#### Memory Allocation Methods

**Aim:**

Write a menu driven program for the implementation of the following Memory

Allocation Methods for **fixed partition**:

1. **First Fit** (b) **Best Fit** (c) **Worst Fit**

**Description:**

**First Fit Strategy**

The First Fit strategy allocates memory by scanning the partitions sequentially and assigning a process to the first partition that is large enough to accommodate it. In the code, this is done by iterating through each process and checking each partition in order until a suitable one is found. Once a partition is allocated to a process, it is marked as occupied. If no suitable partition is found for a process, it remains unallocated. This strategy is straightforward and typically faster since it stops searching as soon as a suitable partition is found. However, it can lead to internal fragmentation because it may leave small unused portions within partitions.

**Algorithm:**

1. start
2. For each process, set its allocated block to -1.
3. For each block, set its allocation status to 0 (unallocated).
4. For each process i from 0 to pr-1:
   * Set alloc to 0.
   * While alloc is 0:
     + For each block j from 0 to bl-1:
       - If the block is unallocated (blocks[j][1] == 0) and the block size is greater than or equal to the process size:
         * Allocate the block to the process (alloc = 1).
         * Mark the block as allocated (blocks[j][1] = 1).
         * Set the process's allocated block to j (page[i][1] = j).
         * Break the loop.
     + Break the loop.
5. Print the allocation result.
6. End

**Best Fit Strategy**

The Best Fit strategy aims to minimize internal fragmentation by finding the smallest partition that is large enough to accommodate each process. This requires checking all partitions to find the best possible fit for each process. In the code, for each process, it searches all partitions to find the one with the minimum difference between the partition size and the process size. Once the optimal partition is found, it is allocated to the process, and the partition is marked as occupied. This method can lead to better memory utilization compared to First Fit, but it may be slower due to the necessity of checking all partitions for each process. Despite aiming to reduce internal fragmentation, it can still leave small unusable portions within partitions.

**Algorithm:**

1. start
2. For each process, set its allocated block to -1.
3. For each block, set its allocation status to 0 (unallocated).
4. For each process i from 0 to pr-1:
   * Initialize cur to -1 and temp to -1.
   * For each block j from 0 to bl-1:
     + If the block is unallocated (blocks[j][1] == 0) and the block size is greater than or equal to the process size:
       - Calculate the difference (diff = blocks[j][0] - page[i][0]).
       - If temp is -1 or diff is less than temp:
         * Set temp to diff.
         * Set cur to j.
   * Allocate the block to the process (page[i][1] = cur).
   * Mark the block as allocated (blocks[cur][1] = 1).
5. Print the allocation result.
6. end

**Worst Fit Strategy**

The Worst Fit strategy allocates a process to the largest available partition that can accommodate it. This approach aims to reduce internal fragmentation by leaving larger remaining partitions unfragmented, which could be useful for accommodating larger processes later. In the code, this is achieved by searching for the partition with the maximum difference between the partition size and the process size for each process. If a suitable partition is found, it is allocated to the process, and the partition is marked as occupied. This strategy can help reduce internal fragmentation by keeping larger partitions intact, but it might leave smaller processes without suitable partitions if large partitions are unnecessarily split.

**Algorithm:**

1. start
2. For each process, set its allocated block to -1.
3. For each block, set its allocation status to 0 (unallocated).
4. For each process i from 0 to pr-1:
   * Initialize cur to 0 and diff to -1.
   * For each block j from 0 to bl-1:
     + If the block is unallocated (blocks[j][1] == 0) and the block size is greater than the process size:
       - Calculate the difference (temp = blocks[j][0] - page[i][0]).
       - If temp is greater than diff:
         * Set diff to temp.
         * Set cur to j.
   * If diff is not -1:
     + Allocate the block to the process (page[i][1] = cur).
     + Mark the block as allocated (blocks[cur][1] = 1).
5. Print the allocation result.
6. end

**Code:**

//Qn 10 Memory Allocation Strategies

//Ivin Mathew Kurian 8-5-24

#include<stdio.h>

int bl,pr;

int blocks[20][2];

int page[20][2];

void firstfit();

void bestfit();

void worstfit();

void main(){

    printf("Enter no of blocks:");

    scanf("%d",&bl);

    printf("Enter block sizes:\n");

    for(int i=0;i<bl;i++){

        printf("block %d size= ",i);

        scanf("%d",&blocks[i][0]);

        blocks[i][1]=0;

    }

    printf("Enter no of processes:");

    scanf("%d",&pr);

    printf("Enter process sizes\n");

    for(int i=0;i<pr;i++){

        printf("process %d size= ",i);

        scanf("%d",&page[i][0]);

        page[i][1]=-1;

    }

    printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n1:First Fit\n2:Best fit\n3:Worst fit\n4:Exit\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

    int opt=0;

    while (opt!=4){

        printf("\n\nEnter option:");

        scanf("%d",&opt);

        if(opt==1){

            firstfit();

        }

        else if(opt==2){

            bestfit();

        }

        else if(opt==3){

            worstfit();

        }

        else

            printf("Exitted.\n");

    }

}

void firstfit(){

    for(int i=0;i<pr;i++)

        page[i][1]=-1;

    for(int i=0;i<bl;i++)

        blocks[i][1]=0;

    for(int i=0;i<pr;i++){

        int alloc=0;

        while(alloc==0){

            for(int j=0;j<bl;j++){

                if(blocks[j][1]==0 && blocks[j][0]>=page[i][0]){

                    alloc=1;

                    blocks[j][1]=1;

                    page[i][1]=j;

                    break;

                }

            }

            break;

        }

    }

    printf("Allocated blocks:\n");

    for(int i=0;i<pr;i++){

        if(page[i][1]==-1)

            printf("Process %d : X\n",i);

        else

            printf("Process %d : Block %d\n",i,page[i][1]);

    }

}

void bestfit(){

    for(int i=0;i<pr;i++)

        page[i][1]=-1;

    for(int i=0;i<bl;i++)

        blocks[i][1]=0;

    for(int i=0;i<pr;i++){

        int cur=-1;

        int temp=-1;

        for(int j=0;j<bl;j++){

            int diff=-1;

            if(blocks[j][1]==0 && blocks[j][0]>=page[i][0]){

                diff=blocks[j][0]-page[i][0];

                if(temp==-1){

                    temp=diff;

                    cur=j;

                }

                else if(diff<temp){

                    temp=diff;

                    cur=j;

                }

            }

        }

        page[i][1]=cur;

        blocks[cur][1]=1;

    }

    printf("Allocated blocks:\n");

    for(int i=0;i<pr;i++){

        if(page[i][1]==-1)

            printf("Process %d : X\n",i);

        else

            printf("Process %d : Block %d\n",i,page[i][1]);

    }

}

void worstfit(){

    for(int i=0;i<pr;i++)

        page[i][1]=-1;

    for(int i=0;i<bl;i++)

        blocks[i][1]=0;

    for(int i=0;i<pr;i++){

        int cur=0;

        int diff=-1;

        for(int j=0;j<bl;j++){

            int temp=0;

            if(blocks[j][1]==0 && blocks[j][0]>page[i][0]){

                temp=blocks[j][0]-page[i][0];

                if(temp>diff){

                    diff=temp;

                    cur=j;

                }

            }

        }

        if(diff!=-1){

            page[i][1]=cur;

            blocks[cur][1]=1;

        }

    }

    printf("Allocated blocks:\n");

    for(int i=0;i<pr;i++){

        if(page[i][1]==-1)

            printf("Process %d : X\n",i);

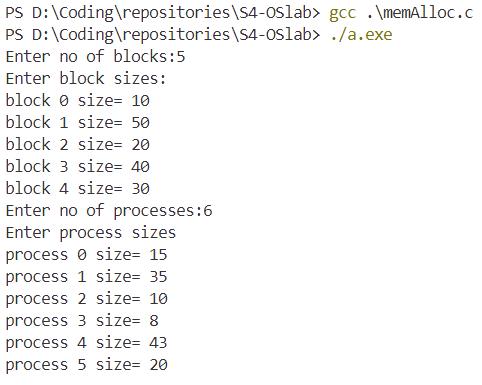
        else

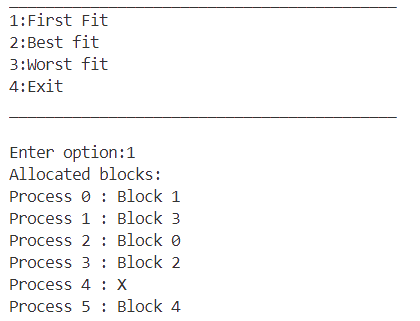
            printf("Process %d : Block %d\n",i,page[i][1]);

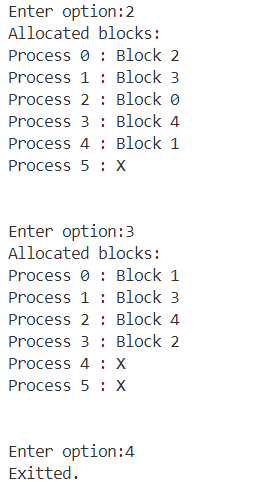
    }

}

**Output:**







**Result:**

The program has been executed and output has been verified.