EXPERIMENT NO. 6

AIM: Write a program to implement Dynamic Memory partitioning algorithm i.e. First fit, Best fit, Worst fit for memory management.

RESOURCES REQUIRED:

H/W Requirements: P-IV and above, Ram 128 MB, Printer, Internet Connection.

S/W Requirements: Python Compiler.

THEORY:

Memory Partitioning:

1. **Fixed Partitioning:**

Main memory is divided into a no. of static partitions at system generation time. A process may be loaded into a partition of equal or greater size. Memory Manager will allocate a region to a process that best fits it unused memory within an allocated partition called internal fragmentation

Advantages:

Simple to implement Little OS overhead

Disadvantages:

It may lead to inefficient use of memory due to internal fragmentation. Main memory utilization is extremely inefficient. Any program, no matter how small, occupies an entire partition. This phenomenon, in which there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition, is referred to as internal fragmentation.

There can be two possibilities as follows:

1. Equal size partitioning
2. Unequal size Partition

Not suitable for systems in which process memory requirements not known ahead of time; i.e. timesharing systems. When the queue for a large partition is empty but the queue for a small partition is full, as is the case for partitions 1 and 3. Here small jobs have to wait to get into memory; even though plenty of memory is free an alternative organization is to maintain a single queue as in

Figure (b). Whenever a partition becomes free, the job closest to the front of the queue that fits in it could be loaded into the empty partition and run.

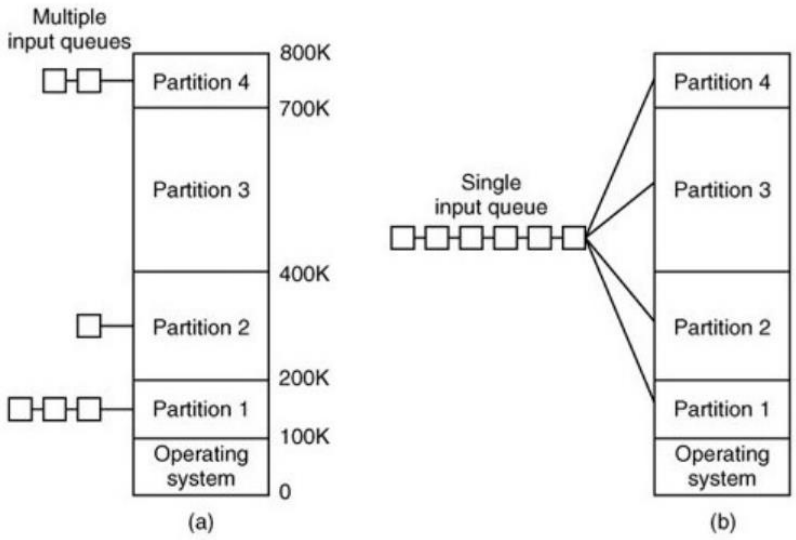

Fig: (a) Fixed memory partitions with separate input queues for each partition. (b) Fixed memory partitions with a single input queue.

2. Dynamic/Variable Partitioning:

To overcome some of the difficulties with fixed partitioning, an approach known as dynamic partitioning was developed. The partitions are of variable length and number. When a process is brought into main memory, it is allocated exactly as much memory as it requires and no more.

An example, using 64 Mbytes of main memory, is shown in figure. Eventually it leads to a situation in which there are a lot of small holes in memory. As time goes on, the data memory becomes more and more fragmented, and memory utilization declines.

This phenomenon is referred to as external fragmentation, indicating that the memory that is external to all partitions becomes increasingly fragmented. One technique for overcoming external fragmentation is compaction.

From time to time, the operating system shifts the processes so that they are contiguous and so that all of the free memory is together in one block. There are various dynamic memory partitioning algorithms such as Best Fit, Worst Fit and First Fit for memory management.

1. **Best Fit:**

The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire

list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.

Advantage:

Memory utilization is much better than first fit as it searches the smallest free partition first available.

Disadvantage:

It is slower and may even tend to fill up memory with tiny useless holes.

2. **Worst fit:**

In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

Advantage:

Reduces the rate of production of small gaps.

Disadvantage:

If a process requiring larger memory arrives at a later stage then it cannot be accommodated as the largest hole is already split and occupied.

3. **First Fit:**

In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.

Advantage:

Fastest algorithm because it searches as little as possible.

Disadvantage:

The remaining unused memory areas left after allocation become waste if it is too smaller. Thus, request for larger memory requirement cannot be accomplished.

CONCLUSION: Hence, we have implemented programs on Dynamic Memory partitioning algorithm with First fit, Best fit, Worst fit for memory management.

CODE:

1. Best Fit:

```
import sys
def printf(format, *args):
    sys.stdout.write(format % args)

class BestFit:
    def __init__(self):
        return None

    def bestFit(self,block,process):

        m,n = len(block),len(process)

        allocation = [-1 for x in range(n)]

        for x in range(0,n):
            best_index = -1
            for y in range(0,m):
                if block[y] >= process[x]:
                    if best_index == -1:
                        best_index = y
                    else:
                        if block[y]<block[best_index]:
                            best_index = y

            if best_index != -1:
```

```
            allocation[x] = best_index
            block[best_index] -=process[x]

        printf("\nProcess no. \t\t Process size \t\t Block No. \n")

        for x in range(0,n):
            if allocation[x] == -1:
                printf("%10d \t\t %10d \t\t Not Allocated\n",(x+1),process[x])
            else:
                printf("%5d \t\t %15d \t\t %8d\n",(x+1),process[x],(1+allocation[x]))
        return None



if __name__=="__main__":
    print("55_Adnan_Shaikh")
    block = [100,500,200,300,600]
    process = [212,417,112,426]
    bestf = BestFit()
    bestf.bestFit(block,process)
```

2. Worst Fit:

```
import sys
def printf(format, *args):
    sys.stdout.write(format % args)

class WorstFit:
```

```
def __init__(self):
    return None

def worstFit(self,block,process):

    m,n = len(block),len(process)

    allocation = [-1 for x in range(n)]

    for x in range(0,n):
        worst_index = -1
        for y in range(0,m):
            if block[y] >= process[x]:
                if worst_index == -1:
                    worst_index = y
                else:
                    if block[y]>block[worst_index]:
                        worst_index = y

        if worst_index != -1:
            allocation[x] = worst_index
            block[worst_index] -=process[x]

    printf("\nProcess no. \t\t Process size \t\t Block No.\n")

    for x in range(0,n):
        if allocation[x] == -1:
```

```
            printf("%5d \t\t %15d \t\t Not Allocated\n",(x+1),process[x])
        else:
            printf("%5d \t\t %15d \t\t %5d\n",(x+1),process[x],(1+allocation[x]))
    return None



if __name__=="__main__":
    print("55_Adnan_Shaikh")
    block = [100,500,200,300]
    process = [500,400,600,426]
    worstf = WorstFit()
    worstf.worstFit(block,process)
```

3. First Fit:

```
import sys
def printf(format, *args):
    sys.stdout.write(format % args)

class FirstFit:
    def __init__(self):
        return None

    def firstFit(self,block,process):

        m,n = len(block),len(process)
```

```
        allocation = [-1 for x in range(n)]

        for x in range(0,n):
            for y in range(0,m):
                if block[y] >= process[x]:
                    allocation[x] = y
                    block[y] -=process[x]
                    break



        printf("\nProcess no. \t\t Process size \t\t Block No.\n")

        for x in range(0,n):
            if allocation[x] == -1:
                printf("%5d \t\t %15d \t\t Not Allocated\n",(x+1),process[x])
            else:
                printf("%5d \t\t %15d \t\t %5d\n",(x+1),process[x],(1+allocation[x]))
        return None



if __name__=="__main__":
    print("55_Adnan_Shaikh")
    block = [100,500,200,300,600]
    process = [500,90,112,426]
    firstf = FirstFit()
```

firstf.firstFit(block,process)

OUTPUT:

1. Best Fit:

```
Command Prompt

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\OS\Fitting algorithm>python bestfit.py
55_Adnan_Shaikh

Process no.             Process size            Block No.
    1                       212                     4
    2                       417                     2
    3                       112                     3
    4                       426                     5
```

2. Worst Fit:

```
Command Prompt

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\OS\Fitting algorithm>python worstfit.py
55_Adnan_Shaikh

Process no.             Process size            Block No.
    1                       500                     2
    2                       400                 Not Allocated
    3                       600                 Not Allocated
    4                       426                 Not Allocated
```

3. First Fit:

```
Command Prompt

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\OS\Fitting algorithm>python firstfit.py
55_Adnan_Shaikh

Process no.             Process size            Block No.
    1                       500                     2
    2                        90                     1
    3                       112                     3
    4                       426                     5
```