|  |
| --- |
| **Exp. 9 - Part A** |
| **Aim:**  Conduct a study for Process Allocation Algorithms and perform following tasks   1. Implementation of First Fit, Worst Fit Algorithm using programming language |
| **Requirements:** C Programming knowledge and understanding of Memory Placement Technique concept. |
| **Outcome:** Understand the Placement Technique algorithm in an operating system.. |
| **Theory:**  **First Fit:**  The first-fit algorithm searches for the first free partition that is large enough to accommodate the process. The operating system starts searching from the beginning of the memory and allocates the first free partition that is large enough to fit the process.  For example, suppose we have the following memory partitions:  | 10 KB | 20 KB | 15 KB | 25 KB | 30 KB |  Now, a process requests 18 KB of memory. The operating system starts searching from the beginning and finds the first free partition of 20 KB. It allocates the process to that partition and keeps the remaining 2 KB as free memory.  **Worst Fit:**  The worst-fit algorithm searches for the largest free partition and allocates the process to it. This algorithm is designed to leave the largest possible free partition for future use.  For example, suppose we have the following memory partitions:  | 10 KB | 20 KB | 15 KB | 25 KB | 30 KB |  Now, a process requests 18 KB of memory. The operating system searches for the largest free partition, which is 30 KB. It allocates the process to that partition and keeps the remaining 12 KB as free memory. |
| **Instructions:**   1. Take input queue of memory request and then draw memory consumption based on the concept of placement technique. |

|  |  |
| --- | --- |
| **Exp.9- Part B** | |
| **Name of Student: Aryan Srivastava** | **Roll No.: A073** |
| **Programme:**  def first\_fit(memory\_blocks, process\_sizes):      allocation = [-1] \* len(process\_sizes)      for i in range(len(process\_sizes)):          for j in range(len(memory\_blocks)):              if memory\_blocks[j] >= process\_sizes[i]:                  allocation[i] = j                  memory\_blocks[j] -= process\_sizes[i]                  break      return allocation  def worst\_fit(memory\_blocks, process\_sizes):      allocation = [-1] \* len(process\_sizes)      for i in range(len(process\_sizes)):          worst\_index = -1          for j in range(len(memory\_blocks)):              if memory\_blocks[j] >= process\_sizes[i]:                  if worst\_index == -1 or memory\_blocks[j] > memory\_blocks[worst\_index]:                      worst\_index = j          if worst\_index != -1:              allocation[i] = worst\_index              memory\_blocks[worst\_index] -= process\_sizes[i]      return allocation  def display\_allocation(process\_sizes, allocation):      print("\nProcess No.\tProcess Size\tBlock No.")      for i in range(len(process\_sizes)):          if allocation[i] != -1:              print(f"   {i+1}   \t   {process\_sizes[i]}   \t   {allocation[i]+1}")          else:              print(f"   {i+1}   \t   {process\_sizes[i]}   \t   Not Allocated")  memory\_blocks = [10, 20, 15, 25, 30]  process\_sizes = [12, 18, 5, 35]  print("First Fit Allocation:")  ff\_memory\_blocks = memory\_blocks.copy()  ff\_allocation = first\_fit(ff\_memory\_blocks, process\_sizes)  display\_allocation(process\_sizes, ff\_allocation)  print("\nWorst Fit Allocation:")  wf\_memory\_blocks = memory\_blocks.copy()  wf\_allocation = worst\_fit(wf\_memory\_blocks, process\_sizes)  display\_allocation(process\_sizes, wf\_allocation) | |

|  |
| --- |
| **Output:** |
| **Observation & Learning:**  Implemented First Fit, Worst Fit Algorithm using python language |
| **Conclusion:**  Implemented First Fit, Worst Fit Algorithm using python language |
| **Questions:**   1. Write all the advantages and disadvantages of First Fit and Worst Fit Placement Technique?   **First Fit Advantages:** First Fit is simple to implement and fast because it stops searching once it finds the first suitable block. It works efficiently for smaller processes and minimizes time complexity by not scanning the entire memory.  **Disadvantages:** It often leads to external fragmentation, leaving small, unusable memory gaps. It can also cause uneven memory utilization, as the lower part of memory gets filled quickly, and it may miss better allocations by not considering larger blocks.  **Worst Fit**  **Advantages:** Worst Fit reduces fragmentation for large processes by allocating the largest available block, ensuring smaller blocks are left for future use. It can be helpful when many large processes need to be allocated.  **Disadvantages:** It is slower than First Fit because it must scan all memory blocks to find the largest one. This approach can waste large blocks on small processes, leading to poor memory utilization and increased fragmentation for smaller requests. |