**Assignment 7**

**Name** : Komal Potdar

**Roll No**.: 92

**PRN No**.: 12320165

**Div**: CS B SY

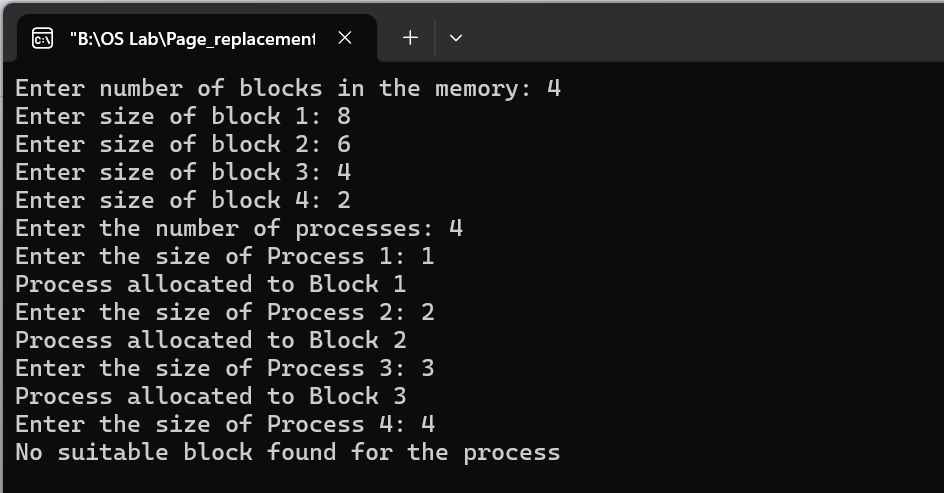
**Batch**: 3

**Page Replacement Strategies**

1. **First Fit**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct Block  {      int size;      int allocated;  };  void firstFit(struct Block \*memory, int numBlocks, int processSize)  {      for (int i = 0; i < numBlocks; i++)      {          if (!memory[i].allocated && memory[i].size >= processSize)          {              memory[i].allocated = 1;              printf("Process allocated to Block %d\n", i + 1);              return;          }      }      printf("No suitable block found for the process\n");  }  int main()  {      int numBlocks; // number of blocks in main memory      printf("Enter number of blocks in the memory: ");      if (scanf("%d", &numBlocks) != 1 || numBlocks <= 0)      {          printf("Invalid input. Please enter a positive integer for the number of blocks.\n");          return 1;      }      struct Block \*memory = (struct Block \*)malloc(numBlocks \* sizeof(struct Block));      for (int i = 0; i < numBlocks; i++)      {          printf("Enter size of block %d: ", i + 1);          if (scanf("%d", &memory[i].size) != 1 || memory[i].size <= 0)          {              printf("Invalid input. Please enter a positive integer for the block size.\n");              free(memory);              return 1;          }          memory[i].allocated = 0;      }      int numProcesses;      printf("Enter the number of processes: ");      if (scanf("%d", &numProcesses) != 1 || numProcesses <= 0)      {          printf("Invalid input. Please enter a positive integer for the number of processes.\n");          free(memory);          return 1;      }      for (int i = 0; i < numProcesses; i++)      {          int processSize;          printf("Enter the size of Process %d: ", i + 1);          if (scanf("%d", &processSize) != 1 || processSize <= 0)          {              printf("Invalid input. Please enter a positive integer for the process size.\n");              free(memory);              return 1;          }          firstFit(memory, numBlocks, processSize);      }      free(memory);      return 0;  } |

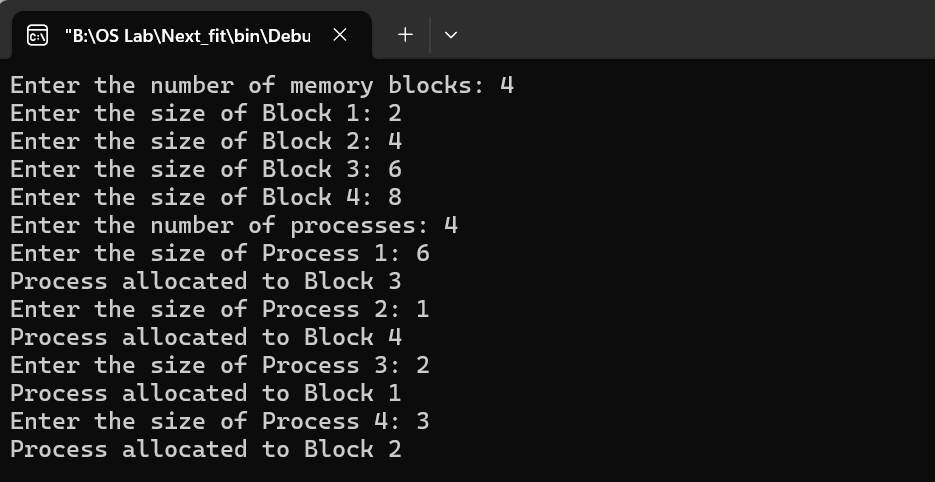
**Output:**



1. **Next Fit**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct Block  {  int size;  int allocated;  };  void nextFit(struct Block \*memory, int numBlocks, int processSize, int \*lastAllocated)  {  int startIndex = \*lastAllocated;  do  {  if (!memory[startIndex].allocated && memory[startIndex].size >= processSize)  {  memory[startIndex].allocated = 1;  printf("Process allocated to Block %d\n", startIndex + 1);  \*lastAllocated = (startIndex + 1) % numBlocks;  return;  }  startIndex = (startIndex + 1) % numBlocks;  } while (startIndex != \*lastAllocated);  printf("No suitable block found for the process.\n");  }  int main()  {  int numBlocks;  printf("Enter the number of memory blocks: ");  scanf("%d", &numBlocks);  struct Block \*memory = (struct Block \*)malloc(numBlocks \* sizeof(struct Block));  for (int i = 0; i < numBlocks; ++i)  {  printf("Enter the size of Block %d: ", i + 1);  scanf("%d", &memory[i].size);  memory[i].allocated = 0;  }  int numProcesses;  int lastAllocated = 0;  printf("Enter the number of processes: ");  scanf("%d", &numProcesses);  for (int i = 0; i < numProcesses; ++i)  {  int processSize;  printf("Enter the size of Process %d: ", i + 1);  scanf("%d", &processSize);  nextFit(memory, numBlocks, processSize, &lastAllocated);  }  free(memory);  return 0;  } |

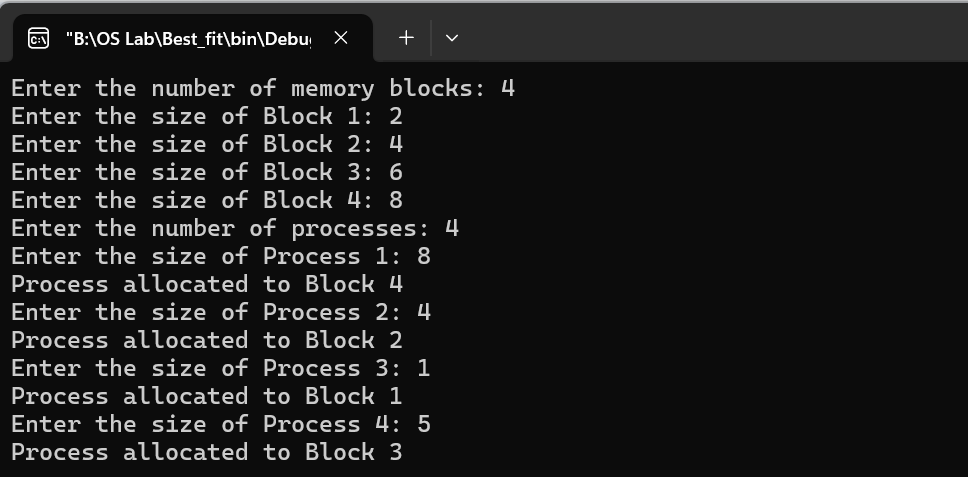
**Output:**



1. **Best Fit**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <limits.h>  struct Block  {  int size;  int allocated;  };  void bestFit(struct Block \*memory, int numBlocks, int processSize)  {  int bestFitIndex = -1;  int minSize = INT\_MAX;  for (int i = 0; i < numBlocks; ++i)  {  if (!memory[i].allocated && memory[i].size >= processSize)  {  if (memory[i].size < minSize)  {  minSize = memory[i].size;  bestFitIndex = i;  }  }  }  if (bestFitIndex != -1)  {  memory[bestFitIndex].allocated = 1;  printf("Process allocated to Block %d\n", bestFitIndex + 1);  }  else  {  printf("No suitable block found for the process.\n");  }  }  int main()  {  int numBlocks;  printf("Enter the number of memory blocks: ");  scanf("%d", &numBlocks);  struct Block \*memory = (struct Block \*)malloc(numBlocks \* sizeof(struct Block));  for (int i = 0; i < numBlocks; ++i)  {  printf("Enter the size of Block %d: ", i + 1);  scanf("%d", &memory[i].size);  memory[i].allocated = 0;  }  int numProcesses;  printf("Enter the number of processes: ");  scanf("%d", &numProcesses);  for (int i = 0; i < numProcesses; ++i)  {  int processSize;  printf("Enter the size of Process %d: ", i + 1);  scanf("%d", &processSize);  bestFit(memory, numBlocks, processSize);  }  free(memory);  return 0;  } |

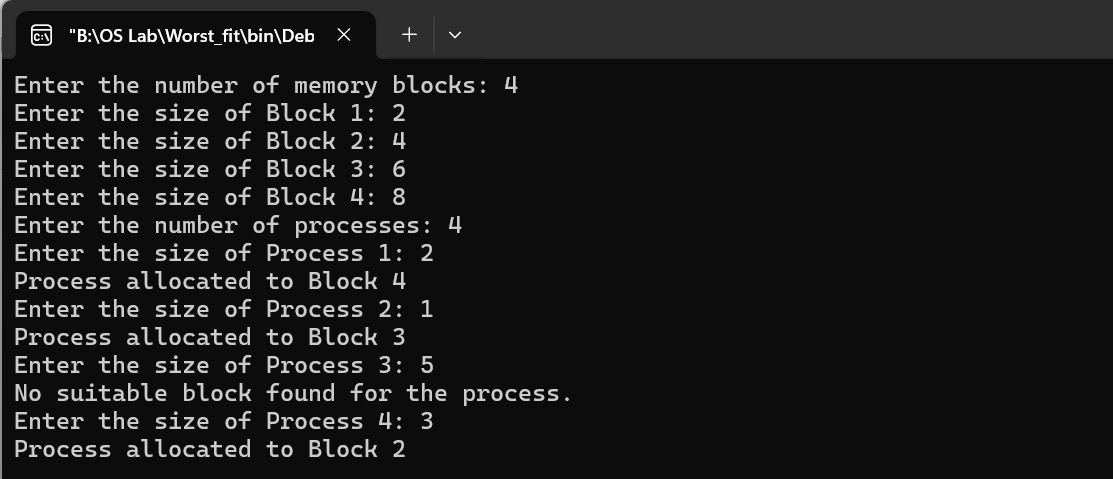
**Output:**



1. **Worst Fit**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct Block  {  int size;  int allocated;  };  void worstFit(struct Block \*memory, int numBlocks, int processSize)  {  int worstFitIndex = -1;  int maxSize = 0;  for (int i = 0; i < numBlocks; ++i)  {  if (!memory[i].allocated && memory[i].size >= processSize)  {  if (memory[i].size > maxSize)  {  maxSize = memory[i].size;  worstFitIndex = i;  }  }  }  if (worstFitIndex != -1)  {  memory[worstFitIndex].allocated = 1;  printf("Process allocated to Block %d\n", worstFitIndex + 1);  }  else  {  printf("No suitable block found for the process.\n");  }  }  int main()  {  int numBlocks;  printf("Enter the number of memory blocks: ");  scanf("%d", &numBlocks);  struct Block \*memory = (struct Block \*)malloc(numBlocks \* sizeof(struct Block));  for (int i = 0; i < numBlocks; ++i)  {  printf("Enter the size of Block %d: ", i + 1);  scanf("%d", &memory[i].size);  memory[i].allocated = 0;  }  int numProcesses;  printf("Enter the number of processes: ");  scanf("%d", &numProcesses);  for (int i = 0; i < numProcesses; ++i)  {  int processSize;  printf("Enter the size of Process %d: ", i + 1);  scanf("%d", &processSize);  worstFit(memory, numBlocks, processSize);  }  free(memory);  return 0; |

**Output:**

****