OPERATING SYSTEM PROJECT REPORT

MEMORY MANAGEMENT SIMULATOR

SUBMITED BY: GROUP 3

SURAJ PANDIT(2021A1R164)

AMANDEEP SINGH (2021A1R168)

VANSHAK CHHABRA (2021A1R153)

SAHIL CHIB (2021A1R160)

SEMESTER 3rd

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
MIET(AUTONOMOUS),JAMMU

ACKNOWLEDGEMENT

Through this section of our report, we want to present our gratitude towards our institute MIET. From here, we are able to work on projects that require you to analyses and solve a problem that exists in the real world. It is an experience that one can achieve rarely and we are happy and thankful to get that chance here. We would also like to thank our mentors Asst. Prof. Sourabh Sharma and Asst. Prof. Pragti Jamwal for being a guiding force throughout this period.

CONTENT

S.NO	TITLE	PAGE NO.
1	PROJECT TITLE	1
2	ACKNOWLEDGEMENT	2
3	CONTENTS	3
4	ABOUT	4-9
5	CODE	9-13
6	OUTPUTS	14-15
7	CONCLUSION	16
8	REFRENCE	16

TITLE: "To design the memory management simulator which simulates the way an Operating System uses to manage the process memory using different contiguous memory allocation schemes (Next Fit, Best Fit, and Worst Fit) and a non-contiguous memory allocation scheme in Linux Environment."

ABSTRACT

Memory allocation is the process of assigning a portion of memory to a process for execution. Partitioning makes easier to organise memory requirement of different processes. It is accomplished through a procedure called memory management. Memory requirement of process may be either contiguous allocation or noncontiguous allocation. If process requirement is contiguous, then consecutive blocks of memory have to be allocated to the process. If the requirement is non-contiguous, then process may be allocated the blocks of memory scattered all around the memory space. The efficiency of memory management depends on the utilization of memory by the process. If the process is allotted exactly the amount of memory requested by it, then it results in the maximum utilization. On the other hand, if the process is allotted more memory than requested, then excess memory is wasted as it cannot be used by another process. Such wastage of memory is referred to as internal fragmentation that has to be minimized. If the process does not get its required contiguous memory due to total available free memory is in non-contiguous form, then, it is referred to as external fragmentation. This incident should also be avoided. All these aspects should be considered while computing the memory utilization. In this paper, we will discuss about some of the popular memory allocation algorithms that are, first fit, best fit and worst fit for fixed sized and variable sized partition of contiguous nature, their performance will be analysed along with internal and external fragmentation and the whole procedure of an algorithm will be illustrated with proper diagrams. The main objective of this paper is to determine the most efficient algorithm in contiguous memory allocation that as minimum fragmentation.

KEYWORDS: best fit, first fit, worst fit, Non-Contiguous Memory Allocation, Contiguous Memory Allocation, Internal Fragmentation, External Fragmentation.

INTRODUCTION:

Memory is central to the operation of a computer system. It consists of a large array of words or bytes each with its own address. In uniprogramming system, main memory has two parts one for the operating system and another part is for the program currently being executed. In the multiprogramming system, the memory part of the user is further divided into accommodate processes. The task of the subdivision is cannot out by the operating system and is known as memory management.

Memory management techniques:

The memory management techniques are divided into two parts...

1. **Uniprogramming:**

In the uniprogramming technique, the RAM is divided into two parts one part is for the

resigning the operating system and other portion is for the user process. Here the border

register is used which contain the last address of the operating system parts. The operating system will compare the user data addresses with the fence register and if it is different that means the user is not entering in the OS area. Border register is also called boundary register and is used to prevent a user from entering

in the operating system area. Here the CPU

utilization is very poor and hence multiprogramming is used.

2. **Multiprogramming:**

In the multiprogramming, the multiple users can share the memory simultaneously. By multiprogramming we mean there will be more than one process in the main memory and if the

running process wants to wait for an event like I/O then instead of sitting ideal CPU will make a context switch and will pick another process.

a. Contiguous memory allocation

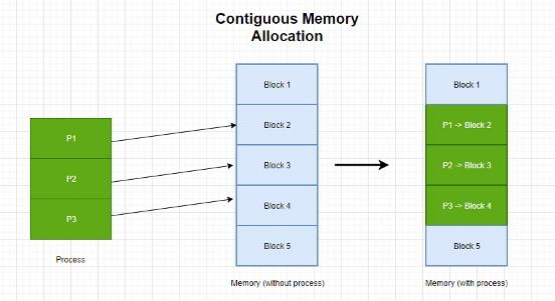
b. Non-contiguous memory allocation

Contiguous memory allocation:

Contiguous memory allocation is a classical memory allocation model. Here, a system assigns consecutive memory blocks (that is, memory blocks having consecutive addresses) to a process.

Contiguous memory allocation is one of the oldest memory allocation methods. Here's how it works: when a process needs to execute, memory is requested by the process. The size of the process is compared with the amount of contiguous main memory available to execute the process.

If sufficient contiguous memory is found, the memory is allocated and the process starts its execution. Otherwise, the process is added to a queue of waiting processes until sufficient free contiguous memory is available.



Swapping:

The kernel swaps out a pr ocess that is not in the running state by writing out its code and data space to a swapping area on the disk. The swapped out process is brought back into memory before it is due for another burst of CPU time. A basic issue in swapping is whether a swapped in process should be loaded back into the same memory area that it occupied before it was swapped out. If so, it's swapping in depends on swapping out of some other process that may have been allocated that memory area in the meanwhile. It would be useful to be able to place the swappedin process elsewhere in memory; however, it would amount to dynamic relocation of the process to a new memory area. As mentioned earlier, only computer systems that provide a relocation register can achieve it.

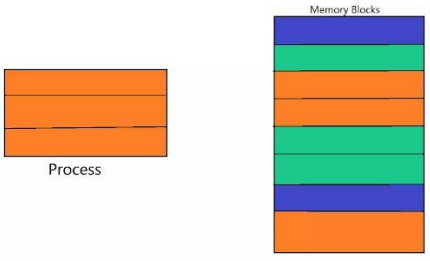
Non-Contiguous memory allocation:

The Non-contiguous memory allocation allows a process to acquire the several memory blocks at the different location in the memory according to its requirement. The non-contiguous memory allocation also reduces the memory wastage caused due to internal and external fragmentation. As it utilizes the memory holes, created during internal and external fragmentation.

 The available free memory space are scattered here and there and all the free memory space is not at one place. So this is timeconsuming.

 A process will acquire the memory space but it is not at one place it is at the different locations according to the process requirement.

 It reduces the wastage of memory which leads to internal and external fragmentation. This utilizes all the free memory space which is created by a different process.



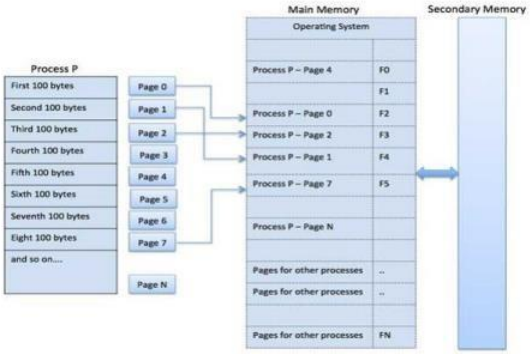
Non-contiguous memory allocation is of different types, 1. Paging

2. Segmentation 3.

Segmentation with

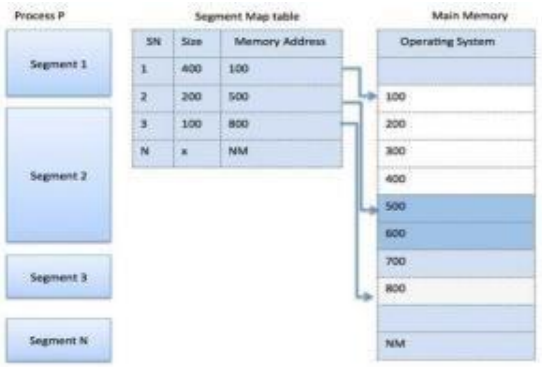
paging **Paging:**

A non-contiguous policy with a fixed size partition is called paging. A computer can address more memory than the amount of physically installed on the system. This extra memory is actually called virtual memory. Paging technique is very important in implementing virtual memory.



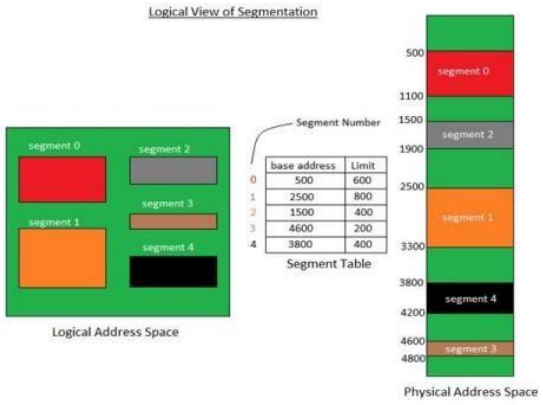
Segmentation:

Segmentation is a programmer view of the memory where instead of dividing a process into equal size partition we divided according to program into partition called segments. The translation is the same as paging but paging segmentation is independent of internal fragmentation but suffers from external fragmentation. Reason of external fragmentation is program can be divided into segments but segment must be contiguous in nature.



Segmentation with Paging:

In segmentation with paging, we take advantages of both segmentation as well as paging. It is a kind of multilevel paging but in multilevel paging, we divide a page table into equal size partition but here in segmentation with paging, we divide it according to segments. All the properties are the same as that of paging because segments are divided into pages.



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.

Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10567	30 K	J1	20 K	Busy	10 K
30457	50 K	J4	50 K	Busy	None
300875	200 K	J2	200 K	Busy	None
809567	700 K	J3	500 K	Busy	200 K
Total available :	980 K	Total used :	770 K		210 K

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.

Process Number	Process Size
P1	30K
P2	100K
P3	45K

MEMORY LOCATION	MEMORY BLOCK SIZE	PROCESS NUMBER	PROCESS SIZE	STATUS	INTERNAL FRAGMENTATION
12345	50K	P3	45K	Busy	5K
45871	100K	P2	100K	Busy	None
1245	400K	P1	30K	Busy	370K
TOTAL AVAILABLE:	550K	TOTAL USED:	175K		375K

Next fit:

The next fit is a modified version of first fit. It begins as the first fit to find a free partition but when called next time it starts searching from where it left off, not from the beginning. This policy makes use of a roving pointer. The pointer moves along the memory chain to search

for the next fit. This helps in, to avoid the usage of memory always from the head (beginning) of the free blockchain.

FRAGMENTATION

Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused. It is also necessary to understand that as programs are loaded and deleted from memory, they generate free space or a hole in the memory. These small blocks cannot be allotted to new arriving processes, resulting in inefficient memory use.

Types of Fragmentation

There are mainly two types of fragmentation in the operating system. These are as follows:

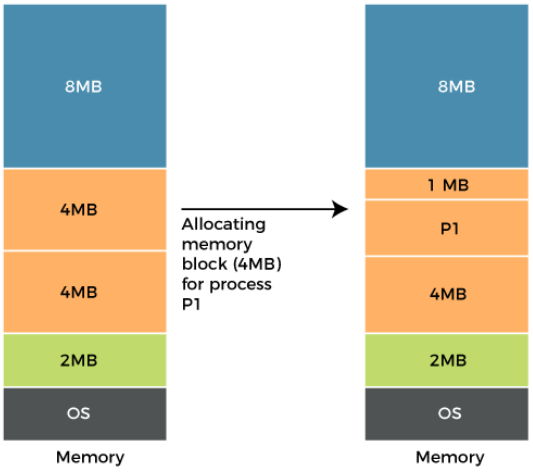
1. **Internal Fragmentation**
2. **External Fragmentation**

Internal Fragmentation

When a process is allocated to a memory block, and if the process is smaller than the amount of memory requested, a free space is created in the given memory block. Due to this, the free space of the memory block is unused, which causes **internal** fragmentation.

For Example:

Assume that memory allocation in RAM is done using fixed partitioning (i.e., memory blocks of fixed sizes). **2MB, 4MB, 4MB**, and **8MB** are the available sizes. The Operating System uses a part of this RAM.

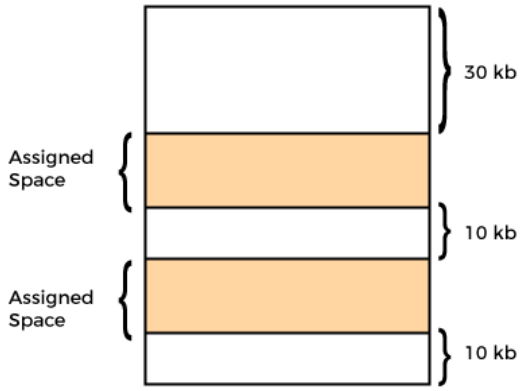


Let's suppose a process **P1** with a size of **3MB** arrives and is given a memory block of **4MB**. As a result, the **1MB** of free space in this block is unused and cannot be used to allocate memory to another process. It is known as **internal fragmentation**.

External Fragmentation

External fragmentation happens when a dynamic memory allocation method allocates some memory but leaves a small amount of memory unusable. The quantity of available memory is substantially reduced if there is too much external fragmentation. There is enough memory space to complete a request, but it is not contiguous. It's known as **external** fragmentation.

For Example:



Process 05 needs 45kb memory space

Let's take the example of external fragmentation. In the above diagram, you can see that there is sufficient space **(50 KB)** to run a process **(05) (need 45KB)**, but the memory is not contiguous. You can use compaction, paging, and segmentation to use the free space to execute a process.

Advantages and Disadvantages of Fragmentation

Advantages

- **Fast Data Writes**

  Data write in a system that supports data fragmentation may be faster than reorganizing data storage to enable contiguous data writes.

- **Fewer Failures**

  If there is insufficient sequential space in a system that does not support fragmentation, the write will fail.

- **Storage Optimization**

  A fragmented system might potentially make better use of a storage device by utilizing every available storage block.

Disadvantages

There are various disadvantages of fragmentation. Some of them are as follows:

Need for regular defragmentation

A more fragmented storage device's performance will degrade with time, necessitating the requirement for time consuming defragmentation operations.

Slower Read Times

The time it takes to read a non-sequential file might increase as a storage device becomes more fragmented.

CODE

```
#include<stdio.h>
#include<stdlib.h>

typedef struct process{                 //process
  int psize;
  int pflag;
}process;

typedef struct block{                   //memory blocks
  int bsize;
  int bflag;
}block;

void accept(process p[],block b[],int *n,int *m){  //accepting no. of blocks and processes
  int i,j;
  printf("\n Enter no. of blocks : ");
  scanf("%d",m);
  for(i=0;i<*m;i++){
    printf(" Enter size of block[%d] : ",i);
    scanf("%d",&b[i].bsize);
  }

  printf("\n Enter no. of processes : ");
  scanf("%d",n);
  for(i=0;i<*n;i++){
    printf(" Enter size of block[%d] : ",i);
    scanf("%d",&p[i].psize);
  }

}
void re_init(process p[],block b[],int n,int m){
  int i;
  for(i=0;i<n;i++)
    p[i].pflag=0;
  for(i=0;i<m;i++)
    b[i].bflag=0;
}

void best_fit(process p[],block b[],int n,int m){
  int i,j,min_frag,in_frag=0,ex_frag=0,id;
  for(i=0;i<n;i++){
    min_frag = 9999;
    id=9999;
    for(j=0;j<m;j++){
```

```
      if(p[i].psize <= b[j].bsize && b[j].bflag == 0 && min_frag > (b[j].bsize -
p[i].psize)){
          min_frag = b[j].bsize - p[i].psize;
          id = j;
      }
    }
    if(min_frag != 9999){
        b[id].bflag = p[i].pflag = 1;
        in_frag += b[id].bsize - p[i].psize;
        printf("\n P[%d]\t-\tB[%d]",i,id);
    }
    if(p[i].pflag == 0)
      printf("\n P[%d]\t-\tUnassigned",i);
  }

  printf("\n\n Total internal fragmentation : %d",in_frag );
  for(j=0;j<m;j++){
    if(b[j].bflag == 0)
      ex_frag+=b[j].bsize;
  }
  printf("\n Total external fragmentation : %d",ex_frag );


}

void worst_fit(process p[],block b[],int n,int m){
  int i,j,max_frag,in_frag=0,ex_frag=0,id;

  for(i=0;i<n;i++){
    max_frag = -1;
    id=9999;
    for(j=0;j<m;j++){
      if(p[i].psize <= b[j].bsize && b[j].bflag == 0 && max_frag < (b[j].bsize -
p[i].psize)){
          max_frag = b[j].bsize - p[i].psize;
          id = j;
      }
    }
    if(max_frag != -1){
        b[id].bflag = p[i].pflag = 1;
        in_frag += b[id].bsize - p[i].psize;
        printf("\n P[%d]\t-\tB[%d]",i,id);
    }
    if(p[i].pflag == 0)
      printf("\n P[%d]\t-\tUnassigned",i);
  }

  printf("\n\n Total internal fragmentation : %d",in_frag );
  for(j=0;j<m;j++){
    if(b[j].bflag == 0)
      ex_frag+=b[j].bsize;
```

```
  }
  printf("\n Total external fragmentation : %d",ex_frag );


}

void next_fit(process p[],block b[],int n,int m){

  int i,j,in_frag=0,ex_frag=0,loc;
  printf("\n Enter block no. to begin allocation : ");
  scanf("%d",&loc);
  for(i=0;i<n;i++){
    for(j=loc;j<m;j++){
      if(p[i].psize <= b[j].bsize && b[j].bflag == 0 && p[i].pflag == 0){
        b[j].bflag = p[i].pflag = 1;
        in_frag += b[j].bsize - p[i].psize;
        printf("\n P[%d]\t-\tB[%d]",i,j);
        if((j+1) == m)
          loc = 0;
        else
          loc = j+1;
        break;
      }

    }
    if(p[i].pflag == 0)
      printf("\n P[%d]\t-\tUnassigned",i);
  }
  printf("\n\n Total internal fragmentation : %d",in_frag );
  for(j=0;j<m;j++){
    if(b[j].bflag == 0)
      ex_frag+=b[j].bsize;
  }
  printf("\n Total external fragmentation : %d",ex_frag );
}


int main(){

  int ch,n=0,m=0;
  process p[10];
  block b[10];
  do{
    printf("\n\n MEMORY MANAGEMENT SIMULATOR");
    printf("\n\n Options:");
    printf("\n 0. Accept");
    printf("\n 1. Best Fit");
    printf("\n 2. Next Fit");
    printf("\n 3. Worst Fit");
    printf("\n 4. Exit");
    printf("\n Select : ");
```

```
    scanf("%d",&ch);
    switch(ch){
      case 0:
        accept(p,b,&n,&m);
        break;
      case 1:
        re_init(p,b,n,m);
        best_fit(p,b,n,m);
        break;
      case 2:
        re_init(p,b,n,m);
        next_fit(p,b,n,m);
        break;
      case 3:
        re_init(p,b,n,m);
        worst_fit(p,b,n,m);
        break;
      case 4:exit(0);
      default:
        printf("\n Invalid selection.");
    }
  }while(ch != 4);
  return 0;
}
```

OUTPUTS

```
PS D:\MIET\3rd Sem> & .\"memory.exe"


 MEMORY MANAGEMENT SIMULATOR

 Options:
 0. Accept
 1. Best Fit
 2. Next Fit
 3. Worst Fit
 4. Exit
 Select :
```

- **ACCEPT**

```
 MEMORY MANAGEMENT SIMULATOR

 Options:
 0. Accept
 1. Best Fit
 2. Next Fit
 3. Worst Fit
 4. Exit
 Select : 0

 Enter no. of blocks : 3
 Enter size of block[0] : 2
 Enter size of block[1] : 4
 Enter size of block[2] : 8

 Enter no. of processes : 3
 Enter size of block[0] : 2
 Enter size of block[1] : 5
 Enter size of block[2] : 7
```

- **BEST FIT**

```
MEMORY MANAGEMENT SIMULATOR

Options:
0. Accept
1. Best Fit
2. Next Fit
3. Worst Fit
4. Exit
Select : 1

P[0]   -       B[0]
P[1]   -       B[2]
P[2]   -       Unassigned

Total internal fragmentation : 3
Total external fragmentation : 4
```

- **NEXT FIT**

```
MEMORY MANAGEMENT SIMULATOR

Options:
0. Accept
1. Best Fit
2. Next Fit
3. Worst Fit
4. Exit
Select : 2

Enter block no. to begin allocation : 1

P[0]   -       B[1]
P[1]   -       B[2]
P[2]   -       Unassigned

Total internal fragmentation : 5
Total external fragmentation : 2
```

- **WORST FIT**

```
MEMORY MANAGEMENT SIMULATOR

Options:
0. Accept
1. Best Fit
2. Next Fit
3. Worst Fit
4. Exit
Select : 3

P[0]   -       B[2]
P[1]   -       Unassigned
P[2]   -       Unassigned

Total internal fragmentation : 6
Total external fragmentation : 6
```

CONCLUSION:

In fixed sized partition, internal fragmentation may or may not occur, but external fragmentation does not occur, because leftover of a partition cannot be allocated to other process. However, in case of variable sized partition, internal fragmentation does not takes place, because leftover of a partition can be allocated to other process, but external fragmentation may have a chance to happen, which may be solved using compaction.
Analysis of the three algorithms: first fit, best fit and worst fit shows that best fit appears to be the most efficient algorithm among three algorithms in fixed sized partition with minimum internal fragmentation. However, worst fit may not be the worst performer in variable sized partition. An improvement of first fit algorithm may be the next fit algorithm, where instead of searching the first unoccupied partition from the beginning; it searches in the set of holes to find the first unoccupied partition. That results in less amount of time consumption. However, its performance is equivalent to first fit algorithm. In future, there may be dynamic memory allocation with combination of fixed sized and variable sized partition according to problem domain to improve the efficiency of the memory allocation.

REFRENCE:

Memory Management in Operating System - GeeksforGeeks

Memory Management - javatpoint

GITHUB ACCOUNTS:

- https://github.com/Surajpandit23/Memorymanagement_os.git
- https://github.com/Amandeep208/OperatingSystem/tree/main/Project-%20Memory%20Management%20Stimulator
- https://github.com/Sahilchib12/Operating-system-/tree/main/Project-%20Memory%20Management%20Stimulator
- https://github.com/Vanshak77/OperatingSystem/tree/main/Project-%20Memory%20Management%20Stimulator