Bahria University

Discovering Knowledge

Course: Operating System

Submitted To: Sir Mudassir

Submitted By: Samaha Munir

Enrollment: 03-134221-034

Semester: 5A

Operating System (CSC 330)

BSCS- 5th A

Bahria University Discovering Knowledge	**Department of Computer Science** **Bahria University, Lahore Campus**

Assignment No 4

Due Date:4/06/2024 **Total marks = 5**

Name:Samaha Munir **Roll No:03-134221-034**

CLOs
CLO3: Analyze the algorithms of the core functions of Operating Systems and explain the major performance issues with respect to the core functions.

Marks Evaluation Criteria:

Feature	Assessment Criteria	Marks
Present ability	The assignment is clean, properly formatted, table of contents, title page contains the name/logo of the institute, submitter's name, teacher's name.	0.5
Assignment Content	Questions solved correctly and presented effectively.	4
Deadline Met	Assignments are submitted within a given deadline.	0.5
Deadline Missed	Assignment deadline is missed / and submitted late.	-0.5 / and -0.5 per day

Question – 1:

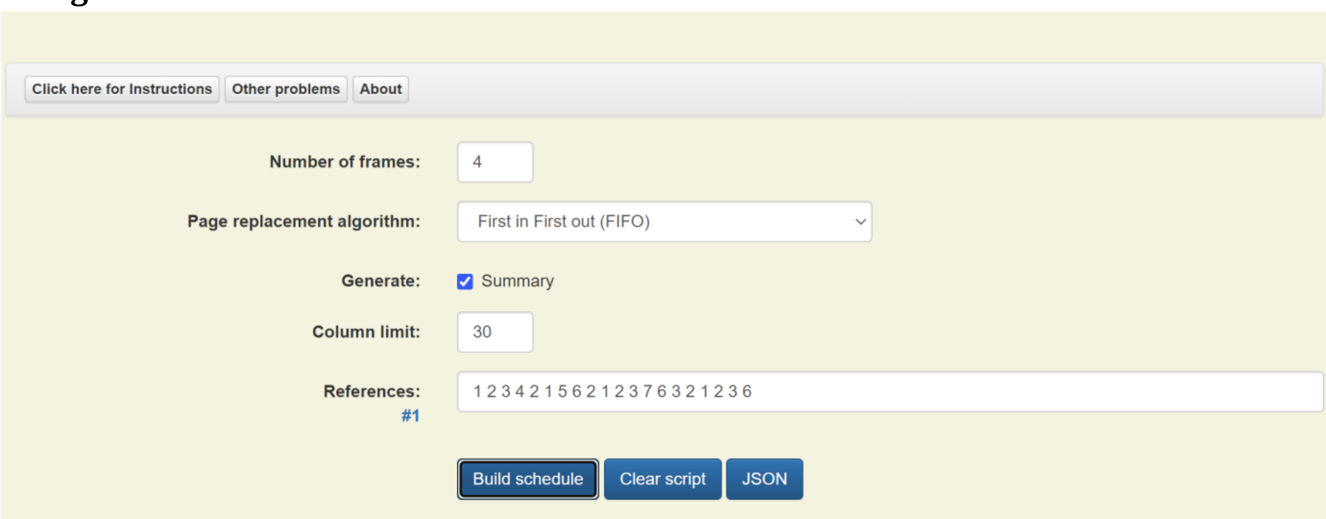
You are supposed to choose any of the six algorithms of Operating System (maximum two from any topic such as from CPU Scheduling Schemes, Process Synchronization, Threads, Memory Management Techniques, or Virtual Memory techniques). Search for the relevant simulation tools that help you to schedule the processes or threads for CPU scheduling scheme, or to implement the memory management, or virtual memory techniques visually (as discussed so far in our regular session).

Once you perform the simulation, you are required to analyze each technique and justify your analysis in your own words in terms of resources utilization, performance, and efficiency.

Page Replacement Algorithm

FIFO Page Replacement Algorithm

Step 1:
Using stimulator



Summary - FIFO algorithm

- Total frames: 4
- Algorithm: FIFO
- Reference string length: 20 references
- String: 1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

Solution visualization

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ref		**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
f		1	2	3	4	4	4	5	6	2	1	1	3	7	6	6	2	1	1	3	3
f			1	2	3	3	3	4	5	6	2	2	1	3	7	7	6	2	2	1	1
f				1	2	2	2	3	4	5	6	6	2	1	3	3	7	6	6	2	2
f					1	1	1	2	3	4	5	5	6	2	1	1	3	7	7	6	6
hit		✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✓	✗	✓
v								1	2	3	4		5	6	2		1	3		7	

- Total references: 20
- Total distinct references: 7
- Hits: 6
- Faults: 14
- **Hit rate:** 6/20 = **30**%
- **Fault rate:** 14/20 = **70**%

Step 2:Analysis

FIFO Page Replacement Algorithm:

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
frame1:	1*	1*	1*	1*			5	5	5	5*		3	3	3		3*	1		1	
frame2:		2	2	2			2*	6	6	6		6*	7	7		7	7*		3	
frame3:			3	3			3	3*	2	2		2	2*	6		6	6		6*	
frame4:				4			4	4	4*	1		1	1	1*		2	2		2	
					H	H					H				H			H		H

Page Hit = 6
Page Miss = Page Fault = 20 - 6 = 14

Page Fault Ratio = 14/20 * 100 = 70%
Page Hit Ration = 6/20 * 100 = 30%

My Analysis of the FIFO Page Replacement Algorithm Implementation

Overview

First In First Out (FIFO) is the simplest page replacement algorithm. The operating system maintains a queue of all pages in memory, with the oldest page at the front. When a page needs to be replaced, the page at the front of the queue is removed.

Analysis and Justification

According to my analysis, the stimulator accurately implements the FIFO page replacement algorithm. It correctly tracks page order and performs replacements by removing the oldest page, as expected. The stimulator also correctly calculates the following :

- **Page Hits**
- **Page Faults**
- **Page Fault Ratio**
- **Page Hit Ratio**

The results from the stimulator match my calculations, confirming the accuracy of its FIFO algorithm implementation.

Metrics Calculation The stimulator provides accurate calculations for key metrics:

- **Page Faults** occur when a page is not found in memory and must be loaded.
- **Page Hits** occur when a requested page is already in memory.
- **Page Fault Ratio** is calculated as:

Page Fault Ratio=Page Faults /Total Page References

- **Page Hit Ratio** is calculated as:

Page Hit Ratio=Page Hits /Total Page Reference

Resource Utilization

The FIFO algorithm optimizes resource utilization by using a straightforward replacement strategy. It ensures that pages are managed in a simple and predictable manner, making it easy to implement and understand.

Performance

The performance of the FIFO algorithm is adequate for systems where simplicity is preferred over efficiency. While it may not always be the most efficient in terms of reducing page faults, its predictable behavior makes it suitable for certain applications.

Efficiency

FIFO is efficient in terms of computational overhead because it uses a simple queue structure to manage pages. However, its efficiency in terms of minimizing page faults may not be as high as more complex algorithms like LRU (Least Recently Used) or LFU (Least Frequently Used).

Conclusion

In conclusion, the stimulator correctly implements the FIFO page replacement algorithm and accurately calculates page hits, page faults, page fault ratio, and page hit ratio. These calculations match my own, confirming the stimulator's accuracy. The FIFO algorithm offers simplicity and predictability, making it efficient in terms of computational overhead, though potentially less efficient in minimizing page faults compared to more complex algorithms. This makes it a reliable choice for systems where ease of implementation and predictability are prioritized.

Page Replacement Algorithm

LRU Page Replacement Algorithm

Step 1:

Using stimulator

Click here for Instructions | Other problems | About

Number of frames: 4

Page replacement algorithm: Least recently used (LRU)

Generate: ☑ Summary

Column limit: 30

References: 1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

#1

Build schedule | Clear script | JSON

Summary - LRU algorithm

- Total frames: 4
- Algorithm: LRU
- Reference string length: 20 references
- String: 1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

Solution visualization

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ref		**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
f		1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
f			1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3
f				1	2	3	4	2	1	5	6	6	1	2	3	7	6	3	3	1	2
f					1	1	3	4	2	1	5	5	6	1	2	2	7	6	6	6	1
hit		✗	✗	✗	✗	✓	✓	✗	✗	✓	✓	✓	✗	✗	✗	✓	✓	✗	✓	✓	✓
v								3	4				5	6	1			7			

- Total references: 20
- Total distinct references: 7
- Hits: 10
- Faults: 10
- **Hit rate:** 10/20 = **50**%
- **Fault rate:** 10/20 = **50**%

Step 2:Analysis

LRU Page Replacement Algorithm:

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
frame1:	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
frame2:		1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3
frame3:			1	2	3	4	2	1	5	6	6	1	2	3	7	6	3	3	1	2
frame4:				1	1	3	4	2	1	5	5	6	1	2	2	7	6	6	6	1
					H	H			H	H	H				H	H		H	H	H

Page Hit = 10
Page Miss = Page Fault = 20 - 10 = 10

Page Fault Ratio = 10/20 * 100 = 50%
Page Hit Ration = 10/20 * 100 = 50%

My Analysis of the LRU Page Replacement Algorithm Implementation

Overview

The Least Recently Used (LRU) page replacement algorithm replaces the page that has not been used for the longest time in memory with a new page. The algorithm operates on the principle that pages not used recently are less likely to be needed soon.

Analysis and Justification

Correct Algorithm Implementation According to my analysis, the stimulator accurately implements the LRU page replacement algorithm. It correctly tracks the usage of pages and replaces the least recently used page, as expected. The stimulator's results match my calculations for the following metrics:

- **Page Hits**
- **Page Faults**
- **Page Fault Ratio**
- **Page Hit Ratio**

These metrics confirm the stimulator's correct implementation of the LRU algorithm.

Metrics Calculation The stimulator accurately calculates the key metrics:

- **Page Faults** occur when a requested page is not in memory and must be loaded.
- **Page Hits** occur when a requested page is already in memory.
- **Page Fault Ratio** is calculated as:

Page Fault Ratio=Page Faults /Total Page References

- **Page Hit Ratio** is calculated as:

Page Hit Ratio=Page Hits /Total Page References

Resource Utilization

The LRU algorithm optimizes resource utilization by effectively managing memory usage.

By prioritizing pages that have been used recently, it reduces the likelihood of replacing pages that are still in active use. This approach ensures that memory is utilized more efficiently, as pages in high demand remain in memory.

Performance

The performance of the LRU algorithm is superior to simpler algorithms like FIFO in terms of reducing page faults. By keeping track of the least recently used pages, LRU provides a more accurate prediction of future page requests, leading to fewer page faults and improved system performance.

Efficiency

LRU is efficient in minimizing page faults, which enhances the overall performance of the system. Although maintaining the LRU order can have some computational overhead, the benefits of reduced page faults generally outweigh these costs. The efficient handling of memory access patterns ensures better performance, particularly in environments with frequent page requests.

Conclusion

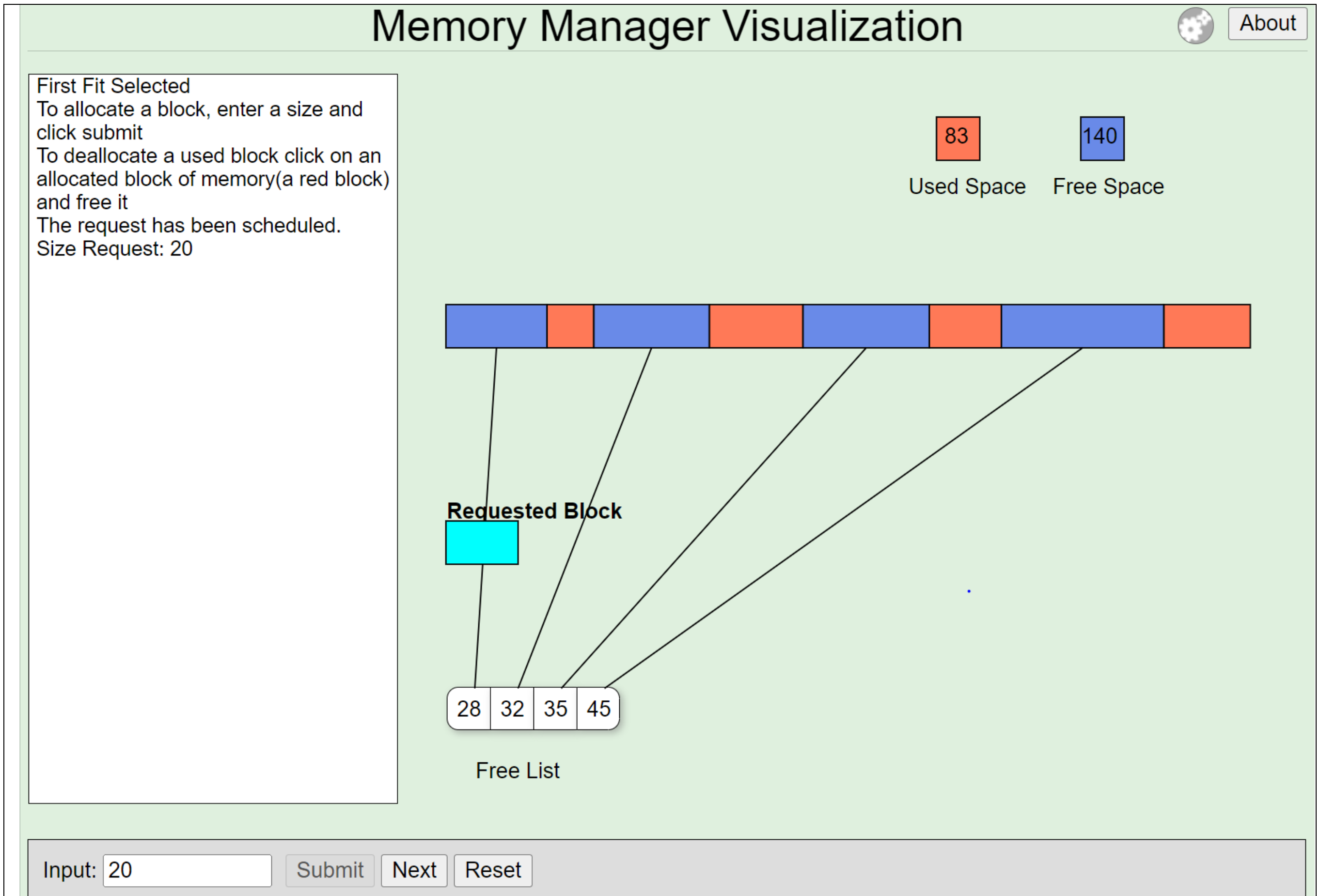
In conclusion, the stimulator correctly implements the LRU page replacement algorithm, accurately calculating page hits, page faults, page fault ratio, and page hit ratio. These calculations match my own, confirming the stimulator's accuracy. The LRU algorithm effectively optimizes resource utilization, enhances performance by reducing page faults, and maintains high efficiency in handling memory access patterns. This makes it a reliable and efficient choice for page replacement in memory management.

Memory Allocation Algorithm

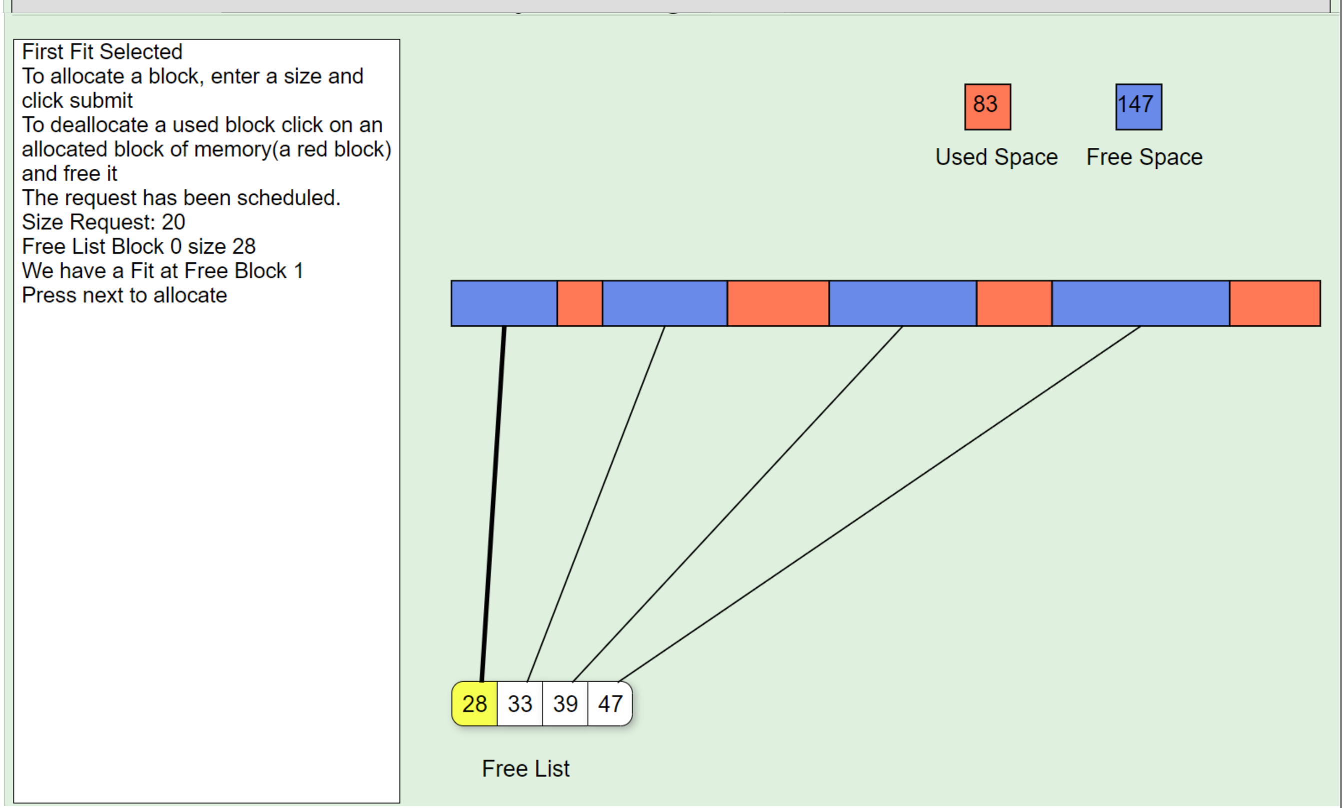
First Fit

Step 1:

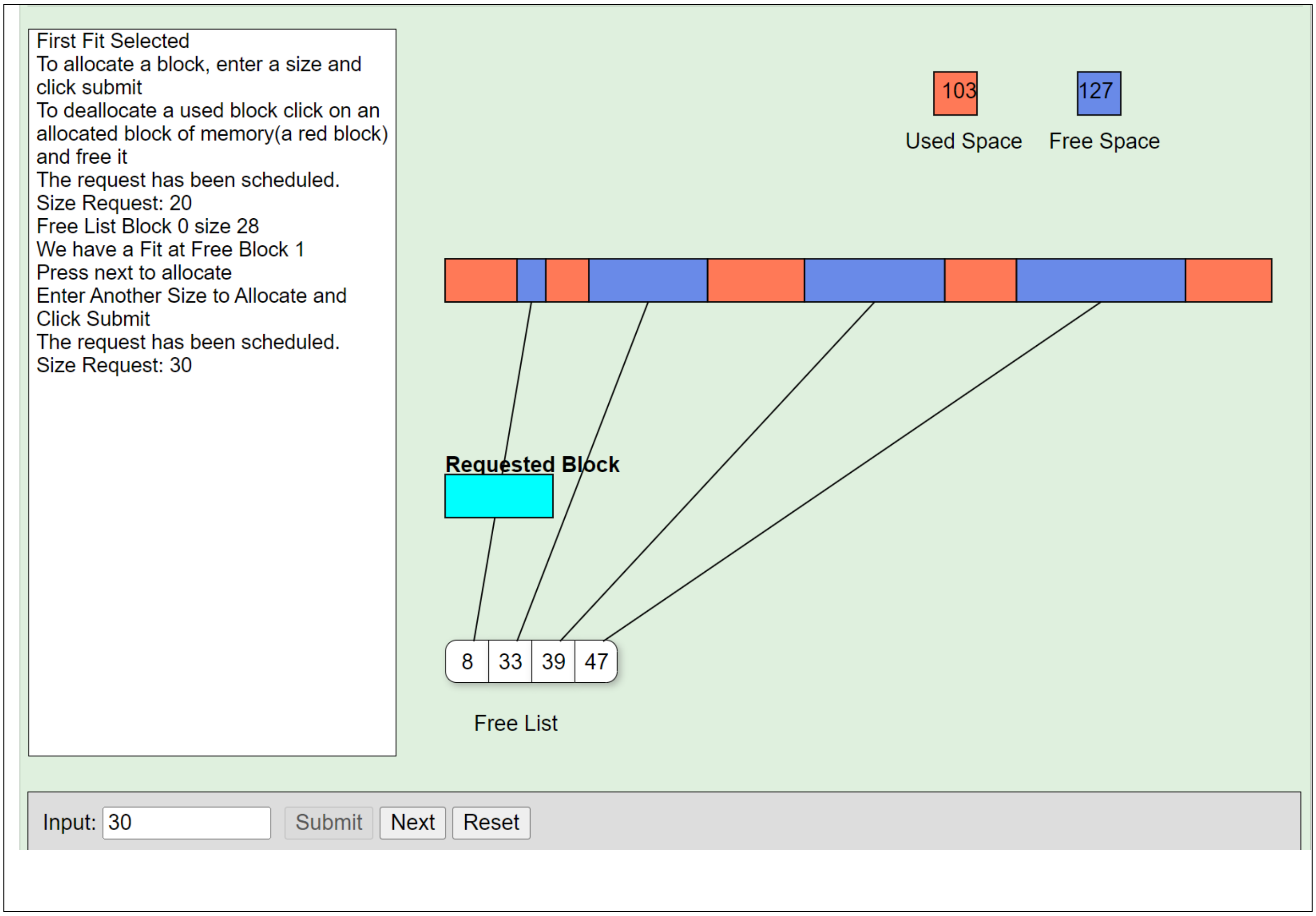
Using stimulator

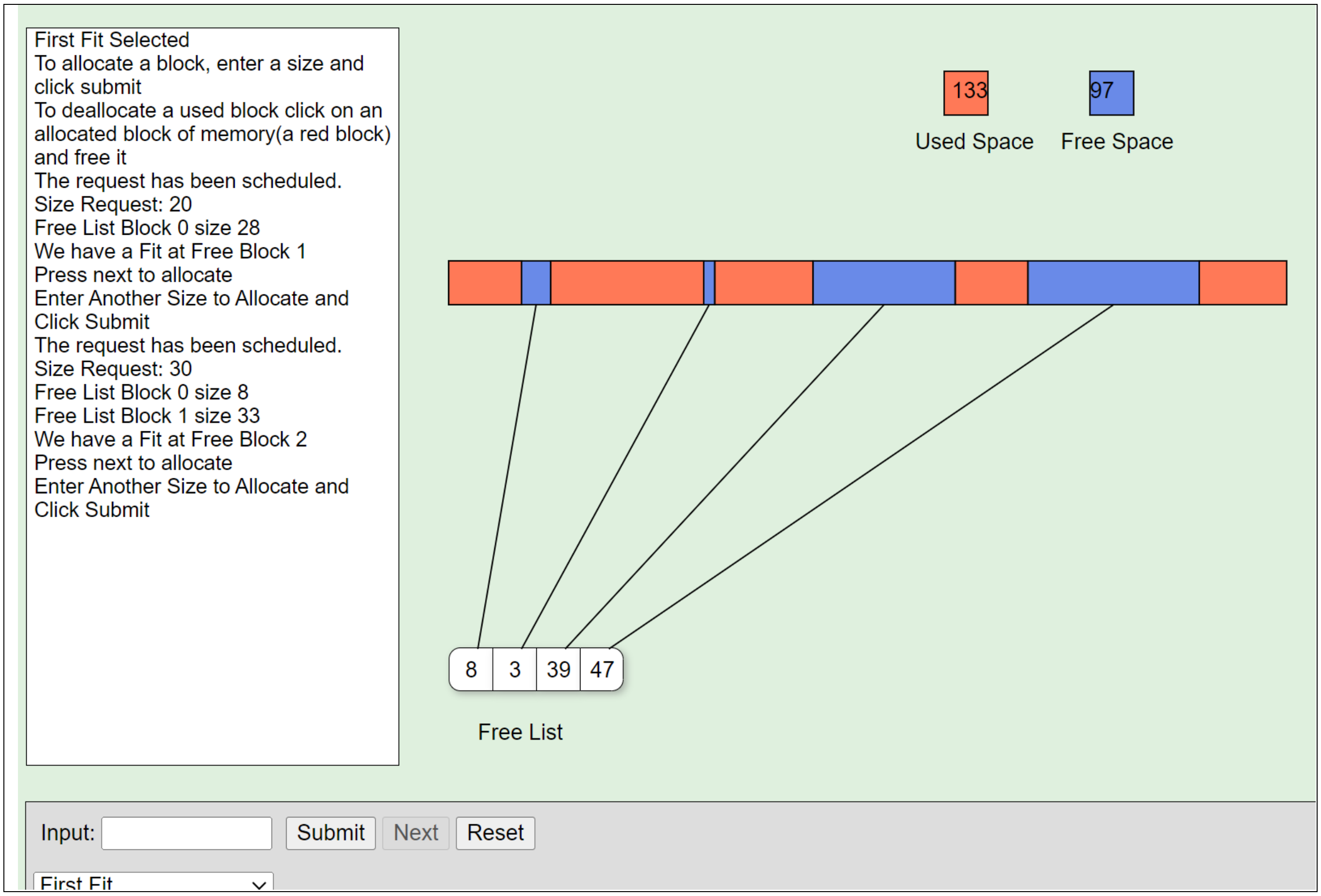


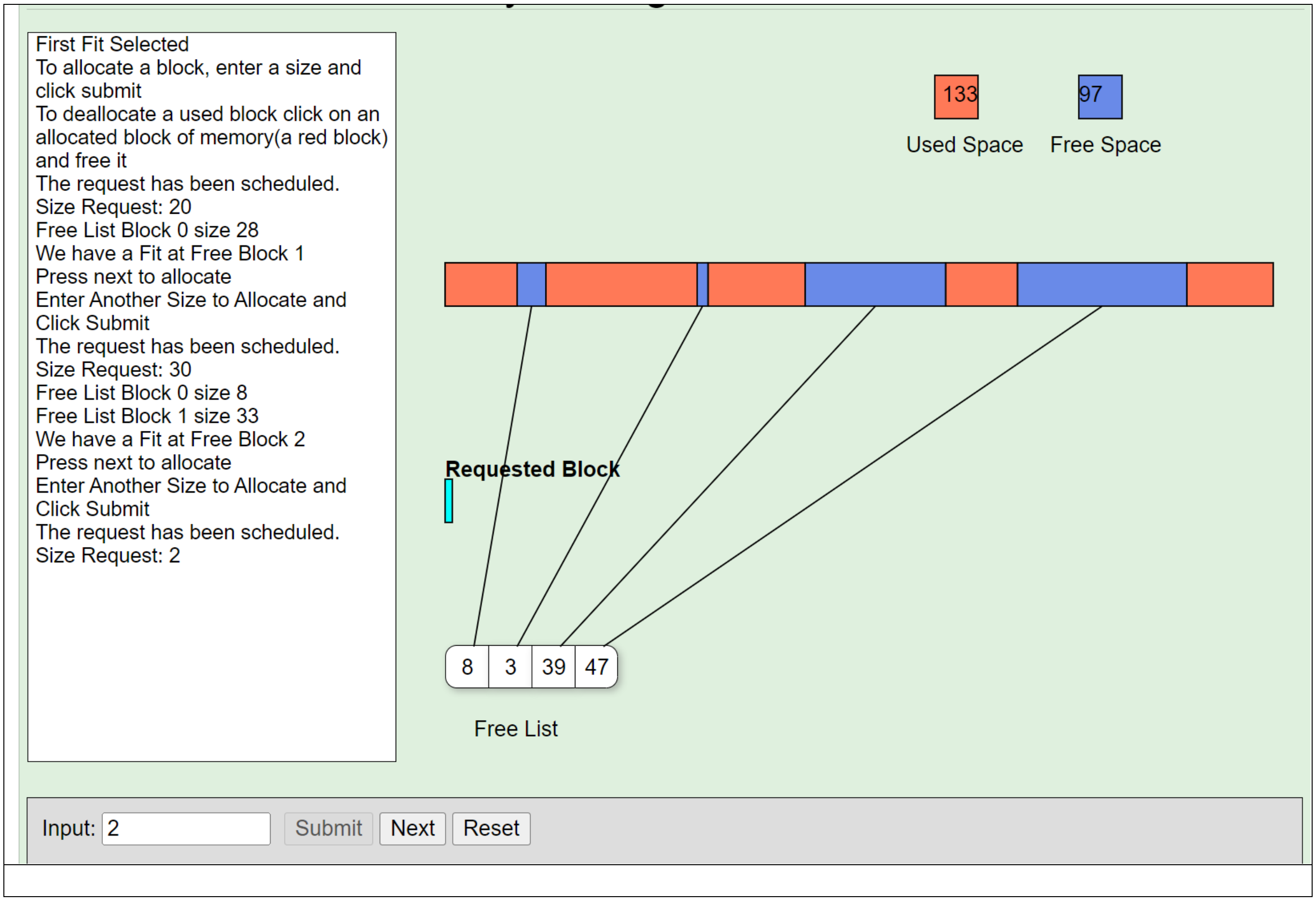
Memory Manager Visualization
About
First Fit Selected
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 20
83
140
Used Space
Free Space
Requested Block
28
32
35
45
Free List
Input: 20
Submit
Next
Reset

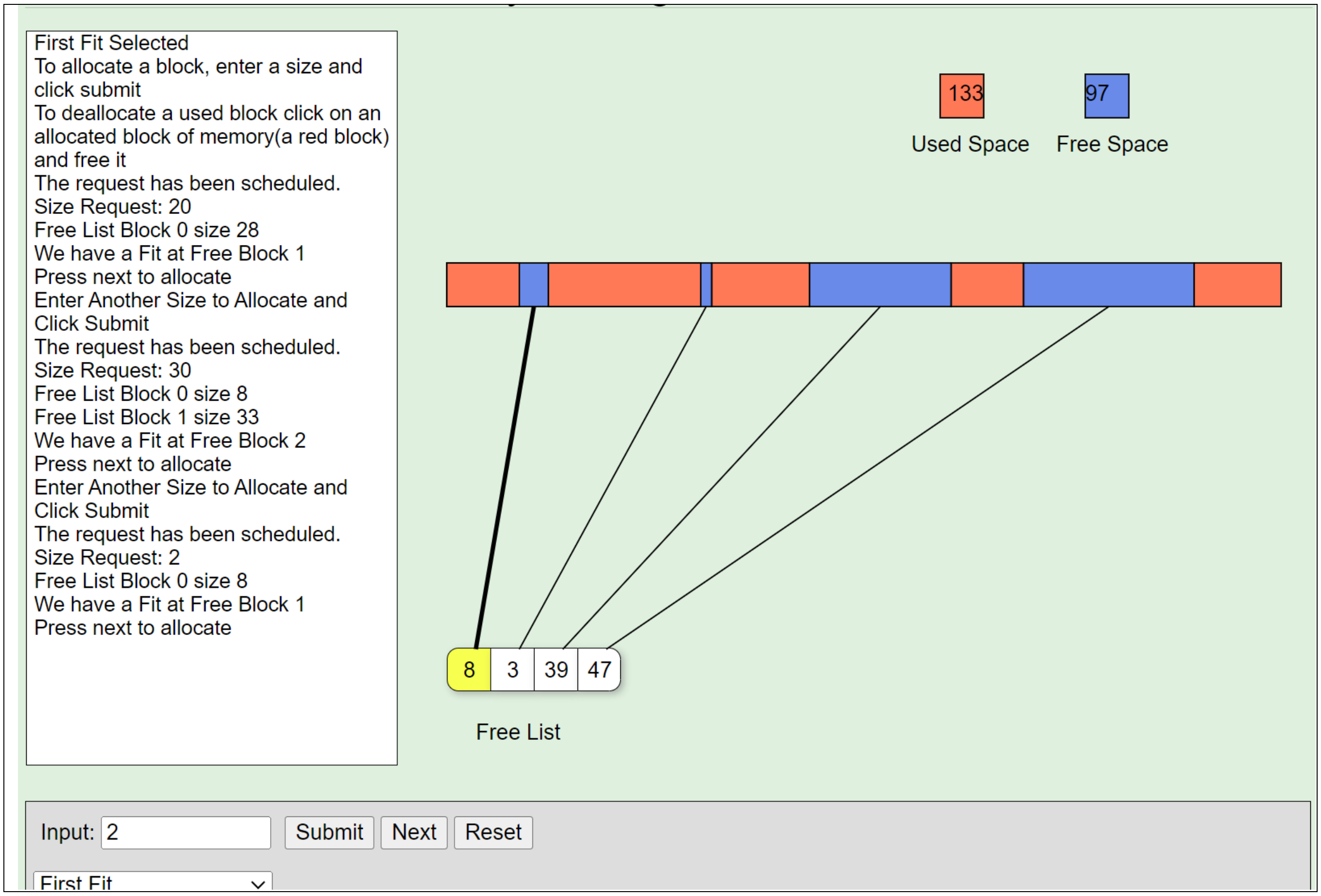


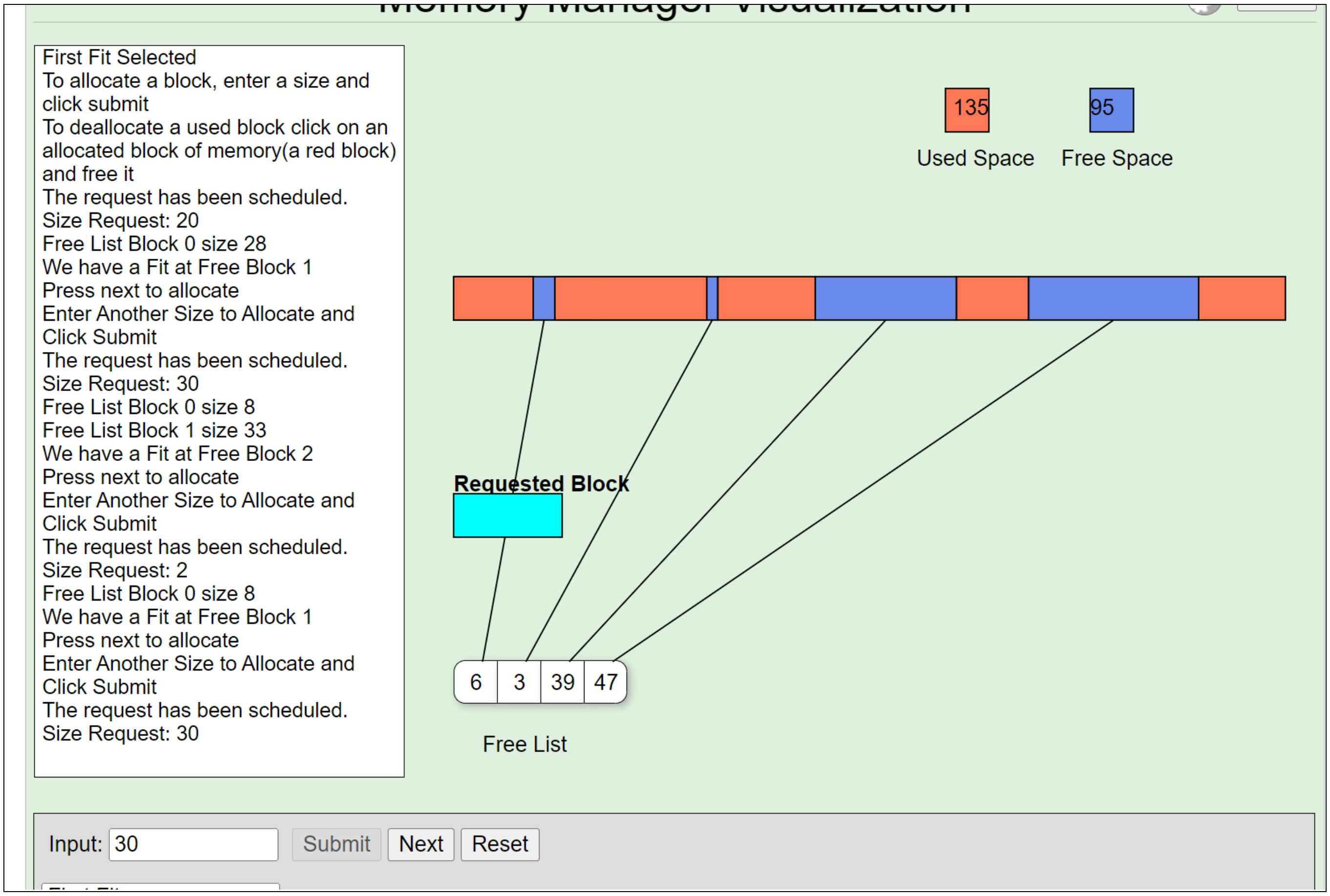
First Fit Selected
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 20
Free List Block 0 size 28
We have a Fit at Free Block 1
Press next to allocate
83
147
Used Space
Free Space
28
33
39
47
Free List

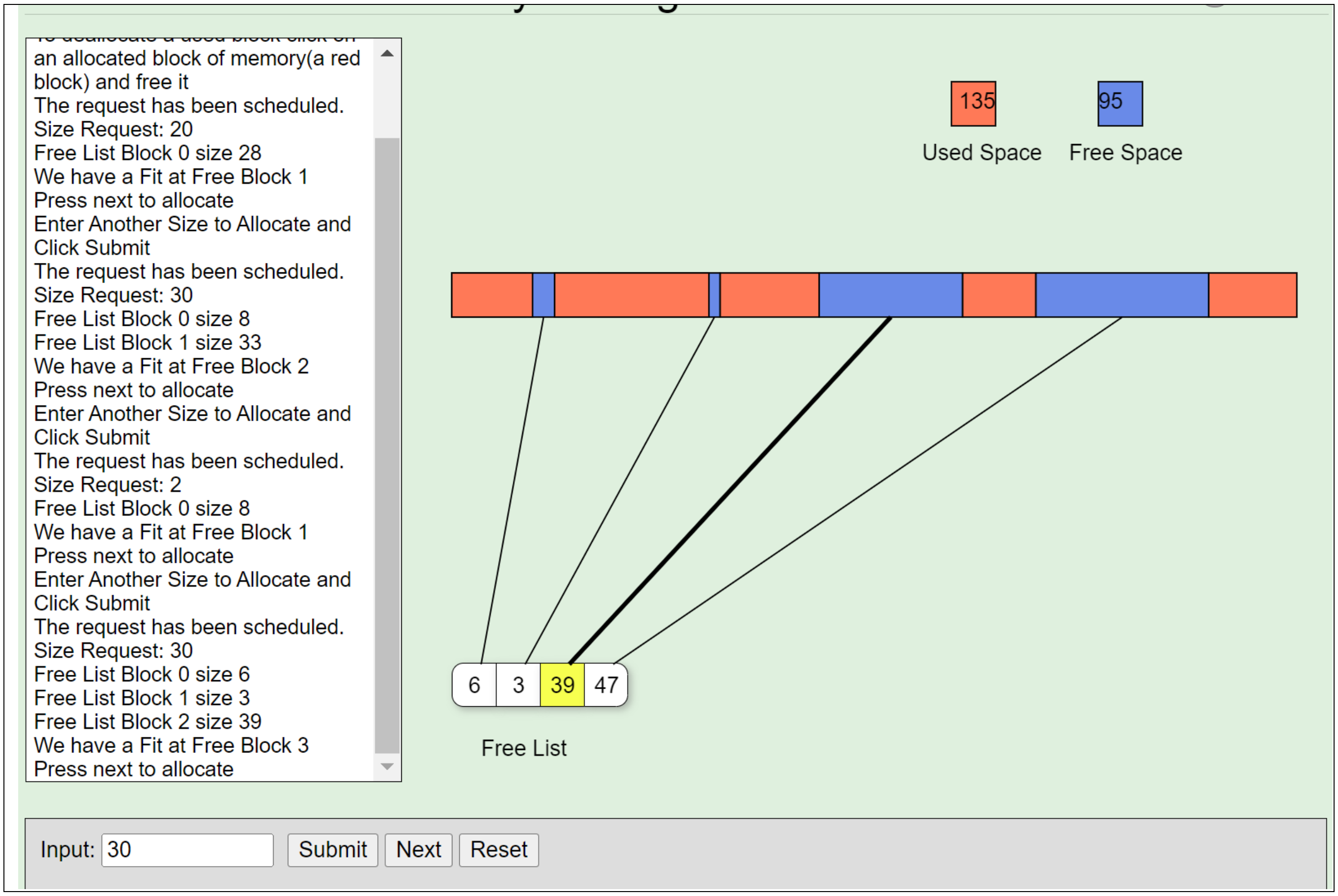

First Fit Selected
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 20
Free List Block 0 size 28
We have a Fit at Free Block 1
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 30
103
127
Used Space
Free Space
Requested Block
8
33
39
47
Free List
Input: 30
Submit
Next
Reset

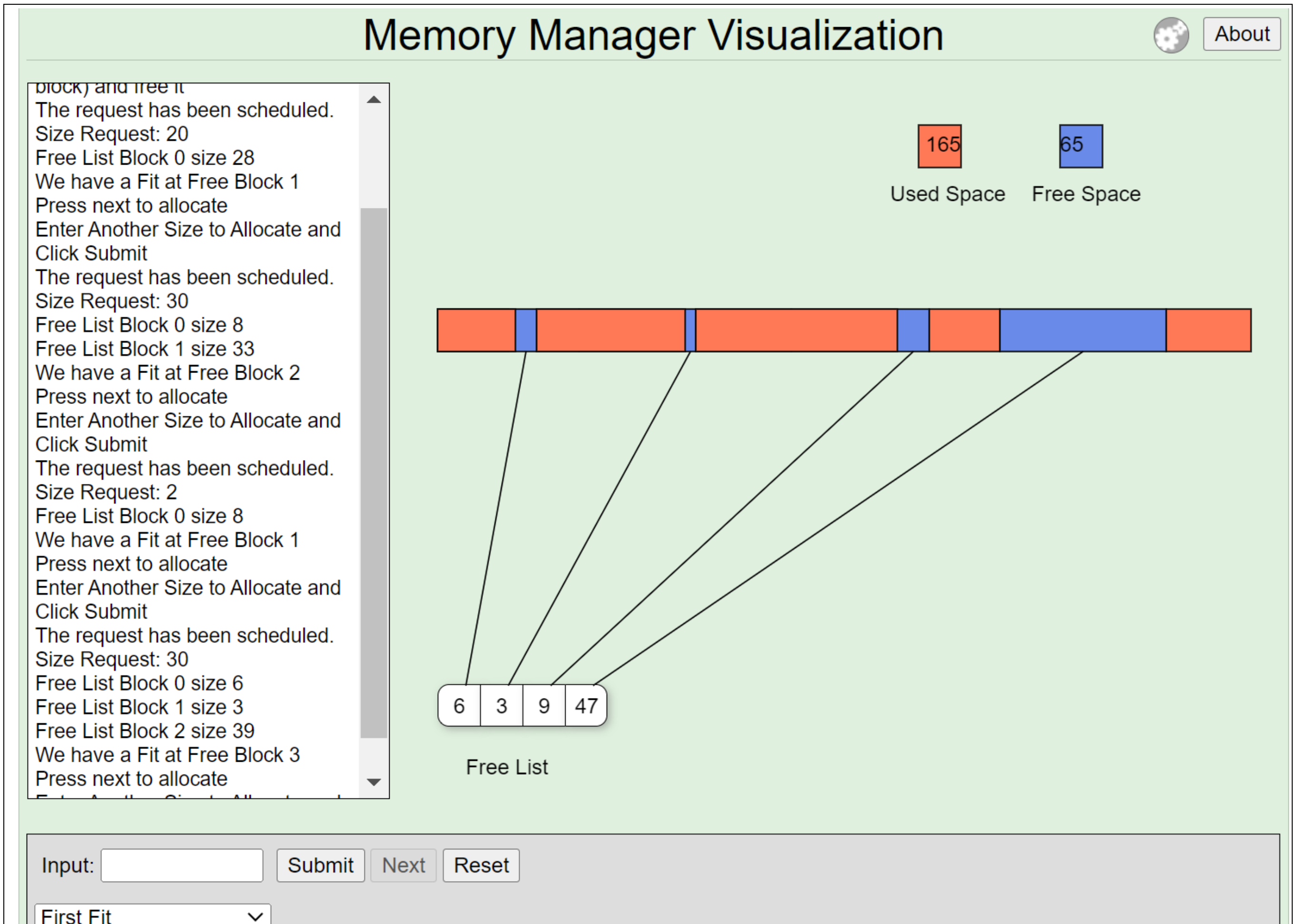

First Fit Selected
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 20
Free List Block 0 size 28
We have a Fit at Free Block 1
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 30
Free List Block 0 size 8
Free List Block 1 size 33
We have a Fit at Free Block 2
Press next to allocate
Enter Another Size to Allocate and Click Submit
133
97
Used Space
Free Space
8
3
39
47
Free List
Input:
Submit
Next
Reset
First Fit


First Fit Selected
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 20
Free List Block 0 size 28
We have a Fit at Free Block 1
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 30
Free List Block 0 size 8
Free List Block 1 size 33
We have a Fit at Free Block 2
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 2
133
97
Used Space
Free Space
Requested Block
8
3
39
47
Free List
Input: 2
Submit
Next
Reset


First Fit Selected
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 20
Free List Block 0 size 28
We have a Fit at Free Block 1
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 30
Free List Block 0 size 8
Free List Block 1 size 33
We have a Fit at Free Block 2
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 2
Free List Block 0 size 8
We have a Fit at Free Block 1
Press next to allocate
133
Used Space
97
Free Space
8
3
39
47
Free List
Input: 2
Submit
Next
Reset
First Fit


First Fit Selected
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 20
Free List Block 0 size 28
We have a Fit at Free Block 1
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 30
Free List Block 0 size 8
Free List Block 1 size 33
We have a Fit at Free Block 2
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 2
Free List Block 0 size 8
We have a Fit at Free Block 1
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 30
135
Used Space
95
Free Space
Requested Block
6
3
39
47
Free List
Input: 30
Submit
Next
Reset


an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 20
Free List Block 0 size 28
We have a Fit at Free Block 1
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 30
Free List Block 0 size 8
Free List Block 1 size 33
We have a Fit at Free Block 2
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 2
Free List Block 0 size 8
We have a Fit at Free Block 1
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 30
Free List Block 0 size 6
Free List Block 1 size 3
Free List Block 2 size 39
We have a Fit at Free Block 3
Press next to allocate
135
Used Space
95
Free Space
6
3
39
47
Free List
Input: 30
Submit
Next
Reset



Step 2:Analysis

First Fit FF

Memory

required memory	28Kb	32Kb	39Kb	47Kb
P1 20Kb	8Kb			
P2 30Kb		3Kb		
P3 2Kb	6Kb			
P4 30Kb			9Kb	

My Analysis of the First-Fit Memory Allocation Algorithm Implementation

Overview

First-Fit Memory Allocation is a method that keeps a list of free and busy memory blocks organized by memory location, from low to high order. In this method, each job claims the first available memory block that is large enough to accommodate its size.

Analysis and Justification

Correct Algorithm Implementation According to my analysis, the stimulator accurately implements the First-Fit memory allocation algorithm. It properly searches through the list of memory blocks and allocates the first block that is sufficiently large to hold the incoming job. This matches my calculations and expectations for the following aspects:

- **Memory Allocation**
- **Memory Utilization**

These confirmations ensure that the First-Fit algorithm is correctly applied.

Metrics and Calculations The stimulator efficiently calculates key metrics for memory allocation:

- **Memory Allocation**: It correctly assigns jobs to the first suitable memory block, ensuring that each job is placed in the lowest available address that fits.
- **Memory Utilization**: It maximizes the use of available memory by filling the earliest available blocks first, reducing fragmentation over time.

Resource Utilization

The First-Fit algorithm optimizes resource utilization by quickly allocating memory to incoming jobs. By choosing the first suitable block, it minimizes the time spent searching for available memory, which speeds up the allocation process and makes efficient use of the system's memory resources.

Performance

The performance of the First-Fit algorithm is notable for its simplicity and speed in allocation. Since it only requires finding the first available block that fits, the allocation process is generally fast. This makes it a suitable choice for environments where quick allocation decisions are necessary.

Efficiency

First-Fit is efficient in terms of computational overhead, as it minimizes the search time for available memory blocks. However, it may lead to external fragmentation over time, as small gaps may be left between allocated blocks. Despite this, its simplicity and speed often outweigh the potential inefficiencies due to fragmentation.

Conclusion

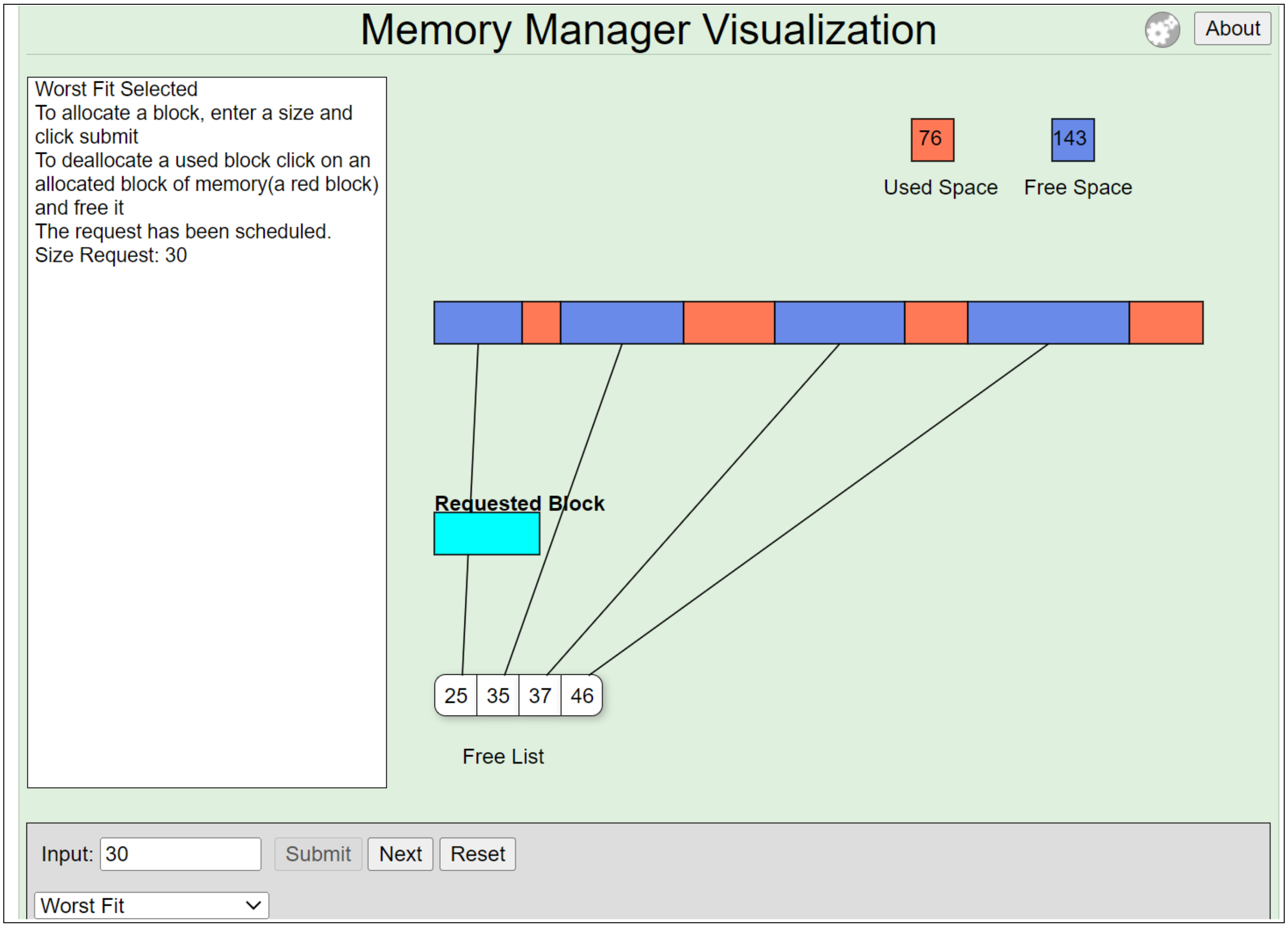
In conclusion, the stimulator accurately implements the First-Fit memory allocation algorithm, correctly allocating memory and calculating metrics that match my own analysis. The First-Fit method ensures efficient resource utilization by quickly allocating the first available memory block that fits the job's size. Its performance is characterized by fast allocation decisions, and its simplicity contributes to its efficiency despite potential fragmentation issues. This makes the First-Fit algorithm a reliable and straightforward choice for memory allocation in many computing environments.

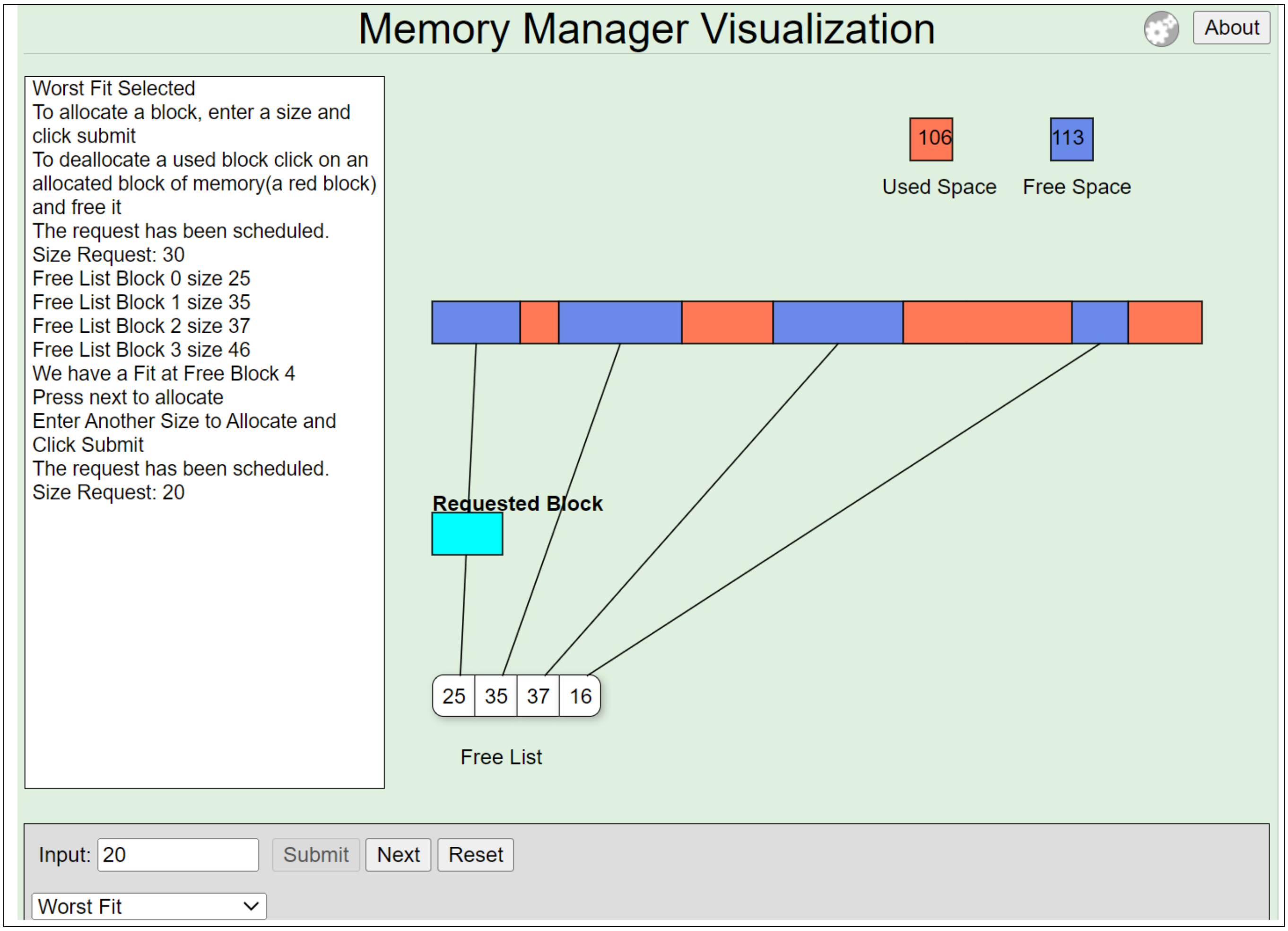
Memory Allocation Algorithm

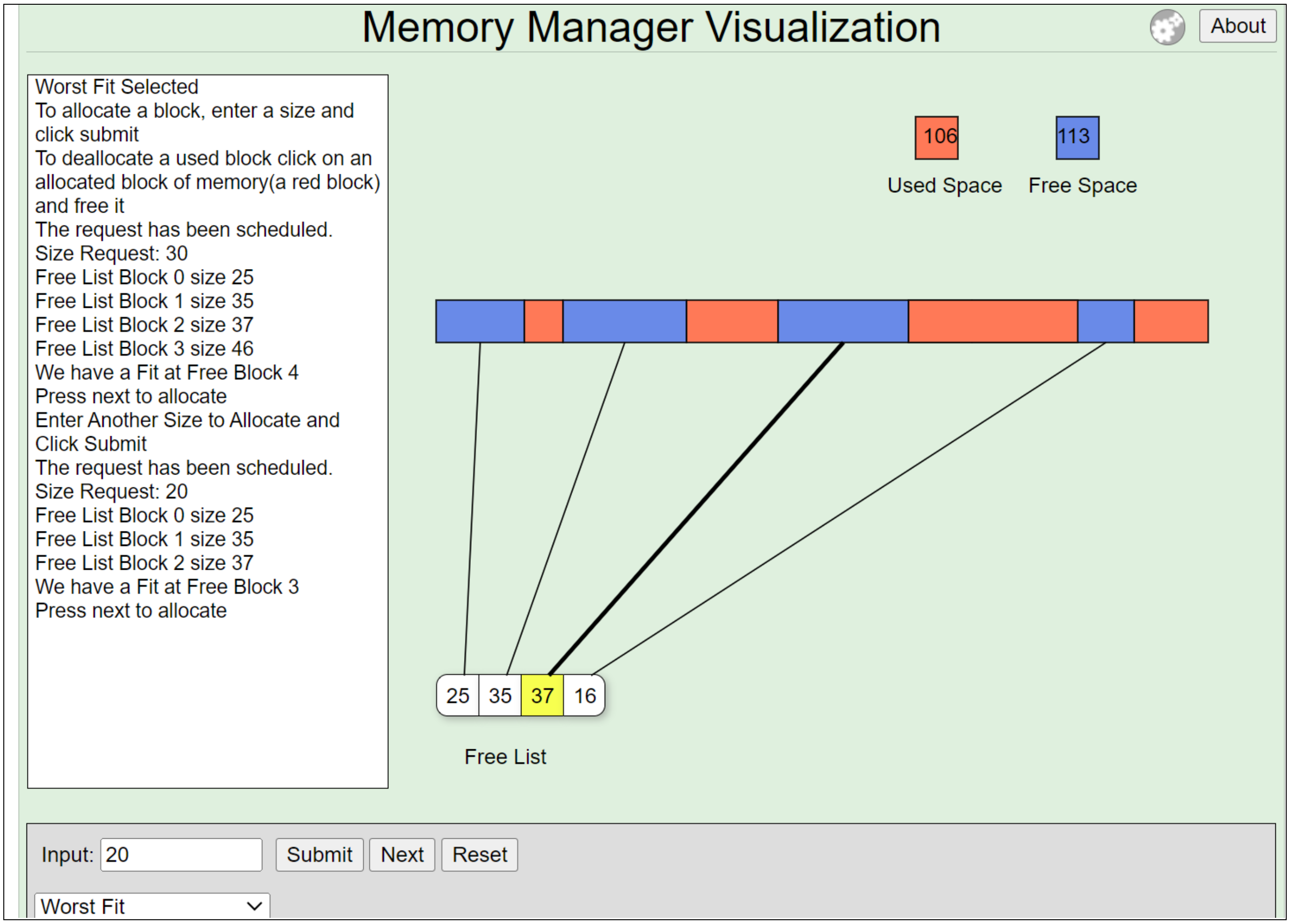
Worst Fit

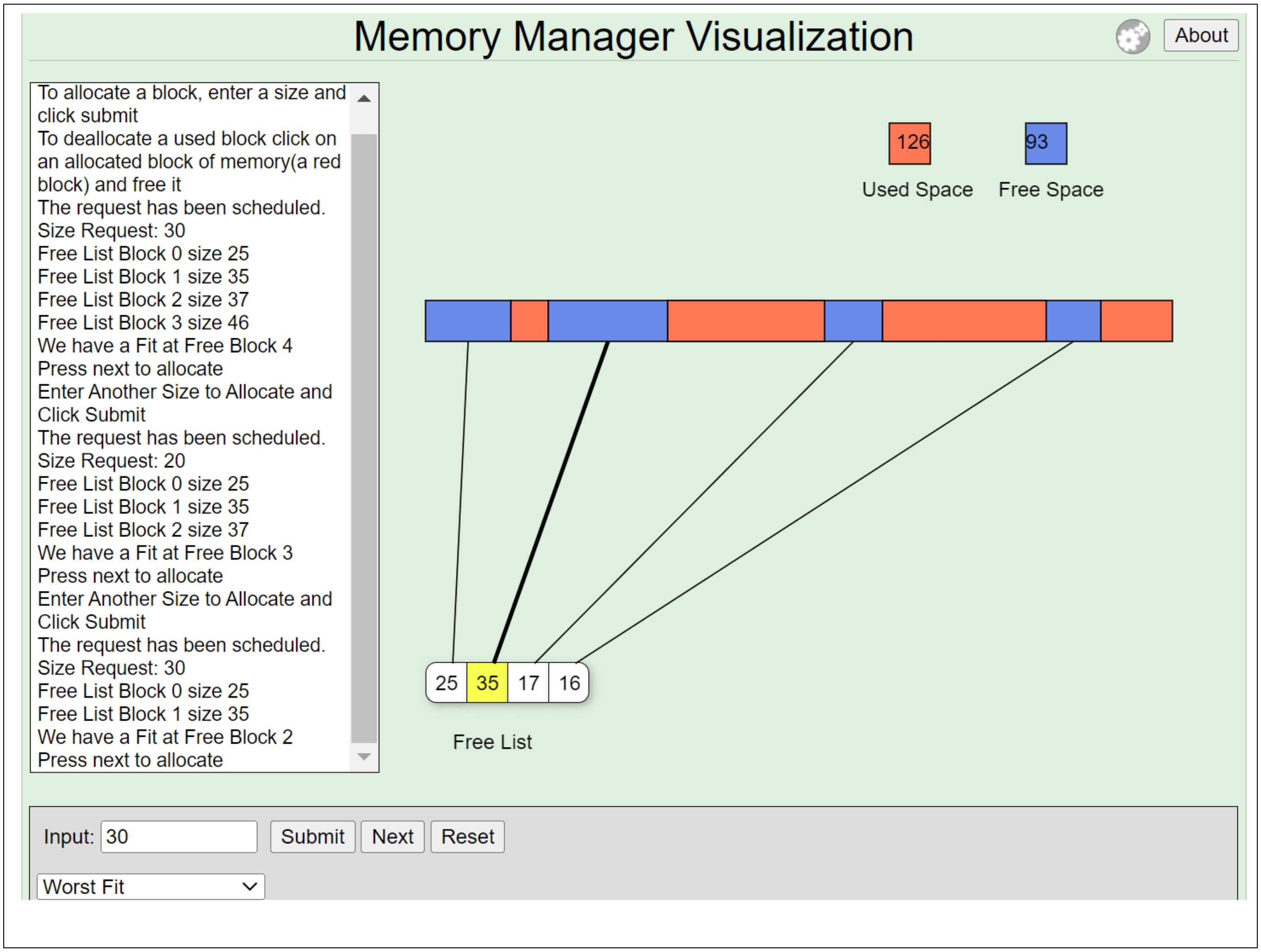
Step 1:

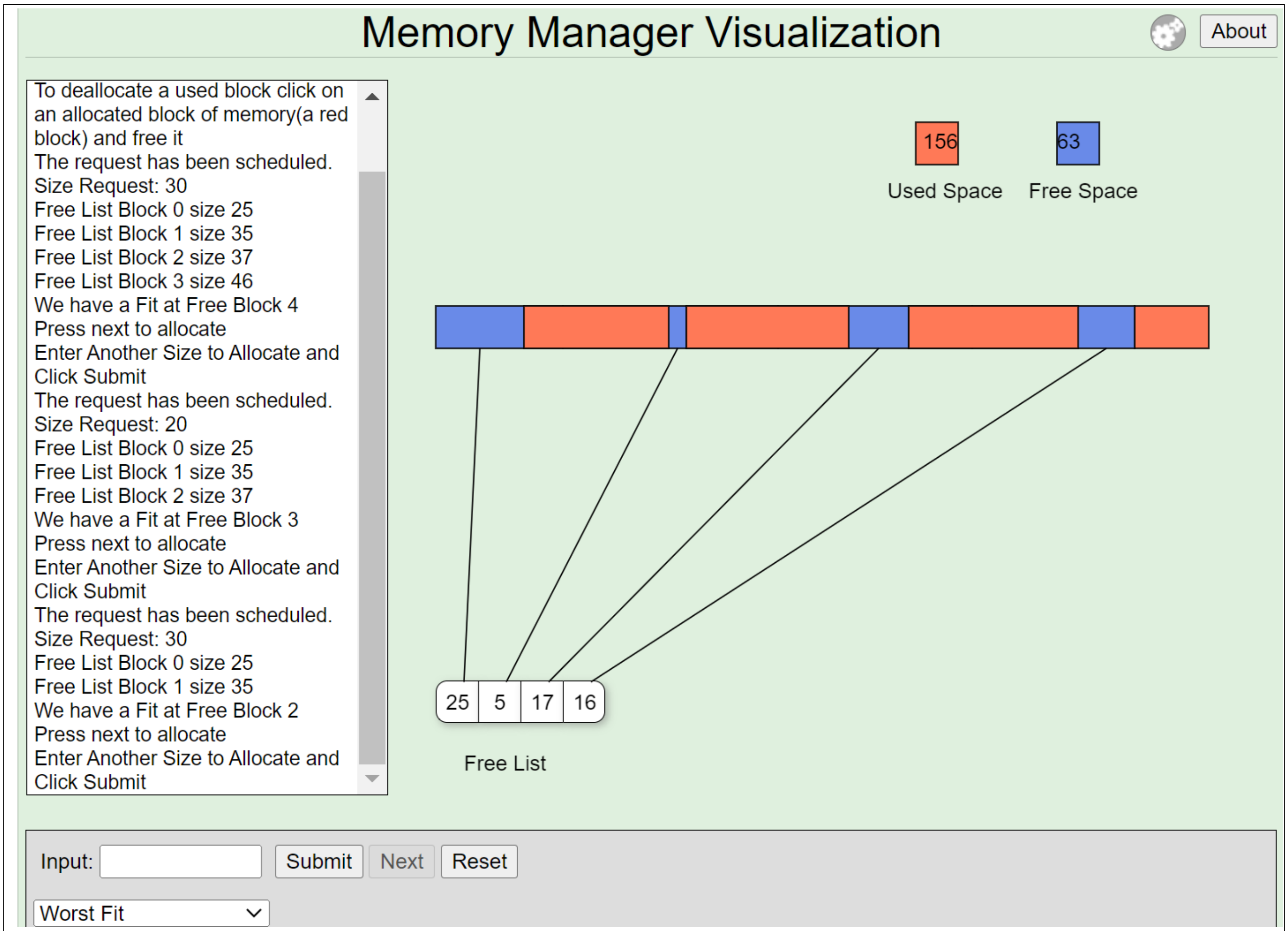
Using stimulator


Memory Manager Visualization
About
Worst Fit Selected
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 30
76
Used Space
143
Free Space
Requested Block
25
35
37
46
Free List
Input: 30
Submit
Next
Reset
Worst Fit


Memory Manager Visualization
About
Worst Fit Selected
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 30
Free List Block 0 size 25
Free List Block 1 size 35
Free List Block 2 size 37
Free List Block 3 size 46
We have a Fit at Free Block 4
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 20
106
Used Space
113
Free Space
Requested Block
25
35
37
16
Free List
Input: 20
Submit
Next
Reset
Worst Fit


Memory Manager Visualization
About
Worst Fit Selected
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 30
Free List Block 0 size 25
Free List Block 1 size 35
Free List Block 2 size 37
Free List Block 3 size 46
We have a Fit at Free Block 4
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 20
Free List Block 0 size 25
Free List Block 1 size 35
Free List Block 2 size 37
We have a Fit at Free Block 3
Press next to allocate
106
Used Space
113
Free Space
25
35
37
16
Free List
Input: 20
Submit
Next
Reset
Worst Fit


Memory Manager Visualization
About
To allocate a block, enter a size and click submit
To deallocate a used block click on an allocated block of memory(a red block) and free it
The request has been scheduled.
Size Request: 30
Free List Block 0 size 25
Free List Block 1 size 35
Free List Block 2 size 37
Free List Block 3 size 46
We have a Fit at Free Block 4
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 20
Free List Block 0 size 25
Free List Block 1 size 35
Free List Block 2 size 37
We have a Fit at Free Block 3
Press next to allocate
Enter Another Size to Allocate and Click Submit
The request has been scheduled.
Size Request: 30
Free List Block 0 size 25
Free List Block 1 size 35
We have a Fit at Free Block 2
Press next to allocate
126
Used Space
93
Free Space
25
35
17
16
Free List
Input: 30
Submit
Next
Reset
Worst Fit



Step 2:Analysis

wrost fit

WF

	memory / required memory	25 Kb	35Kb	37Kb	46Kb
P_1	30Kb				16Kb
P_2	20Kb			17Kb	
P_3	30Kb		5Kb		

My Analysis of the Worst-Fit Memory Allocation Algorithm Implementation
Overview

Worst-Fit Memory Allocation is a technique where the system traverses the entire memory to find the largest available hole/partition and places the process in that partition. This method assumes that the largest hole will leave a substantial portion of the memory unfragmented for future allocations. However, it is a slow process because it requires searching through the entire memory.

Analysis

Correct Algorithm Implementation According to my analysis, the stimulator accurately implements the Worst-Fit memory allocation algorithm. It correctly identifies the largest available memory block and allocates it to the incoming process. This matches my calculations and expectations, confirming the stimulator's correctness.

The stimulator effectively handles key metrics associated with memory allocation:

- **Memory Allocation**: Each job is assigned to the largest available memory block, ensuring that the most substantial partition is used first.
- **Memory Utilization**: By using the largest available hole, the algorithm attempts to leave larger contiguous free spaces, potentially reducing fragmentation for future allocations.

Resource Utilization

The Worst-Fit algorithm optimizes resource utilization by ensuring that the largest partitions are used first, which may help in keeping larger free blocks available for future allocations. This can be beneficial in scenarios where memory requests vary significantly in size.

Performance

The performance of the Worst-Fit algorithm can be slower compared to other algorithms like First-Fit or Best-Fit because it requires a complete traversal of the memory to find the largest hole. This increases the allocation time, particularly in systems with a large number of memory partitions.

Efficiency

While Worst-Fit may lead to more efficient use of larger memory blocks, its need to traverse the entire memory makes it less efficient in terms of allocation speed. However, it can be more effective in reducing fragmentation compared to other algorithms by keeping large free blocks available.

Conclusion

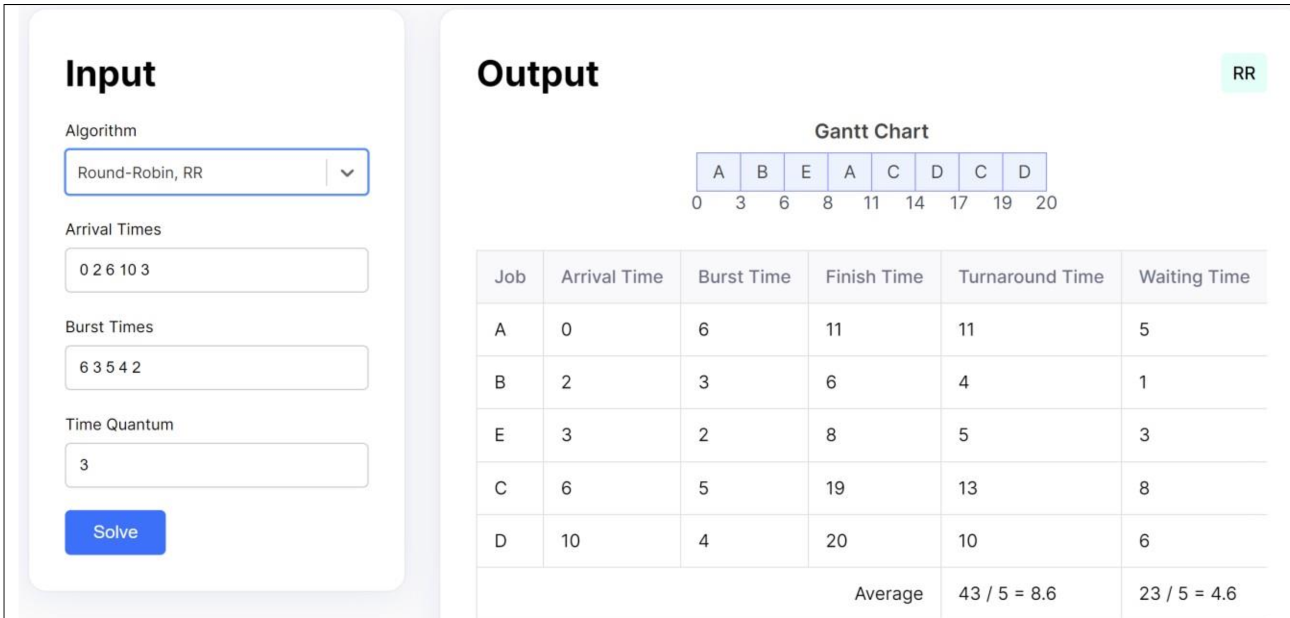
In conclusion, the stimulator correctly implements the Worst-Fit memory allocation algorithm, accurately allocating memory to the largest available block and correctly calculating relevant metrics. The Worst-Fit method ensures that the largest memory blocks are utilized first, potentially reducing fragmentation for future allocations. While this method can optimize resource utilization, its performance is slower due to the need to traverse the entire memory

CPU Scheduling Schemes

Round-Robin,RR

Step 1:

Using stimulator



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	6	11	11	5
B	2	3	6	4	1
E	3	2	8	5	3
C	6	5	19	13	8
D	10	4	20	10	6
			Average	43 / 5 = 8.6	23 / 5 = 4.6

Step 2: Analysis

[LPT – PET – AT]

$$\text{Avg. W. T.} = \frac{(14-3-0)+(3-0-2)+(17-3-6)+(19-3-10)+(6-0-3)}{5} = \frac{29}{5} = 5.8$$

[CT – AT]

$$\text{Avg. T. A. T} = \frac{(17-0)+(6-2)+(19-6)+(20-10)+(8-3)}{5} = \frac{49}{5} = 9.8$$

$$\text{Throughput} = 5/20 = 0.25$$

My Analysis of the Round Robin CPU Scheduling Algorithm Implementation Overview

Round Robin is a CPU scheduling algorithm where each process is cyclically assigned a fixed time slot (quantum)..

Analysis and Justification

Correct Gantt Chart Generation According to my analysis, the stimulator correctly generates the Gantt chart. This chart visually represents the order in which processes are executed and their time slices, ensuring that each process gets its fair share of CPU time in a cyclic manner.

Issues in Waiting and Turnaround Time Calculation

However, the stimulator does not subtract the arrival time when calculating waiting time and turnaround time. Correct calculations should consider the arrival time to accurately reflect the time each process spends waiting and the total time from arrival to completion:

- **Waiting Time should be calculated as:**

Waiting Time=Turnaround Time−Burst TimeWaiting

- **Turnaround Time should be:**

Turnaround Time=Completion Time−Arrival

This ensures that the waiting and turnaround times reflect the actual performance experienced by each process.

Throughput Calculation Additionally, the stimulator does not calculate throughput, which is a key performance metric. Throughput measures the number of processes completed per unit time and is calculated as:

Throughput=Total Number of Processes/Total Time Taken

Including this metric would provide a more comprehensive view of the system's performance.

Resource Utilization

The Round Robin algorithm ensures fair CPU time distribution among all processes, optimizing CPU utilization. This fair allocation helps in maintaining balanced resource utilization across all processes.

Performance

The algorithm provides good performance in terms of response time, as each process gets CPU time in a cyclic manner. However, the accuracy of waiting time and turnaround time calculations is crucial for evaluating the system's performance correctly. Including throughput would also enhance performance assessment.

Efficiency

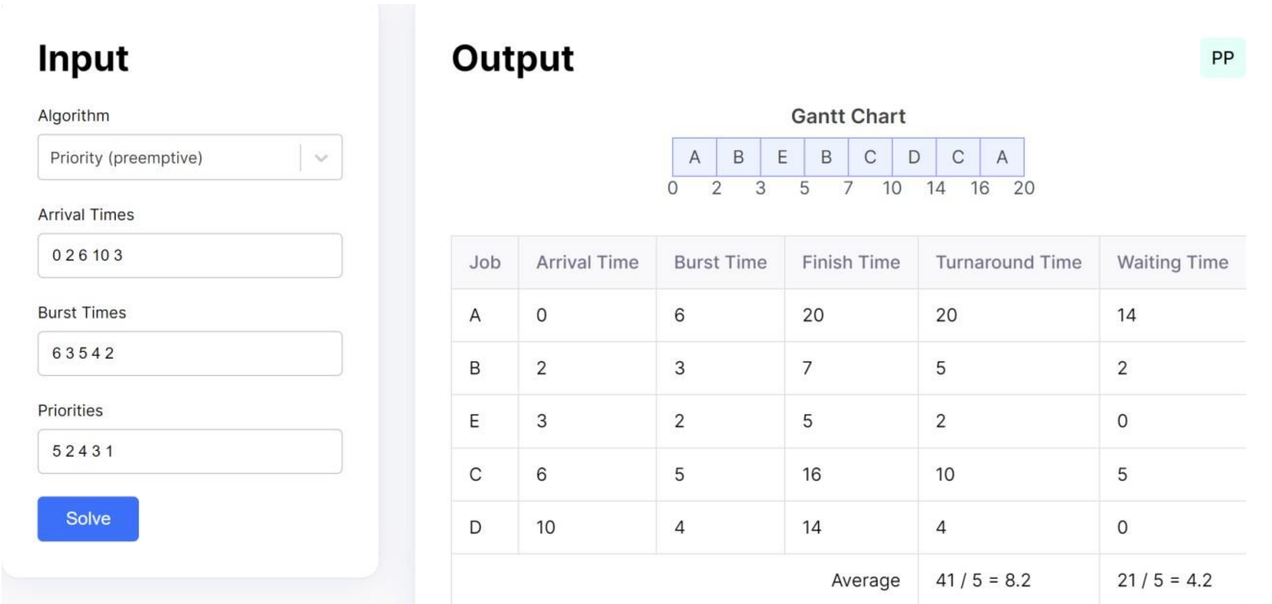
Round Robin is efficient for time-sharing systems as it ensures that all processes receive attention within a fixed time quantum. Correctly calculating waiting and turnaround times, along with throughput, would provide a clearer picture of system efficiency and help in identifying any potential bottlenecks.

Conclusion

In conclusion, while the stimulator accurately generates the Gantt chart for the Round Robin CPU scheduling algorithm, it needs to correct the calculation of waiting and turnaround times by considering the arrival time. Additionally, incorporating throughput calculation would provide a more complete evaluation of system performance. These adjustments will enhance the analysis in terms of resource utilization, performance, and efficiency, making the Round Robin algorithm implementation more effective and reliable.

CPU Scheduling Schemes
Priority

Step 1:
Using stimulator



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	6	20	20	14
B	2	3	7	5	2
E	3	2	5	2	0
C	6	5	16	10	5
D	10	4	14	4	0
			Average	41 / 5 = 8.2	21 / 5 = 4.2

Step 2:Analysis

[LPT – PET – AT]

$$\text{Avg. W. T.} = \frac{(16-2-0)+(5-1-2)+(14-3-6)+(10-0-10)+(3-0-3)}{5} = \frac{21}{5} = 4.2$$

[CT – AT]

$$\text{Avg. T. A. T} = \frac{(20-0)+(7-2)+(16-6)+(14-10)+(5-3)}{5} = \frac{41}{5} = 8.2$$

$$\text{Throughput} = 5/20 = 0.25$$

My Analysis of the Priority Scheduling Algorithm Implementation

Overview

Priority scheduling in operating systems is a scheduling algorithm that schedules processes based on the priority assigned to each process. Higher-priority processes are executed before lower-priority ones.

Analysis and Justification

Correct Gantt Chart Generation According to my analysis, the stimulator correctly generates the Gantt chart, accurately reflecting the order in which processes are executed based on their priority. This visual representation ensures that the higher-priority processes are given precedence in the CPU scheduling.

Issues in Waiting and Turnaround Time Calculation However, the stimulator does not subtract the arrival time when calculating waiting time and turnaround time. Correct calculations should account for arrival time to accurately measure the performance:

- **Waiting Time** should be calculated as:

Waiting Time=Turnaround Time−Burst TimeWaiting Time=Turnaround Time−Burst Time

Here, Turnaround Time should already account for arrival time.

- **Turnaround Time** should be:

Turnaround Time=Completion Time−Arrival TimeTurnaround Time=Completion Time−Arrival Time

This adjustment ensures that both metrics accurately reflect the actual performance experienced by each process.

Throughput Calculation Additionally, the stimulator does not calculate throughput, which is an important performance metric. Throughput measures the number of processes completed per unit time and can be calculated as:

Throughput=Total Number of ProcessesTotal Time TakenThroughput=Total Time TakenTotal Number of Processes

Including this metric would provide a more comprehensive view of the system's performance.

Resource Utilization

The Priority scheduling algorithm ensures that critical processes are given priority, optimizing resource utilization by ensuring that important tasks are completed first. This prioritization helps in maximizing the effective use of CPU resources.

Performance

The algorithm performs well in terms of prioritizing critical tasks, ensuring that high-priority processes are executed promptly. However, the accuracy of waiting time and turnaround time calculations is crucial for evaluating the system's overall performance. Including throughput would enhance the assessment of how effectively the system handles process execution.

Efficiency

Priority scheduling is efficient in handling processes based on their importance. Correctly calculating waiting and turnaround times, along with throughput, would provide a clearer picture of the system’s efficiency. This would help in identifying any potential delays or inefficiencies in processing lower-priority tasks.

Conclusion

In conclusion, while the stimulator accurately generates the Gantt chart for the Priority Scheduling algorithm, it needs to correct the calculation of waiting and turnaround times by considering the arrival time. Additionally, incorporating throughput calculation would provide a more complete evaluation of system performance. These adjustments will enhance the analysis in terms of resource utilization, performance, and efficiency, making the Priority Scheduling algorithm implementation more effective and reliable

BANKER'S ALGORITHM

Step 1:

Using stimulator

All Bookmarl

No. of instances of A: No. of instances of B: No. of instances of C: Sample Example Find available Find need Find process sequence Reset

	Allocation			Max			Need		
	A	B	C	A	B	C	A	B	C
P1	0	0	1	0	0	1	0	0	0
P2	1	0	0	1	8	6	0	8	6
P3	1	3	5	2	3	5	1	0	0
P4	0	6	2	0	6	5	0	0	3
P5	0	0	1	0	6	5	0	6	4

Available		
A	B	C
2	6	2

Safe Process Sequence :

Operations: Calculating the Need Matrix....
Need[n][n] = Max[n][n] - Allocation[n][n]
Need[0][0] = 0 - 0 = 0 Need[0][1] = 0 - 0 = 0 Need[0][2] = 1 - 1 = 0
Need[1][0] = 1 - 1 = 0 Need[1][1] = 8 - 0 = 8 Need[1][2] = 6 - 0 = 6
Need[2][0] = 2 - 1 = 1 Need[2][1] = 3 - 3 = 0 Need[2][2] = 5 - 5 = 0
Need[3][0] = 0 - 0 = 0 Need[3][1] = 6 - 6 = 0 Need[3][2] = 5 - 2 = 3
Need[4][0] = 0 - 0 = 0 Need[4][1] = 6 - 0 = 6 Need[4][2] = 5 - 1 = 4

Reset

	Allocation			Max			Need		
	A	B	C	A	B	C	A	B	C
P1	0	0	0	0	0	1	0	0	0
P2	0	0	0	1	8	6	0	8	6
P3	0	0	0	2	3	5	1	0	0
P4	0	0	0	0	6	5	0	0	3
P5	0	0	0	0	6	5	0	6	4

Available		
A	B	C
4	15	11

Safe Process Sequence : P1 P3 P4 P5 P2

Operations: Calculating the Final Order....
Step1: Available Matrix = 2, 6, 2 As Need[1] = (0, 0, 0) < Available = (2, 6, 2) => Process P1 is selected. And New Available Matrix is (2, 6, 2) + (0, 0, 1) = (2, 6, 3)
Step2: Available Matrix = 2, 6, 3 As Need[3] = (1, 0, 0) < Available = (2, 6, 3) => Process P3 is selected. And New Available Matrix is (2, 6, 3) + (1, 3, 5) = (3, 9, 8)
Step3: Available Matrix = 3, 9, 8 As Need[4] = (0, 0, 3) < Available = (3, 9, 8) => Process P4 is selected. And New Available Matrix is (3, 9, 8) + (0, 6, 2) = (3, 15, 10)
Step4: Available Matrix = 3, 15, 10 As Need[5] = (0, 6, 4) < Available = (3, 15, 10) => Process P5 is selected. And New Available Matrix is (3, 15, 10) + (0, 0, 1) = (3, 15, 11)
Step5: Available Matrix = 3, 15, 11 As Need[2] = (0, 8, 6) < Available = (3, 15, 11) => Process P2 is selected. And New Available Matrix is (3, 15, 11) + (1, 0, 0) = (4, 15, 11)

Step 2: Analysis

BANKER'S ALGORITHM

	Allocation A B C	Maximum A B C	Available A B C
P_1	0 0 1	0 0 1	2 6 2
P_2	1 0 0	1 8 6	
P_3	1 3 5	2 3 5	
P_4	0 6 2	0 6 5	
P_5	0 0 1	0 6 5	

NEED (maximum - allocation)

	A B C
P_1	0 0 0
P_2	0 8 6
P_3	1 0 0
P_4	0 0 3
P_5	0 6 4

work

A B C

work	finish
2 6 2	~~F~~ T
+ 0 0 1 (P_1)	~~F~~ T
2 6 3	
+ 1 3 5 (P_3)	~~F~~ T
3 9 8	~~F~~ T
+ 0 6 2 (P_4)	~~F~~ T
3 15 10	
+ 0 0 1 (P_5)	
3 15 11	
+ 1 0 0	
4 15 11 (P_2)	

(allocation + available)
4 15 11 ← Recheck.

Safe sequence

<P_1, P_3, P_4, P_5, P_2>

My Analysis of the Banker's Algorithm Implementation

The Banker's Algorithm is used for resource allocation and deadlock avoidance. It checks for safety by simulating resource allocation based on the maximum possible resource requests.

Correct Output Calculation

According to my analysis, the stimulator accurately applies the Banker's Algorithm with the formula: Need=Maximum−Allocation This calculation ensures that the system knows exactly how much more of each resource each process will need to reach its maximum request, making sure the allocation can be done safely without causing deadlock.

Available Matrix Calculation

The available resources are calculated as: Available=Total Resources−Allocation The stimulator then rechecks this with: New Available=Allocation+Available This helps to confirm that the resources are accurately accounted for and reassesses availability after allocation.

Resource Utilization

This algorithm optimizes resource utilization by adjusting allocations based on current needs and maximum claims, preventing waste and ensuring resources are used effectively.

Performance

The algorithm performs well, quickly determining if a state is safe by comparing resource requests with available resources. This quick calculation ensures the system can respond to requests efficiently.

Efficiency

By preemptively checking for safe states before granting resources, the algorithm avoids deadlocks, leading to smoother and more efficient system operations without the need for costly deadlock resolution.

Conclusion

In conclusion, the stimulator implements the Banker's Algorithm correctly, performing the necessary calculations for safe resource allocation. This ensures optimal resource utilization, high performance, and efficient operation by preventing deadlocks and maintaining balanced resource distribution. The algorithm is a reliable method for effective system resource management.

Do your own, some One is watching!

Best of Luck!