Name: K Srikanth Registration Number: 17ETCS002124

Laboratory 8

Title of the Laboratory Exercise: Programs for memory management algorithms

1. Introduction and Purpose of Experiment

In a multiprogramming system, the user part of memory must be further subdivided to accommodate multiple processes. This task of subdivision is carried out dynamically done by the operating system known as memory management. By solving these problems students will become familiar with the implementations of memory management algorithms in dynamic memory partitioning scheme of operating system.

2. Aim and Objectives

Aim

- To develop a simulator for memory management algorithms

Objectives

At the end of this lab, the student will be able to

- Apply memory management algorithms wherever required
- Develop simulators for the algorithms

3. Experimental Procedure

I. Analyse the problem statement

II. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code

III. Implement the algorithm in C language

IV. Compile the C program

V. Test the implemented program

VI. Document the Results

VII. Analyse and discuss the outcomes of your experiment

Name: K Srikanth Registration Number: 17ETCS002124

4. Questions

Implement a simulator for the memory management algorithms with the provision of compaction and garbage collection

a) First fit

b) Best fit

b) Worst fit

5. Calculations/Computations/Algorithms

First Fit Algorithm

1. **Start**
2. Input the number of blocks
3. Input the size of each block.
4. Input the number of processes
5. Input the size of each process
6. If (size of the block > = size of the process and dirty bit not 0)
    a. allocate the block to the process
    b. Mark dirty bit
7. Else
    a. moves on to the next block
8. Display the table with the blocks allocated to a respective process
9. **End**

Best Fit Algorithm

1. Start
2. Input the number of memory blocks
3. Input the size of each block.
4. Input the number of processes
5. Input the size of each process
6. Set all the memory blocks as free.(dirty bit 0)

7. Pick a process
8. Find the block that is best to assign to the current process which when allocated leaves out a smallest fragment.
9. After checking all the processes, assign to the one which creates the smallest hole. (mark dirty bit)
10. If there is no free block that can be allocated to the process or any block that is big enough to fit the current process, don't allocate.
11. Repeat previous 4 steps for all other processes.
12. **End**

Worst Fit Algorithm

1. **Start**
2. Input the number of memory blocks
3. Input the size of each block.
4. Input the number of processes
5. Input the size of each process
6. Set all the memory blocks as free.(dirty bit 0)
7. Pick a process
8. Find the block that is too big to assign for the current process which results in largest possible fragment.
9. After checking all the processes, assign to the one which creates the largest possible hole. (mark dirty bit)
10. If there is no free block that can be allocated to the process or any block that is big enough to fit the current process, don't allocate.
11. Repeat previous 4 steps for all other processes.
12. **End**

6. Presentation of Results

Main Function

```c
int main(int argc, char **argv)
{
    // Declaration Section 17ETCS002124 K Srikanth
    int *partitions_size;
    int *partitions;
    int *processes;
    int *partition_order;
    int n, nl;
    int loop = 0;
    int i, j;
    printf("Enter the Number of Partitions : ");
    scanf("%d", &n);
    partition_order = (int *)malloc(sizeof(int) * n);
    processes = (int *)malloc(sizeof(int) * nl);
    partitions_size = (int *)malloc(sizeof(int) * n);
    partitions = (int *)malloc(sizeof(int) * n);
    for (i = 0; i < n; i++)
    {
        printf("Enter the Partition %d Size :", i + 1);
        scanf("%d", (partitions_size + i));
        partitions[i] = 0;
    }
    printf("The Partitions are : \n");
    for (i = 0; i < n; i++)
    {
        printf("%d |", partitions_size[i]);
    }
    printf("\n");
    printf("Enter the Number of Processes : ");
    scanf("%d", &nl);
    for (i = 0; i < nl; i++)
    {
        printf("Enter the Partition %d Size :", i + 1);
        scanf("%d", (processes + i));
    }
    printf("Enter the Choice : \n");
    printf("Press 1 For First Fit \n");
    printf("Press 2 For Best Fit \n");
    printf("Press 3 For Worst Fit \n");
    int choice;
    scanf("%d", &choice);
```

Figure 1 Main Function

a) First fit

Code

```c
case 1:
    printf("\n");
    for (i = 0; i < nl; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (partitions[j] == 0 && *(processes + i) <= partitions_size[j])
            {
                partitions[j] = i + 1;
                *(processes + i) = 0;
                printf("Executing First Fit Algorithm  \nPartitions after sorting : \n");
                printf("Process %d fit in Partition %d \n", partitions[j], j + 1);
                break;
            }
        }
    }
    for (i = 0; i < n; i++)
    {
        if (processes[i] != 0)
        {
            printf("Process %d Size %d Couldn't Fit\n", i + 1, processes[i]);
        }
    }
    return (EXIT_SUCCESS);
    break;
```

Figure 2 Case 1 For First Fit Algorithm

Result

```
❯ ./a.out
Enter the Number of Partitions : 2
Enter the Partition 1 Size :100
Enter the Partition 2 Size :200
The Partitions are :
100 |200 |
Enter the Number of Processes : 2
Enter the Partition 1 Size :100
Enter the Partition 2 Size :200
Enter the Choice :
Press 1 For First Fit
Press 2 For Best Fit
Press 3 For Worst Fit
1

Executing First Fit Algorithm
Partitions after sorting :
Process 1 fit in Partition 1
Executing First Fit Algorithm
Partitions after sorting :
Process 2 fit in Partition 2
```

Figure 3 C Program Output For First Fit Algorithm

b) Best fit

Code

```
case 2:
    for (i = 0; i < n; i++){
        partition_order[i] = i;}
    for (i = 0; i < (n - 1); i++){
        for (j = 0; j < (n - i - 1); j++){
            if (partitions_size[j] > partitions_size[j + 1]){
                int temp = partitions_size[j];
                partitions_size[j] = partitions_size[j + 1];
                partitions_size[j + 1] = temp;
                temp = partition_order[j];
                partition_order[j] = partition_order[j + 1];
                partition_order[j + 1] = temp;}}}
    printf("\n");
    printf("Executing Best Fit Algorithm \nPartitions after sorting : \n");
    for (i = 0; i < n; i++){
        printf("%d |", partitions_size[i]);}
    printf("\n");
    for (i = 0; i < nl; i++){
        for (j = 0; j < n; j++){
            if (partitions[j] == 0 && *(processes + i) <= partitions_size[j]){
                partitions[j] = i + 1;
                printf("Process %d [Size = %d] fit in partition %d [Size =%d]\n", i + 1,
                processes[i], partition_order[j] + 1, partitions_size[j]);
                processes[i] = 0;
                break;}} }
    break;
```

Figure 4 Case 2 For Best Fit Algorithm

Result

```
❯ ./a.out
Enter the Number of Partitions : 2
Enter the Partition 1 Size :100
Enter the Partition 2 Size :200
The Partitions are :
100 |200 |
Enter the Number of Processes : 2
Enter the Partition 1 Size :100
Enter the Partition 2 Size :200
Enter the Choice :
Press 1 For First Fit
Press 2 For Best Fit
Press 3 For Worst Fit
2

Executing Best Fit Algorithm __
Partitions after sorting :
100 |200 |
Process 1 [Size = 100] fit in partition 1 [Size =100]
Process 2 [Size = 200] fit in partition 2 [Size =200]
```

Figure 5 C Program Output For Best Fit Algorithm

c) Worst fit

Code

```
case 3:
    for (i = 0; i < n; i++){
        partition_order[i] = i;}
    for (i = 0; i < (n - 1); i++) {
        for (j = 0; j < (n - i - 1); j++){
            if (partitions_size[j] > partitions_size[j + 1]){
                int temp = partitions_size[j];
                partitions_size[j] = partitions_size[j + 1];
                partitions_size[j + 1] = temp;
                temp = partition_order[j];
                partition_order[j] = partition_order[j + 1];
                partition_order[j + 1] = temp;}}}
    printf("\n");
    printf("Executing Worst Fit Algorithm  \nPartitions after sorting : \n");
    for (i = 0; i < nl; i++){
        printf("%d |", partitions_size[i]);}
    printf("\n");
    for (i = 0; i < nl; i++){
        for (j = n - 1; j > -1; j--){
            if (partitions[j] == 0 && *(processes + i) <= partitions_size[j]){
                partitions[j] = i + 1;
                printf("Process %d [Size = %d] fit in partition %d [Size =%d]\n", i + 1,
                processes[i], partition_order[j] + 1, partitions_size[j]);
                processes[i] = 0;
                break;}}}
    break;
    default:
    break;
}
```

Figure 6 Case 3 For Worst Fit Algorithm

Result

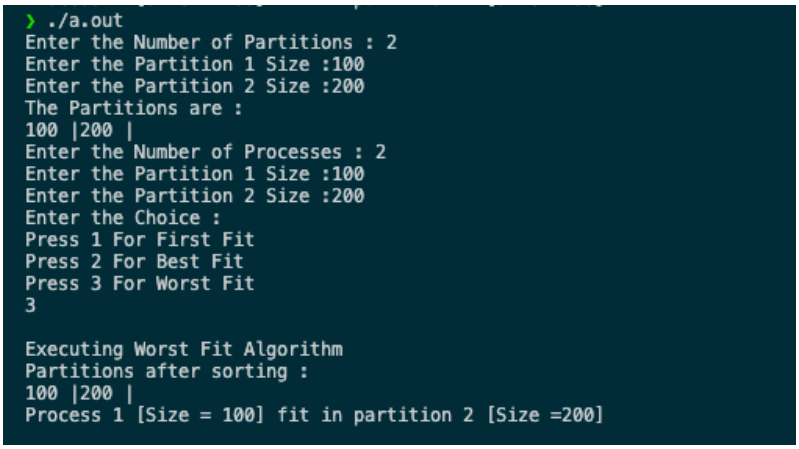

Figure 7 C Program Output For Worst Fit Algorithm

7. Analysis and Discussions

Memory allocation is one of the important tasks of the operating system. As the different processes need the CPU processing time, they also need a certain amount of memory. This memory is allocated to different processes according to their demand. The three most common types of memory allocation schemes are the best fit, first fit, and worst fit.

First – fit: The first fit memory allocation scheme checks the empty memory block in a sequential manner. This means that the memory block found empty at the first attempt is checked for size; if the size is not less than the required size, it is allocated. One of the biggest issues in this memory allocation scheme is, when a process is allocated to a comparatively larger space than needed, it creates huge chunks of memory space.

Best fit: In the case of the best fit memory allocation scheme, the operating system searches for the empty memory block. When the operating system finds the memory block with minimum wastage of memory, it is allocated to the process. This scheme is considered as the best approach as it results in most optimised memory allocation. However, finding the best fit memory allocation may be time-consuming. Best fit allocates the process to a partition which is the smallest sufficient partition among the free available partitions.

Worst – fit: The processes need empty memory slots during processing time. This memory is allocated to the processes by the operating system which decides depending on the free memory and the demanded memory by the process in execution. In the case of the worst fit memory allocation scheme, the operating system searches for free memory blocks demanded by the operating system. An empty block is assigned to the processes as soon as the CPU identifies it. The scheme is also said as the worst fit memory management scheme as sometimes a process is allocated a memory block which is much larger to the actual demand resulting in a huge amount of wasted memory.

The operating system conducts this memory allocation using an algorithm.

8. Conclusions

In this lab session the student has performed partition allocation memory management technique. In Partition Allocation, when there is more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation. Memory management is one of the important tasks in the operating system. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Advantages

- **Best-fit:** Memory utilization is much better than first fit as it searches the smallest free partition.
- **First-fit:** Fastest algorithm because it searches as little as possible.
- **Worst-fit:** Reduces the rate of production of small gaps.

9. Comments

1. Limitations of Experiments

- Although, best fit minimizes the wastage space, it consumes a lot of processor time for searching the block which is close to required size. Also, Best-fit may perform poorer than other algorithms in some cases.
- It may have problems of not allowing processes to take space even if it was possible to allocate.

- Worst Fit allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. If a large process comes at a later stage, then memory will

not have space to accommodate it.

2. Learning happened

To, apply memory management algorithms wherever required and to develop simulators for same.