|  |
| --- |
| **Subject: Operating System Sub Teacher: Prof. S.S.Shethe.**  **Class: S. E. Computer Engg Roll no :-**  **Practical No.: Date:** |

**Title -** Write a program to implementmemory management algorithm for first fit, best fit & worst fit.

**Aim - S**tudy of to implement memory management and algorithm of

1. First fit
2. Worst fit
3. Burst fit

**Theory -**

**Memory Management -**

Memory management is the process of controlling and coordinating computer memory, assigning portions calledblocks. To various running programs to optimize overall systemperformance. Memory management resid.es in hardware, in the Os (operating system), and in programs and applications.

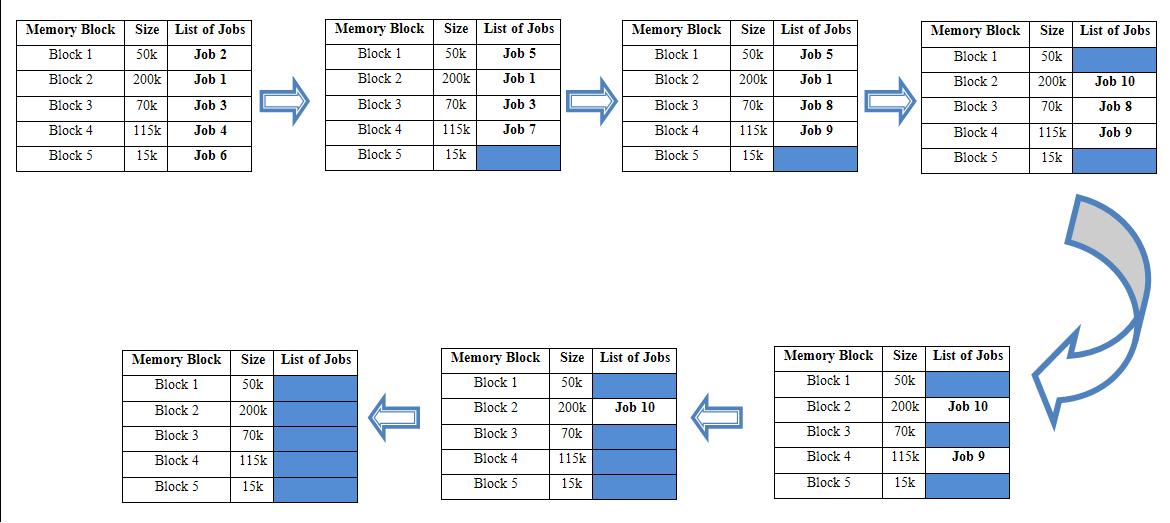
In the OS, memory management involves the allocation (and constant reallocation) of specific memory blocks to individual programs as user demands change. At the application level, memory management ensures the availability of adequate memory for the objects and data structures of each running program at all times. Application memory management combines two related tasks, known as allocation and recycling.

* When the program requests a block of memory, a part of the memory manager called the allocator assigns that block to the program.
* When a program no longer needs the data in previously allocated memory blocks, those blocks become available for reassignment. This task can be done manually (by the programmer) or automatically (by the memory manager).

One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided. When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate. Best-fit strategy chooses the block that is closest in size to the request. First-fit chooses the first available block that is large enough. Worst-fit chooses the largest available block.

**1) F I R S T – F I T**

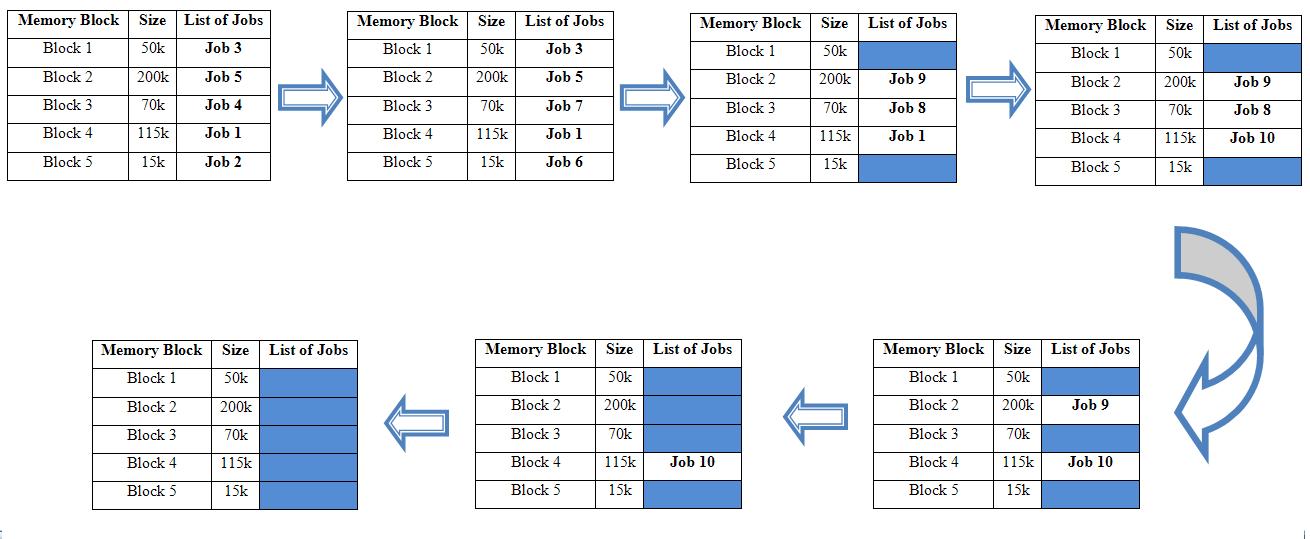
First-fit memory allocation is faster in making allocation but leads to memory waste. The illustration below shows that on the first cycle, job 1 to job 4 are submitted first while job 6 occupied block 5 because the remaining memory space is enough to its required memory size to be process. While  job 5 is in waiting queue because the memory size in block 5 is not enough for the job 5 to be process. Then on the next cycle, job 5 replace job 2 on block 1 and job 7 replace job 4 on block 4 after both job 2 and job 4 finish their process. Job 8 is in waiting queue because the remaining block is not enough to accommodate the memory size of job 8. On the third cycle, job 8 replace job 3 and job 9 occupies block 4 after processing job 7.  While Job 1 and job 5 remain on its designated block. After the third cycle block 1 and block 5 are free to serve the incoming jobs but since there are 10 jobs so it will remain free. And job 10 occupies block 2 after job 1 finish its turns. On the other hand, job 8 and job 9 remain on their block. Then on the fifth cycle, only job 9 and job 10 are to be process while there are 3 memory blocks free. In the sixth cycle, job 10 is the only remaining job to be process and lastly in the seventh cycle, all jobs are successfully process and executed and all the memory blocks are now free.



**2 ) B E S T - F I T**

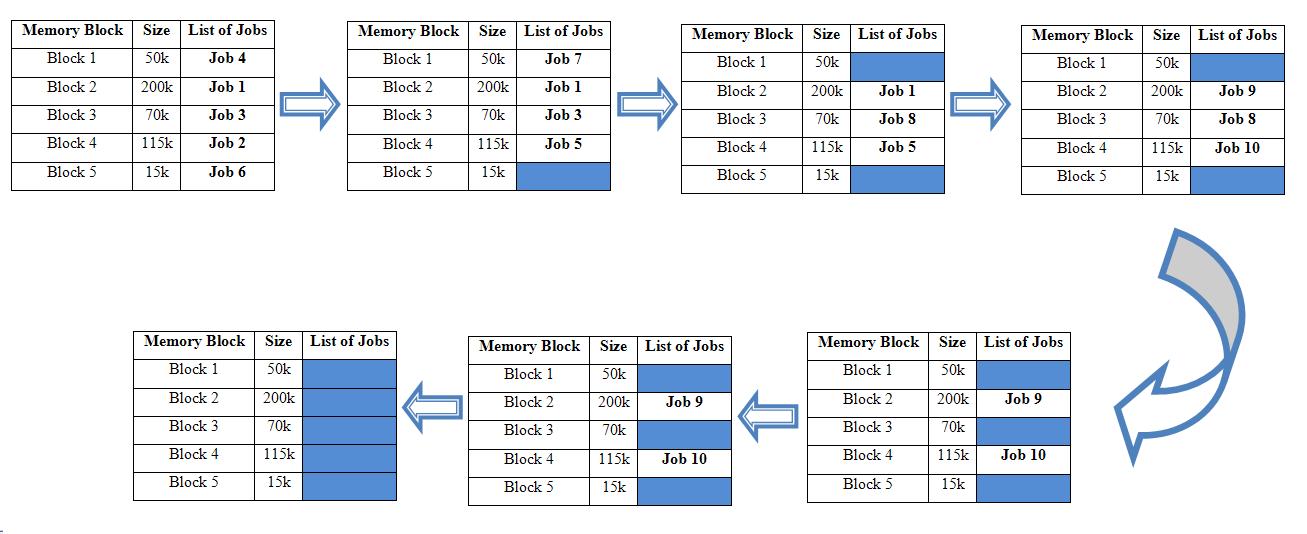
Best-fit memory allocation makes the best use of memory space but slower in making allocation. In the illustration below, on the first processing cycle, jobs 1 to 5 are submitted and be processed first. After the first  cycle, job 2 and 4 located on block 5 and block 3 respectively and both having one turnaround are replace by job 6 and 7 while  job 1, job 3 and job 5 remain on their designated block. In the third cycle, job 1 remain on block 4, while job 8 and job 9 replace job 7 and job 5 respectively (both having 2 turnaround). On the next cycle, job 9 and job 8 remain on their block while job 10 replace job 1 (having 3 turnaround). On the fifth cycle only job 9 and 10 are the remaining jobs to be process and there are 3 free memory blocks  for the incoming jobs.

But since there are only 10 jobs, so it will remain free. On the sixth cycle, job 10 is the only remaining job to be process and finally on the seventh cycle, all jobs are successfully process and executed and all the memory blocks are now free.



**3)W O R S T - F I T**

Worst-fit memory allocation is opposite to best-fit. It allocates free available block to the new job and it is not the best choice for an actual system. In the illustration, on the first cycle, job 5 is in waiting queue while job 1 to job 4 and job 6 are the jobs to be first process. After then,  job 5 occupies the free block replacing job 2. Block 5 is now free to accommodate the next job which is job 8 but since the size in block 5 is not enough for job 8, so job 8 is in waiting queue. Then on the next cycle, block 3 accommodate job 8 while job 1 and job 5 remain on their memory block. In this cycle, there are 2 memory blocks are free. In the fourth cycle, only job 8 on block 3 remains while job 1 and job 5 are respectively replace by job 9 and job 10. Just the same in the previous cycle, there are still two free memory blocks. At fifth cycle, job 8 finish its job while the job 9 and job 10 are still on block 2 and block 4 respectively and there is additional memory block free. The same scenario happen on the sixth cycle. Lastly, on the seventh cycle, both job 9 and job 10 finish its process and in this cycle, all jobs are successfully process and executed. And all the memory blocks are now free.



**CONCLUSION:**

In this practical we learn how to implement memory management algorithms for

First fit, Best fit and Worst fit.

//\* \*\*\*\*\*\*\*\*\*\*\*\*\*PROGRAM FOR MEMORY MANAGEMENT\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*//

#include<stdio.h>

#include<conio.h>

#define MAX\_MP 20

#define MAX\_P 20

int block[MAX\_MP];

intoccupiedMP[MAX\_MP];

int size;

int P[MAX\_P];

intoccupiedP[MAX\_P];

intsizeP;

int i;

void main()

{

void input();

voidfirstfit();

voidbestfit();

voidworstfit();

void display();

clrscr();

input();

printf("\n\n FIRST FIT");

firstfit();

display();

printf("\n\nBEST FIT");

bestfit();

display();

printf("\n\n WORST FIT");

worstfit();

display();

getch();

}

void input()

{

printf("\n\n Enter how many memory blocks you want to create: ");

scanf("%d",&size);

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

{

printf("\n Enter the size of memory block %d (in KB): ",i+1);

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

occupiedMP[i]=-1;

}

printf("\n\n Enter the no. of processes: ");

scanf("%d",&sizeP);

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

{

printf("\n Enter the size of process %d(in KB):",i+1);

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

}

}

void display() //check availability of memory partition

{

int i;

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

{

if(occupiedP[i]!=-1)

printf("\n Process %d fit into the memory block %d",i+1,occupiedP[i]+1);

else

printf("\n Process %d has no memory block to fit in.",i+1);

}

}

voidfirstfit() //check availability of first free memory partition

{

inti,indexMP;

intgetMPFF(int block[],intoccupiedMP[],intsize,int P[],intpindex);

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

occupiedMP[i]=-1;

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

occupiedP[i]=-1;

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

{

indexMP=getMPFF(block,occupiedMP,size,P,i);

if(indexMP!=-1)

occupiedMP[indexMP]=i;

occupiedP[i]=indexMP;

}

}

intgetMPFF(int block[],intoccupiedMP[],intsize,int P[],intpindex)

{

int i;

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

{

if(occupiedMP[i]==-1 && block[i]>P[pindex])

return i;

}

return i;

}

voidbestfit() //check availability of best tightest free memory partition

{

int i;

intindexMP;

intgetMPBF(int block[],intoccupiedMP[],intSIze,int P[],intplndex);

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

occupiedMP[i]=-1;

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

occupiedMP[i]=-1;

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

{

indexMP=getMPBF(block,occupiedMP,size,P,i);

if(indexMP!=-1)

occupiedMP[indexMP]=i;

occupiedP[i]=indexMP;

}

}

intgetMPBF(int block[],intoccupiedMP[],intsize,int p[],intplndex)

{

int i;

intprevMinHighestIndex;

prevMinHighestIndex=-1;

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

{

if(occupiedMP[i]==-1&&block[i]>p[plndex])

{

if(prevMinHighestIndex==-1 || block[i]<block[prevMinHighestIndex])

prevMinHighestIndex=i;

}

}

returnprevMinHighestIndex;

}

voidworstfit() //check availability of biggest free memory partition

{

int i;

intindexMP;

intgetMPWF(int block[],intoccupiedMP[],intsize,int p[],intplndx);

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

occupiedMP[i]=-1;

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

occupiedMP[i]=-1;

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

{

indexMP=getMPWF(block,occupiedMP,size,P,i);

if(indexMP!=-1)

occupiedMP[indexMP]=i;

occupiedP[i]=indexMP;

}

}

intgetMPWF(int block[],int Occupied[],intsize,int P[],intplndex)

{

int i;

intprevMaxHighestIndex=-1;

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

{

if(Occupied[i]==-1&& block[i]>P[plndex])

{

if(prevMaxHighestIndex==-1 || block[i]>block[prevMaxHighestIndex])

prevMaxHighestIndex=i;

}

}

returnprevMaxHighestIndex;

}

**OUTPUT:**

Enter how many memory blocks you want to create: 3

Enter the size of memory block 1 (in KB): 100

Enter the size of memory block 2 (in KB): 200

Enter the size of memory block 3 (in KB): 300

Enter the no. of processes: 3

Enter the size of process 1(in KB):90

Enter the size of process 2(in KB):150

Enter the size of process 3(in KB):100

FIRST FIT

Process 1 fit into the memory block 1

Process 2 fit into the memory block 2

Process 3 fit into the memory block 3

BEST FIT

Process 1 fit into the memory block 1

Process 2 fit into the memory block 2

Process 3 fit into the memory block 3

WORST FIT

Process 1 fit into the memory block 3

Process 2 fit into the memory block 2

Process 3 has no memory block to fit in.