**Sachin Chaudhary**

**230701402**

**Ex. No. 10a:**

**Best Fit Memory Allocation in Python**

Aim:

To implement the Best Fit memory allocation technique using Python.

Algorithm:

Input memory blocks and processes with sizes.

Initialize all memory blocks as free.

Start by picking each process and find the minimum block size that can be assigned to the current process.

If a suitable block is found, assign it to the current process.

If no suitable block is found, leave that process and continue checking the further processes.

Program Code (Best Fit):

def best\_fit(memory\_blocks, processes):

n = len(memory\_blocks)

m = len(processes)

# Initialize memory block allocation

allocation = [-1] \* m

# Sort the memory blocks

for i in range(m):

min\_diff = float('inf')

block\_index = -1

# Find the smallest block that can fit the current process

for j in range(n):

if memory\_blocks[j] >= processes[i] and memory\_blocks[j] - processes[i] < min\_diff:

min\_diff = memory\_blocks[j] - processes[i]

block\_index = j

# If a block is found for the current process

if block\_index != -1:

allocation[i] = block\_index

memory\_blocks[block\_index] -= processes[i]

# Display the allocation results

print("Process No. Process Size Block no.")

for i in range(m):

if allocation[i] != -1:

print(f"{i + 1}\t\t{processes[i]}\t\t{allocation[i] + 1}")

else:

print(f"{i + 1}\t\t{processes[i]}\t\tNot Allocated")

# Example input

memory\_blocks = [500, 400, 300, 200, 100]

processes = [212, 417, 112, 426]

best\_fit(memory\_blocks, processes)

Sample Output (Best Fit):

Process No. Process Size Block no.

1 212 4

2 417 2

3 112 3

4 426 5

Result:

The program successfully implements the Best Fit memory allocation technique, finding the smallest available block that can accommodate each process.

**Ex. No. 10b:  
 First Fit Memory Allocation in C**

Aim:

To implement memory allocation using the First Fit method.

Algorithm:

Define max as 25 (the maximum number of blocks).

Declare variables: frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0, bf[max], ff[max].

Get the number of blocks and files, and their sizes.

In a loop, check if bf[j] != 1, if so, calculate temp = b[j] - f[i].

Find the highest block that fits and assign the block to the file.

Program Code (First Fit):

#include <stdio.h>

#define MAX 25

void firstFit(int block\_sizes[], int process\_sizes[], int m, int n) {

int allocation[m];

// Initialize all allocations as -1 (unassigned)

for (int i = 0; i < m; i++) {

allocation[i] = -1;

}

// Assign blocks to processes

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

if (block\_sizes[j] >= process\_sizes[i]) {

allocation[i] = j;

block\_sizes[j] -= process\_sizes[i];

break;

}

}

}

// Display the allocation results

printf("Process No. Process Size Block no.\n");

for (int i = 0; i < n; i++) {

if (allocation[i] != -1) {

printf("%d %d %d\n", i + 1, process\_sizes[i], allocation[i] + 1);

} else {

printf("%d %d Not Allocated\n", i + 1, process\_sizes[i]);

}

}

}

int main() {

int block\_sizes[] = {500, 400, 300, 200, 100}; // Memory blocks

int process\_sizes[] = {212, 417, 112, 426}; // Processes

int m = sizeof(block\_sizes) / sizeof(block\_sizes[0]);

int n = sizeof(process\_sizes) / sizeof(process\_sizes[0]);

firstFit(block\_sizes, process\_sizes, m, n);

return 0;

}

Sample Output (First Fit):

Process No. Process Size Block no.

1 212 1

2 417 2

3 112 3

4 426 Not Allocated

Result:

The program successfully implements the First Fit memory allocation technique, where each process is allocated the first block that is large enough to fit its size.