayed / presented alongside with the summary of each algorithm's performance metrics.
- Users can navigate back to the landing page or to the navigation bar by clicking on the respective buttons

IV. ALGORITHMS EXPLANATION

Fixed Partition (BF, FF, WF)

Fixed Partitioning is a memory management method in which, during system startup, the main memory is partitioned into fixed-size sections. During the system's runtime, the size of each division stays fixed and is assigned to a particular job or process. This technique permits multiprogramming, meaning that several processes can run concurrently in memory. Its disadvantage is that fragmentation, such as internal fragmentation(unused space inside a partition), may cause memory space to be wasted.

Allocation Techniques:

First Fit

Every process is assigned to the first partition big enough to fit it under the First Fit allocation technique. The process is assigned to the first partition that best suits its size by the system, which iterates through the available partitions sequentially. Although this method is straightforward and simple to use, it may not always select the best partition, which might result in fragmentation.

Best Fit

This procedure assigns a process to the smallest partition that can accommodate it. Best Fit seeks to reduce wasted space by choosing the most appropriate partition among all those that are available, in contrast to First Fit, which allots to the first partition that fits. Although this approach tends to lessen fragmentation, the overhead is great and it is considerably slower since it needs to check all of the partitions.

Worst Fit

Worst Fit assigns a process to the biggest partition that is available. The idea behind this method is that assigning a process to the largest partition makes more room for allocations in the future, which may lessen fragmentation. When it comes to allocation techniques, Worst Fit often leads to more internal fragmentation. When a sizable partition is set aside for a comparatively tiny process, a sizable portion of that partition is probably going to remain unutilized.

In conclusion, These allocation techniques are essential components of operating system memory management. The degree of memory fragmentation and system efficiency can be greatly impacted by the allocation technique selection. Every technique has benefits and drawbacks, and the choice is made based on the particular needs and traits of the system at hand.

Page Replacement Policy (FIFO)

First In First Out Algorithm (FIFO) is one of the types of algorithm used in Page Replacement Policy. We can think of it as a queue of people in a cinema ticket booth. It's a straight forward algorithm which replaces the first (or oldest) page that has been residing for a long time in the memory. The way this algorithm works by tracking all the pages by using a queue in the main memory. The oldest page is at the front of the queue, the newest page is at the back of the queue. When a new page arrives and there is no space (frames) available in the memory, the oldest page should be removed.

There are few concepts in this algorithm. The Page Fault (Page Interrupt) and Page Hit. A Page Fault occurs when a page request is not present in the memory, this creates an alert for the Operating System. On the other hand, when a page request is already existing in the main memory is referred to as Page Hit. Determining the effectiveness of the algorithm is dependent on the number of page faults, having a less (fewer) number of page faults signifies the effectiveness of the FIFO algorithm.

One of the advantages of FIFO Algorithm is its simplicity, easy to understand and implement. It's more efficient for small systems as it can provide excellent performance. In contrast, having a large number of pages is one of the downsides of the algorithm that may result in a slower execution process. In Conclusion, the FIFO Algorithm is a good choice for a system working with a small set of pages, while having a huge set of pages may slow down the performance of the algorithm.

Non-Preemptive CPU Scheduling (Priority)

The Non-Preemptive Priority Scheduling Algorithm involves several steps to efficiently schedule processes in a computing system. Initially, the user inputs the number of processes along with their respective burst times, arrival times, and scheduling priorities. Using an enhanced bubble sort technique, the processes are sorted in ascending order based on their arrival times. In cases where multiple processes have the same arrival time, the sorting extends to their priorities, with lower values representing higher priority. These sorted processes form a ready queue.

Subsequently, calculations are performed to determine the Finish Time, Turn Around Time, and Waiting Time for each process. The Finish Time for the initial process is determined by adding its arrival time to its burst time. For subsequent processes, the Finish Time is calculated by adding the burst time to the Finish Time of the preceding process. Turn Around Time is derived by subtracting the Arrival Time from the Finish Time, while Waiting Time is computed by subtracting the Burst Time from the Turnaround Time.

The CPU executes the processes based on their arrival times, prioritizing those with lower arrival times without necessarily considering their priority levels. After executing all processes, the algorithm calculates the Average Waiting Time and Average Turnaround Time, providing crucial insights into the overall efficiency of the scheduling approach. Finally, the algorithm concludes, having completed the scheduling and analysis process.

Preemptive CPU Scheduling (SRTF)

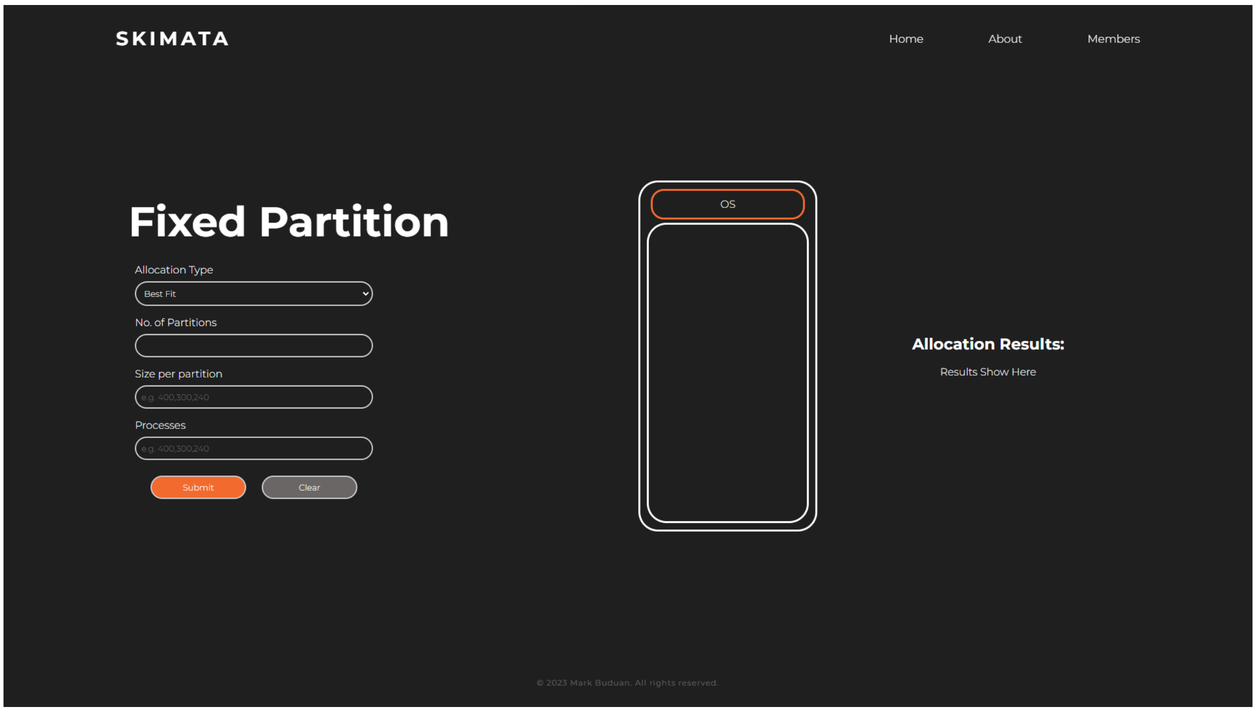
Shortest Remaining Time First(SRTF) is the preemptive version of the Shortest Job Next (SJN) algorithm. Both functions almost identically; however, the difference between the behavior of both is that with SJN, once a process executes, it will continue to be executed, while for SRTF, a new process can interrupt a currently executing process if the new process has a shorter burst time. Nonetheless, SRTF is an improvement over SJN in terms of responsiveness and turnaround time.

The allocation process of SRTF's scheduling algorithm is an iterative process, and it begins by adding processes that have arrived and are ready for execution in the ready queue. The remaining burst times of each process in the ready queue are then compared to each other; the process at the front of the queue is the one with the shortest remaining burst time. Another comparison takes place, where the remaining time of the process at the front of the ready queue is compared with the remaining time of the currently executing process. If the remaining burst time of the process at the front of the ready queue is shorter than the remaining burst time of the currently executing process, the current process in the CPU is preempted and placed back in the ready queue. The process at the front of the ready queue becomes the new current process. This process continues until there are no more processes left to execute. The scheduler always prioritizes the process with the shortest remaining burst time. When a process ends, its turnaround time (completion time minus arrival time) and waiting time (turnaround time - burst time) is calculated. After all of the processes are executed, the average turnaround time and waiting time is then calculated. Additionally, a table summary and Gantt chart can be used for visualization and analysis purposes. Both complement each other in providing a comprehensive view of the scheduling process and behavior under different scenarios.

SRTF scheduling algorithm offers several notable advantages in regards to performance optimization of the CPU. Its primary strengths include achieving optimal turnaround time and reduced waiting time. This makes SRTF more suitable for interactive systems and real-time systems. However, despite its advantages, there are still potential challenges to consider, such as the risk of starvation for longer processes and the increased overhead associated with frequent preemption. The suitability of the SRTF algorithm will depend on the characteristics and requirements of a system or application.

V. RESULT AND ANALYSIS

Fixed Partition (BF, FF, WF)



Primary State of Fixed Partition Page

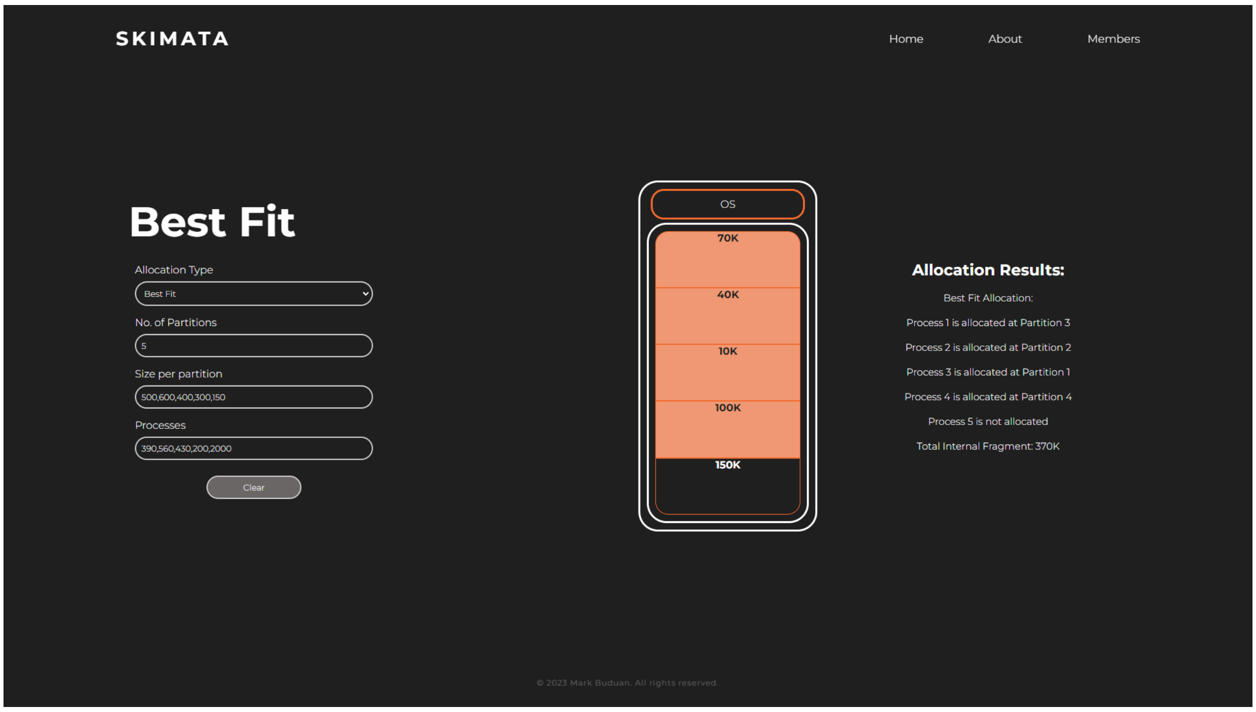
After inputting valid inputs inside the fields, the user can hit the submit button in order for the run() function to be called and execute all the processes within it which includes the validation of the inputs and the main function runAlgo(). Within the process, the function will dynamically create partitions based on the given user input (e.g. 5 = 5 partitions made).

The inputs in these simulations are the following:
No. of Partitions = 5
Size Per Partition:

Partition1	Partition2	Partition3	Partition4	Partition5
500	600	400	300	150

Processes:

Process1	Process2	Process3	Process4	Process5
350	500	430	200	2000



Best Fit

Allocation Type: Best Fit

No. of Partitions: 5

Size per partition: 500,600,400,300,150

Processes: 390,560,430,200,2000

Allocation Results:

Best Fit Allocation:

Process 1 is allocated at Partition 3

Process 2 is allocated at Partition 2

Process 3 is allocated at Partition 1

Process 4 is allocated at Partition 4

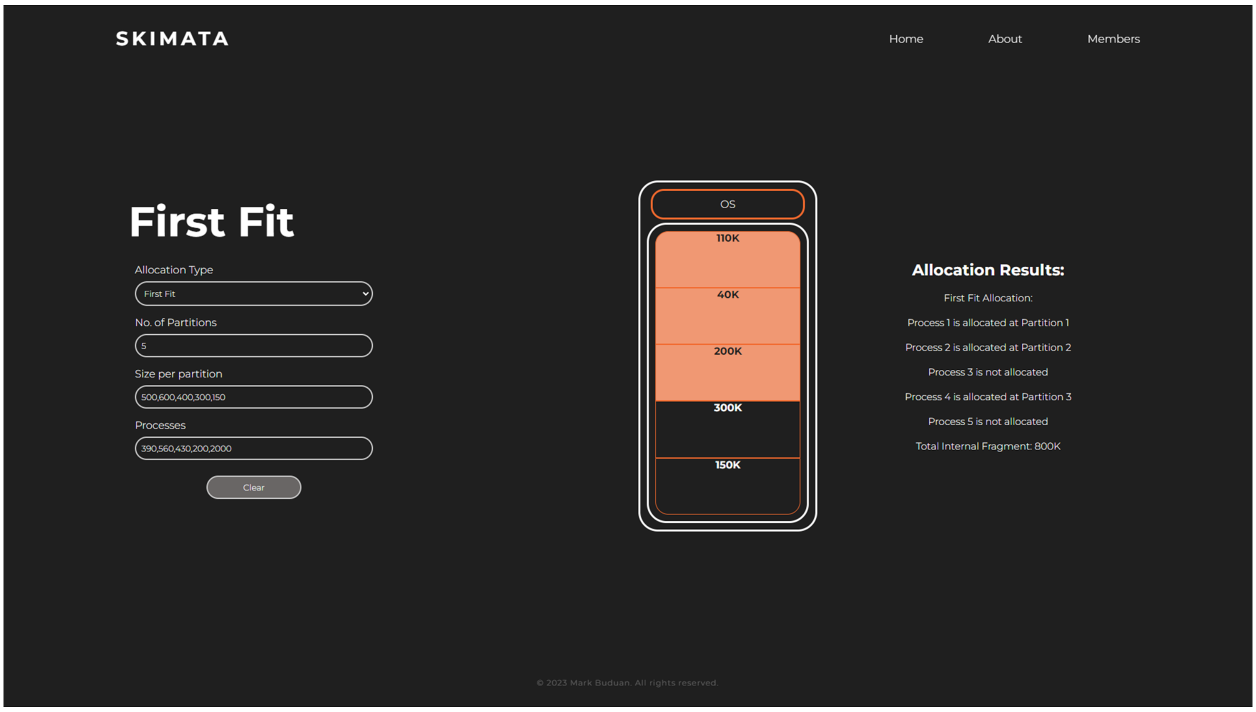
Process 5 is not allocated

Total Internal Fragment: 370K

Best Fit Allocation Technique Results and Analysis:

In this particular simulation, using the inputs mentioned previously, produced correct results based on the allocation technique utilized. Each process was correctly allocated and the graph shows a correct representation (filled partition is used/busy while hollow partition is unused/free) and the internal fragments inside the partitions were also properly reflected. Additionally, the results also reflected correct internal fragmentation count.

Partitions	Partition Size	Jobs	Job Sizes	Status	Internal Frags
Partition1	500K	J3	430K	BUSY	70K
Partition2	600K	J2	560K	BUSY	40K
Partition3	400K	J1	390K	BUSY	10K
Partition4	300K	J4	200K	BUSY	100K
Partition5	150K			FREE	150K
Total Available	1950K	Total Used	1580K		370K



First Fit

Allocation Type: First Fit

No. of Partitions: 5

Size per partition: 500,600,400,300,150

Processes: 390,560,430,200,2000

Allocation Results:

First Fit Allocation:

Process 1 is allocated at Partition 1

Process 2 is allocated at Partition 2

Process 3 is not allocated

Process 4 is allocated at Partition 3

Process 5 is not allocated

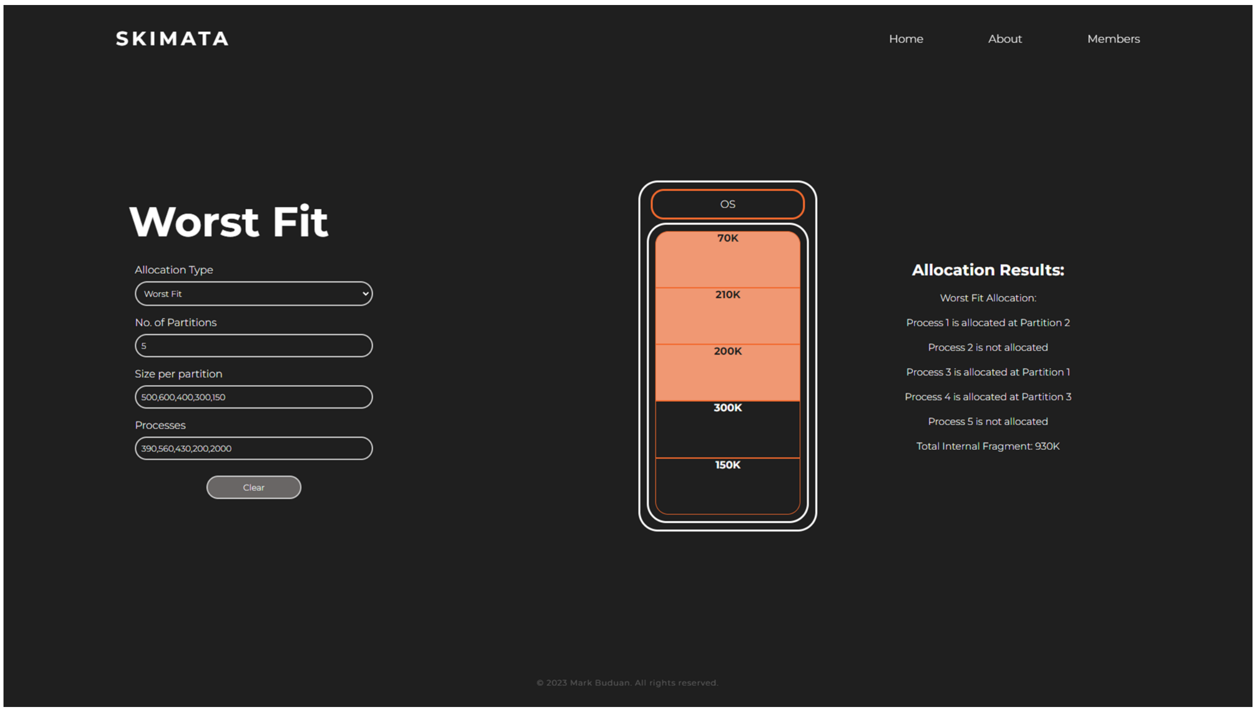
Total Internal Fragment: 800K

© 2023 Mark Buduan. All rights reserved.

First Fit Allocation Technique Results and Analysis:

The same with Best Fit, all results are correct and the graph properly reflects the computations based on the allocation technique used. Upon first observation, the First Fit simulation has reflected different results compared to Best Fit which is a good indicator that is functioning properly. Upon analysis, the First Fit simulation is able to reflect accurate results based on our comparison with our manual computation.

Partitions	Partition Size	Jobs	Job Sizes	Status	Internal Frags
Partition1	500K	J1	390K	BUSY	110K
Partition2	600K	J2	560K	BUSY	40K
Partition3	400K	J4	200K	BUSY	200K
Partition4	300K			FREE	300K
Partition5	150K			FREE	150K
Total Available	1950K	Total Used	1150K		800K



Worst Fit

Allocation Type: Worst Fit

No. of Partitions: 5

Size per partition: 500,600,400,300,150

Processes: 390,560,430,200,2000

Allocation Results:

Worst Fit Allocation:

Process 1 is allocated at Partition 2

Process 2 is not allocated

Process 3 is allocated at Partition 1

Process 4 is allocated at Partition 3

Process 5 is not allocated

Total Internal Fragment: 930K

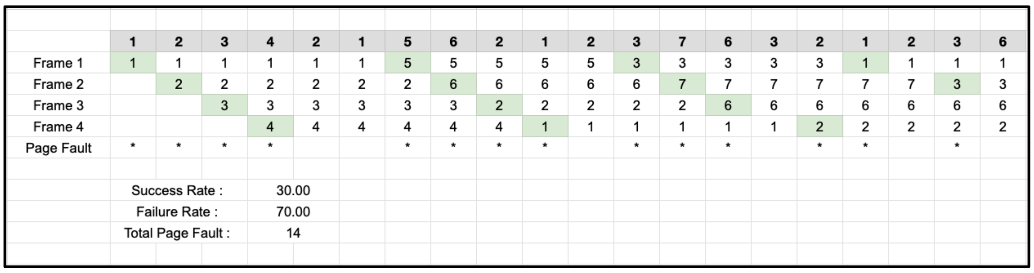
© 2023 Mark Buduan. All rights reserved.

Worst Fit Allocation Technique Results and Analysis:

The same with First Fit, all results are correct and the graph properly reflects the computations based on the allocation technique used. Although the graph is the same with First Fit, it shows different details inside since a different allocation method was used. Internal fragments inside the partitions are significantly larger compared to the previous allocation techniques. Once again, the simulation was counter checked by comparing the results on our manual computation in order to ensure whether the results are reflected correctly.

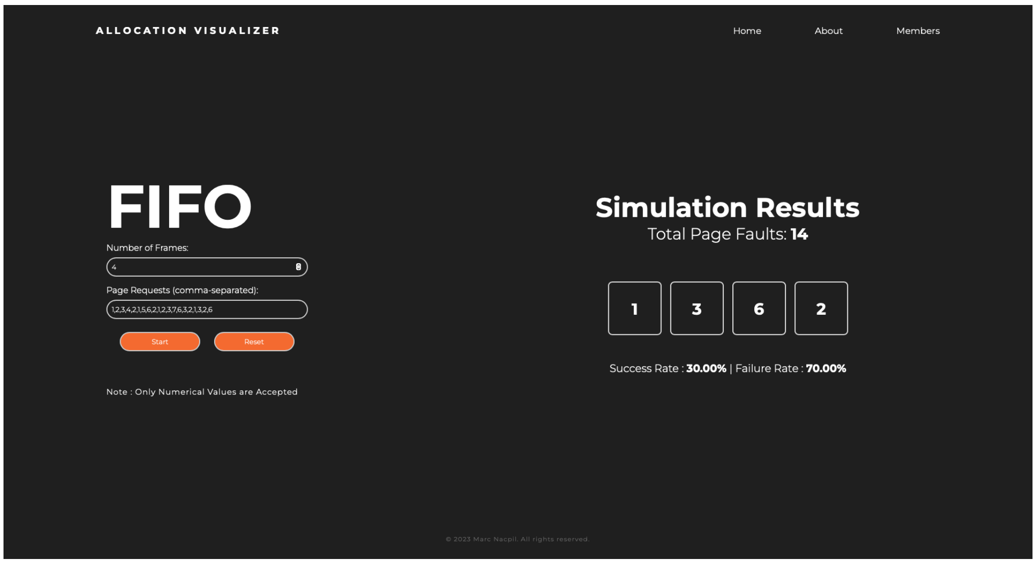
Partitions	Partition Size	Jobs	Job Sizes	Status	Internal Frags
Partition1	500K	J3	430K	BUSY	70K
Partition2	600K	J1	390K	BUSY	210K
Partition3	400K	J4	200K	BUSY	200K
Partition4	300K			FREE	300K
Partition5	150K			FREE	150K
Total Available	1950K	Total Used	1020K		930K

Page Replacement Policy (FIFO)



	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
Frame 1	1	1	1	1	1	1	5	5	5	5	5	3	3	3	3	3	1	1	1	1
Frame 2		2	2	2	2	2	2	6	6	6	6	6	7	7	7	7	7	7	3	3
Frame 3			3	3	3	3	3	3	2	2	2	2	2	6	6	6	6	6	6	6
Frame 4				4	4	4	4	4	4	1	1	1	1	1	1	2	2	2	2	2
Page Fault	*	*	*	*			*	*	*	*		*	*	*		*	*		*	

Success Rate :	30.00
Failure Rate :	70.00
Total Page Fault :	14



FIFO

Number of Frames: 4

Page Requests (comma-separated): 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,3,2,6

Note : Only Numerical Values are Accepted

Simulation Results

Total Page Faults: **14**

1	3	6	2

Success Rate : **30.00%** | Failure Rate : **70.00%**

© 2023 Marc Nacpil. All rights reserved.

Sample Exection using FIFO Algorithm

Upon entering the required fields for the algorithm and clicking the start button, the startSimulation function is initiated. This function begins by extracting all the specified pages and frames from the text fields. The number of page frames to be utilized is stored in a variable. Furthermore, pages are being split as the user input pages are retrieved and being stored in an array.

As the user-provided values are appropriately stored in their respective variables within suitable data structures, the program executes updateFrames function that is the responsible of updating the number of displayed frames within the application. Afterwards, the animateSimulation function takes the responsibility of executing the first-in, first-out (FIFO) algorithm and rendering the animation or visualization.

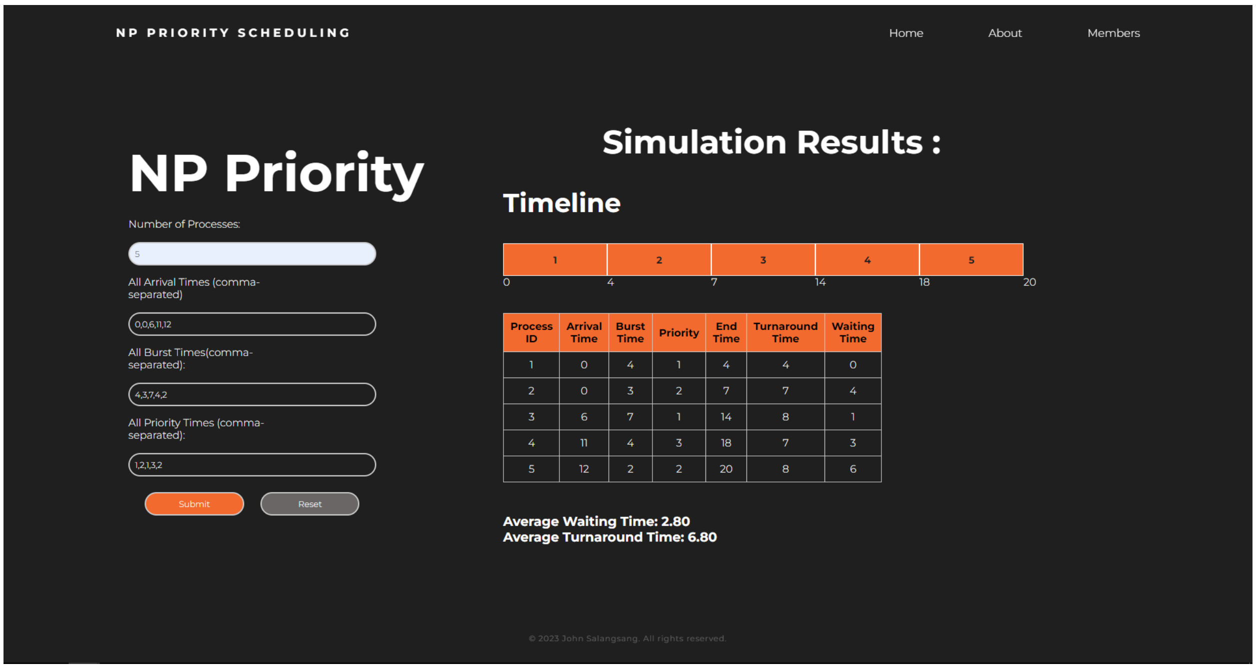
Initially, the function checks for any page requests. It validates the current page and determines if it's a page fault, updating the pageFaults variable accordingly if so. Furthermore, page replacement using FIFO is performed, the FIFO queue is updated, and the displayed frames are refreshed with a simple animation. This process continues until no further page requests are encountered. Finally, the program calculates the success and failure rates and displays the total page faults for each execution.

Non-Preemptive CPU Scheduling (Priority)

First, processes get executed according to their arrival time and priority. In the following implementation, first the number of processes and their respective arrival, burst timings and priorities are accepted by the class constructor having the following method variables: processes, arrivalTime, burstTime, priority.

After accepting the required input for processes, method name priorityNonPreemptiveAlgorithm is called where, first the processes are sorted according there respective arrival time in ascending order (less arrival time high priority) and processes having same arrival timings are sorted according to their priorities (less value denotes high priority) using enhanced bubble sort method in class method having following signature: sortAccordingArrivalTimeAndPriority().

For instance,



NP Priority

Number of Processes: 5

All Arrival Times (comma-separated): 0,0,6,11,12

All Burst Times(comma-separated): 4,3,7,4,2

All Priority Times (comma-separated): 1,2,1,3,2

Simulation Results :

Timeline

1	2	3	4	5

0 4 7 14 18 20

Process ID	Arrival Time	Burst Time	Priority	End Time	Turnaround Time	Waiting Time
1	0	4	1	4	4	0
2	0	3	2	7	7	4
3	6	7	1	14	8	1
4	11	4	3	18	7	3
5	12	2	2	20	8	6

Average Waiting Time: 2.80
Average Turnaround Time: 6.80

© 2023 John Salangsang. All rights reserved.

1. At start P0 arrives and gets executed because its arrival time is 0. Its duration of execution is 0-4 seconds.

2. P1 arrives at time 0 sec during which CPU was busy with Process P0, though their arrival timing is scheduled as per their priority values. After completion of P3, P2 is executed for duration 4-7 seconds.

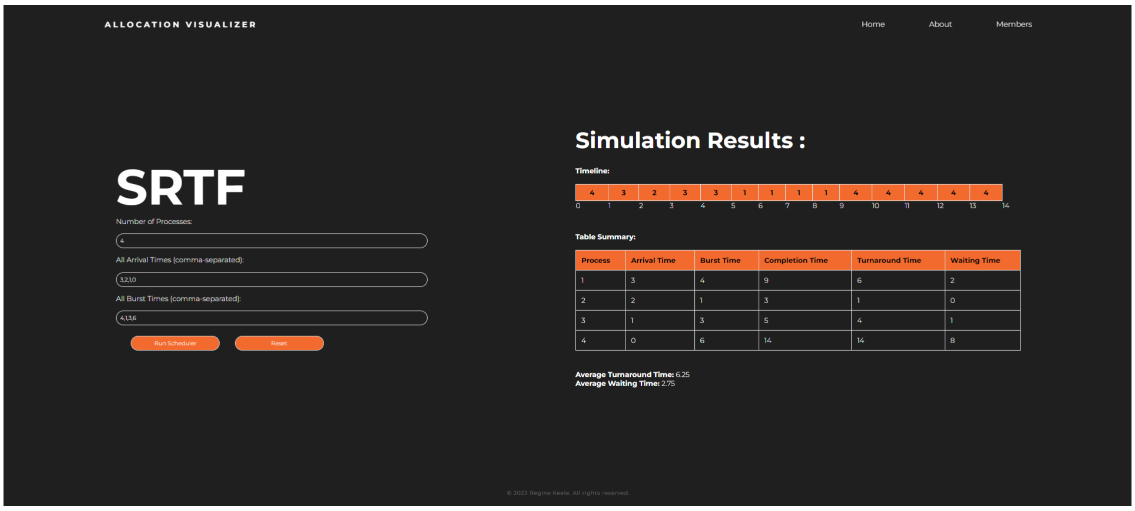
3. P2 arrives at time 6 sec but its execution is started at 7 sec after complete execution of process P1, for a duration of 7-14 seconds.

4. P3 arrives at time 11 sec but gets resources of CPU at 11th second for execution. Its execution period is 14-18 seconds.

5. Similarly P4 arrives at time 12 sec but its execution gets started at time 18th second and lasts for a period 18-20 seconds.

Lastly, after getting sorted according to arrival time and priority, finish, waiting, turn around timings are calculated in the class method having the following signature: priorityNonPreemptiveAlgorithm(). At last, the order of processes executed by the NonPreemptivePriority class is displayed in the form of a table by buildTable() method and a timeline by displayTimeline() method, also the average turn around time and average waiting time are also displayed.

Preemptive CPU Scheduling (SRTF)



SRTF

Number of Processes: 4

All Arrival Times (comma-separated): 3,2,1,0

All Burst Times (comma-separated): 4,1,3,6

Simulation Results :

Timeline:

4	3	2	3	3	1	1	1	1	4	4	4	4	4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Table Summary:

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
1	3	4	9	6	2
2	2	1	3	1	0
3	1	3	5	4	1
4	0	6	14	14	8

Average Turnaround Time: 6.25
Average Waiting Time: 2.75

© 2023 Regine Keele. All rights reserved.

Sample SRTF Algorithm Simulation

Users are required to input data into the fields in order for the simulation to begin. Upon clicking Run Scheduler, all of the inputs are first validated and once every value in the fields are valid then the simulation begins. When the run simulator is clicked, the runScheduler() function is initiated. The input field values are then retrieved and placed on a variable. The variables are numProcesses, allArrivalTimes, and allBurstTimes which holds the value that is inputted in the number of processes field, all arrival time field, all burst time field, respectively.

A dictionary type object is created to represent individual processes and their respective attributes. The attributes for each process are the following process id, arrival time, burst time, remaining time, completion time, turnaround time, and waiting time. It loops until it reaches the last process and once finished, the algorithm for scheduling begins.

A remaining process list is used to keep track if there are still processes to allocate or none. All processes that have a remaining time greater than 0 are placed here. A while loop is used to continue performing the scheduling algorithm until the remainingProcesses length is equal to zero. Inside the while loop, the process with the shortest remaining time is selected. The remaining time of the selected process is decremented, and the process is added to the Gantt chart. If the remaining time becomes zero, the process is marked as completed, and its completion, turnaround, and waiting times are calculated.

At the end of the simulation a gantt chart, result table summary, and average statistics is displayed. The displayGanttChart(ganttChart) function creates and displays the Gantt chart once allocation of all processes is finished; it is based on the simulation results. Next, the displayTable(completedProcesses) function creates and displays a table summarizing the completed processes, along with their arrival time, burst time, completion time, turnaround time, and waiting time. Lastly, the function displayAverage(completedProcesses) is responsible for calculating and displaying the average turnaround time and average waiting time for all completed processes. After all necessary results are displayed, a user can modify the entered inputs then clicking run scheduler again or they could just simply click the reset button which triggers the resetScheduler() function to rest the simulation and clear all field

VI. CONCLUSION

The SKIMATA is a simulation algorithm application that empowers users to explore and experiment with various algorithms. The application consist a various range of algorithms, including Fixed Partition (BF, FF, WF), Paged Replacement Policy (FIFO), Non-Preemptive Scheduling (Priority), and Preemptive Scheduling (SRTF). It provides real-time visual representations of these algorithms, coupled with user-friendly controls that enable dynamic parameter configuration. The SKIMATA application is designed to foster an engaging learning experience, serving as an valuable tool for fully comprehending the underlying concepts of these algorithms.

The SKIMATA application is particularly beneficial for students and learners in the field of computer science. It serves as a valuable tool for three primary reasons: engagement, visualization, and interactivity.

Engagement: An engaging learning method is crucial for fully grasping complex concepts. The SKIMATA application promotes engagement by providing users with flexibility and control in understanding these algorithms, promoting a deeper interest and motivation.

Visualization: Visualization plays a vital role to ease the difficulty associated with comprehending algorithms. The SKIMATA application utilizes visual representations to enhance understanding, allowing users to effortlessly grasp the mechanisms of each algorithm.

Interactivity: Interactive applications establish a profound understanding of algorithms by enabling users to experiment with diverse inputs across various scenarios. The SKIMATA application incorporates interactivity to facilitate a deeper comprehension of algorithms.

Despite its current strengths, the SKIMATA application offers opportunities for further improvement as it continues to evolve. One area for enhancement is the delivery of more detailed animations and simulations. Additionally, the app could benefit from the incorporation of a broader range of algorithms. Furthermore, integrating an explanation feature that provides step-by-step guidance would enhance the user experience and facilitate a deeper understanding of the algorithms.

In conclusion, the SKIMATA application stands as a powerful tool for learning and understanding algorithms. Its engaging nature, coupled with its visualization and interactive capabilities, makes it an valuable asset for students and learners in the field of computer science.

VII. REFERENCES

- Chaudhari, P. R. (2020, April 2). Non-Preemptive Priority CPU Scheduling Algorithm. OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/non-preemptive-priority-cpu-scheduling/
- Datta, S., & Datta, S. (2022, November 11). *How does FIFO page replacement work? | Baeldung on Computer Science*. Baeldung on Computer Science. https://www.baeldung.com/cs/fifo-page-replacement
- GeeksForGeeks. (2023, May 10). *Introduction of Shortest Remaining Time First (SRTF) algorithm.* https://www.geeksforgeeks.org/introduction-of-shortest-remaining-time-first-srtf-algorithm
- GeeksforGeeks. (2023, April 4). *Priority CPU Scheduling with different arrival time – Set 2*. https://www.geeksforgeeks.org/priority-cpu-scheduling-with-different-arrival-time-set-2/
- GeeksforGeeks. (2023, September 15). *Program for page replacement Algorithms Set 2 FIFO*. https://www.geeksforgeeks.org/program-page-replacement-algorithms-set-2-fifo/
- GeeksForGeeks. (2023, September 24). *Shortest Remaining Time First (SRTF) With predicted Time.* https://www.geeksforgeeks.org/shortest-remaining-time-first-srtf-with-predicted-time/
- GeekforGeeks. (2019, October 16). First Fit Allocation in Operating Systems. GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/first-fit-allocation-in-operating-systems/
- GeekforGeeks. (2019, October 16). First Fit Allocation in Operating Systems. GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/first-fit-allocation-in-operating-systems/
- GeekforGeeks. (2020, June 9). Worst Fit Allocation in Operating Systems. GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/worst-fit-allocation-in-operating-systems/
- MyCareerWise. (n.d.). *SRTF Process and Examples*. MyCareerwise. https://mycareerwise.com/content/srtf-process-and-examples/content/exam/gate/computer-science
- Sharma, V. (2022, October 8). FIFO Page Replacement Algorithm - Scaler Topics. *Scaler Topics*. https://www.scaler.com/topics/fifo-page-replacement-algorithm/