ITE2006 Operating Systems

Digital Assignment 4

Regno: 19BIT0096

Name: B. Joseph Prasanth Kumar Reddy

Q) You are provided with memory partitions of 250k, 400k, 95k, 500k respectively. How will you perform first fit, best fit and worst fit algorithms with process of 260k, 450k,100k and 650k (in order). Which algorithm makes the most efficient use of memory.

Ans:

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.

Advantage: It is the fastest searching Algorithm as we not to search only first block, we do not need to search a lot.

Disadvantage: The main disadvantage of First Fit is that the extra space cannot be used by any other processes. If the memory is allocated it creates large amount of chunks of memory space.

Aim:

To write a c program to implement first fit algorithm for memory management.

Algorithm:

1- Input memory blocks with size and processes with size.

2- Initialize all memory blocks as free.

3- Start by picking each process and check if it can be assigned to current block.

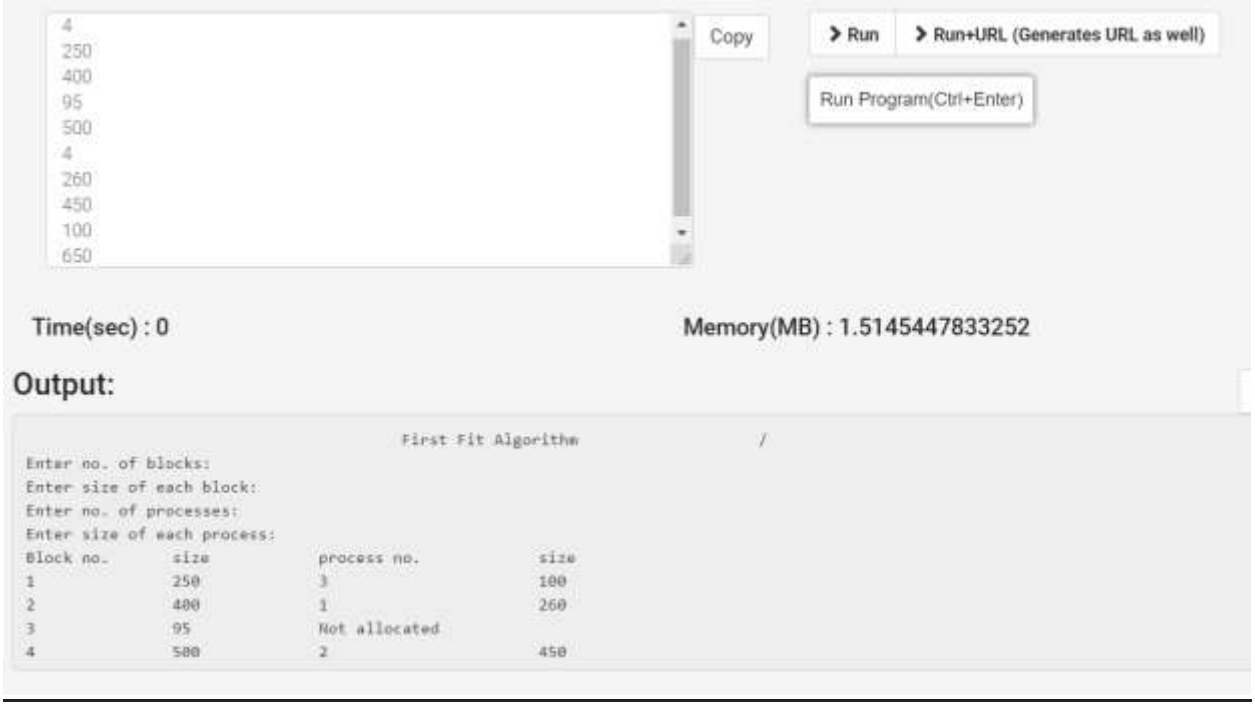
4- If size-of-process <= size-of-block if yes then assign and check for next process.

5- If not then keep checking the further blocks.

PROGRAM:

```
#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("\t\t\t\t\t First Fit Algorithm\t\t\t/\t\t\n");
	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");
	}
}
```

Input and Output:



Best Fit:

The Best Fit algorithm allocates the smallest free partition available in the memory that is sufficient enough to hold the process within the system. It searches the complete memory for available free partitions and allocates the process to the memory partition which is the smallest enough to hold the process. This is a very slow searching algorithm since it searches a lot of memory spaces to find the best fit memory for the process. Therefore, memory utilization is much better as compared to other memory management algorithms.

Aim:

To write a c program to implement best fit algorithm for memory management.

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 minimum block size that can be assigned to current process i.e., find min(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current].

4-If found then assign it to the current process.

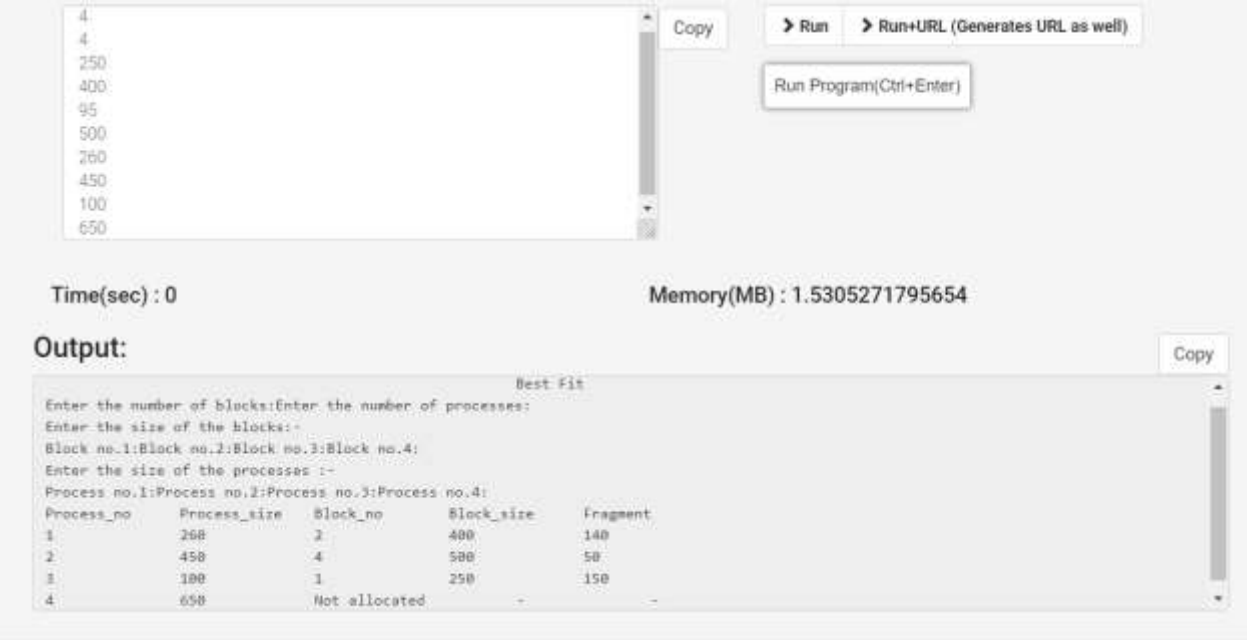
5- If not then leave that process and keep checking the further processes.

Program:

```
#include<stdio.h>
void main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999; static
int barray[20],parray[20];
printf("\n\t\t\tMemory Management Scheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of processes:");
scanf("%d",&np);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++){
printf("Block no.%d:",i);
scanf("%d",&b[i]);
}
printf("\nEnter the size of the processes :-\n");
for(i=1;i<=np;i++){
printf("Process no.%d:",i);
scanf("%d",&p[i]);
}
for(i=1;i<=np;i++){
for(j=1;j<=nb;j++){
if(barray[j]!=1){
temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp){
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
lowest=10000;
```

```
}
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i=1;i<=np;i++){
if(parray[i]!=0)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i
]],fragment[i]);
else
printf("\n%d\t\t%d\t\tNot allocated\t\t-\t\t-",i,p[i]);
}
}
```

Input and Output:



Worst Fit:

In this allocation technique, the process traverses the whole memory and always search for the largest hole/partition, and then the process is placed in that hole/partition. It is a slow process because it has to traverse the entire memory to search the largest hole.

Advantages: Since this process chooses the largest hole/partition, therefore there will be large internal fragmentation. Now, this internal fragmentation will be quite big so that other small processes can also be placed in that leftover partition.

Disadvantages: It is a slow process because it traverses all the partitions in the memory and then selects the largest partition among all the partitions, which is a time-consuming process.

Aim:

To write a c program to implement worst fit algorithm for memory management.

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].

4- If found then assign it to the current process.

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

Program:

```
#include<stdio.h>
#define max 25
void main(){
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0; static
int bf[max],ff[max];
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of Process:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++){
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the Process:-\n");
for(i=1;i<=nf;i++){
printf("Process %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++){
for(j=1;j<=nb;j++){
if(bf[j]!=1){
temp=b[j]-f[i];
```

```
if(temp>=0)
if(highest<temp){
ff[i]=j;
highest=temp;
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nProcess_no \tProcess_size \tBlock_no
\tBlock_size \tFragment");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[ i]);
}
```

Input:

```
4
4
250
400
95
500
260
450
100
650
```

Copy

Output:

```
Enter the number of blocks:Enter the number of Process:
Enter the size of the blocks:-
Block 1:Block 2:Block 3:Block 4:Enter the size of the Process:-
Process 1:Process 2:Process 3:Process 4:
Preocess_no     Process_size    Block_no    Block_size      Fragment
1       260     4       500     240
2       450     0       0       0
3       100     2       400     300
4       650     0       0       0
```
