***PAL Mini Project Report on***

***“Memory management- first fit, best fit and Worst fit*”**

**Group Members:**

**118A1067 Anandteertha Rao**

**118A1070 Amit Sabkale**

**\*CONTENTS\***

**1.Introduction of the topic**

**2.Project Goals**

**3.Problem definition**

**4.** **System component (H/W &amp; S/W)**

**5.code**

**6.Output**

**7.Conclusion**

***Introduction of the topic:***

The modern computer system has memory space divided into blocks of varying sizes. The operating system assigns these memory block to the different process demanding empty memory spaces. This memory is assigned in the main memory segment depending on the demanded memory and the empty memory slots present in the main memory.

Assigning memory blocks to the process is challenging as the primary memory is needed to be divided among the operating system, user process, and the operating system process. Therefore, the system uses different algorithms to allocate memory from the main memory segment. These algorithms are also known as the memory partitioning algorithms are broadly categorized under the following algorithms:

**1)First Fit**

**2)Best Fit**

**3)Worst Fit**

**Project Goals:**

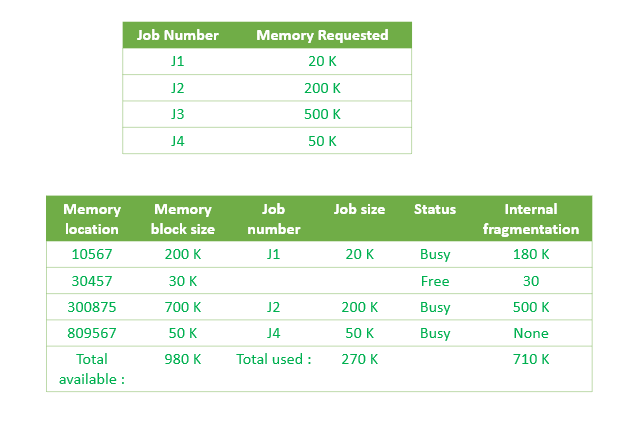
For Memory Management Best fit chooses the block that is closest in size to the request. First fit begins to scan memory from the beginning and chooses the first available block that is large enough. Worst fit begin to scan memory from the location of the last placement and chooses the next available block that is large enough.

**Problem definition:**

A Program to perform First fit, Best fit and Worst fit for Memory Management.

**1)First-Fit:**

This method keeps the free/busy list of jobs organized by memory location, low-ordered to high-ordered memory. In this method, first job claims the first available memory with space more than or equal to it’s size. The operating system doesn’t search for appropriate partition but just allocate the job to the nearest memory partition available with sufficient size.



As illustrated above, the system assigns J1 the nearest partition in the memory. As a result, there is no partition with sufficient space is available for J3 and it is placed in the waiting list.

**Advantages of First-Fit:**

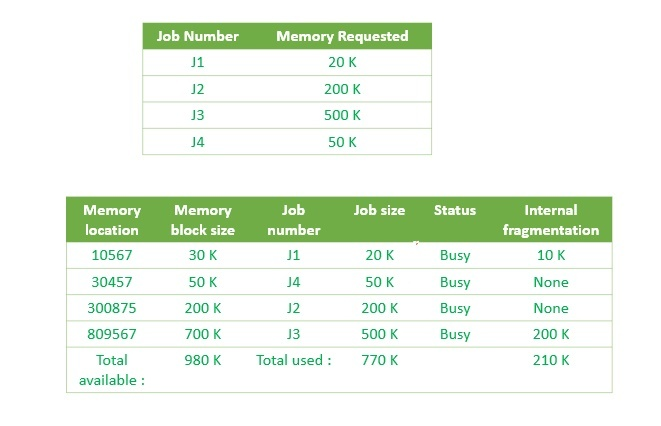
It is fast in processing. As the processor allocates the nearest available memory partition to the job, it is very fast in execution.

**Disadvantages of Fist-Fit:**

It wastes a lot of memory. The processor ignores if the size of partition allocated to the job is very large as compared to the size of job or not. It just allocates the memory. As a result, a lot of memory is wasted and many jobs may not get space in the memory, and would have to wait for another job to complete.

**2)Best-Fit:**

This method keeps the free/busy list in order by size – smallest to largest. In this method, the operating system first searches the whole of the memory according to the size of the given job and allocates it to the closest-fitting free partition in the memory, making it able to use memory efficiently. Here the jobs are in the order from smallest job to largest job.



As illustrated in above figure, the operating system first search throughout the memory and allocates the job to the minimum possible memory partition, making the memory allocation efficient.

**Advantages of Best-Fit:**

Memory Efficient. The operating system allocates the job minimum possible space in the memory, making memory management very efficient. To save memory from getting wasted, it is the best method.

**Disadvantages of Best-Fit:**

It is a Slow Process. Checking the whole memory for each job makes the working of the operating system very slow. It takes a lot of time to complete the work.

**3)Worst fit:**

In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit**.**

**Advantage: -**

Reduces the rate of production of small gaps.

**Disadvantage: -**

If a process requiring larger memory arrives at a later stage then it cannot be accommodated as the largest hole is already split and occupied.

**System component (H/W &amp; S/W):**

**SOFTWARE REQUIREMENTS:**

**1.Operating System: -**

Windows 7 and above, Linux Ubuntu, etc.

**2.Front End: -** TURBO C

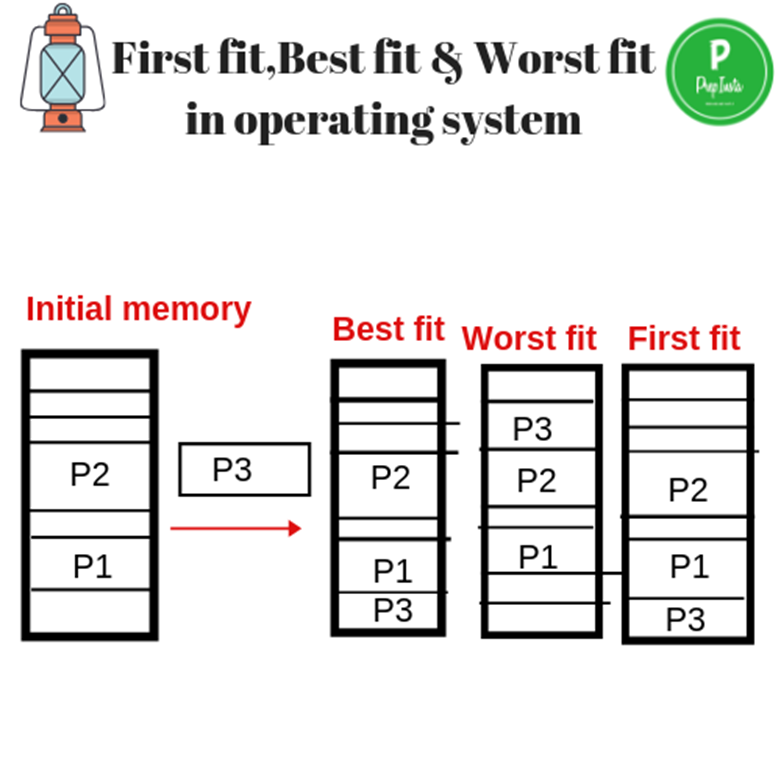
**HARDWARE SPECIFICATIONS:**

**1.** **Processor: -** i3 and above

**2.** **RAM: -** 2 GB and above

**3.Hard disk: -**

**-**No requirements because we are using TURBO C, it stores it’s programs in C drive, where the OS resides, enough space for the program should be required.

****

***CODE:***

*//WAP TO IMPLEMENT BEST FIT FIRST FIT AND WORST FIT ALGORITHMS OF*

*//MEMORY ALLOCATION .*

*#include<stdio.h>*

*#include<conio.h>*

*int memory\_partitions[20];*

*int processes[20];*

*int number\_of\_processes;*

*int number\_of\_memory\_partitions;*

*int ff[20]; //array for first fit*

*int bf[20]; //array for best fit*

*int wf[20]; //array for worst fit*

*int array[20]; //temporary array*

*int temp\_process[20]; //temp array to hold all the processes*

*int temp\_memory\_partitions[20]; //temp array to hold all the memory partitions*

*int min(int arr[20],int number\_of\_elements)*

*{*

*int i; //for iteration*

*int M=arr[0];*

*//clrscr();*

*for(i=0;i<number\_of\_elements;i++)*

*{*

*if(M>arr[i])*

*M=arr[i];*

*}*

*return M;*

*}*

*void first\_fit()*

*{*

*int i; //for iteration of processes array*

*int j; //for iteration of memory partition array*

*for(i=0;i<number\_of\_processes;i++)*

*{*

*for(j=0;j<number\_of\_memory\_partitions;j++)*

*{*

*if(memory\_partitions[j]>processes[i])*

*{*

*ff[i]=memory\_partitions[j];*

*memory\_partitions[j]-=processes[i];*

*break;*

*}*

*}*

*if(j==number\_of\_memory\_partitions)*

*ff[i]=9999;*

*}*

*//printing the first fit array*

*for(i=0;i<number\_of\_processes;i++)*

*{*

*if(ff[i]!=9999)*

*{*

*printf("\nprocess %d = %d gets the memory =%d\n",*

*i+1,processes[i],ff[i]);*

*}*

*else*

*{*

*printf("\nprocess %d = %d has to wait\n",i+1,*

*processes[i]);*

*}*

*}*

*}*

*void bestfit()*

*{*

*int i; //for iteration of processes array*

*int j; //for iteration of memory partion array*

*int count; //to count the number of elements which has*

*//memory\_partition greater than processes*

*int k;*

*for(i=0;i<number\_of\_processes;i++)*

*{*

*count=0;*

*//initialise the array*

*for(k=0;k<20;k++)*

*array[k]=9999;*

*for(j=0;j<number\_of\_memory\_partitions;j++)*

*{*

*if(memory\_partitions[j]>processes[i])*

*{*

*array[j]=memory\_partitions[j];*

*count++;*

*}*

*}*

*/\*printf("\nthe array = \n");*

*for(k=0;k<count;k++)*

*printf("%d\t",array[k]);\*/*

*if(count!=0){*

*bf[i]=min(array,number\_of\_memory\_partitions);*

*for(j=0;j<number\_of\_memory\_partitions;j++)*

*{*

*if(memory\_partitions[j]==bf[i])*

*{*

*memory\_partitions[j]-=processes[i];*

*}*

*}*

*}*

*else*

*{*

*bf[i]=9999;*

*}*

*}*

*//printing the best fit*

*for(i=0;i<number\_of\_processes;i++)*

*{*

*if(bf[i]!=9999)*

*{*

*printf("\nprocess %d = %d gets memory =%d\n",*

*i+1,processes[i],bf[i]);*

*}*

*else*

*{*

*printf("\nprocess %d = %d has to wait\n",i+1,processes[i]);*

*}*

*}*

*}*

*void worstfit()*

*{*

*int i; //for iteration of the processes array*

*int j; //for iteraion of the memory partition array \*

*int max;*

*int k;*

*for(i=0;i<number\_of\_processes;i++)*

*{*

*max=memory\_partitions[0];*

*for(j=0;j<number\_of\_memory\_partitions;j++)*

*{*

*if(max<memory\_partitions[j])*

*{*

*max=memory\_partitions[j];*

*k=j;*

*}*

*}*

*if(max>=processes[i])*

*{*

*wf[i]=memory\_partitions[k];*

*memory\_partitions[k]-=processes[i];*

*}*

*else*

*{*

*wf[i]=9999;*

*}*

*}*

*for(i=0;i<number\_of\_processes;i++)*

*{*

*if(wf[i]!=9999)*

*printf("\nprocess %d = %d gets memory =%d\n",*

*i+1,processes[i],wf[i]);*

*else*

*printf("\nprocess %d = %d has to wait\n",i+1,processes[i]);*

*}*

*}*

*void main()*

*{*

*int i; //for iteration*

*int choice; //for switch case*

*clrscr();*

*printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");*

*printf("\nenter the number of processes\n");*

*scanf("%d",&number\_of\_processes);*

*printf("\nenter the number of memory partitions\n");*

*scanf("%d",&number\_of\_memory\_partitions);*

*printf("\nenter the processes size one by one\n");*

*for(i=0;i<number\_of\_processes;i++)*

*{*

*printf("\nenter the process %d size : ",i+1);*

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

*temp\_process[i]=processes[i];*

*}*

*printf("\nenter the memory partitions one by one\n");*

*for(i=0;i<number\_of\_memory\_partitions;i++)*

*{*

*printf("\nenter the memory partition %d size : ",i+1);*

*scanf("%d",&memory\_partitions[i]);*

*temp\_memory\_partitions[i]=memory\_partitions[i];*

*}*

*clrscr();*

*printf("\nour process table now becomes\n");*

*printf("\nprocess | size\n");*

*for(i=0;i<number\_of\_processes;i++)*

*{*

*printf("\nP%d | %d\n",i+1,processes[i]);*

*}*

*//printf("\nour memory partition table is :\n");*

*do{*

*//clrscr();*

*for(i=0;i<number\_of\_processes;i++)*

*{*

*processes[i]=temp\_process[i];*

*}*

*for(i=0;i<number\_of\_memory\_partitions;i++)*

*{*

*memory\_partitions[i]=temp\_memory\_partitions[i];*

*}*

*printf("\n1.first fit\n2.best fit\n3.worst fit\n4.exit\n");*

*printf("\nenter your choice\n");*

*scanf("%d",&choice);*

*switch(choice)*

*{*

*case 1:*

*printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");*

*printf("\nfirst fit\n");*

*first\_fit();*

*//printf("\n\n\n");*

*break;*

*case 2:*

*printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");*

*printf("\nbest fit\n");*

*bestfit();*

*break;*

*case 3:*

*printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");*

*printf("\nworst fit\n");*

*worstfit();*

*break;*

*case 4:*

*printf("\n end of program \n");*

*break;*

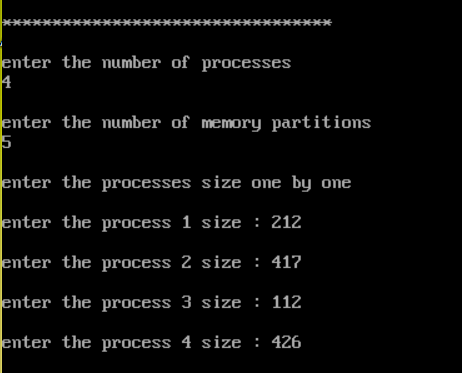
*}*

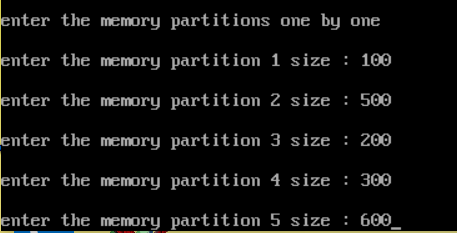
*}while(choice!=4);*

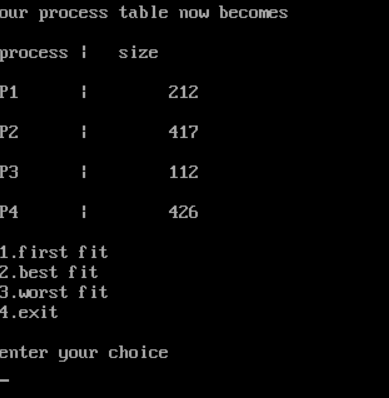
*getch();*

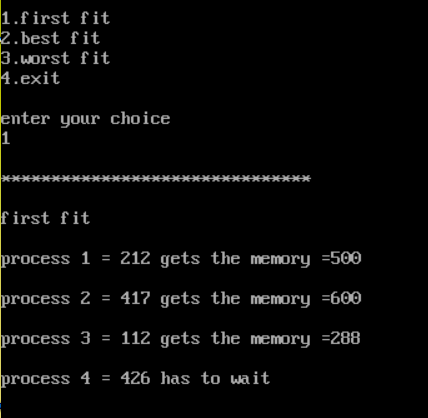
*}*

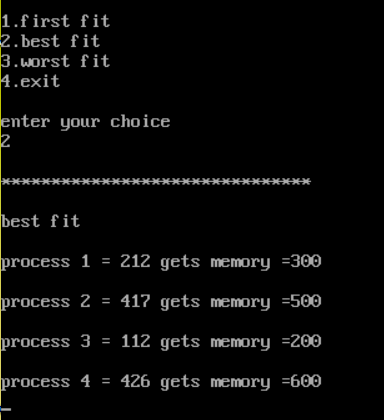
***OUTPUT:***

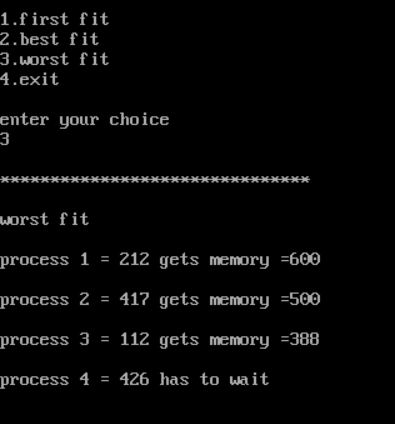
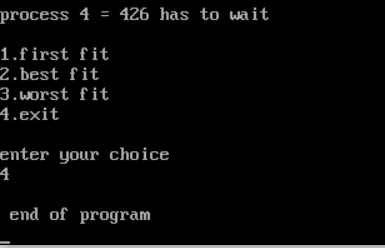










**Conclusion: -**

Hence, Program have been successfully implemented for First fit, Best fit and Worst fit memory management.