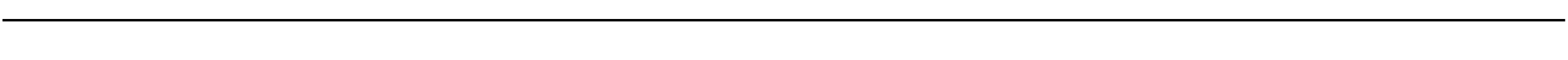
**Experiment 6**

**Title:** Memory Management: Memory Allocation Strategies in Dynamic Partitioning

Write a program to demonstrate the concept of dynamic partitioning placement algorithms i.e. Best Fit, First Fit, Next Fit and Worst-Fit.

**Estimated time to complete this experiment:** 2 hours



**Objective:** Learning about memory allocation strategies. Implementing a program for allocating memory blocks of unequal size to the requesting processes using various techniques and concluding about the most efficient technique for memory allocation.

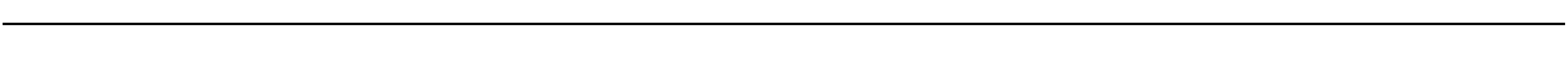


**Expected Outcome of Experiment:** To map jobs to blocks of memory of unequal size and mention the efficient techniques.



**Books/ Journals/ Websites referred:**

1. William Stallings, Operating System: Internals and Design Principles, Prentice Hall, 8thEdition, 2014, ISBN-10: 0133805913 • ISBN-13: 9780133805918.
2. Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Operating System Concepts, John Wiley &Sons, Inc., 9thEdition, 2016, ISBN 978-81-265-5427-0



**Pre Lab/ Prior Concepts:** Any Programming platform. Linked list management.

**Brief description:**

The real challenge of efficiently managing memory is seen in the case of a system which has multiple processes running at the same time. Since primary memory can be space-multiplexed, the memory manager can allocate a portion of primary memory to each process for its own use. However, the memory manager must keep track of which processes are running in which memory locations, and it must also determine how to allocate and deallocate available memory when new processes are created and when old processes complete execution.



**New Concepts to be learned:** Memory allocation strategies.



**Requirements:** PC with any programming platform.



**Theory:**

While various different strategies are used to allocate space to processes competing for memory, four of the most popular are Best fit, Worst fit, and First fit.

* + - 1. **Best fit:** The allocator places a process in the smallest block of unallocated memory in which it will fit.
  1. Advantages of Best-Fit Allocation :
     1. Memory Efficient.
     2. The operating system allocates the job minimum possible space in the memory, making memory management very efficient. To save memory from getting wasted, it is the best method.
  2. Disadvantages of Best-Fit Allocation :
     1. It is a Slow Process. Checking the whole memory for each job makes the working of the operating system very slow. It takes a lot of time to complete the work.
        1. **Worst fit:** The memory manager places a process in the largest block of unallocated memory available. The idea is that this placement will create the largest hold after the allocations, thus increasing the possibility that, compared to best fit, another process can use the remaining space.
           1. Advantages of Worst-Fit Allocation :

Since this process chooses the largest hole/partition, therefore there will be large internal fragmentation. Now, this internal fragmentation will be quite big so that other small processes can also be placed in that leftover partition.

* 1. Disadvantages of Worst-Fit Allocation :
     1. It is a slow process because it traverses all the partitions in the memory and then selects the largest partition among all the partitions, which is a time-consuming process.
        1. **First fit:** There may be many holes in the memory, so the operating system, to reduce the amount of time it spends analyzing the available spaces, begins at the start of primary memory and allocates memory from the first hole it encounters large enough to satisfy the request.
  2. Advantages:
     1. It is fast in processing.
     2. As the processor allocates the nearest available memory partition to the job, it is very fast in execution.
  3. Disadvantages:
     1. The processor ignores if the size of partition allocated to the job is very large as compared to the size of job or not. It just allocates the memory. As a result, a lot of memory is wasted and many jobs may not get space in the memory, and would have to wait for another job to complete.
     2. Internal fragmentation
        1. **Next Fit:** Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.



**Program:**

#include <iostream>

#include <vector>

using namespace std;

void Take1D(int n, vector<int> &array)

{

    for (int i = 0; i < n; i++)

    {

        int temp;

        cin >> temp;

        array.push\_back(temp);

    }

}

void BestFit(int NoOfProcess, vector<int> incoming, int NoOfHoles, vector<int> Holes)

{

    vector<int> allocation(NoOfProcess, -1); *// to store allocation info*

    for (int i = 0; i < NoOfProcess; i++)

    {

        int bestIdx = -1;

        for (int j = 0; j < NoOfHoles; j++)

        {

            if (Holes[j] >= incoming[i]) *// if hole is big enough*

            {

                if (bestIdx == -1 || Holes[bestIdx] > Holes[j]) *// if no hole is selected yet or if the current hole is smaller than the selected hole*

                {

                    bestIdx = j;

                }

            }

        }

        if (bestIdx != -1) *// a hole is found*

        {

            allocation[i] = bestIdx; *// allocate process i to hole bestIdx*

            Holes[bestIdx] -= incoming[i]; *// reduce the size of the hole*

        }

    }

    cout << "Process No.\tProcess Size\tBlock no." << endl;

    for (int i = 0; i < NoOfProcess; i++)

    {

        cout << i+1 << "\t\t" << incoming[i] << "\t\t";

        if (allocation[i] != -1)

        {

            cout << allocation[i]+1;

        }

        else

        {

            cout << "Not Allocated";

        }

        cout << endl;

    }

}

void WorstFit(int NoOfProcess, vector<int> incoming, int NoOfHoles, vector<int> Holes)

{

    vector<int> allocation(NoOfProcess, -1); *// to store allocation info*

    for (int i = 0; i < NoOfProcess; i++)

    {

        int worstIdx = -1;

        for (int j = 0; j < NoOfHoles; j++)

        {

            if (Holes[j] >= incoming[i]) *// if hole is big enough*

            {

                if (worstIdx == -1 || Holes[worstIdx] < Holes[j]) *// if no hole is selected yet or if the current hole is bigger than the selected hole*

                {

                    worstIdx = j;

                }

            }

        }

        if (worstIdx != -1) *// a hole is found*

        {

            allocation[i] = worstIdx; *// allocate process i to hole worstIdx*

            Holes[worstIdx] -= incoming[i]; *// reduce the size of the hole*

        }

    }

    cout << "Process No.\tProcess Size\tBlock no." << endl;

    for (int i = 0; i < NoOfProcess; i++)

    {

        cout << i+1 << "\t\t" << incoming[i] << "\t\t";

        if (allocation[i] != -1)

        {

            cout << allocation[i]+1;

        }

        else

        {

            cout << "Not Allocated";

        }

        cout << endl;

    }

}

void NextFit(int NoOfProcess, vector<int> incoming, int NoOfHoles, vector<int> Holes)

{

    vector<int> allocation(NoOfProcess, -1); *// to store allocation info*

    int start = 0;

    for (int i = 0; i < NoOfProcess; i++)

    {

        int j;

        for (j = start; j < NoOfHoles; j++)

        {

            if (Holes[j] >= incoming[i]) *// if hole is big enough*

            {

                allocation[i] = j; *// allocate process i to hole j*

                Holes[j] -= incoming[i]; *// reduce the size of the hole*

                start = j; *// update start for next iteration*

                break;

            }

        }

        if (allocation[i] == -1) *// no hole is found*

        {

*// search from beginning*

            for (j = 0; j < start; j++)

            {

                if (Holes[j] >= incoming[i]) *// if hole is big enough*

                {

                    allocation[i] = j; *// allocate process i to hole j*

                    Holes[j] -= incoming[i]; *// reduce the size of the hole*

                    start = j; *// update start for next iteration*

                    break;

                }

            }

        }

    }

    cout << "Process No.\tProcess Size\tBlock no." << endl;

    for (int i = 0; i < NoOfProcess; i++)

    {

        cout << i+1 << "\t\t" << incoming[i] << "\t\t";

        if (allocation[i] != -1)

        {

            cout << allocation[i]+1;

        }

        else

        {

            cout << "Not Allocated";

        }

        cout << endl;

    }

}

void FirstFit(int NoOfProcess, vector<int> incoming, int NoOfHoles, vector<int> Holes)

{

    vector<int> allocation(NoOfProcess, -1); *// initialize allocation vector to -1*

    int completed = 0;

    int i = 0;

    int missed = 0;

    int j = 0;

    while (1)

    {

        if (Holes[i] >= incoming[j])

        {

            Holes[i] = Holes[i] - incoming[i];

            allocation[j] = i; *// update allocation vector*

            completed++;

            i = -1;

            j++;

        }

        else

        {

            missed++;

        }

        if (missed == NoOfHoles)

        {

            cout << "Not Possible" << endl;

            cout << "As process of size " << incoming[j] << " cannot be allocated by FirstFit" << endl;

            return;

        }

        if (completed == NoOfProcess)

        {

            cout << "Possible" << endl;

*// print allocation information*

            for (int k = 0; k < NoOfProcess; k++) {

                cout << "Process " << k << " is allocated to hole " << allocation[k] << endl;

            }

            return;

        }

        i = (i + 1) % NoOfHoles;

    }

}

void FirstFit1(int NoOfProcess, vector<int> incoming, int NoOfHoles, vector<int> Holes)

{

    vector<int> allocation(NoOfProcess, -1); *// to store allocation info*

    int start = 0;

    for (int i = 0; i < NoOfProcess; i++)

    {

        int j;

        for (j = 0; j < NoOfHoles; j++)

        {

            if (Holes[j] >= incoming[i]) *// if hole is big enough*

            {

                allocation[i] = j; *// allocate process i to hole j*

                Holes[j] -= incoming[i]; *// reduce the size of the hole*

                start = j; *// update start for next iteration*

                break;

            }

        }

    }

    cout << "Process No.\tProcess Size\tBlock no." << endl;

    for (int i = 0; i < NoOfProcess; i++)

    {

        cout << i+1 << "\t\t" << incoming[i] << "\t\t";

        if (allocation[i] != -1)

        {

            cout << allocation[i]+1;

        }

        else

        {

            cout << "Not Allocated";

        }

        cout << endl;

    }

}

int main()

{

    int choice;

*// declare variables to store input*

    int NoOfProcess, NoOfHoles;

    vector<int> incoming, Holes;

    cout << "Enter the number of processes: ";

    cin >> NoOfProcess;

    Take1D(NoOfProcess, incoming);

    cout << "Enter the number of holes: ";

    cin >> NoOfHoles;

    Take1D(NoOfHoles, Holes);

    while (1)

    {

        cout << "Choose an algorithm:" << endl;

        cout << "1. First Fit" << endl;

        cout << "2. Best Fit" << endl;

        cout << "3. Worst Fit" << endl;

        cout << "4. Next Fit" << endl;

        cout << "5. Exit" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

            FirstFit1(NoOfProcess, incoming, NoOfHoles, Holes);

            break;

        case 2:

            BestFit(NoOfProcess, incoming, NoOfHoles, Holes);

            break;

        case 3:

            WorstFit(NoOfProcess, incoming, NoOfHoles, Holes);

            break;

        case 4:

            NextFit(NoOfProcess, incoming, NoOfHoles, Holes);

            break;

        case 5:

            return 0;

        default:

            cout << "Invalid choice. Please try again." << endl;

            break;

        }

    }

    return 0;

}

**Output:**

**Text

Description automatically generated**



**Conclusion:** Hence we have understoodhow processes are allocated using various memory allocation strategies.

