## AIM: Implementation of various memory allocation algorithms, (First fit, Best fit and Worst fit).

**Objective:** To study and implement various memory allocation algorithms.

#### Theory:

**First Fit:**

The First Fit memory allocation checks the empty memory blocks in a sequential manner. It means that the memory Block which found empty in the first attempt is checked for size. But if the size is not less than the required size then it is allocated.

#### Algorithm:

1. START.
2. At first get the no of processes and blocks.
3. Allocate the process by if(size of block>=size of the process) then allocate the process else move to the next block.
4. Now Display the processes with blocks and allocate to respective process.
5. STOP.

**Source code:**

#include<stdio.h>

void main()

{

int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

for(i = 0; i < 10; i++)

{

flags[i] = 0;

allocation[i] = -1;

}

printf("Enter no. of blocks: ");

scanf("%d", &bno);

printf("\nEnter size of each block: ");

for(i = 0; i < bno; i++)

scanf("%d", &bsize[i]);

printf("\nEnter no. of processes: ");

scanf("%d", &pno);

printf("\nEnter size of each process: ");

for(i = 0; i < pno; i++)

scanf("%d", &psize[i]);

for(i = 0; i < pno; i++) //allocation as per first fit

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

if(flags[j] == 0 && bsize[j] >= psize[i])

{

allocation[j] = i;

flags[j] = 1;

break;

}

//display allocation details

printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");

for(i = 0; i < bno; i++)

{

printf("\n%d\t\t%d\t\t", i+1, bsize[i]);

if(flags[i] == 1)

printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);

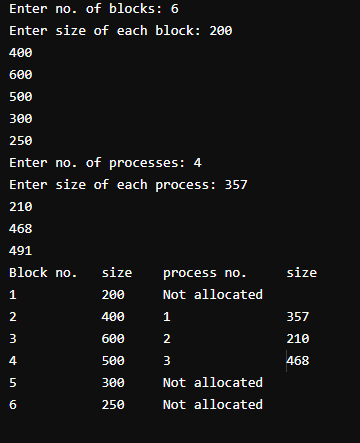
else

printf("Not allocated");

}

}

**Output:**



#### Worst Fit:

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.

**Algorithm**

1. Input memory blocks and processes with sizes.
2. Initialize all memory blocks as free.
3. Start by picking each process and find the maximum block size that can be assigned to current process i.e., find max(bockSize[1], blockSize[2],. blockSize[n]) >

processSize[current], if found then assign it to the current process.

1. If not then leave that process and keep checking the further processes.

**Source code:**

#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int processes)

{

// This will store the block id of the allocated block to a process

int allocation[processes];

int occupied[blocks];

// initially assigning -1 to all allocation indexes

// means nothing is allocated currently

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

allocation[i] = -1;

}

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

occupied[i] = 0;

}

// pick each process and find suitable blocks

// according to its size ad assign to it

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

{

int indexPlaced = -1;

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

{

// if not occupied and block size is large enough

if(blockSize[j] >= processSize[i] && !occupied[j])

{

// place it at the first block fit to accomodate process

if (indexPlaced == -1)

indexPlaced = j;

// if any future block is larger than the current block where

// process is placed, change the block and thus indexPlaced

else if (blockSize[indexPlaced] < blockSize[j])

indexPlaced = j;

}

}

// If we were successfully able to find block for the process

if (indexPlaced != -1)

{

// allocate this block j to process p[i]

allocation[i] = indexPlaced;

// make the status of the block as occupied

occupied[indexPlaced] = 1;

// Reduce available memory for the block

blockSize[indexPlaced] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocation[i] != -1)

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

else

printf("Not Allocated\n");

}

}

// Driver code

int main()

{

int blockSize[] = {100, 50, 30, 120, 35};

int processSize[] = {40, 10, 30, 60};

int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

int processes = sizeof(processSize)/sizeof(processSize[0]);

implimentWorstFit(blockSize, blocks, processSize, processes);

return 0;

}

**Output:**

Process No. Process Size Block no.

1 40 4

2 10 1

3 30 2

4 60 Not Allocated

#### Best Fit:

Best fit uses the best memory block based on the Process memory request. In best fit implementation the algorithm first selects the smallest block which can adequately fulfill the memory request by the respective process.

Algorithm:

* 1. Get no. of Processes and no. of blocks.
  2. After that get the size of each block and process requests.
  3. Then select the best memory block that can be allocated using the above definition.
  4. Display the processes with the blocks that are allocated to a respective process.
  5. Value of Fragmentation is optional to display to keep track of wasted memory.
  6. Stop.

**Source code:**

#include <stdio.h>

void implimentBestFit(int blockSize[], int blocks, int processSize[], int processes)

{

// This will store the block id of the allocated block to a process

int allocation[processes];

// initially assigning -1 to all allocation indexes

// means nothing is allocated currently

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

allocation[i] = -1;

}

// pick each process and find suitable blocks

// according to its size ad assign to it

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

{

int indexPlaced = -1;

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

{

if (blockSize[j] >= processSize[i])

{

// place it at the first block fit to accomodate process

if (indexPlaced == -1)

indexPlaced = j;

// if any future block is better that is

// any future block with smaller size encountered

// that can accomodate the given process

else if (blockSize[j] < blockSize[indexPlaced])

indexPlaced = j;

}

}

// If we were successfully able to find block for the process

if (indexPlaced != -1)

{

// allocate this block j to process p[i]

allocation[i] = indexPlaced;

// Reduce available memory for the block

blockSize[indexPlaced] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocation[i] != -1)

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

else

printf("Not Allocated\n");

}

}

// Driver code

int main()

{

int blockSize[] = {50, 20, 100, 90};

int processSize[] = {10, 30, 60, 30};

int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

int processes = sizeof(processSize)/sizeof(processSize[0]);

implimentBestFit(blockSize, blocks, processSize, processes);

return 0 ;

}

**Output:**

Process No. Process Size Block no.

1 10 2

2 30 1

3 60 4

4 30 4