Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

Experiment Title: MEMORY MANAGEMENT

Aim/Objective:

The aim and objectives of memory management in operating systems are focused on effectively managing the computer's memory resources to optimize system performance, enable efficient execution of processes, and provide a secure and stable environment for running applications.

Description:

Memory management in operating systems refers to the management and organization of computer memory resources to efficiently allocate and control memory for processes and applications. It involves various techniques, algorithms, and data structures to optimize memory utilization, ensure data integrity, provide protection between processes, and enhance overall system performance. Here's a description of the key aspects of memory management in operating systems:

1. Memory Organization:
2. Memory Allocation:
3. Memory Protection and Isolation:
4. Virtual Memory Management:
5. Memory Deallocation:
6. Memory Fragmentation Management:
7. Memory Swapping and Page Replacement:

Pre-Requisites:

- **General Idea on memory management**
- **Concept of Internal Fragmentation and External Fragmentation**

Pre-Lab Task:

Memory Management	**FUNCTIONALITY**
Memory Fixed Partitioning Technique (MFT)	Divides memory into fixed-size partitions for process allocation.
Memory Variable Partitioning Technique (MVT)	Allocates memory dynamically, adjusting partition sizes based on process needs.

Memory Management	FUNCTIONALITY
Internal Fragmentation	Wasted memory within an allocated partition due to size mismatches.
External Fragmentation	Free memory is scattered, making it difficult to allocate large processes.
Dynamic Memory Allocation	Memory allocated during program execution, using techniques like malloc or calloc.
Static Memory Allocation	Memory allocated at compile-time, with fixed sizes for variables.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

In-Lab

1. Write a C Program to implement the Memory Fixed Partitioning Technique (MFT) algorithm.

```
#include <stdio.h>
#include <stdlib.h>
#define MEMORY_SIZE 1024
#define PARTITION_SIZE 256
int partitions[MEMORY_SIZE / PARTITION_SIZE];
int partitionCount = MEMORY_SIZE / PARTITION_SIZE;

void initializeMemory() {
   for (int i = 0; i < partitionCount; i++) {
      partitions[i] = -1;
   }
}

int allocateMemory(int processSize) {
   for (int i = 0; i < partitionCount; i++) {
      if (partitions[i] == -1 && (i + 1) * PARTITION_SIZE >= processSize) {
         partitions[i] = processSize;
         return i;
      }
   }
   return -1;
}

void deallocateMemory(int partitionNumber) {
   partitions[partitionNumber] = -1;
}

int main() {
   initializeMemory();
   int process1Size = 200;
   int process2Size = 400;
   int process3Size = 300;
   int partition1 = allocateMemory(process1Size);
   int partition2 = allocateMemory(process2Size);
   int partition3 = allocateMemory(process3Size);
   if (partition1 == -1 || partition2 == -1 || partition3 == -1) {
      printf("Memory allocation failed.\n");
```

```
 } else {

      printf("Memory allocated for Process 1 in Partition %d\n", partition1);
      printf("Memory allocated for Process 2 in Partition %d\n", partition2);
      printf("Memory allocated for Process 3 in Partition %d\n", partition3);
      deallocateMemory(partition1);
      deallocateMemory(partition2);

      deallocateMemory(partition3);
    }
    return 0;
}
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

1. Write a C program to implement the Memory Variable Partitioning Technique (MVT) algorithm.

```
#include <stdio.h>
#include <stdlib.h>
#define MEMORY_SIZE 1024

typedef struct Partition {
   int size;
   int isAllocated;
} Partition;

Partition memory[MEMORY_SIZE];

void initializeMemory() {
   for (int i = 0; i < MEMORY_SIZE; i++) {
      memory[i].size = 0;
      memory[i].isAllocated = 0;
   }
}

int allocateMemory(int processSize) {
   for (int i = 0; i < MEMORY_SIZE; i++) {
      if (!memory[i].isAllocated && memory[i].size >= processSize) {
         memory[i].isAllocated = 1;
         return i;
      }
   }
   return -1;
}

void deallocateMemory(int partitionNumber) {
   memory[partitionNumber].isAllocated = 0;
}

int main() {
   initializeMemory();
   int process1Size = 200;
   int process2Size = 400;
   int process3Size = 300;
   int partition1 = allocateMemory(process1Size);
   int partition2 = allocateMemory(process2Size);
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

```
    int partition3 = allocateMemory(process3Size);

   if (partition1 == -1 || partition2 == -1 || partition3 == -1) {
      printf("Memory allocation failed.\n");
   } else {


printf("Memory allocated for Process 1 in Partition %d\n", partition1);

      printf("Memory allocated for Process 2 in Partition %d\n", partition2);
      printf("Memory allocated for Process 3 in Partition %d\n", partition3);
      deallocateMemory(partition1);
      deallocateMemory(partition2);
      deallocateMemory(partition3);
   }
   return 0;
}
```

Post – Lab:

1. **Write a Program to simulate Dynamic Memory Allocation in C using malloc ().**

```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int n;
   printf("Enter the number of integers you want to allocate: ");
   scanf("%d", &n);
   int *dynamicArray = (int *)malloc(n * sizeof(int));
   if (dynamicArray == NULL) {
      printf("Memory allocation failed. Exiting...\n");
      return 1;
   }
   printf("Enter %d integers:\n", n);
   for (int i = 0; i < n; i++) {
      scanf("%d", &dynamicArray[i]);
   }
   printf("You entered the following integers:\n");
   for (int i = 0; i < n; i++) {
      printf("%d ", dynamicArray[i]);
   }
   printf("\n");
   free(dynamicArray);
   return 0;
}
```

Data and Results

Data:

The program dynamically allocates memory for integers and takes user input for processing.

Result:

The entered integers are displayed, and memory is successfully freed after use.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

Analysis and inferences:

Analysis:

Memory allocation ensures flexibility in handling different sizes of integer arrays for user inputs.

Inferences:

Dynamic memory allocation helps efficiently manage memory based on user input size.

2. Write a Program to simulate Dynamic Memory Allocation in C using free ().

```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int n;
   printf("Enter the number of integers you want to allocate: ");
   scanf("%d", &n);
   int *dynamicArray = (int *)malloc(n * sizeof(int));
   if (dynamicArray == NULL) {
      printf("Memory allocation failed. Exiting...\n");
      return 1;
   }
   printf("Enter %d integers:\n", n);
   for (int i = 0; i < n; i++) {
      scanf("%d", &dynamicArray[i]);
   }
   printf("You entered the following integers:\n");
   for (int i = 0; i < n; i++) {
      printf("%d ", dynamicArray[i]);
   }
   printf("\n");
   free(dynamicArray);
   return 0;
}
```

3. Write a Program to simulate Dynamic Memory Allocation in C using Calloc ().

```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int n;
   printf("Enter the number of integers you want to allocate: ");
   scanf("%d", &n);
   int *dynamicArray = (int *)calloc(n, sizeof(int));
   if (dynamicArray == NULL) {
      printf("Memory allocation failed. Exiting...\n");
      return 1;
   }
   printf("Enter %d integers:\n", n);
   for (int i = 0; i < n; i++) {
      scanf("%d", &dynamicArray[i]);
   }
   printf("You entered the following integers:\n");
   for (int i = 0; i < n; i++) {
      printf("%d ", dynamicArray[i]);
   }
   printf("\n");
   free(dynamicArray);
   return 0;
}
```

4. Write a Program to simulate Dynamic Memory Allocation in C using Relloc ().

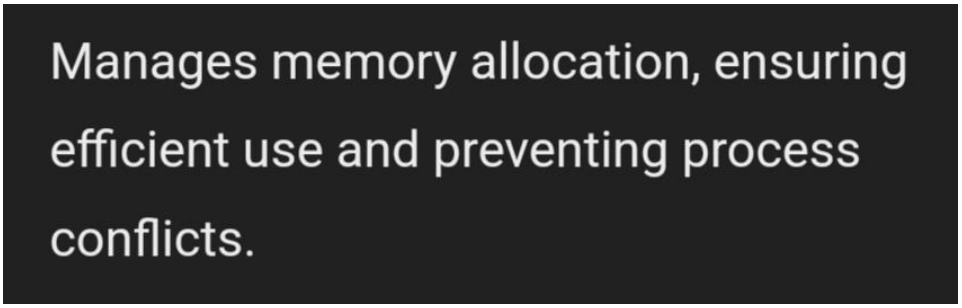
```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int n, newSize;

   printf("Enter the number of integers you want to allocate: ");
   scanf("%d", &n);

   int *dynamicArray = (int *)malloc(n * sizeof(int));
   if (dynamicArray == NULL) {
      printf("Memory allocation failed. Exiting...\n");
      return 1;
   }

   printf("Enter %d integers:\n", n);
   for (int i = 0; i < n; i++) {
      scanf("%d", &dynamicArray[i]);
   }

   printf("You entered the following integers:\n");
   for (int i = 0; i < n; i++) {
      printf("%d ", dynamicArray[i]);
   }
   printf("\n");

   printf("Enter the new size for the array: ");
   scanf("%d", &newSize);

   dynamicArray = (int *)realloc(dynamicArray, newSize * sizeof(int));
   if (dynamicArray == NULL) {
      printf("Memory reallocation failed. Exiting...\n");
      return 1;
   }

   printf("Enter %d integers for the resized array:\n", newSize);
   for (int i = n; i < newSize; i++) {
      scanf("%d", &dynamicArray[i]);
```

```
 }

   printf("You entered the following integers after resizing:\n");
   for (int i = 0; i < newSize; i++) {
      printf("%d ", dynamicArray[i]);
   }


   printf("\n");

   free(dynamicArray);

   return 0;
}
```

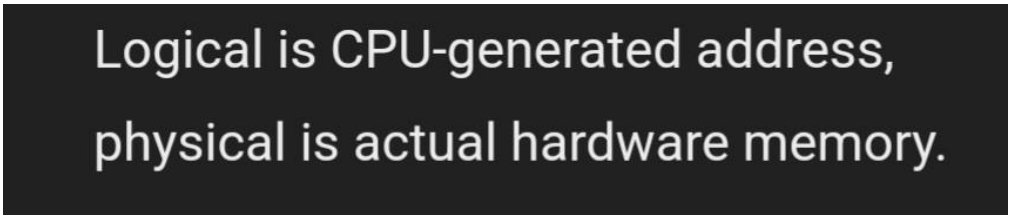
Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

Sample VIVA-VOCE Questions:

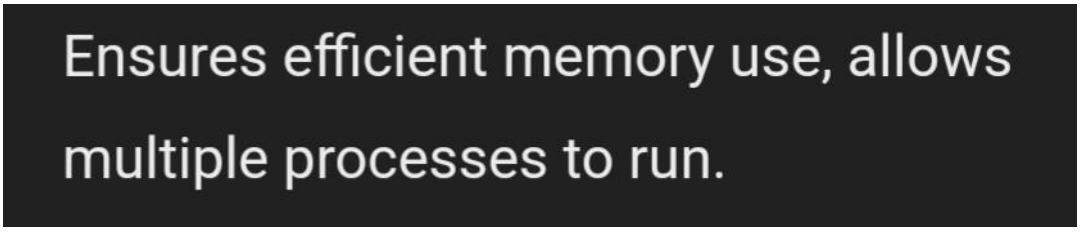
1. What is the purpose of memory management in an operating system?



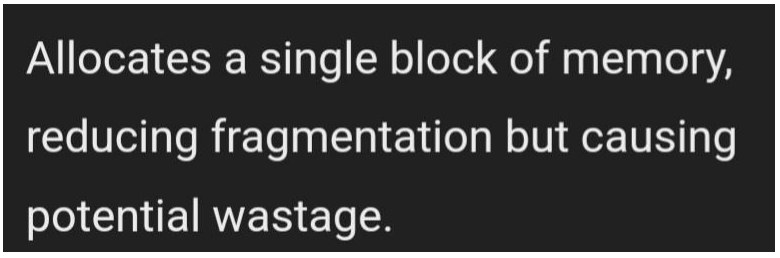
2. Explain the difference between logical and physical memory?



3. Why is memory allocation important in operating systems?



4. How does contiguous memory allocation work?



5. Differentiate between fixed partitioning and dynamic partitioning.

Fixed has fixed-sized partitions, dynamic adjusts based on process size.

6. What is fragmentation? Explain internal and external fragmentation with examples.

Fragmentation is wasted memory due to inefficient allocation.

- **Internal**: Unused space within allocated blocks.
- **External**: Small free blocks scattered across memory.

Evaluator Remark (if any):	**Marks Secured _____out of 50**
	Signature of the Evaluator with Date

Note: Evaluator MUST ask Viva-voce before signing and posting marks for each experiment.