OS - Experiment 5

Name: Kartik Jolapara Sapid: 60004200107

Div./Batch: B/B1 Branch: Computer Engineering

AIM:

To implement various memory allocation techniques like first fit, best fit and worst fit.

THEORY:

- **First fit**

This method keeps the free/busy list of jobs organized by memory location, low-ordered to high-ordered memory. In this method, first job claims the first available memory space more than or equal to its size. The operating system doesn't search for appropriate partition but just allocate the job to the nearest memory partition available with sufficient size.

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int noOfPartitions;
    cout << "Enter the number of patitions: ";
    cin >> noOfPartitions;
    int partitionMemory[noOfPartitions], tempPMemory[noOfPartitions];
    for (int i = 0; i < noOfPartitions; i++)
    {
        cout << "Partition " << i + 1 << ": ";
        cin >> partitionMemory[i];
```

```
        tempPMemory[i] = partitionMemory[i];
    }
    int noOfProcesses;
    cout << "Enter the number of processes: ";
    cin >> noOfProcesses;
    int processesMemory[noOfProcesses];
    for (int i = 0; i < noOfProcesses; i++)
    {
        cout << "Partition " << i + 1 << ": ";
        cin >> processesMemory[i];
    }
    string firstFit[noOfPartitions], notAllocatedProcesses;
    for (int i = 0; i < noOfPartitions; i++)
    {
        firstFit[i] = "X";
    }
    for (int i = 0; i < noOfProcesses; i++)
    {
        bool isAlloc = false;
        for (int j = 0; j < noOfPartitions; j++)
        {
            if (processesMemory[i] <= partitionMemory[j])
            {
                partitionMemory[j] -= processesMemory[i];
                if (firstFit[j] == "X")
                {

                    firstFit[j] = "P" + to_string(i + 1);
                }
                else
                {
                    firstFit[j] += ", P" + to_string(i + 1);
                }
                isAlloc = true;
                break;
            }
        }
        if (!isAlloc && notAllocatedProcesses.empty())
        {
            notAllocatedProcesses = "P" + to_string(i + 1);
        }
        else if (!isAlloc)
        {
            notAllocatedProcesses += ", P" + to_string(i + 1);
        }
    }
    cout << "\nPartitions\t\tFirst Fit\n";
    for (int i = 0; i < noOfPartitions; i++)
```

```
    {
        cout << tempPMemory[i] << "\t\t\t" << firstFit[i] << "\n";
    }
    if (notAllocatedProcesses.empty())
    {
        cout << "There are no unallocated processes!\n";
    }
    else
    {
        cout << "The unallocated processes are: " <<
notAllocatedProcesses << "\n";
    }

    return 0;
}
```

Output:

```
Enter the number of patitions: 4
Partition 1: 10
Partition 2: 50
Partition 3: 30
Partition 4: 20
Enter the number of processes: 3
Partition 1: 20
Partition 2: 60
Partition 3: 10

Partitions              First Fit
10                      P3
50                      P1
30                      X
20                      X
The unallocated processes are: P2
```

- **Best fit**

This method keeps the free/busy list in order by size-smallest to largest. In this method, the operating system first searches the whole of the memory according to the size of the given job and allocates it to the closestfitting free partition in the memory, making it able to use memory efficiently. Here the jobs are in the order from smallest job to largest job.

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int noOfPartitions;
    cout << "Enter the number of patitions: ";
    cin >> noOfPartitions;
    int partitionMemory[noOfPartitions], tempPMemory[noOfPartitions];
    for (int i = 0; i < noOfPartitions; i++)
    {
        cout << "Partition " << i + 1 << ": ";
        cin >> partitionMemory[i];
        tempPMemory[i] = partitionMemory[i];
    }
    int noOfProcesses;
    cout << "Enter the number of processes: ";
    cin >> noOfProcesses;
    int processesMemory[noOfProcesses];
    for (int i = 0; i < noOfProcesses; i++)
    {
        cout << "Partition " << i + 1 << ": ";
        cin >> processesMemory[i];
    }
    string bestFit[noOfPartitions], notAllocatedProcesses;
    for (int i = 0; i < noOfPartitions; i++)
    {
        bestFit[i] = "X";
    }
    for (int i = 0; i < noOfProcesses; i++)
    {
        bool isAlloc = false;
        int minSize;
        for (int j = 0; j < noOfPartitions; j++)
        {
            if (processesMemory[i] <= partitionMemory[j] && !isAlloc)
            {
```

```
                minSize = j;
                isAlloc = true;
            }
            else if (processesMemory[i] <= partitionMemory[j])
            {
                if (partitionMemory[j] < partitionMemory[minSize])
                {
                    minSize = j;
                }
            }
        }
        if (isAlloc)
        {
            partitionMemory[minSize] -= processesMemory[i];
            if (bestFit[minSize] == "X")
            {

                bestFit[minSize] = "P" + to_string(i + 1);
            }
            else
            {
                bestFit[minSize] += ", P" + to_string(i + 1);
            }
            isAlloc = true;
        }
        else if (!isAlloc && notAllocatedProcesses.empty())
        {
            notAllocatedProcesses = "P" + to_string(i + 1);
        }
        else if (!isAlloc)
        {
            notAllocatedProcesses += ", P" + to_string(i + 1);
        }
    }
    cout << "\nPartitions\t\tBest Fit\n";
    for (int i = 0; i < noOfPartitions; i++)
    {
        cout << tempPMemory[i] << "\t\t\t" << bestFit[i] << "\n";
    }
    if (notAllocatedProcesses.empty())
    {
        cout << "There are no unallocated processes!\n";
    }
    else
    {
        cout << "The unallocated processes are: " <<
notAllocatedProcesses << "\n";
    }
```

```
    return 0;
}
```

Output:

```
Enter the number of patitions: 4
Partition 1: 10
Partition 2: 50
Partition 3: 30
Partition 4: 20
Enter the number of processes: 3
Partition 1: 20
Partition 2: 60
Partition 3: 10

Partitions              Best Fit
10                      P3
50                      X
30                      X
20                      P1
The unallocated processes are: P2
```

- **Worst fit**

In this allocation technique, the process traverses the whole memory and always search for the largest hole/partition, and then the process is placed in that hole/partition. It is a slow process because it has to traverse the entire memory to search the largest hole.

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int noOfPartitions;
    cout << "Enter the number of patitions: ";
    cin >> noOfPartitions;
    int partitionMemory[noOfPartitions], tempPMemory[noOfPartitions];
    for (int i = 0; i < noOfPartitions; i++)
    {
        cout << "Partition " << i + 1 << ": ";
        cin >> partitionMemory[i];
        tempPMemory[i] = partitionMemory[i];
    }
    int noOfProcesses;
    cout << "Enter the number of processes: ";
    cin >> noOfProcesses;
    int processesMemory[noOfProcesses];
    for (int i = 0; i < noOfProcesses; i++)
    {
        cout << "Partition " << i + 1 << ": ";
        cin >> processesMemory[i];
    }
    string worstFit[noOfPartitions], notAllocatedProcesses;
    for (int i = 0; i < noOfPartitions; i++)
    {
        worstFit[i] = "X";
    }
    for (int i = 0; i < noOfProcesses; i++)
    {
        bool isAlloc = false;
        int maxSize;
        for (int j = 0; j < noOfPartitions; j++)
        {
            if (processesMemory[i] <= partitionMemory[j] && !isAlloc)
            {
                maxSize = j;
```

```
                isAlloc = true;
            }
            else if (processesMemory[i] <= partitionMemory[j])
            {
                if (partitionMemory[j] > partitionMemory[maxSize])
                {
                    maxSize = j;
                }
            }
        }
        if (isAlloc)
        {
            partitionMemory[maxSize] -= processesMemory[i];
            if (worstFit[maxSize] == "X")
            {

                worstFit[maxSize] = "P" + to_string(i + 1);
            }
            else
            {
                worstFit[maxSize] += ", P" + to_string(i + 1);
            }
            isAlloc = true;
        }
        else if (!isAlloc && notAllocatedProcesses.empty())
        {
            notAllocatedProcesses = "P" + to_string(i + 1);
        }
        else if (!isAlloc)
        {
            notAllocatedProcesses += ", P" + to_string(i + 1);
        }
    }
    cout << "\nPartitions\t\tWorst Fit\n";
    for (int i = 0; i < noOfPartitions; i++)
    {
        cout << tempPMemory[i] << "\t\t\t" << worstFit[i] << "\n";
    }
    if (notAllocatedProcesses.empty())
    {
        cout << "There are no unallocated processes!\n";
    }
    else
    {
        cout << "The unallocated processes are: " <<
notAllocatedProcesses << "\n";
    }
```

```
    return 0;
}
```

Output:

```
Enter the number of patitions: 4
Partition 1: 10
Partition 2: 50
Partition 3: 30
Partition 4: 20
Enter the number of processes: 3
Partition 1: 20
Partition 2: 60
Partition 3: 10

Partitions              Worst Fit
10                      X
50                      P1, P3
30                      X
20                      X
The unallocated processes are: P2
```