Experiment No. 8
Memory Management a. Write a program to demonstrate the concept of dynamic partitioning placement algorithms i.e. Best Fit, First Fit, Worst-Fit
Date of Performance:
Date of Submission:
Marks:
Sign:

Aim: To study and implement memory allocation strategy First fit.

Objective:

a. Write a program to demonstrate the concept of dynamic partitioning placement algorithms i.e. Best Fit, First Fit, Worst-Fit etc.

Theory:

The primary role of the memory management system is to satisfy requests for memory allocation. Sometimes this is implicit, as when a new process is created. At other times, processes explicitly request memory. Either way, the system must locate enough unallocated memory and assign it to the process.

Partitioning: The simplest methods of allocating memory are based on dividing memory into areas with fixed partitions.

Selection Policies: If more than one free block can satisfy a request, then which one should we pick? There are several schemes that are frequently studied and are commonly used.

First Fit: In 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.

- **Advantage:** Fastest algorithm because it searches as little as possible.
- **Disadvantage:** The remaining unused memory areas left after allocation become waste if it is too smaller. Thus request for larger memory requirement cannot be accomplished
- 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 actual process size needed.
- Worst fit: In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

Next Fit: If we want to spread the allocations out more evenly across the memory space, we often use a policy called next fit. This scheme is very similar to the first fit approach, except for the place where the search starts.

Program:

```
// C implementation of First - Fit algorithm
#include<stdio.h>

// Function to allocate memory to
// blocks as per First fit algorithm
void firstFit(int blockSize[], int m, int processSize[], int n)
{
```

```
int i, j;
// Stores block id of the
// block allocated to a process
int allocation[n];

// Initially no block is assigned to any process
for(i = 0; i < n; i++)
{
        allocation[i] = -1;
}

// pick each process and find suitable blocks
// according to its size ad assign to it
for (i = 0; i < n; i++)    //here, n -> number of processes
{
        for (j = 0; j < m; j++)   //here, m -> number of blocks
        {
                if (blockSize[j] >= processSize[i])
                {
                        // allocating block j to the ith process
                        allocation[i] = j;

                        // Reduce available memory in this block.
                        blockSize[j] -= processSize[i];

                        break; //go to the next process in the queue
                }
        }
}

printf("\nProcess No.\tProcess Size\tBlock no.\n");
```

```
	for (int i = 0; i < n; i++)
	{
		printf(" %i\t\t\t", i+1);
		printf("%i\t\t\t\t", processSize[i]);
		if (allocation[i] != -1)
			printf("%i", allocation[i] + 1);
		else
			printf("Not Allocated");
		printf("\n");
	}
}

// Driver code
int main()
{
	int m; //number of blocks in the memory
	int n; //number of processes in the input queue
	int blockSize[] = {100, 500, 200, 300, 600};
	int processSize[] = {212, 417, 112, 426};
	m = sizeof(blockSize) / sizeof(blockSize[0]);
	n = sizeof(processSize) / sizeof(processSize[0]);

	firstFit(blockSize, m, processSize, n);

	return 0 ;
}
```

Output:

```
/tmp/Msa04QZ3Nq.o
Incoming     Frame 1     Frame 2     Frame 3
4            4           -           -
1            4           1           -
2            4           1           2
4            4           1           2
5            5           1           2
Total Page Faults:  4


=== Code Execution Successful ===
```

Conclusion:

Why do we need memory allocation strategies?

Memory allocation strategies are essential for efficient utilization of computer memory in any computing system. These strategies govern how memory is allocated and deallocated to processes or programs running on the system. There are several reasons why these strategies are necessary: 1. **Optimal Resource Utilization**: Memory is a finite resource, and efficient allocation ensures that it is used optimally. By employing appropriate allocation strategies, the system can maximize the amount of memory available for running processes, thereby improving overall system performance. 2. **Prevention of Fragmentation**: Memory fragmentation occurs when memory is allocated and deallocated in a way that leaves small pockets of unused memory scattered throughout the address

space. This fragmentation can lead to inefficient memory usage and may eventually result in fragmentation-related issues such as memory exhaustion. Memory allocation strategies help mitigate fragmentation by organizing memory allocation and deallocation in a structured manner. 3. **Fairness and Prioritization**: Different processes or programs running on a system may have varying memory requirements. Memory allocation strategies can prioritize certain processes over others based on factors such as process priority, resource usage, or specific requirements. This ensures fair allocation of memory resources and prevents resource starvation for critical processes. 4. **Performance Optimization**: Efficient memory allocation strategies can significantly impact system performance. For example, strategies that minimize memory overhead or reduce access latency can lead to faster program execution and improved responsiveness of the system as a whole. 5. **Memory Protection and Security**: Memory allocation strategies play a crucial role in enforcing memory protection and security mechanisms. By carefully managing memory allocation and access permissions, these strategies help prevent unauthorized access to sensitive data and mitigate security vulnerabilities such as buffer overflows or memory corruption attacks. Overall, memory allocation strategies are indispensable for ensuring the smooth operation, performance, and security of computing systems by effectively managing the allocation and utilization of memory resources