Assignment - 5

OPERATING SYSTEM

Topic: Memory Management

Q: Write a C program to simulate the MVT and MFT memory management techniques.

DESCRIPTION:

MFT (Multiprogramming with a Fixed Number of Tasks) is one of the old memory management techniques in which the memory is partitioned into fixed-size partitions and each job is assigned to a partition. The memory assigned to a partition does not change. *MVT (Multiprogramming with a Variable Number of Tasks)* is a memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning of memory is dynamic and changes as jobs enter and leave the system. MVT is a more ``efficient'' user of resources. MFT suffers from the problem of internal fragmentation, and MVT suffers from external fragmentation.

CODE :

```
#include <stdio.h>
#define MAX_PROCESSES 100
void mft();
void mvt();
int main() {
    int choice;
    printf("Choose memory management technique:\n");
    printf("1. MFT (Multiprogramming with a Fixed number of Tasks)\n");
    printf("2. MVT (Multiprogramming with a Variable number of Tasks)\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);
    if (choice == 1) {
        mft();
    } else if (choice == 2) {
        mvt();
    } else {
        printf("Invalid choice.\n");
    }
    return 0;
}
void mft() {
    int totalMemory, blockSize, numProcesses;
    int processMemory[MAX_PROCESSES];
    int blockCount, i, j;
    int internalFragmentation = 0;
    printf("Enter the total memory available (in Bytes): ");
    scanf("%d", &totalMemory);
    printf("Enter the block size (in Bytes): ");
    scanf("%d", &blockSize);
    printf("Enter the number of processes: ");
    scanf("%d", &numProcesses);
    blockCount = totalMemory / blockSize;
    printf("No. of Blocks available in memory -- %d\n", blockCount);
    for (i = 0; i < numProcesses; i++) {
        printf("Enter memory required for process %d (in Bytes): ", i + 1);
        scanf("%d", &processMemory[i]);
```

```
    }
    printf("\nPROCESS\tMEMORY REQUIRED\tALLOCATED\tINTERNAL FRAGMENTATION\n");
    for (i = 0; i < numProcesses; i++) {
        int allocated = 0;
        int fragmentation = 0;
        for (j = 0; j < blockCount; j++) {
            if (processMemory[i] <= blockSize) {
                allocated = 1;
                fragmentation = blockSize - processMemory[i];
                internalFragmentation += fragmentation;
                break;
            }
        }
        if (allocated) {
            printf("%d\t%d\t\tYES\t\t%d\n", i + 1, processMemory[i], fragmentation);
        } else {
            printf("%d\t%d\t\tNO\t\t—\n", i + 1, processMemory[i]);
        }
    }
    printf("\nMemory is full; the remaining processes cannot be accommodated.\n");
    printf("The total internal fragmentation is %d.\n", internalFragmentation);
    printf("Total External Fragmentation is %d\n", totalMemory - (blockCount * blockSize));
}
void mvt() {
    int totalMemory;
    int processMemory[MAX_PROCESSES];
    int memoryAllocated = 0;
    int totalMemoryAllocated = 0;
    printf("Enter the total memory available (in Bytes): ");
    scanf("%d", &totalMemory);
    int total = totalMemory;
    while (memoryAllocated < MAX_PROCESSES) {
        printf("Enter memory required for process %d (in Bytes): ", memoryAllocated + 1);
        int memoryRequired;
        scanf("%d", &memoryRequired);
        if (memoryRequired > totalMemory) {
            printf("Memory required for process %d is more than the remaining memory available.\n",
memoryAllocated + 1);
            break;  // Stop taking further inputs once memory exceeds remaining memory.
        }
        processMemory[memoryAllocated] = memoryRequired;
        totalMemory -= memoryRequired;
        totalMemoryAllocated += memoryRequired;
        memoryAllocated++;
        printf("Memory is allocated for Process %d\n", memoryAllocated);
        if (totalMemory == 0) {
            printf("Memory is Full\n");
            break;  // Stop if no more memory is available.
        }
        char choice;
        printf("Do you want to continue (y/n): ");
        scanf(" %c", &choice);
        if (choice == 'n') break;
    }
    printf("Memory is Full");
```

```
    printf("\nTotal Memory Available -- %d Bytes\n", total);
    printf("\nPROCESS\tMEMORY ALLOCATED\n");
    for (int i = 0; i < memoryAllocated; i++) {
        printf("%d\t%d Bytes\n", i + 1, processMemory[i]);
    }
    printf("\nTotal Memory Allocated is %d Bytes\n", totalMemoryAllocated);
    printf("Total External Fragmentation is %d Bytes\n", totalMemory);
}
```

```
namrata@NamraRio:~/MCA2023/Namrata_B_34/assignment5$
 nano memoryManagement.c
namrata@NamraRio:~/MCA2023/Namrata_B_34/assignment5$
 gcc memoryManagement.c -o memoryManagement
namrata@NamraRio:~/MCA2023/Namrata_B_34/assignment5$
 ./memoryManagement
Choose memory management technique:
1. MFT (Multiprogramming with a Fixed number of Task
s)
2. MVT (Multiprogramming with a Variable number of T
asks)
Enter your choice (1 or 2): 1
Enter the total memory available (in Bytes): 1000
Enter the block size (in Bytes): 300
Enter the number of processes:
5
No. of Blocks available in memory -- 3
Enter memory required for process 1 (in Bytes): 275
Enter memory required for process 2 (in Bytes): 400
Enter memory required for process 3 (in Bytes): 290
Enter memory required for process 4 (in Bytes): 293
Enter memory required for process 5 (in Bytes): 100

PROCESS MEMORY REQUIRED ALLOCATED       INTERNAL FRA
GMENTATION
1       275             YES             25
2       400             NO              -
3       290             YES             10
4       293             YES             7
5       100             YES             200

Memory is full; the remaining processes cannot be ac
commodated.
The total internal fragmentation is 242.
Total External Fragmentation is 100
```

```
namrata@NamraRio:~/MCA2023/Namrata_B_34/assi
gnment5$ ./memoryManagement
Choose memory management technique:
1. MFT (Multiprogramming with a Fixed number
 of Tasks)
2. MVT (Multiprogramming with a Variable num
ber of Tasks)
Enter your choice (1 or 2): 2
Enter the total memory available (in Bytes):
 1000
Enter memory required for process 1 (in Byte
s): 400
Memory is allocated for Process 1
Do you want to continue (y/n): y
Enter memory required for process 2 (in Byte
s): 275
Memory is allocated for Process 2
Do you want to continue (y/n): y
Enter memory required for process 3 (in Byte
s): 550
Memory required for process 3 is more than t
he remaining memory available.
Memory is Full
Total Memory Available -- 1000 Bytes

PROCESS MEMORY ALLOCATED
1       400 Bytes
2       275 Bytes

Total Memory Allocated is 675 Bytes
Total External Fragmentation is 325 Bytes
```