Experiment No.8
Implement memory allocation strategy First fit.
Date of Performance:
Date of Submission:

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

Objective: The objective of memory allocation strategies is to provide ways to dynamically allocate portions of memory to programs at their request, and free it for reuse when no longer needed as the main memory is limited in size.

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.

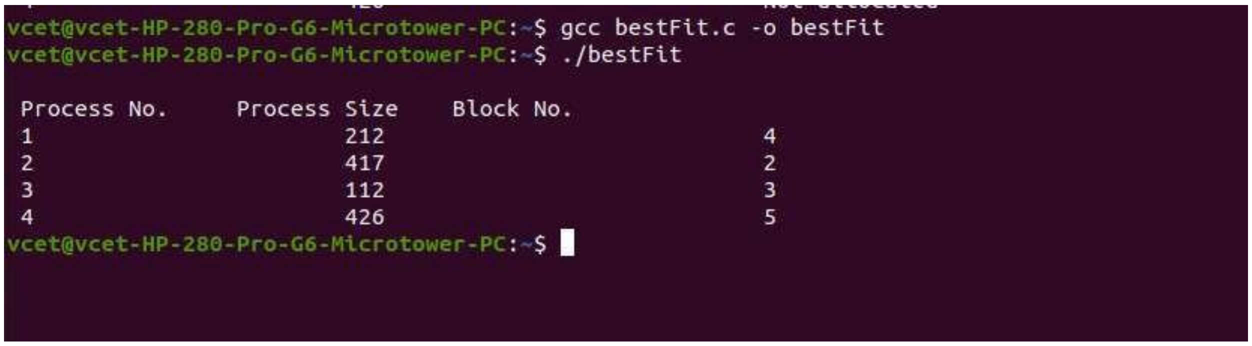
Result:

CODE:

BEST FIT:-

```
#include <stdio.h>
void bestFit(int blockSize[], int m, int processSize[], int n) {
  int allocation[n];
  for (int i = 0; i < n; i++) {
    allocation[i] = -1;
  }
  for (int i = 0; i < n; i++) {
    int bestIdx = -1; // Index of the best fitting block
    for (int j = 0; j < m; j++) {
      if (blockSize[j] >= processSize[i]) {
        if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx]) {
          bestIdx = j;
        }
      }
    }
    if (bestIdx != -1) {
      allocation[i] = bestIdx;
      blockSize[bestIdx] -= processSize[i];
    }
  }
  printf("\n Process No. \t Process Size \t Block 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");
  }
}
int main() {
  int m;
  int n;
  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]);
  bestFit(blockSize, m, processSize, n);
  return 0;
}
```

OUTPUT



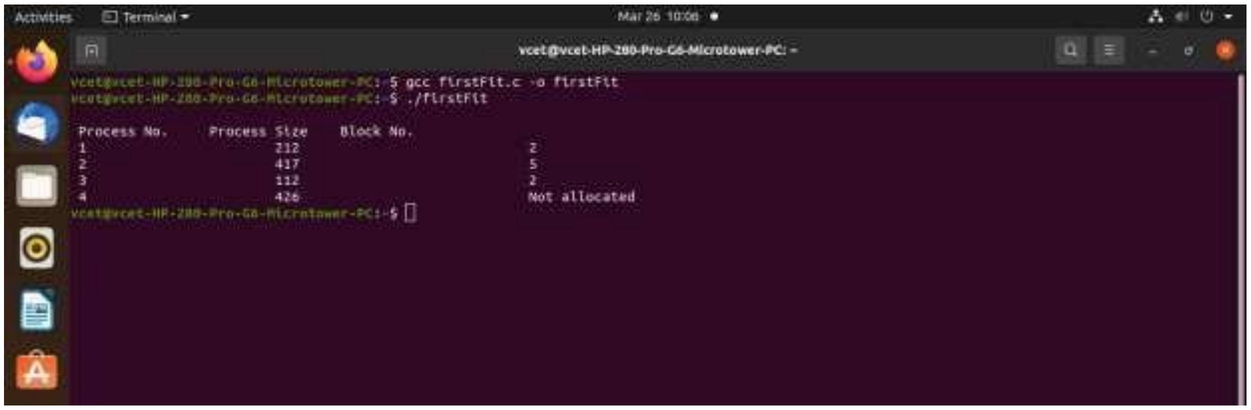
```
vcet@vcet-HP-280-Pro-G6-Microtower-PC:~$ gcc bestFit.c -o bestFit
vcet@vcet-HP-280-Pro-G6-Microtower-PC:~$ ./bestFit

 Process No.     Process Size    Block No.
 1                   212                        4
 2                   417                        2
 3                   112                        3
 4                   426                        5
vcet@vcet-HP-280-Pro-G6-Microtower-PC:~$ 
```

FIRST FIT:-

```
#include<stdio.h>
void firstFit(int blockSize[],int m, int processSize[],int n)
{
  int i,j;
  int allocation [n];
  for(i=0;i<n;i++)
  {
  allocation[i]=-1;
  }
  for(i=0;i<n;i++)
  {
    for(j=0;j<m;j++)
    {
       if(blockSize[j]>=processSize[i])
       {
       allocation[i]=j;
```

```
        blockSize[j]-=processSize[i];
        break;
        }
     }
   }
   printf("\n Process No. \t Process Size \t Block 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");
    }
}
int main()
{
      int m;
      int n;
      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



```
vcet@vcet-HP-280-Pro-G6-Microtower-PC:~$ gcc firstFit.c -o firstFit
vcet@vcet-HP-280-Pro-G6-Microtower-PC:~$ ./firstFit

 Process No.     Process Size    Block No.
 1                       212                         2
 2                       417                         5
 3                       112                         2
 4                       426                         Not allocated
vcet@vcet-HP-280-Pro-G6-Microtower-PC:~$ 
```

Conclusion: 1. Simplicity: The First Fit technique is an effective choice for memory management systems because it is simple to understand and apply.

2. Efficiency: By choosing the first memory block that is accessible and satisfies the needs of the operation, it usually provides a speedy allocation procedure. This efficiency stands out especially when contrasted with more intricate allocation techniques.

3. Fragmentation: On the other hand, the experiment brought attention to a possible disadvantage of the First Fit approach, namely memory fragmentation. Fragmentation is the result of little pockets of idle memory building up when processes are created and deallocated. In order to counteract the consequences of this fragmentation, which might lower overall memory use efficiency, further memory management strategies could be needed.

4. Space Utilization: The First Fit approach may not always choose the best block in terms of size, even if it seeks to allocate memory as soon as a suitable block is discovered. Smaller blocks may be allocated in lieu of bigger ones, resulting in inefficient space use.