**EXERCISE-3**

Memory Management I**: Best Fit**

**Aim:**

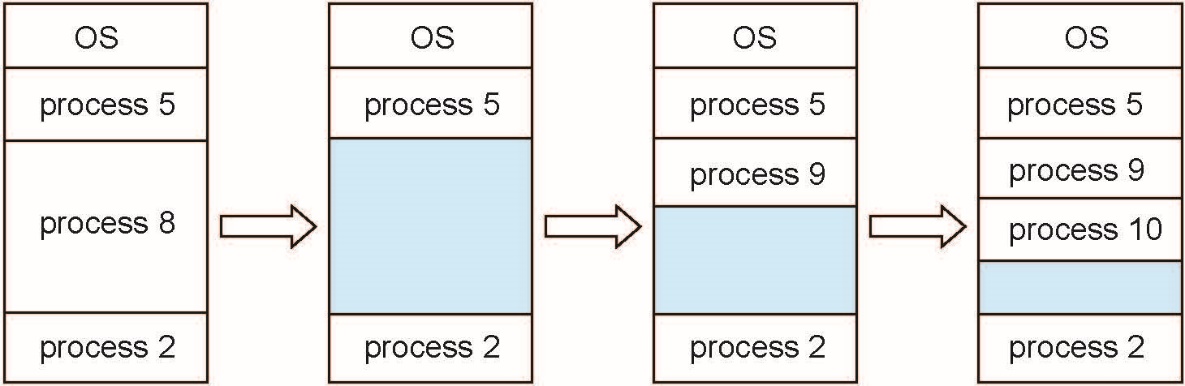
To implement Memory Management I**: Best Fit**

**Description:**

**Multiple-partition allocation**

* Degree of multiprogramming limited by number of partitions
* Variable-partition sizes for efficiency (sized to a given process’ needs)
* **Hole** – block of available memory; holes of various size are scattered throughout memory
* When a process arrives, it is allocated memory from a hole large enough to accommodate it
* Process exiting frees its partition, adjacent free partitions combined
* Operating system maintains information about:

a) allocated partitions b) free partitions (hole)



**Dynamic Storage-Allocation Problem:**

How to satisfy a request of size ***n*** from a list of free holes?

Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size

Produces the smallest leftover hole

**Program:**

#include<stdio.h>

//#include<conio.h>

#include<stdlib.h>

struct process

{

char name[20];

int size,sadd,min,max;

}p[20];

struct hole

{

int name;

int min,max,size;

}h[100],temp;

void main()

{

int m,tms,z,bs,d,condition,con,u,k,fs,ra,v;

int exfr=0,i=0, n=1,j=0;

//clrscr();

printf("\nenter total memory size:");

scanf("%d",&tms);

printf("\n hole0 starting is :0");

printf("\n hole0 ending is :%d",(tms-1));

fs=tms;

h[j].size=tms;

h[j].min=0;

h[j].max=tms-1;

do

{

i++;

printf("\nenter the process name:");

scanf("%s",p[i].name);

printf("\nenter the size of the %s process:",p[i].name);

scanf("%d",&p[i].size);

//printf("the entered size is: %d",p[i].size);

if(p[i].size<fs)

{

if(i==1)

{

m=h[j].max-p[i].size+1;

//printf("the value of m is:%d",m);

ra=rand();

// printf("random munber is :%d",ra);

p[i].sadd=ra%m;

//printf("the choice is :%d",(ra%m));

printf("the starting address is :%d",p[i].sadd);

p[i].min=p[i].sadd;

p[i].max=p[i].min+p[i].size-1;

printf("\nthe procees %d min is: %d",i,p[i].min);

printf("\nthe process %d max is: %d",i,p[i].max);

fs=m;

printf("\nthe amount of free size availabe:%d",fs);

if(m==p[i].sadd)

{

h[j].max=p[i].sadd-1;

h[j].size=h[j].max-h[j].min+1;

//printf("\nthe hole %d min is: %d",j,h[j].min);

//printf("\nthe hole %d max is: %d",j,h[j].max);

//printf("\nthe hole %d size is: %d",j,h[j].size);

}

else if(p[i].sadd==h[j].min)

{

h[j].min=p[i].max+1;

h[j].size=h[j].max-h[j].min+1;

// printf("\nthe hole %d min is: %d",j,h[j].min);

// printf("\nthe hole %d max is: %d",j,h[j].max);

//printf("\nthe hole %d size is: %d",j,h[j].size);

}

else

{

h[j+1].min=p[i].max+1;

h[j+1].max=h[j].max;

h[j].max=p[i].sadd-1;

h[j].size=h[j].max-h[j].min+1;

h[j+1].size=h[j+1].max-h[j+1].min+1;

n=n+1;

//printf("\nthe hole %d min is: %d",j,h[j].min);

//printf("\nthe hole %d max is: %d",j,h[j].max);

//printf("\nthe hole %d min is: %d",j+1,h[j+1].min);

//printf("\nthe hole %d max is: %d",j+1,h[j+1].max);

//printf("\nthe hole %d size is: %d",j,h[j].size);

//printf("\nthe hole %d size is: %d",j+1,h[j+1].size);

}

}

else

{

k=0;

do

{

if(h[k].size>=p[i].size)

{

p[i].min=h[k].min;

p[i].max=p[i].min+p[i].size-1;

h[k].min=p[i].max+1;

h[k].size=h[k].size-p[i].size;

fs=fs-p[i].size;

printf("\nthe procees %d min is: %d",i,p[i].min);

printf("\nthe process %d max is: %d",i,p[i].max);

condition=1;

}

else

{

printf("\nthe hole %d is not sufficeient because its size is %d",k,h[k].size);

k=k+1;

condition=0;

//printf("\n the value of k is %d",k);

}

}while(k<n&&condition==0);

}

}

else

{

i=i-1;

printf("\nmemory allocation is not possible ");

}

for(u=0;u<n;u++)

{

printf("\nthe hole %d min is: %d",u,h[u].min);

printf("\nthe hole %d max is: %d",u,h[u].max);

printf("\nthe hole %d size now is%d",u,h[u].size);

}

for(u=0;u<n-1;u++)

{

if(h[u].size>h[u+1].size)

{

temp=h[u+1];

h[u+1]= h[u];

h[u]=temp;

}

}

for(u=0;u<n;u++)

{

printf("\nafter sortingthe hole %d min is: %d",u,h[u].min);

printf("\n after sorting the hole %d max is: %d",u,h[u].max);

printf("\nafter sorting the hole %d size now is%d",u,h[u].size);

}

if(k==n&&condition==0)

{

printf("\n unable to allocate the memory due to external fragmentation");

con=2;

}

if (con!=2)

{

printf("\ndo you want to enter the another process1:enter 0:Noneed");

scanf("%d",&con);

}

}while(con==1);

printf("\n\tpname\tpmin\tpmax");

if(con==2)

{

for(z=1;z<i;z++)

{

printf("\n\t%s \t%d \t%d \n", p[z].name, p[z].min, p[z].max );

}

for(k=0;k<n;k++)

{

exfr=exfr+h[k].size;

}

printf("external fragmentation is %d:",exfr);

}

if(con==0)

{

for(z=1;z<i+1;z++)

{

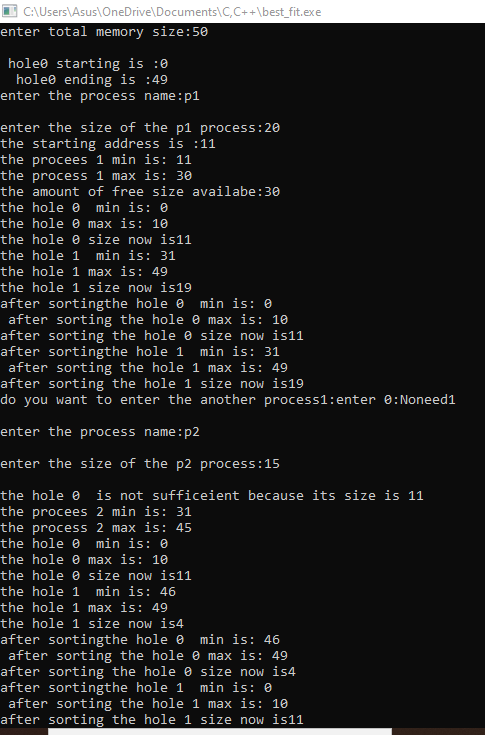
printf("\n\t%s \t%d \t%d \n", p[z].name, p[z].min, p[z].max );

}

}

}

**Output:**

****

