EXPERIMENT

Simulation of continuous memory allocation policies

Objective: Write a program to simulate memory allocation policies:

a)First-fit algorithm b)Best-fit algorithm c)Next-fit algorithm d)Worst-fit algorithm

Software Used: C/C++/Java

Theory:

First Fit: This policy starts searching for a block of memory that has enough space for the incoming process from start (lower memory locations) and allocates it there. It is faster than best fit as in this we don't have to search the whole cache for the best location to allocate the process. But wastage of space can occur.

Best Fit: This policy allocates the process to the smallest available free block of memory. The best fit may result into a bad fragmentation, but in practice this is not commonly observed. Wastage of space is avoided at the cost of time for search the free space size equal to or just above the required size in the complete cache.

Next Fit: This policy makes use of a roving pointer. The pointer roves along the memory chain to search for a next fit. Thus each time a request is made the pointer begins searching from the place it last finished. This helps in, to avoid the usage of memory always from the head (beginning) of the free block chain. Wastage of space can occur as it doesn't necessarily select the best slot.

Worst Fit: This policy allocates the process to the largest available free block of memory. This leads to elimination of all large blocks of memory, thus requests of processes for large memory cannot be met. It is not practically used.

Program:

```
#include<stdio.h>
#include<stdlib.h>
void display(void);
void first(int, int);
void next(int, int);
void best(int, int);
void worst(int, int);

int f=0;
struct memory
   {
  int pnum;
  int space;
 }loc[6];

int main()
{
   int ch,no,size;
```

```
loc[0].pnum = 1;
loc[0].space = 2;

loc[1].pnum = -1;
loc[1].space = 4;

loc[2].pnum = 2;
loc[2].space = 6;

loc[3].pnum = -2;
loc[3].space = 3;

loc[4].pnum = 3;
loc[4].space = 1;

loc[5].pnum = -3;
loc[5].space = 2;

do
{
```

```
if (f==0)
{
 printf("Initial situtation\n");
 display();
 printf("\n");
 f=1;
}


printf("\nEnter process no. : ");
scanf("%d",&no);

printf("\nEnter size of the new process: ");
scanf("%d",&size);

printf("\nWhich algo\n1.First Fit\n2.Next Fit\n3.Best Fit\n4.Worst Fit\n5.Exit\n");
scanf("%d",&ch);

switch(ch)
{



 case 1:first(no, size);
```

```
        break;

  case 2:next(no, size);
        break;

    case 3:best(no, size);
        break;

  case 4:worst(no, size);
        break;

  case 5:exit(0);

  default:printf("\nInvalid Choice !!");
         break;

 }
}while(ch!=5);
}

void first(int no, int size)
```

```
{
 int i,flag=0;

 for(i=0;i<6;i++)
 {
  if(loc[i].space>=size && loc[i].pnum<0)
  {
   loc[i].pnum = no;
   loc[i].space -= size;
   display();
   flag = 1;
   break;
  }
 }

 if(flag==0)
 printf("\nInsufficient Memory !!!\n");
}

void next(int no, int size)



{
```

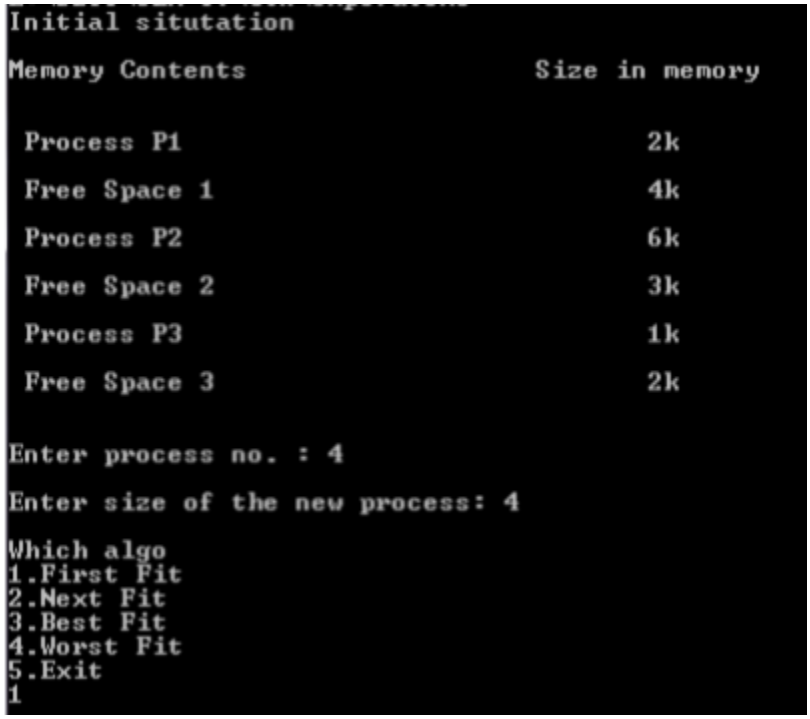
```
 int i = 5,flag=0;


 if(loc[5].space>=size && loc[i].pnum<0)
 {
  loc[i].pnum = no;
  loc[i].space -= size;
  display();
  flag=1;
 }


 else
 first(no, size);
}


void best(no, size)
{
 int i,flag=0;
 int min = 1;


 for(i=0;i<6;i++)




 {
```

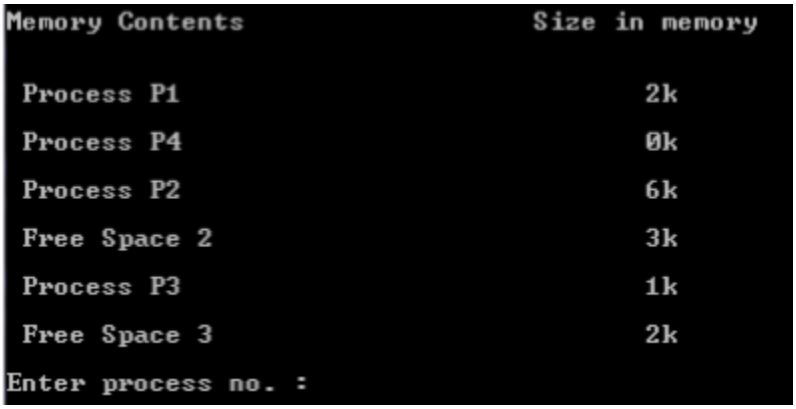
```
 if(loc[i].space>=size && loc[i].pnum<0)
 {
  flag=1;

  if(loc[i].space<=loc[min].space)
  min = i;
 }
}

loc[min].pnum = no;
loc[min].space -= size;

if(flag==1)
display();

else
printf("\nInsufficient Memory !!!\n");
}

void worst(no, size)



{
```

```
int i,flag=0;
int max = 1;

for(i=0;i<6;i++)
{
 if(loc[i].space>=size && loc[i].pnum<0)
 {
  flag=1;
  if(loc[i].space>=loc[max].space)
  max = i;
 }
}

loc[max].pnum = no;
loc[max].space -= size;

if(flag==1)
display();

else



printf("\nInsufficient Memory !!!\n");
```

```
}

void display()
{
 int i;

 printf("\nMemory Contents\t\t\t Size in memory\n");

 for(i=0;i<6;i++)
 {
  if(loc[i].pnum>0)
  printf("\n\n Process P%d\t\t\t\t%dk",loc[i].pnum,loc[i].space);

  else
  printf("\n\n Free Space %d\t\t\t\t%dk",loc[i].pnum*1,loc[i].space);
 }
   printf("\n");
}
```

Output:





Conclusion: The Simulation of continuous memory allocation policies was implemented successfully.