Session	**2025-26 (ODD)**	**Course Name**	**Operating System Lab**
Semester	**5**	**Course Code**	**23IOT1504**
Roll No	**35**	**Name of Student**	**Bhushan Tayade**

Practical Number	**9**
Course Outcome	1. Understand Computer System Configuration and Simulate system resources efficiently using Linux Commands (CO1) 2. Analyse operating system functionalities utilizing system calls, thread programming and process scheduling algorithms (CO2) 3. Apply Synchronization primitives to implement a Deadlock-free solution(CO3) 4. Simulate Disk scheduling, Memory allocation, File allocation, page replacement algorithms (CO4)
Aim	**Stimulate All Files Allocation Strategies**
Problem Definition	**1. Sequential** **2. Indexed** **3. Linked**
Theory	**1. Sequential (Contiguous) Allocation** In sequential allocation, each file occupies a set of contiguous or adjacent blocks on the disk. This means that all the blocks of a file are stored one after another in sequence. The file's directory entry maintains the starting block address and the total number of blocks allocated to the file. This method provides very fast access to the file, as all its blocks are stored continuously, making it efficient for both sequential and direct access operations. However, it suffers from **external fragmentation**, meaning that over time, free disk space may become scattered and it may be difficult to find large continuous areas for new files. Moreover, it is hard to expand a file once it is allocated, since the adjacent blocks may already be occupied. For example, if a file starts at block 10 and has a length of 4 blocks, it will occupy blocks 10, 11, 12, and 13. **2. Linked Allocation** In linked allocation, files are stored as a linked list of disk blocks that may be scattered anywhere on the disk. Each block contains not

only the data but also a pointer to the next block in the file. The directory entry for the file holds the address of the first block, and from there, each block's pointer leads to the next one until the end of the file is reached. This method eliminates external fragmentation since any free block can be used. It also allows files to grow easily by adding more blocks to the chain. However, it has some drawbacks — accessing a file is slower because the system must follow pointers from one block to another, and it only supports **sequential access**, not direct access. Additionally, the storage space used by pointers slightly reduces the space available for data. For example, a file might be stored across blocks 5, 9, and 12, connected as 5 → 9 → 12.

3. Indexed Allocation

In indexed allocation, all the addresses of the file's data blocks are stored in a separate block called the **index block**. Each file has its own index block that keeps track of the locations of all the disk blocks used by that file. The directory entry stores the address of this index block. To read or write a file, the operating system simply refers to the index block to locate any required data block. This method supports both **direct** and **sequential access**, avoids external fragmentation, and makes it easy to expand files. However, it requires additional storage space for the index block, and for very large files, multiple index blocks may be needed. For example, if the index block is located at block 5 and the file's data blocks are stored at 11, 15, and 17, then block 5 contains the addresses [11, 15, 17].

Procedure and Execution (100 Words)	**File Allocation Strategies** File allocation is the method used by an operating system to store and manage files on secondary storage like hard disks. The main goal is to use disk space efficiently while providing quick access to files. There are three major file allocation strategies: **Sequential (Contiguous) Allocation**, **Linked Allocation**, and **Indexed Allocation**. Each has its own way of storing file blocks, along with different advantages and limitations. In **Sequential or Contiguous Allocation**, each file occupies a set of adjacent blocks on the disk. The file directory maintains the starting block address and the total number of blocks. This method offers very fast access because all blocks are stored continuously, supporting both sequential and direct access efficiently. However, it suffers from **external fragmentation**, as finding a large enough continuous free area becomes difficult. It is also hard to grow files dynamically because adjacent space may already be occupied. In **Linked Allocation**, each file is stored as a linked list of disk blocks scattered anywhere on the disk. Each block contains data and a pointer to the next block. The directory entry stores the address of the first block. This method eliminates external fragmentation and makes it easy to expand files. However, it has slower access because the system must follow the pointer chain from one block to another. It supports only sequential access, not direct access, and the pointer field slightly reduces usable storage space. In **Indexed Allocation**, all addresses of a file's blocks are kept in a separate index block. Each file has its own index block that contains pointers to all data blocks. This method supports both sequential and direct access and avoids fragmentation. It is easy to grow files, but it requires additional space for the index block. 1. Compute turnaround time = burst time + waiting time. 2. Find average waiting time and turnaround time. 3. Display process order, burst, waiting, and turnaround times.

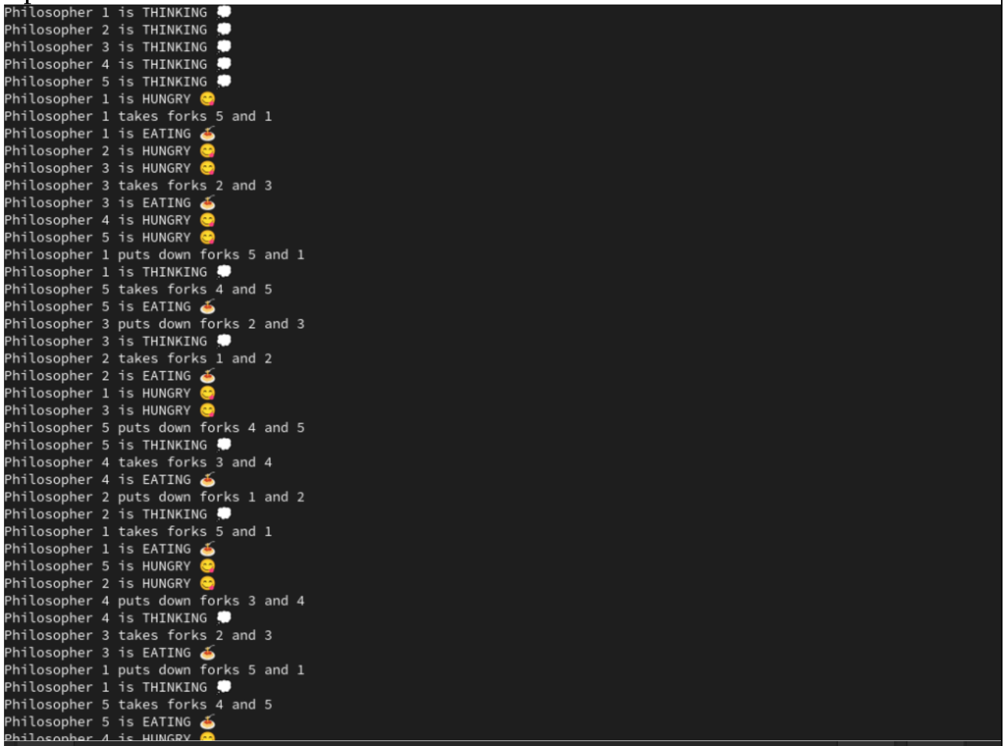
Code:

```
#include <stdio.h>

#include <stdlib.h>



void sequentialAllocation() {

    int startBlock, length, i;

    int blocks[50] = {0};



    printf("\n--- Sequential File Allocation ---
\n");

    printf("Enter the starting block: ");

    scanf("%d", &startBlock);

    printf("Enter the length of the file: ");

    scanf("%d", &length);



    printf("File allocated in blocks: ");

    for (i = startBlock; i < startBlock + length;
i++) {

        if (blocks[i] == 0) {

            blocks[i] = 1;

            printf("%d ", i);

        } else {

            printf("\nBlock %d already allocated.
Allocation failed.\n", i);

            break;
        }
    }
    printf("\n");
}

void linkedAllocation() {
    int n, i, block, next;
    int blocks[50] = {0};

    printf("\n--- Linked File Allocation ---\n");
    printf("Enter number of blocks to allocate: 
");
    scanf("%d", &n);

    printf("Enter block numbers:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &block);
        if (blocks[block] == 0) {
            blocks[block] = 1;
        } else {
            printf("Block %d already allocated.\n", 
block);
        }
    }
    printf("File allocated using linked list:\n");
    for (i = 0; i < n; i++) {
        printf("%d", i);
        if (i != n - 1)
            printf(" -> ");
    }
    printf("\n");
}

void indexedAllocation() {
    int indexBlock, n, i, block;
    int blocks[50] = {0};

    printf("\n--- Indexed File Allocation ---\n");
    printf("Enter the index block: ");
    scanf("%d", &indexBlock);
    blocks[indexBlock] = 1;

    printf("Enter number of blocks needed: ");
    scanf("%d", &n);

    int index[n];
    printf("Enter block numbers: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &block);
        if (blocks[block] == 0) {
            blocks[block] = 1;
            index[i] = block;
        } else {
            printf("Block %d already allocated.\n",
block);
            i--;
        }
    }


    printf("Index Block %d -> ", indexBlock);
    for (i = 0; i < n; i++)
        printf("%d ", index[i]);
    printf("\n");
}


int main() {
    int choice;
    printf("File Allocation Strategies
Simulation\n");
    printf("1. Sequential Allocation\n");
    printf("2. Linked Allocation\n");
    printf("3. Indexed Allocation\n");
    printf("Enter your choice (1-3): ");
    scanf("%d", &choice);

    switch (choice) {
        case 1: sequentialAllocation(); break;
        case 2: linkedAllocation(); break;
        case 3: indexedAllocation(); break;
        default: printf("Invalid choice!\n");
    }

    return 0;
}
```

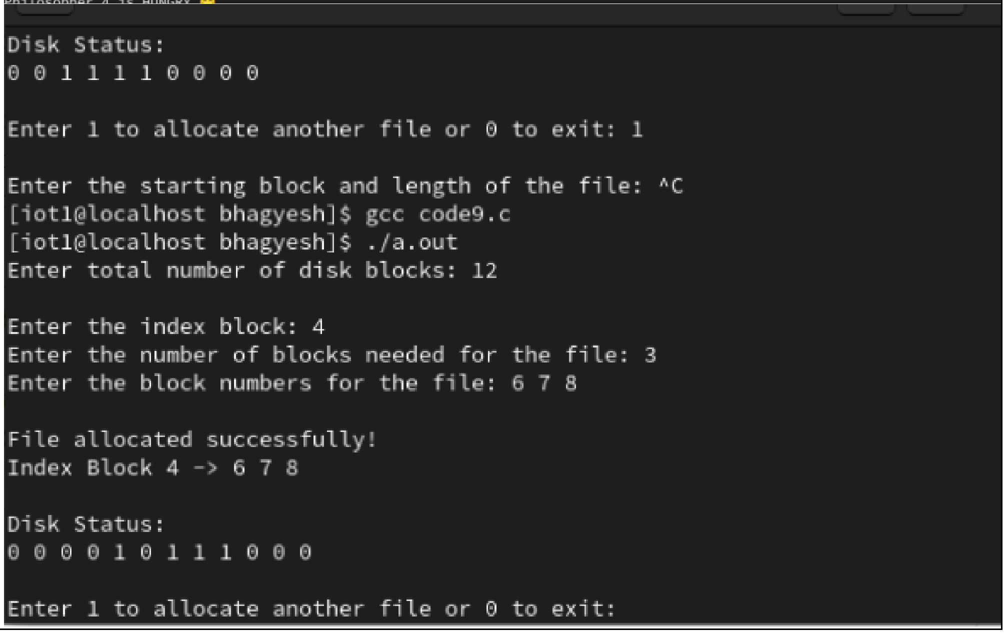

	`scanf("%d", &A[i][1]);`
	`A[i`

```
printf("Ave
```

Output:

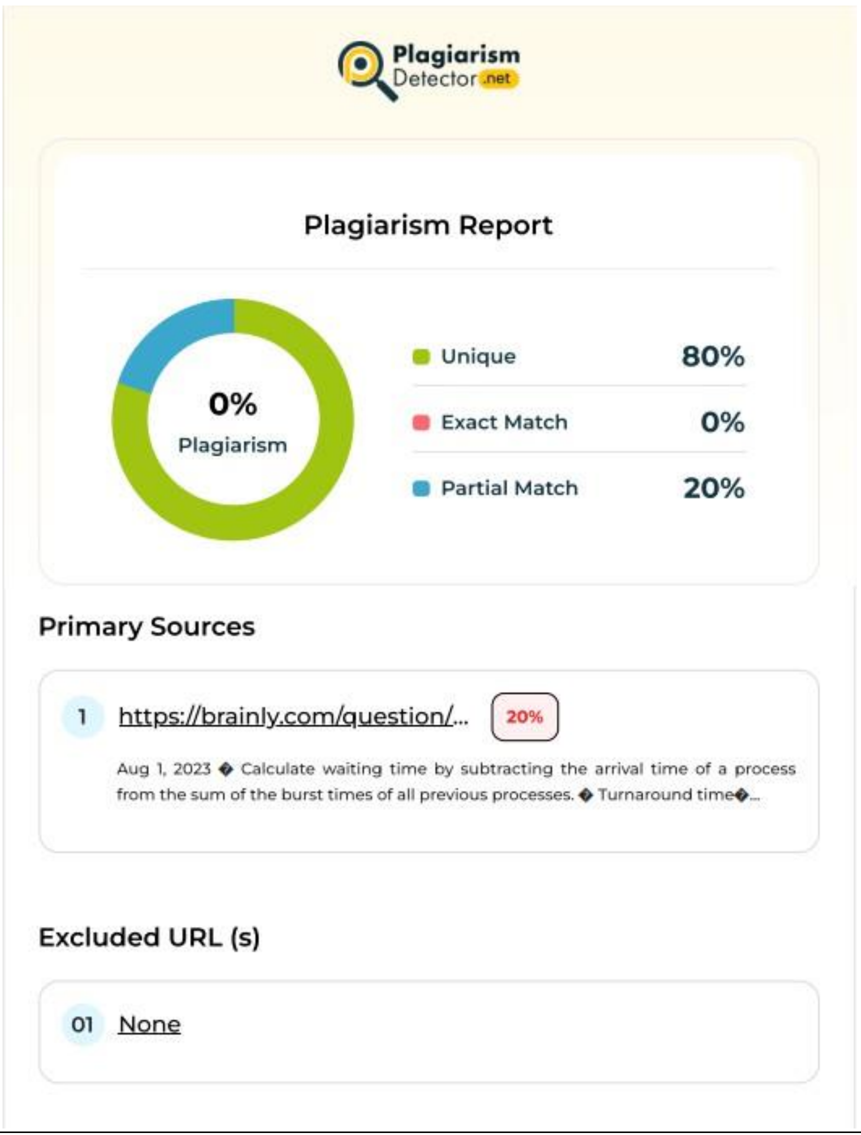


```
Philosopher 1 is THINKING 💭
Philosopher 2 is THINKING 💭
Philosopher 3 is THINKING 💭
Philosopher 4 is THINKING 💭
Philosopher 5 is THINKING 💭
Philosopher 1 is HUNGRY 😋
Philosopher 1 takes forks 5 and 1
Philosopher 1 is EATING 🍝
Philosopher 2 is HUNGRY 😋
Philosopher 3 is HUNGRY 😋
Philosopher 3 takes forks 2 and 3
Philosopher 3 is EATING 🍝
Philosopher 4 is HUNGRY 😋
Philosopher 5 is HUNGRY 😋
Philosopher 1 puts down forks 5 and 1
Philosopher 1 is THINKING 💭
Philosopher 5 takes forks 4 and 5
Philosopher 5 is EATING 🍝
Philosopher 3 puts down forks 2 and 3
Philosopher 3 is THINKING 💭
Philosopher 2 takes forks 1 and 2
Philosopher 2 is EATING 🍝
Philosopher 1 is HUNGRY 😋
Philosopher 3 is HUNGRY 😋
Philosopher 5 puts down forks 4 and 5
Philosopher 5 is THINKING 💭
Philosopher 4 takes forks 3 and 4
Philosopher 4 is EATING 🍝
Philosopher 2 puts down forks 1 and 2
Philosopher 2 is THINKING 💭
Philosopher 1 takes forks 5 and 1
Philosopher 1 is EATING 🍝
Philosopher 5 is HUNGRY 😋
Philosopher 2 is HUNGRY 😋
Philosopher 4 puts down forks 3 and 4
Philosopher 4 is THINKING 💭
Philosopher 3 takes forks 2 and 3
Philosopher 3 is EATING 🍝
Philosopher 1 puts down forks 5 and 1
Philosopher 1 is THINKING 💭
Philosopher 5 takes forks 4 and 5
Philosopher 5 is EATING 🍝
Philosopher 4 is HUNGRY 😋
```



```
Disk Status:
0 0 1 1 1 1 0 0 0 0

Enter 1 to allocate another file or 0 to exit: 1

Enter the starting block and length of the file: ^C
[iot1@localhost bhagyesh]$ gcc code9.c
[iot1@localhost bhagyesh]$ ./a.out
Enter total number of disk blocks: 12

Enter the index block: 4
Enter the number of blocks needed for the file: 3
Enter the block numbers for the file: 6 7 8

File allocated successfully!
Index Block 4 -> 6 7 8

Disk Status:
0 0 0 0 1 0 1 1 1 0 0 0

Enter 1 to allocate another file or 0 to exit:
```

Output Analysis	**Enter the starting block: 5** **Enter the length of the file: 4** **File allocated in blocks: 5 6 7 8**
Link of student Github profile where lab assignment has been uploaded	"https://github.com/Bhushan-Tayade/YCCN-23071391.git"
Conclusion	**Each file allocation strategy has its own advantages and limitations. Sequential allocation offers fast access but suffers from fragmentation. Linked allocation removes fragmentation and allows flexible file growth but is slower due to pointer traversal. Indexed allocation provides efficient direct access and easy file expansion, though it requires extra space for the index block. The choice of allocation method depends on the system requirements, such as speed, memory efficiency, and access type.**

Plag Report (Similarity index < 12%)	 Plagiarism Detector .net **Plagiarism Report** 0% Plagiarism 🟩 Unique 80% 🟥 Exact Match 0% 🟦 Partial Match 20% **Primary Sources** 1 https://brainly.com/question/... 20% Aug 1, 2023 � Calculate waiting time by subtracting the arrival time of a process from the sum of the burst times of all previous processes. � Turnaround time�... **Excluded URL (s)** 01 None
Date	**30/10/25**