12113029
Harshvardhan
IT-02

Submitted to :

Dr. Priyanka Ahalawat

Experiment-1

1.1
UNIX OS

UNIX is a powerful Operating System initially developed by Ken Thompson, Dennis Ritchie at AT&T Bell laboratories in 1970.

It has features such as multitasking, flexible .

Hardware and Software Requirements:

Ram : 1 GB
Processor : Any with the clock speed of 375 Mhz or faster
1 GB free space

Linux OS

Linux is a free, open source operating system.
It is supported on most of the systems.

Hardware and Software Requirements:

Ram : 512 MB
Processor : 32-bit intel compatible with clock speed of 2 Ghz or faster
Space : 2.5 GB
DVD-ROM Drive

Windows XP

Hardware and Software Requirements:

Ram : 64 MB
Processor : pentium 233 MHz or faster (300Mhz best)
Space : 1.5 GB
DVD-ROM Drive

Windows 7

Ram : 1 GB(32 bit) or 2GB(64 bit)
Processor : 1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64)
Space : 16 to 20 GB
DVD-ROM Drive
1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64)

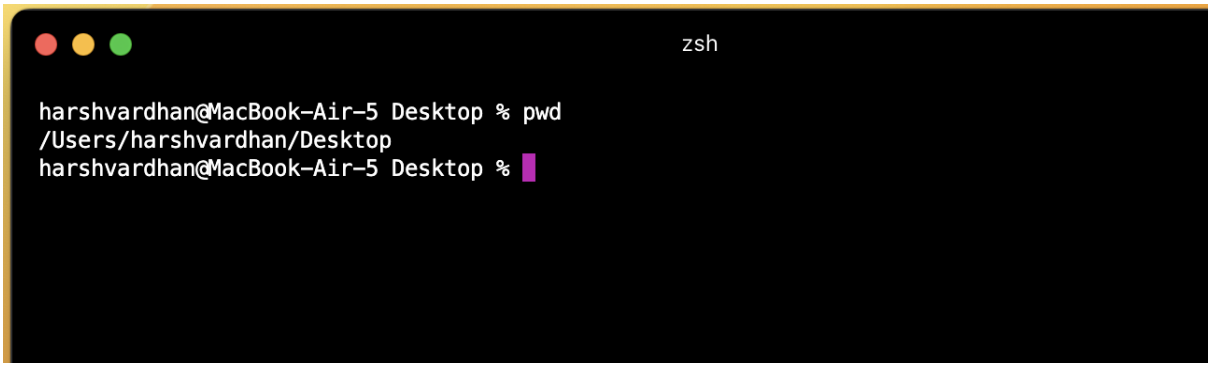
Windows 8

Ram : 1 GB(32 bit) or 2GB(64 bit)
Processor : 1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64)
Space : 16 to 20 GB
DVD-ROM Drive
1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64)

1.2

Pwd

Used to print the present working directory. In my case it is Desktop which has been printed.

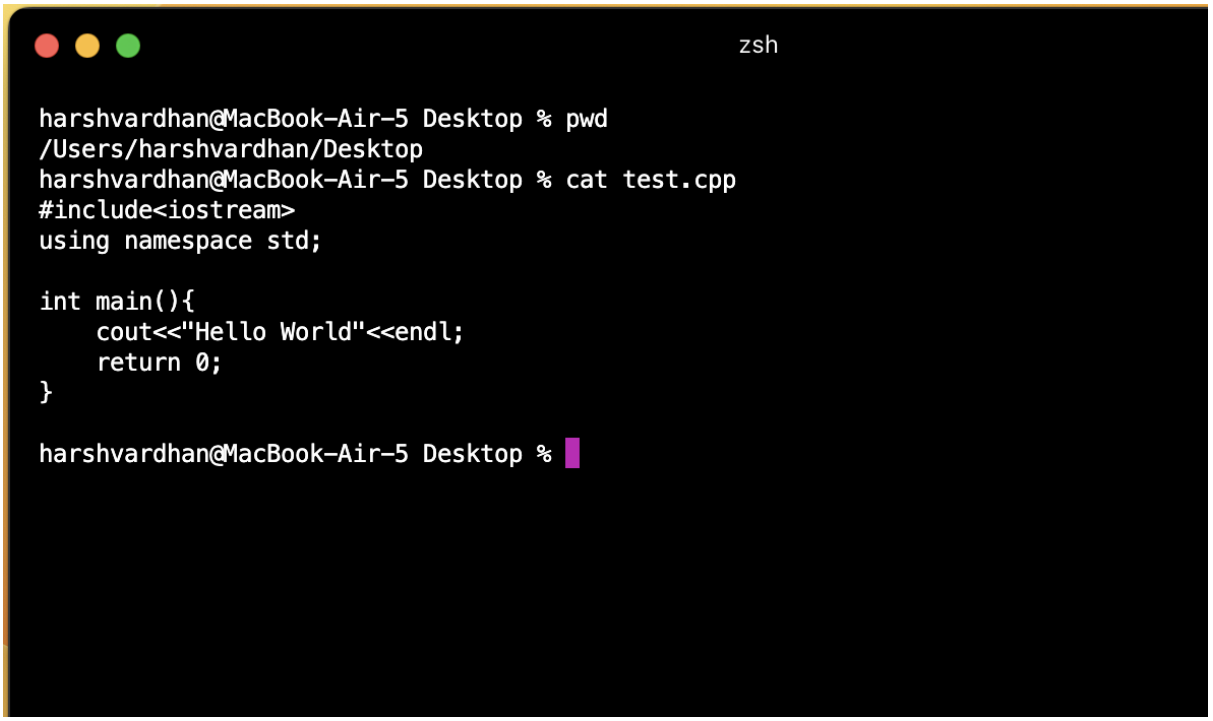


cd

Used to change the directory. In my case it is Desktop and it changes to MERN folder which has been printed.

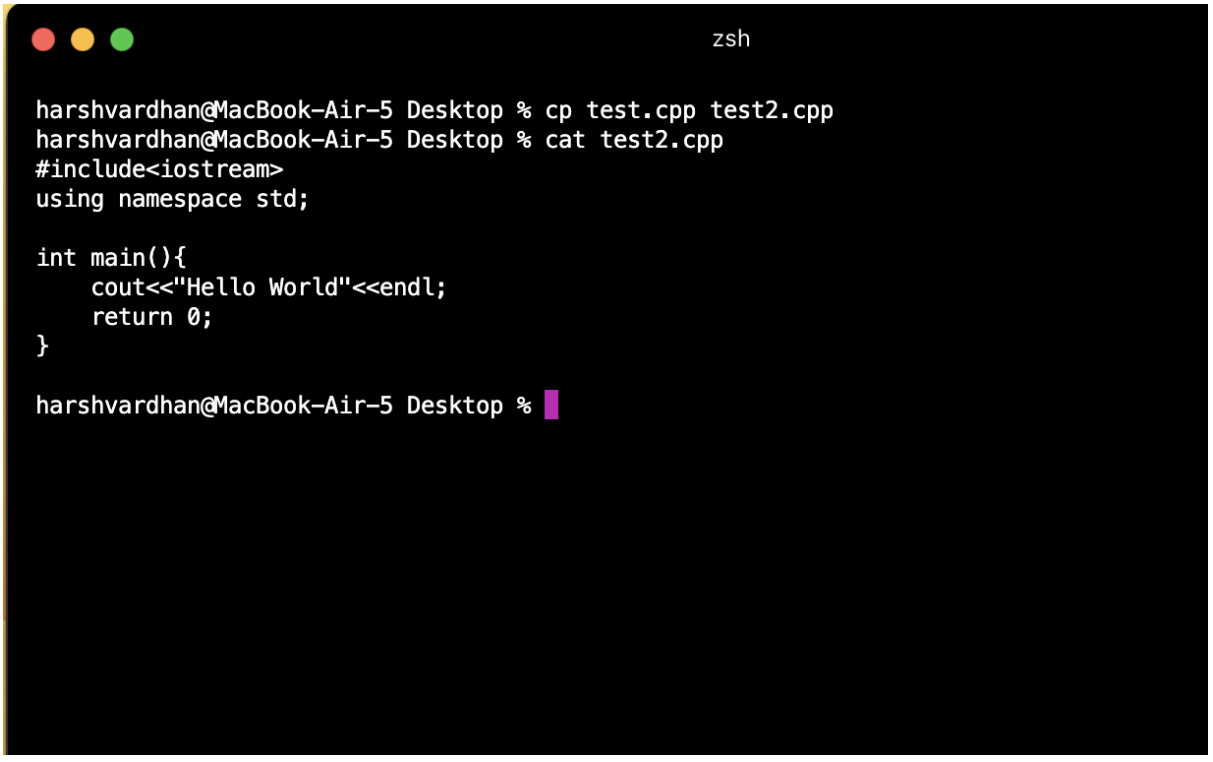
cat

This command is used to print the contents of the file and also used to concatenate the content of two files.



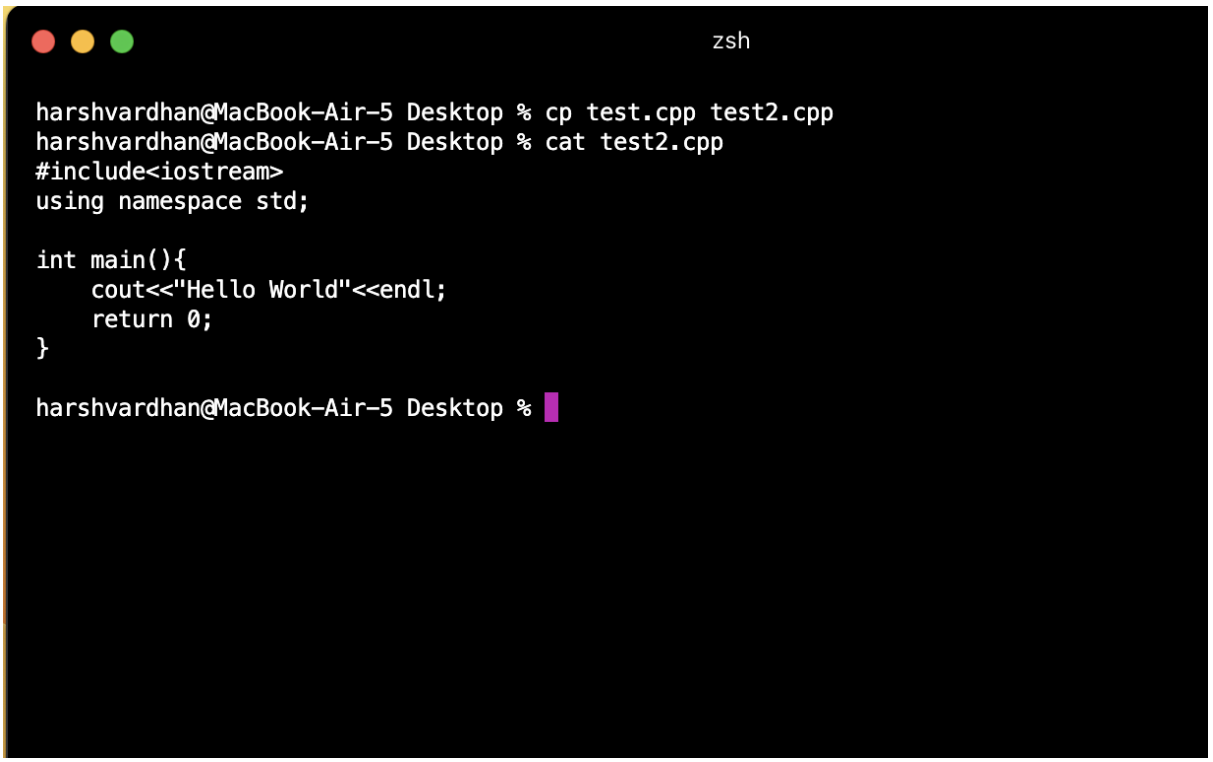
cp

This command is used to copy of the contents of the file,directory ,etc from one destination to other location.
In this is case contents of the file test.cpp has been copied to test2.cpp.



chmod

This command is used to change access of the mode of a file.



df

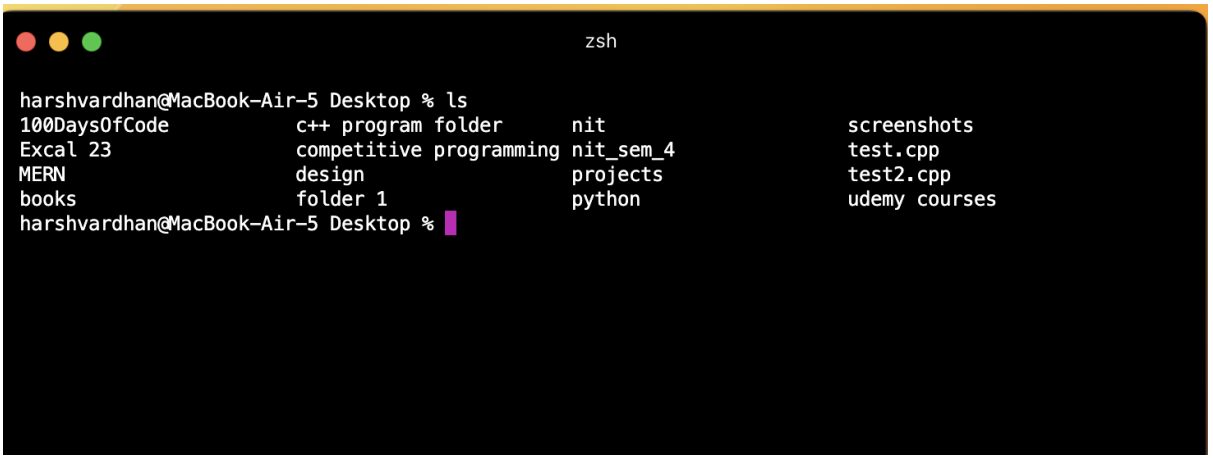
This command is used to display the information about the storage space and available space allocated to a given file.

less

This command is used to read a file page by page. It is useful when we have to read a large file and have to view only a small part.

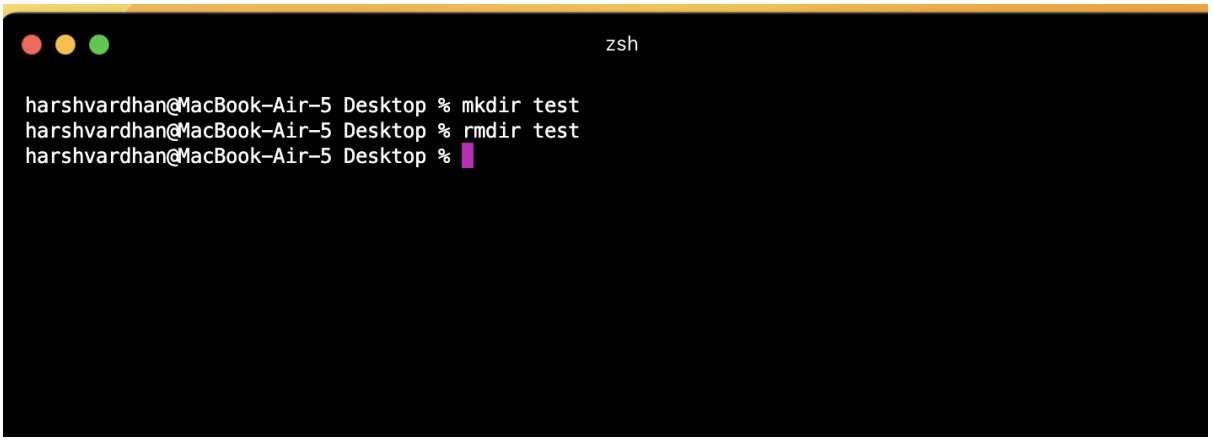
ls

This command is used to list the items of the present working directory.



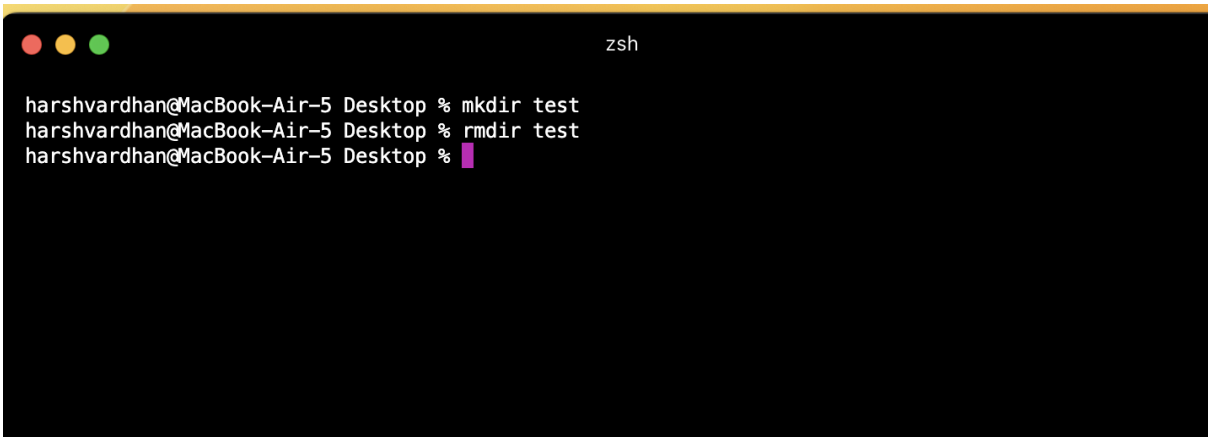
mkdir

This command is used to make a new directory.



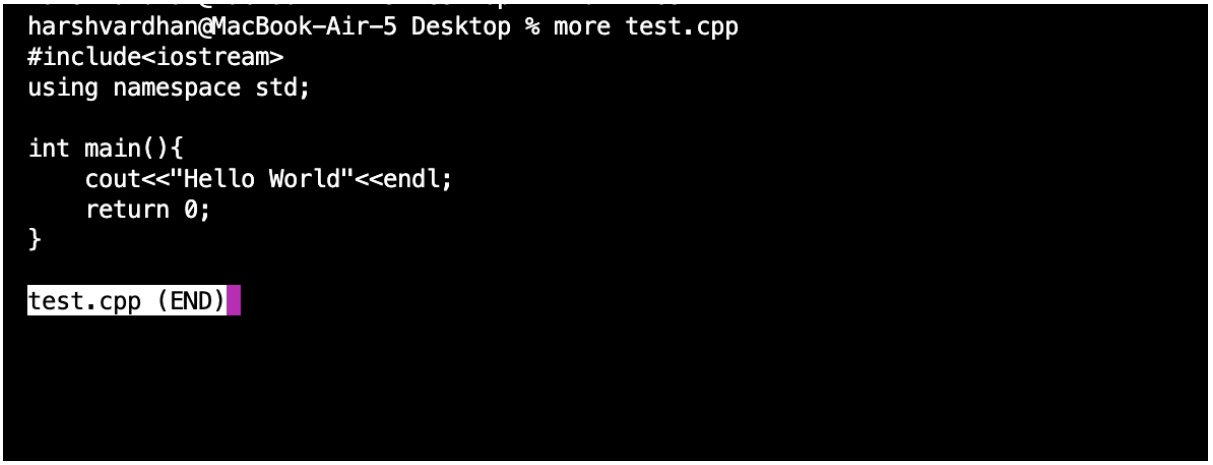
rmdir

This command is used to remove a existing directory.



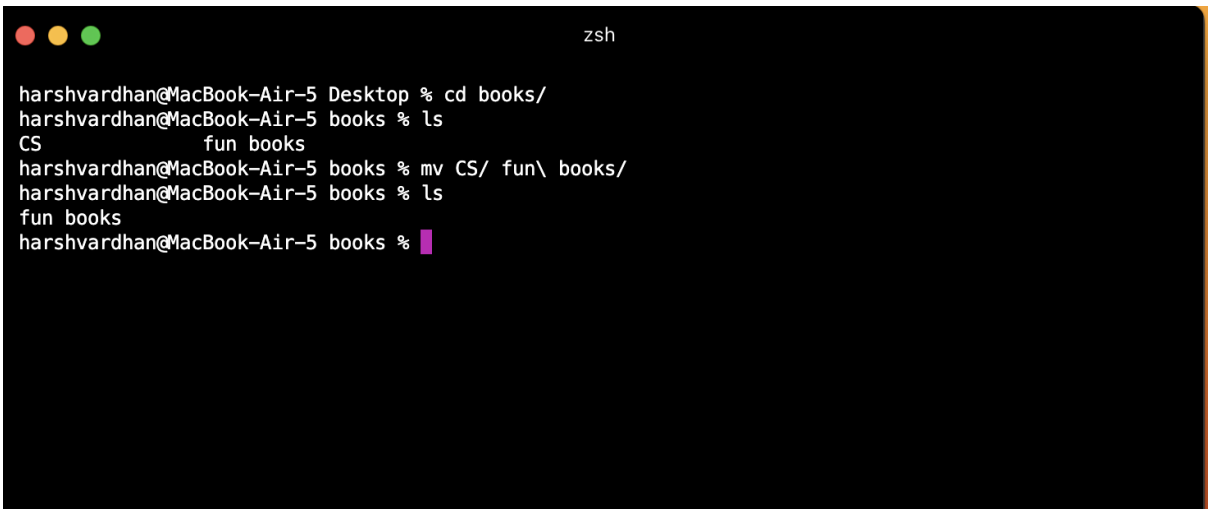
more

This is used to view the text files in the command prompt, displaying one screen at a time in case the file is large (For example log files).



mv

This command is used to move items in a existing directory.

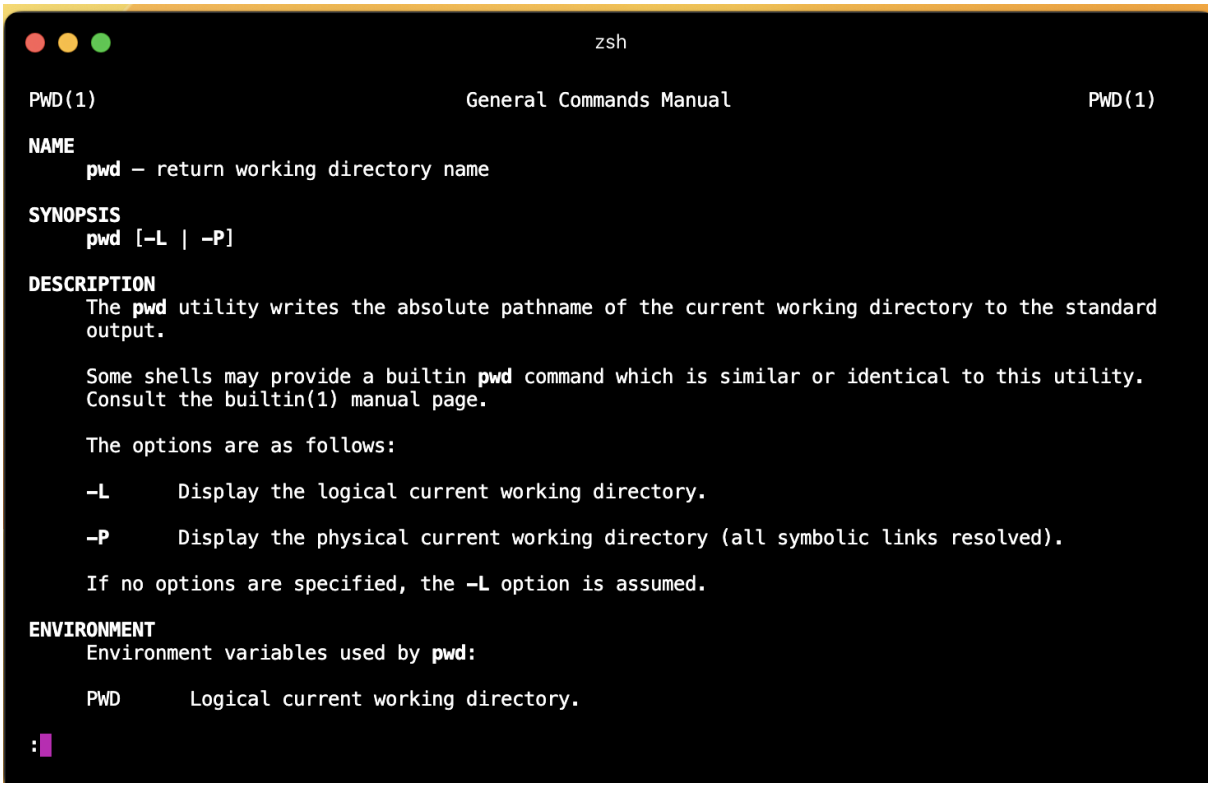


rm

This command is used to delete the items of the existing directory . It can also remove non-empty directory as well.

man

This command is used to display the user manual of any command.

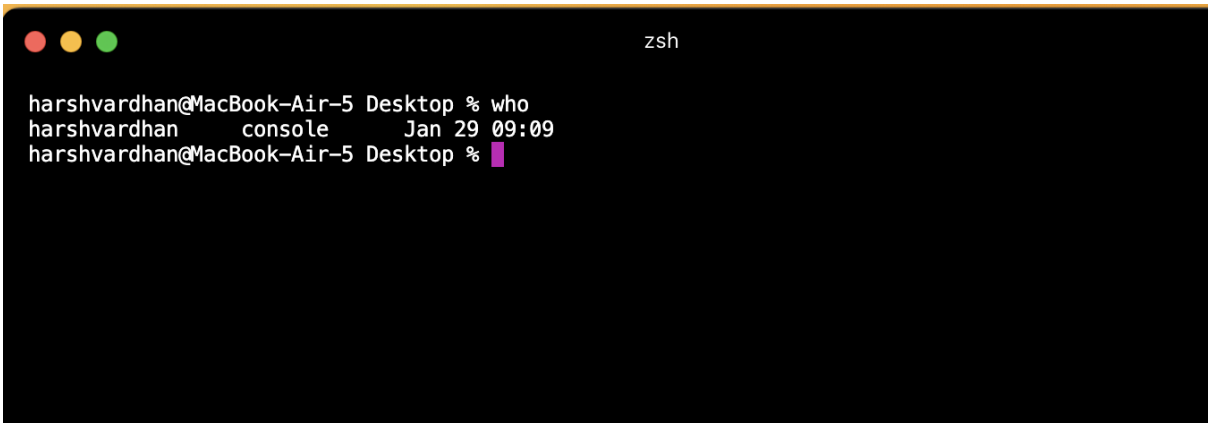


uname

This command is used to print the info about kernel,version and other stuff.

who

This command is used to print information about the users who are currently logged in, currently run level and time of last boot.

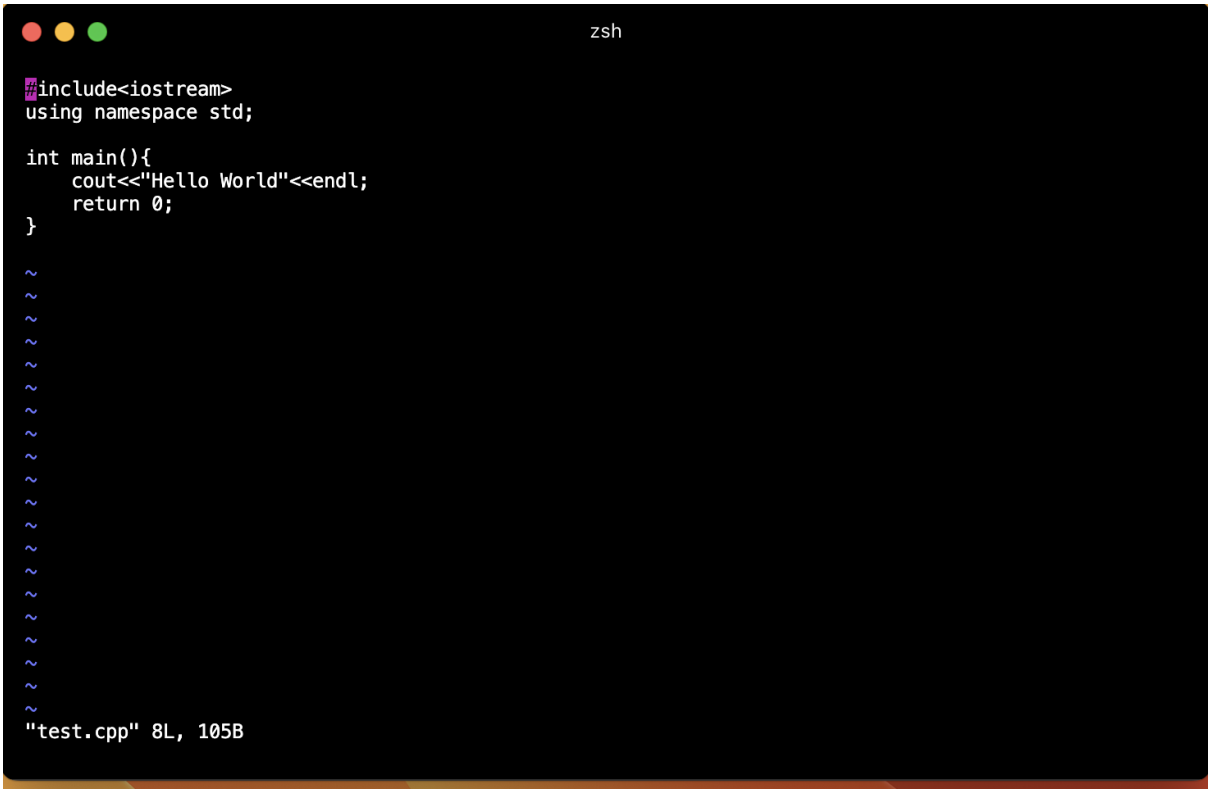


ps

This command is used to view the currently running processes on the system.

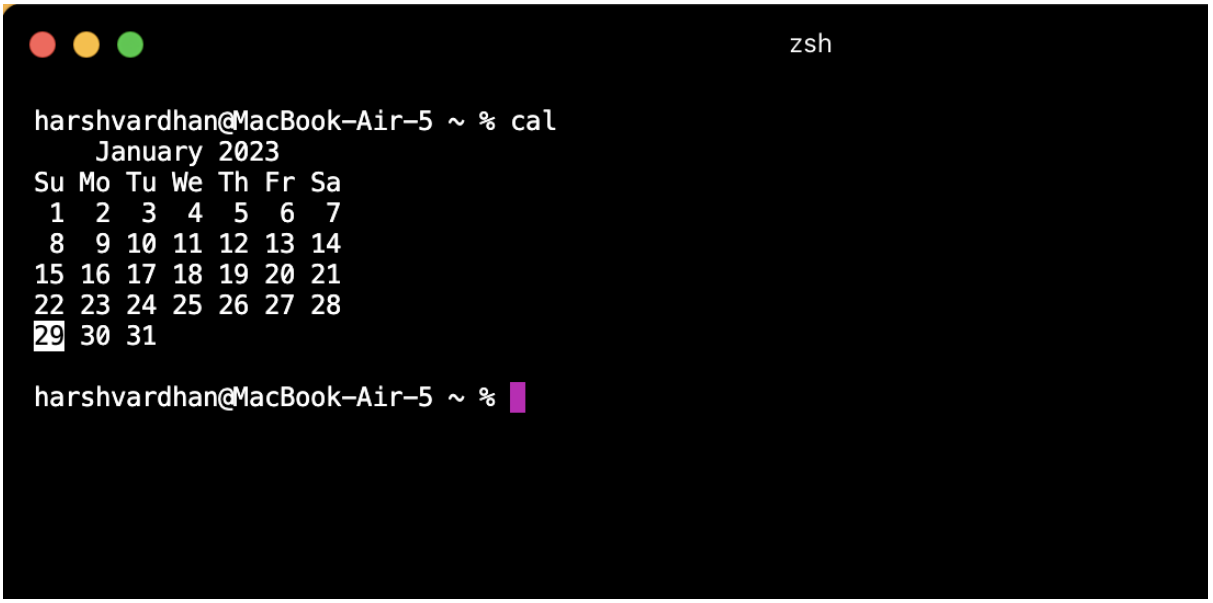
vi

This command is used to open a text editor where you can write code.



cal

This command is used to display the calender of specific month or year.

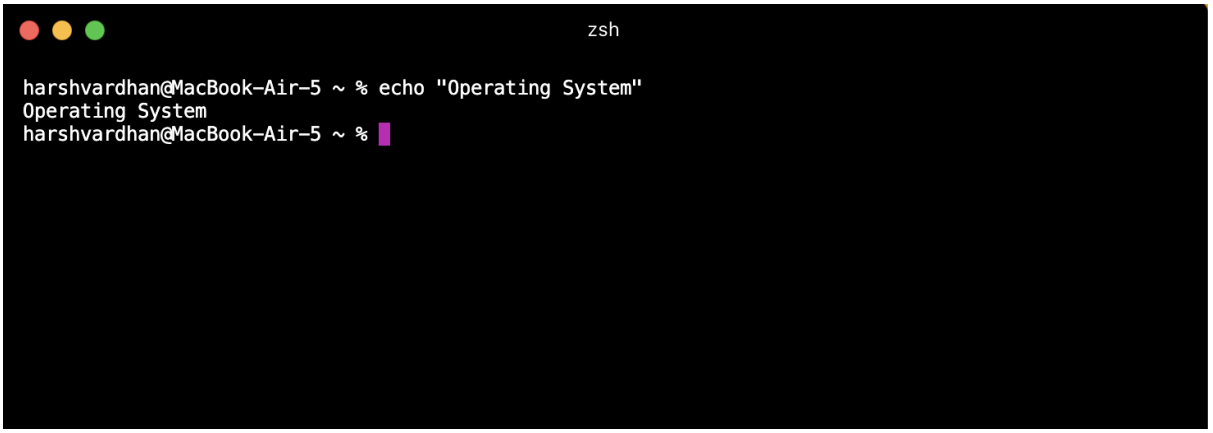


date

This command is used to display the system date and time.

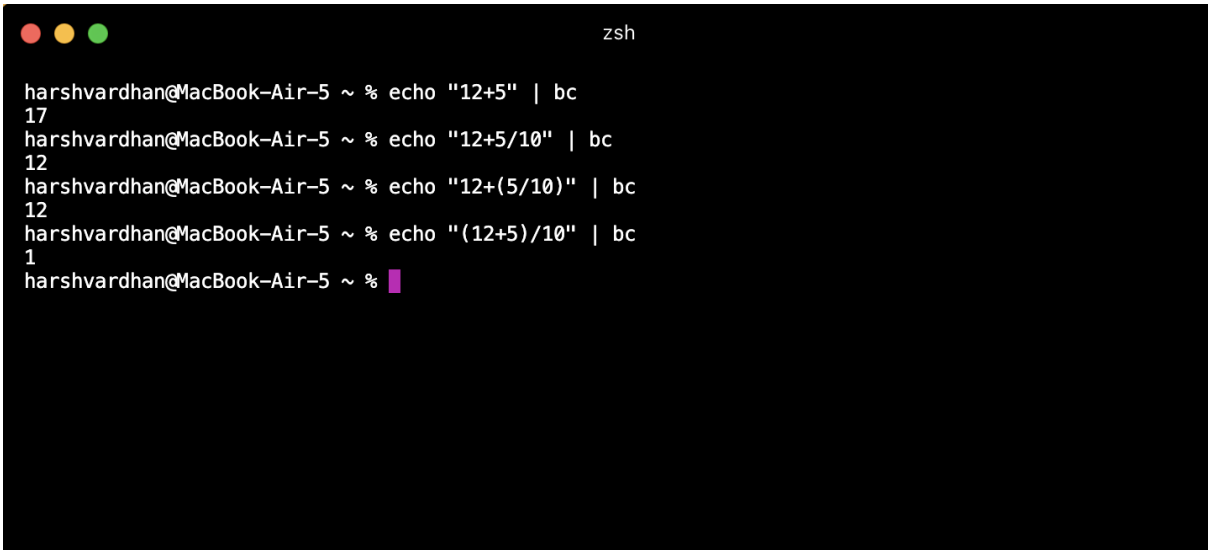
echo

This command is used to display the line of string that has been passed as an argument.



bc

This command is used to perform the command line calculations



grep

This command is used to search for a particular expression in the file. It simply prints the lines which contains that regular expression.

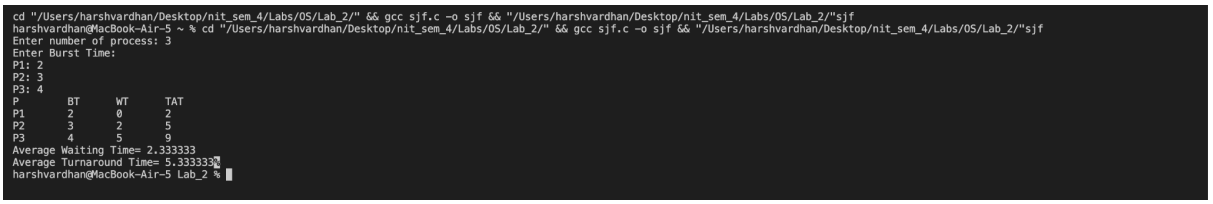
Experiment - 2

Q1. Implement SJF (Shortest Job First) CPU scheduling algorithm taking user inputs and calculate Average TAT and WT.

```
#include
int main()
{
        int A[100][4];
    // Matrix for storing Process Id, Burst
        // Time, Average Waiting Time & Average
        // Turn Around Time.
        int i, j, n, total = 0, index, temp;

        float avg_wt, avg_tat;

        printf("Enter number of process: ");
        scanf("%d", &n);
        printf("Enter Burst Time:\n");
        // User Input Burst Time and alloting Process Id.
        for (i = 0; i < n; i++) {
                printf("P%d: ", i + 1);
                scanf("%d", &A[i][1]);
                A[i][0] = i + 1;
        }

        // Sorting process according to their Burst Time.

        for (i = 0; i < n; i++) {
                index = i;
                for (j = i + 1; j < n; j++)
                        if (A[j][1] < A[index][1])
                                index = j;
                temp = A[i][1];
                A[i][1] = A[index][1];
                A[index][1] = temp;

                temp = A[i][0];
                A[i][0] = A[index][0];
```

```
            A[index][0] = temp;
        }
        A[0][2] = 0;
        for (i = 1; i < n; i++) {
            A[i][2] = 0;
            for (j = 0; j < i; j++)
                A[i][2] += A[j][1];
            total += A[i][2];
        }

        avg_wt = (float)total / n;
        total = 0;
        printf("P         BT      WT      TAT\n");

        for (i = 0; i < n; i++) {
            A[i][3] = A[i][1] + A[i][2];
            total += A[i][3];
            printf("P%d     %d      %d      %d\n", A[i][0],
                A[i][1], A[i][2], A[i][3]);
        }
        avg_tat = (float)total / n;
        printf("Average Waiting Time= %f", avg_wt);
        printf("\nAverage Turnaround Time= %f", avg_tat);

    return 0;
}
```

Output:



Q:2 Implement FCFS (First Come First Serve) CPU scheduling algorithm taking user inputs and calculate Average TAT and WT.

```
#include<stdio.h>

 int main()
```

```
{
    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
    printf("Enter total number of processes(maximum 20):");
    scanf("%d",&n);

    printf("Enter Process Burst Time\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d]:",i+1);
        scanf("%d",&bt[i]);
    }

    wt[0]=0;

    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
    }

   // printf("\nProcessttBurst TimetWaiting TimetTurnaround Time");

    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        avwt+=wt[i];
        avtat+=tat[i];
       // printf("nP[%d]tt%dtt%dtt%d",i+1,bt[i],wt[i],tat[i]);
    }

    avwt/=i;
    avtat/=i;
    printf("\nnAverage Waiting Time:%d",avwt);
    printf("\nAverage Turnaround Time:%d",avtat);

    return 0;
}
```

Output:

Q3. Implement Priority queue CPU scheduling algorithm taking user inputs and calculate Average TAT and WT.

```
#include

struct process
{
   int pid, arrival_time, burst_time, priority;
};

void priority_queue()
{

   int num_processes, i, j;

   float avg_tat = 0, avg_wt = 0;

   printf("Enter the number of processes: ");

   scanf("%d", &num_processes);

   struct process p[num_processes], temp;

   int ct[num_processes], tat[num_processes], wt[num_processes];

   for (i = 0; i < num_processes; i++)
   {
      printf("Enter the arrival time for process %d: ", i + 1);
      scanf("%d", &p[i].arrival_time);
      printf("Enter the burst time for process %d: ", i + 1);
      scanf("%d", &p[i].burst_time);
      printf("Enter the priority for process %d: ", i + 1);
      scanf("%d", &p[i].priority);
      p[i].pid = i + 1;
   }

   for (i = 0; i < num_processes - 1; i++)
```

```
    {
        for (j = 0; j < num_processes - i - 1; j++)
        {
            if (p[j].priority < p[j + 1].priority)
            {
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
    ct[0] = p[0].arrival_time + p[0].burst_time;
    tat[0] = ct[0] - p[0].arrival_time;
    wt[0] = tat[0] - p[0].burst_time;

    for (i = 1; i < num_processes; i++)
    {
        if (p[i].arrival_time > ct[i - 1])
        {
            ct[i] = p[i].arrival_time + p[i].burst_time;
        }
        else
        {
            ct[i] = ct[i - 1] + p[i].burst_time;
        }
        tat[i] = ct[i] - p[i].arrival_time;
        wt[i] = tat[i] - p[i].burst_time;
    }

    for (i = 0; i < num_processes; i++)
    {
        avg_tat += tat[i];
        avg_wt += wt[i];
    }
    printf("Process\tAT\tBT\tPri\tCT\tTAT\tWT\n");
    for (i = 0; i < num_processes; i++)
    {
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival_time,
p[i].burst_time, p[i].priority, ct[i], tat[i], wt[i]);
    }
    printf("Average Turnaround Time: %.2f\n", avg_tat / num_processes);
    printf("Average Waiting Time: %.2f\n", avg_wt / num_processes);
}
```

```
int main()
{
   priority_queue();
   return 0;
}
```

Output:

Q4. Implement Multi level queue CPU scheduling algorithm taking user inputs and calculate Average TAT and WT.

```
#include<stdio.h>

int main()
{
        int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp;


        float wtavg, tatavg;

        printf("Enter the number of processes:");
        scanf("%d",&n);


        for(i=0;i<n;i++)
        {
                p[i] = i;
                printf("Enter the Burst Time of Process%d:", i);
                scanf("%d",&bt[i]);
                printf("System/User Process (0/1) ? ");
                scanf("%d", &su[i]);
        }

        for(i=0;i<n;i++)
                for(k=i+1;k<n;k++)
                        if(su[i] > su[k])
```

```
                {
                temp=p[i];
                p[i]=p[k];
                p[k]=temp;

                temp=bt[i];
                bt[i]=bt[k];
                bt[k]=temp;

                temp=su[i];
                su[i]=su[k];
                su[k]=temp;
                }

        wtavg = wt[0] = 0;
        tatavg = tat[0] = bt[0];

        for(i=1;i<n;i++)
        {
                wt[i] = wt[i-1] + bt[i-1];
                tat[i] = tat[i-1] + bt[i];
                wtavg += wt[i];
                tatavg += tat[i];
        }

        printf("\nPROCESS\t\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING
TIME\tTURNAROUND TIME");
        for(i=0;i<n;i++)
                printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],su[i],bt[i],wt[i],tat[i]);
        printf("\nAverage Waiting Time is --- %f",wtavg/n);
        printf("\nAverage Turnaround Time is --- %f",tatavg/n);
        return 0;
}
```

Output:

Experiment - 3

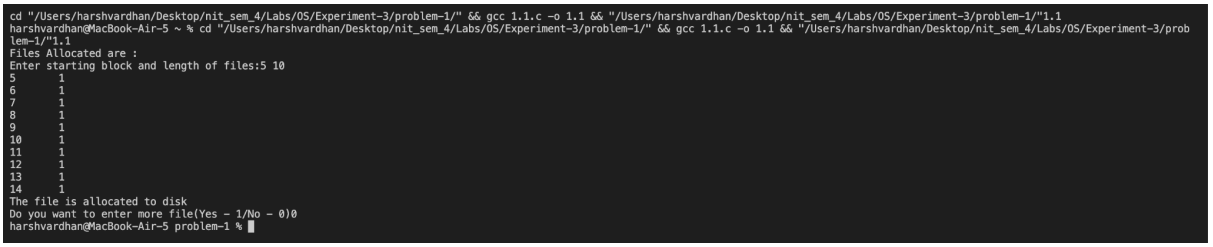
Q 1. Implement file storage allocation techniques –
1..1 Contiguous (using array)
1..2 linked –list (using linked list)
1..3 indirect allocation (indexing)

1.1

```
#include <stdio.h>

int main()
{
   int f[50], i, st, len, j, c, k, count = 0;

   for (i = 0; i < 50; i++)
   {
      f[i] = 0;
      printf("Files Allocated are : \n");
   x:
      count = 0;
      printf("Enter starting block and length of files:");
      scanf("%d%d", &st, &len);
      for (k = st; k < (st + len); k++)
         if (f[k] == 0)
            count++;
      if (len == count)
      {
         for (j = st; j < (st + len); j++)
            if (f[j] == 0)
            {
               f[j] = 1;
               printf("%d\t%d\n", j, f[j]);
            }
         if (j != (st + len - 1))
            printf("The file is allocated to disk\n");
      }
      else
         printf("The file is not allocated \n");
```

```
        printf("Do you want to enter more file(Yes - 1/No - 0)");
        scanf("%d", &c);
        if (c == 1)
            goto x;
        else
        {
            return 0;
        }
    }
}
```

Output

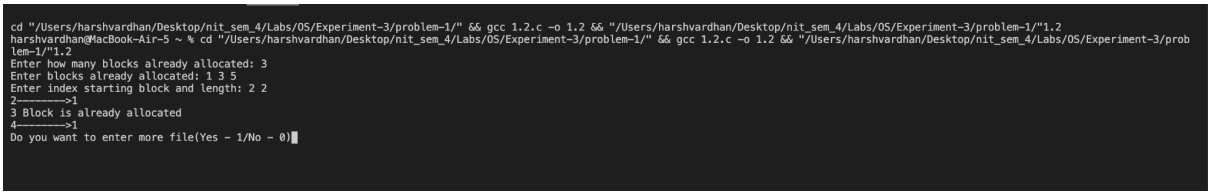


1.2

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

int main()
{
    int f[50], p, i, st, len, j, c, k, a;

    for (i = 0; i < 50; i++)
        f[i] = 0;

    printf("Enter how many blocks already allocated: ");
    scanf("%d", &p);
    printf("Enter blocks already allocated: ");
    for (i = 0; i < p; i++)
    {
        scanf("%d", &a);
        f[a] = 1;
    }
x:
    printf("Enter index starting block and length: ");
    scanf("%d%d", &st, &len);
    k = len;
```

```
    if (f[st] == 0)
    {
        for (j = st; j < (st + k); j++)
        {
            if (f[j] == 0)
            {
                f[j] = 1;
                printf("%d-------->%d\n", j, f[j]);
            }
            else
            {
                printf("%d Block is already allocated \n", j);
                k++;
            }
        }
    }
    else
        printf("%d starting block is already allocated \n", st);

    printf("Do you want to enter more file(Yes - 1/No - 0)");
    scanf("%d", &c);
    if (c == 1)
        goto x;
    else
        return 0;
}
```

Output



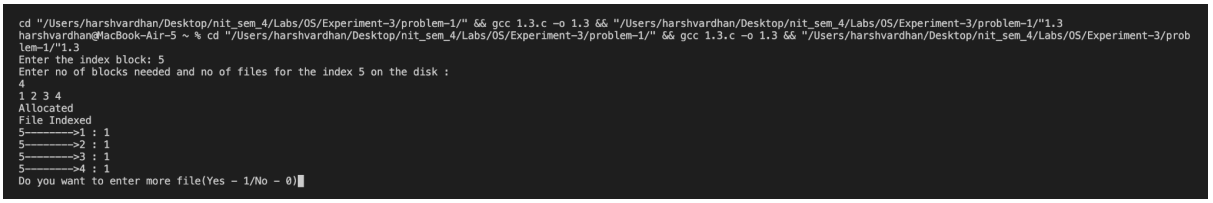
1.3

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


int main()
{
    int f[50], index[50], i, n, st, len, j, c, k, ind, count = 0;
```

```
    for (i = 0; i < 50; i++)
        f[i] = 0;
x:
    printf("Enter the index block: ");
    scanf("%d", &ind);

    if (f[ind] != 1)
    {
        printf("Enter no of blocks needed and no of files for the index %d on the disk :
\n", ind);
        scanf("%d", &n);
    }
    else
    {
        printf("%d index is already allocated \n", ind);
        goto x;
    }

y:

    count = 0;
    for (i = 0; i < n; i++)
    {
        scanf("%d", &index[i]);
        if (f[index[i]] == 0)
            count++;
    }
    if (count == n)
    {
        for (j = 0; j < n; j++)
            f[index[j]] = 1;
        printf("Allocated\n");
        printf("File Indexed\n");

        for (k = 0; k < n; k++)
            printf("%d-------->%d : %d\n", ind, index[k], f[index[k]]);
    }
    else
    {
        printf("File in the index is already allocated \n");
        printf("Enter another file indexed");
        //goto y;
    }
    printf("Do you want to enter more file(Yes - 1/No - 0)");
```

```
    scanf("%d", &c);
    if (c == 1)
        goto x;
    else
        return 0;

}
```

Output



Experiment - 4

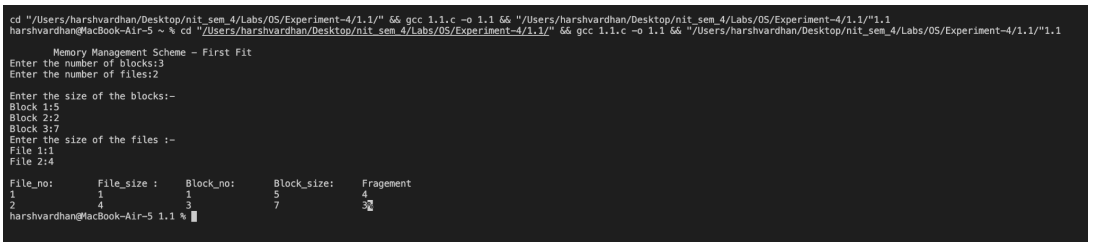
Q : 1. Implementation of Contiguous allocation techniques –
1.1 Worst-fit
1.2 Best-fit
1.3 First-fit

1.1
// worst fit

```
#include <stdio.h>
#define max 25


int main()
{
   int frag[max], b[max], f[max], i, j, nb, nf, temp;
   static int bf[max], ff[max];

   printf("\n\tMemory Management Scheme - First Fit");
   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);
   printf("Enter the number of files:");
   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 files :-\n");
   for (i = 1; i <= nf; i++)
   {
      printf("File %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)
            {
               ff[i] = j;
```

```
                break;
            }
        }
    }
    frag[i] = temp;
    bf[ff[i]] = 1;
  }
  printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
  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]);

}
```

Output

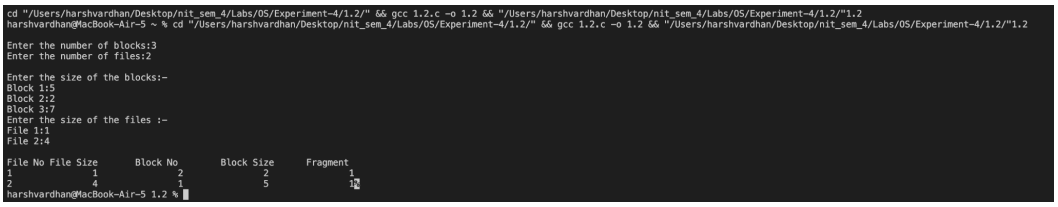


1.2

```
// Best fit program

#include
#define max 25


int main()
{
  int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest = 10000;
  static int bf[max], ff[max];

  printf("\nEnter the number of blocks:");
  scanf("%d", &nb);
  printf("Enter the number of files:");
  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 files :-\n");
  for (i = 1; i <= nf; i++)
  {
    printf("File %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 (lowest > temp)
                    {
                        ff[i] = j;

                        lowest = temp;
                    }
            }
        }
        frag[i] = lowest;
        bf[ff[i]] = 1;
        lowest = 10000;
    }
    printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
    for (i = 1; i <= nf && ff[i] != 0; 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]);

}
```

Output



1.3

```
// first fit program

#include
#define max 25

int main()
{
    // b mein size of blocks
    // f mein size of files to be allocated

    int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0;
    static int bf[max], ff[max];

    printf("\n\tMemory Management Scheme - Worst Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
```

```
    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 files :-\n");
    for (i = 1; i <= nf; i++)
    {
        printf("File %d:", i);
        scanf("%d", &f[i]);
    }
    for (i = 1; i <= nf; i++)
    {

        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1) // if bf[j] is not allocated
            {
                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("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    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]);

}
```

Output

```
cd "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-4/1.3/" && gcc 1.3.c -o 1.3 && "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-4/1.3/"1.3
harshvardhan@MacBook-Air-5 ~ % cd "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-4/1.3/" && gcc 1.3.c -o 1.3 && "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-4/1.3/"1.3

        Memory Management Scheme - Worst Fit
Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               1               3               7               6
2               4               1               5               1
harshvardhan@MacBook-Air-5 1.3 %
```

Experiment - 5

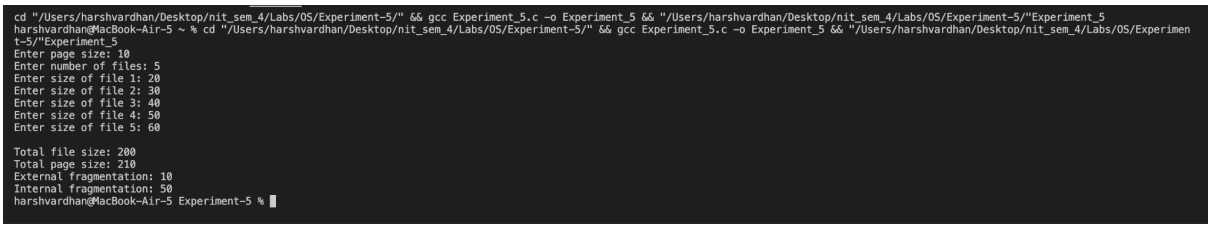
Q. Calculation of external and internal fragmentation.
It is assumed that files are generated randomly having varying size.
Every time program is executed for different page size.

Code:

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

int main() {
  int page_size, file_count, i;
  int *file_sizes;
  int total_file_size = 0, total_page_size = 0;
  int external_fragmentation = 0, internal_fragmentation = 0;

  printf("Enter page size: ");
  scanf("%d", &page_size);

  printf("Enter number of files: ");
  scanf("%d", &file_count);

  // Allocate memory for file sizes array
  file_sizes = (int *) malloc(file_count * sizeof(int));

  // Read file sizes from user input
  for(i = 0; i < file_count; i++) {
    printf("Enter size of file %d: ", i+1);
    scanf("%d", &file_sizes[i]);
    total_file_size += file_sizes[i];
  }

  // Calculate total page size
  total_page_size = (total_file_size / page_size + 1) * page_size;

  // Calculate external fragmentation
  external_fragmentation = total_page_size - total_file_size;

  // Calculate internal fragmentation for each file
  for(i = 0; i < file_count; i++) {
    internal_fragmentation += page_size - (file_sizes[i] % page_size);
  }

  printf("\nTotal file size: %d\n", total_file_size);
  printf("Total page size: %d\n", total_page_size);
  printf("External fragmentation: %d\n", external_fragmentation);
```

```
    printf("Internal fragmentation: %d\n", internal_fragmentation);

    // Free memory for file sizes array
    free(file_sizes);

    return 0;
}
```

Output:



Experiment - 6

Q. Implementation of Compaction for the continually changing memory layout and calculate total movement of data.
Compaction is a technique used to remove internal fragmentation.
Assumption that has to be taken into consideration is that the files must be inserted and deleted continuously. User must provide memory image at different instances of time.
In the assignment 5, we have obtained total internal and external fragmentation. To the above practical, we continue performing the compaction. Hereby, this activity is left for students to think and perform compaction to the above practical.

Code:

```
#include
#include
// #include
void create(int, int);
void del(int);
void compaction();
void display();

int fname[10], fsize[10], fstart[10], freest[10], freesize[10], m = 0, n = 0, start;

void create(int name, int size)
{
   int i, flag = 1, j, a;
   for (i = 0; i <= m; i++)
      if (freesize[i] >= size)
         a = i, flag = 0;
   if (!flag)
   {
      for (j = 0; j < n; j++)
         ;
      n++;
      fname[j] = name;
      fsize[j] = size;
      fstart[j] = freest[a];
      freest[a] = freest[a] + size;
      freesize[a] = freesize[a] - size;
      printf("\n The memory map will now be: \n\n");
      display();
   }
   else
   {
      printf("\nNo enough space is available.System compaction.....");
      flag = 1;
      compaction();
      display();
```

```
        for (i = 0; i <= m; i++)
            if (freesize[i] >= size)
                a = i, flag = 0;
        if (!flag)
        {
            for (j = 0; j < n; j++)
                ;
            n++;
            fname[j] = name;
            fsize[j] = size;
            fstart[j] = freest[a];
            freest[a] += size;
            freesize[a] -= size;
            printf("\n The memory map will now be: \n\n");
            display();
        }
        else
        {
          printf("\nNo enough space.\n");
        }

    }
}

void del(int name)
{
    int i, j, k, flag = 1;
    for (i = 0; i < n; i++)
        if (fname[i] == name)
            break;
    if (i == n)
    {
        flag = 0;
        printf("\nNo such process exists.....\n");
    }
    else
    {
        m++;
        freest[m] = fstart[i];
        freesize[m] = fsize[i];
        for (k = i; k < n; k++)
        {
            fname[k] = fname[k + 1];
            fsize[k] = fsize[k + 1];
            fstart[k] = fstart[k + 1];
        }
        n--;
    }
    if (flag)
    {
        printf("\n\n After deletion of this process the memory map will be : \n\n");
        display();
```

```
    }
}

void compaction()
{
    int i, j, size1 = 0, f_size = 0;
    if (fstart[0] != start)
    {
        fstart[0] = start;
        for (i = 1; i < n; i++)
            fstart[i] = fstart[i - 1] + fsize[i - 1];
    }
    else
    {
        for (i = 1; i < n; i++)
            fstart[i] = fstart[i - 1] + fsize[i - 1];
    }
    f_size = freesize[0];
    for (j = 0; j <= m; j++)
        size1 += freesize[j];
    freest[0] = freest[0] - (size1 - f_size);
    freesize[0] = size1;
    m = 0;
}

void display()
{
    int i;
    printf("\n  *** MEMORY MAP TABLE  *** \n");

    printf("\n\nNAME SIZE STARTING ADDRESS \n\n");

    for (i = 0; i < n; i++)
        printf(" %d%10d%10d\n", fname[i], fsize[i], fstart[i]);

    printf("\n\n");

    printf("\n\n***  FREE SPACE TABLE  ***\n\n");

    printf("FREE START ADDRESS FREE SIZE \n\n");

    for (i = 0; i <= m; i++)
        printf(" %d %d\n", freest[i], freesize[i]);
}

int main()
{
    int name, size, ch, i;
    int *ptr;
    // clrscr();

    ptr = (int *)malloc(sizeof(int) * 100);
```

```
    start = freest[0] = (int)ptr;

    freesize[0] = 500;

    printf("\n\n");

    printf(" Free start address Free Size \n\n");

    for (i = 0; i <= m; i++)
       printf(" %d %d\n", freest[i], freesize[i]);
    printf("\n\n");
    while (1)
    {
       printf("1.Create.\n");
       printf("2.Delete.\n");
       printf("3.Compaction.\n");
       printf("4.Exit.\n");
       printf("Enter your choice: ");
       scanf("%d", &ch);
       switch (ch)
       {
       case 1:
          printf("\nEnter the name of file: ");
          scanf("%d", &name);
          printf("\nEnter the size of the file: ");
          scanf("%d", &size);
          create(name, size);
          break;
       case 2:
          printf("\nEnter the file name which u want to delete: ");
          scanf("%d", &name);
          del(name);
          break;
       case 3:
          compaction();
          printf("\nAfter compaction the tables will be:\n");
          display();
          break;
       case 4:
          exit(1);
       default:
          printf("\nYou have entered a wrong choice.\n");
       }
    }
}
```

Output: 

Experiment - 7

Q:1. Implementation of resource allocation graph.

Code:

```
#include
#include

#define MAX_RESOURCES 10
#define MAX_PROCESSES 10

typedef struct Node
{
   int data;
   struct Node *next;
} Node;

typedef struct List
{
   Node *head;
} List;

int n,m;                    // number of processes and resources
int avail[MAX_RESOURCES];  // array to store number of available instances of each resource
List alloc[MAX_PROCESSES]; // array of lists to store allocation info
List req[MAX_PROCESSES];   // array of lists to store request info
int finish[MAX_PROCESSES]; // array to track finished processes

void init()
{
   // function to initialize data structures
   int i;
   for (i = 0; i < n; i++)
   {
      finish[i] = 0;
      alloc[i].head = NULL;
      req[i].head = NULL;
   }
}

void add_node(List *list, int data)
{
   // function to add a node to the end of a list
   Node *new_node = (Node *)malloc(sizeof(Node));
   new_node->data = data;
   new_node->next = NULL;
   if (list->head == NULL)
```

```
    {
        list->head = new_node;
    }
    else
    {
        Node *curr_node = list->head;
        while (curr_node->next != NULL)
        {
            curr_node = curr_node->next;
        }
        curr_node->next = new_node;
    }
}

void print_graph()
{
    // function to print the resource allocation graph
    int i, j;
    printf("\nResource Allocation Graph:\n\n");
    printf("Processes: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", i);
    }
    printf("\n");
    printf("Resources: ");
    for (j = 0; j < n; j++)
    {
        printf("%d ", j);
    }
    printf("\n\n");
    printf("Allocation List:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d: ", i);
        Node *curr_node = alloc[i].head;
        while (curr_node != NULL)
        {
            printf("%d ", curr_node->data);
            curr_node = curr_node->next;
        }
        printf("\n");
    }
    printf("\nRequest List:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d: ", i);
        Node *curr_node = req[i].head;
        while (curr_node != NULL)
        {
            printf("%d ", curr_node->data);
            curr_node = curr_node->next;
```

```
        }
        printf("\n");
    }
    printf("\nAvailable Resources: ");
    for (j = 0; j < n; j++)
    {
        printf("%d ", avail[j]);
    }
    printf("\n\n");
}

int main()
{
    // initialize resources
    int i, j, res, choice;
    printf("Enter the number of resources: ");
    scanf("%d", &n);
    printf("Enter the number of processes: ");
    scanf("%d", &m);
    for (i = 0; i < n; i++)
    {
        printf("Enter the number of instances of resource %d: ", i);
        scanf("%d", &res);
        avail[i] = res;
    }
    // initialize processes
    for (i = 0; i < m; i++)
    {
        printf("Enter the resource allocated by process %d: ", i);
        for (j = 0; j < n; j++)
        {
            scanf("%d", &choice);
            if (choice > 0)
            {
                add_node(&alloc[i], j);
                add_node(&alloc[i], j);
                avail[j] -= choice;
            }
            add_node(&req[i], choice);
        }
    }

    print_graph();


    return 0;
}
```

Output:

```
cd "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-7/" && gcc RAC_linkedList.c -o RAC_linkedList && "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-7/"RAC_linkedList
harshvardhan@MacBook-Air-5 ~ % cd "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-7/" && gcc RAC_linkedList.c -o RAC_linkedList && "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Exper
iment-7/"RAC_linkedList
Enter the number of resources: 2
Enter the number of processes: 3
Enter the number of instances of resource 0: 2
Enter the number of instances of resource 1: 3
Enter the resource allocated by process 0: 1 1
Enter the resource allocated by process 1: 0 1
Enter the resource allocated by process 2: 1 1

Resource Allocation Graph:

Processes: 0 1
Resources: 0 1

Allocation List:
0: 0 0 1 1
1: 1 1

Request List:
0: 1 1
1: 0 1

Available Resources: 0 0

harshvardhan@MacBook-Air-5 Experiment-7 %
```

```
cd "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-7/" && gcc RAC_linkedList.c -o RAC_linkedList && "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-7/"RAC_linkedList
harshvardhan@MacBook-Air-5 ~ % cd "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-7/" && gcc RAC_linkedList.c -o RAC_linkedList && "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Exper
iment-7/"RAC_linkedList
Enter the number of resources: 2
Enter the number of processes: 2
Enter the number of instances of resource 0: 1
Enter the number of instances of resource 1: 1
Enter the resource allocated by process 0: 1 0
Enter the resource allocated by process 1: 0 1

Resource Allocation Graph:

Processes: 0 1
Resources: 0 1

Allocation List:
0: 0 0
1: 1 1

Request List:
0: 1 0
1: 0 1

Available Resources: 0 0

harshvardhan@MacBook-Air-5 Experiment-7 %
```

Code:

```
#include
#include

#define MAX_RESOURCES 100
#define MAX_PROCESSES 100

int n,m;                              // number of processes and resources
int alloc[MAX_PROCESSES][MAX_RESOURCES]; // matrix to store allocation info
int req[MAX_PROCESSES][MAX_RESOURCES];   // matrix to store request info
int avail[MAX_RESOURCES];               // array to store number of available instances of each
resource
int finish[MAX_PROCESSES];              // array to track finished processes

void init()
{
   // function to initialize data structures
   int i, j;
   for (i = 0; i < n; i++)
   {
      finish[i] = 0;
      for (j = 0; j < m; j++)
      {
         alloc[i][j] = 0;
         req[i][j] = 0;
      }
   }
}

void print_graph()
```

```
{
    // function to print the resource allocation graph
    int i, j;
    printf("\nResource Allocation Graph:\n\n");
    printf("Processes: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", i);
    }
    printf("\n");
    printf("Resources: ");
    for (j = 0; j < m; j++)
    {
        printf("%d ", j);
    }
    printf("\n\n");
    printf("Allocation Matrix:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            printf("%d ", alloc[i][j]);
        }
        printf("\n");
    }
    printf("\nRequest Matrix:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            printf("%d ", req[i][j]);
        }
        printf("\n");
    }
    printf("\nAvailable Resources: ");
    int sum=0;
    for(int i=0;i
```

Output:

```
harshvardhan@MacBook-Air-5 ~ % cd "/Users/harshvardhan/Desktop/Htt_Sem_4/Labs/OS/Experiment-7/" && gcc RAC_matrix.c -o RAC_matrix && "/Users/harshvardhan/Desktop/Htt_Sem_4/Labs/OS/Experiment-7/
"RAC_matrix
Enter the number of processes : 2
Enter the number of resources :3
Enter the number of instances of each resource: 2 2 2

Enter the allocation matrix:
2 1 1
1 0 1

Enter the request matrix:
1 1 2
0 0 1

Resource Allocation Graph:

Processes: 0 1
Resources: 0 1 2

Allocation Matrix:
2 1 1
1 0 1

Request Matrix:
1 1 2
0 0 1

Available Resources: 2 2 2

harshvardhan@MacBook-Air-5 Experiment-7 %
```

Experiment - 8

Q: 1. Implementation of bankers algorithm.
Resource-request algorithm
Safety algorithm.

Code:

```
#include

int main()
{
    int n, m, i, j, k;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the number of resources: ");
    scanf("%d", &m);

    int alloc[n][m], max[n][m], avail[m];

    printf("Enter the allocation matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }

    printf("Enter the max matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    printf("Enter the available resources:\n");
    for (j = 0; j < m; j++) {
        scanf("%d", &avail[j]);
    }

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }
    int need[n][m];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {
```

```
            int flag = 0;
            for (j = 0; j < m; j++) {
                if (need[i][j] > avail[j]) {
                    flag = 1;
                    break;
                }
            }

            if (flag == 0) {
                ans[ind++] = i;
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y];
                f[i] = 1;
            }
        }
    }
}

int flag = 1;

// To check if sequence is safe or not
for (i = 0; i < n; i++) {
    if (f[i] == 0) {
        flag = 0;
        printf("The given sequence is not safe\n");
        break;
    }
}

if (flag == 1) {
    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d\n", ans[n - 1]);
}

return 0;
}
```

Output:

Experiment - 9

Code:

```
#include
#include

#define MAX_NODES 100

typedef struct node {
    int vertex;
    struct node *next;
} Node;

Node *RAG[MAX_NODES], *WFG[MAX_NODES];
int visited[MAX_NODES], path[MAX_NODES], n, path_len, cycle;

// function adds a node
void add_edge(Node **graph, int src, int dest) {
    Node *new_node = (Node*) malloc(sizeof(Node));
    new_node->vertex = dest;
    new_node->next = *graph;
    *graph = new_node;
}

// initialse some basic arrays which arre visisted path and WFG
void initialize() {
    int i;
    for(i = 0; i < n; i++) {
        visited[i] = 0;
        path[i] = 0;
        WFG[i] = NULL;
    }
}


// code for dfs search algo

int dfs(int node, int start) {
    visited[node] = 1;
    path[path_len++] = node;
    Node *curr = RAG[node];
    while(curr != NULL) {
        int i = curr->vertex;
```

```
        if(!visited[i]) {
            add_edge(&WFG[start], start, i);
            if(dfs(i, start)) {
                return 1;
            }
        } else if(visited[i] == 1) {
            int j;
            for(j = 0; j < path_len; j++) {
                if(path[j] == i) {
                    int k;
                    for(k = j; k < path_len; k++) {
                        add_edge(&WFG[path[k]], path[k], i);
                    }
                    cycle = 1;
                    return 1;
                }
            }
        }
        curr = curr->next;
    }
    path_len--;
    visited[node] = 2;
    return 0;
}


// main code to convert RAG to WAG

void rag_to_wfg() {
    int i;
    for(i = 0; i < n; i++) {
        if(!visited[i]) {
            dfs(i, i);
        }
    }
}

void print_graph(Node **graph) {
    int i;
    for(i = 0; i < n; i++) {
        printf("%d -> ", i);
        Node *curr = graph[i];
        while(curr != NULL) {
            printf("%d ", curr->vertex);
            curr = curr->next;
        }
        printf("\n");
    }
}

int main() {
    // to take input for number of vertices , edges and edges info
```

```
    int i, j, m;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    printf("Enter the number of edges: ");
    scanf("%d", &m);
    printf("Enter the edges:\n");
    for(i = 0; i < m; i++) {
        int src, dest;
        scanf("%d%d", &src, &dest);
        add_edge(&RAG[src], src, dest);
    }

    initialize();
    rag_to_wfg();
    printf("\nResource Allocation Graph:\n");
    print_graph(RAG);
    printf("\nWait-for Graph:\n");
    print_graph(WFG);

    // To detect whether a cycle is present or not

    if(cycle) {
        printf("\nWait-for Graph contains a cycle\n");
    } else {
        printf("\nWait-for Graph does not contain a cycle\n");
    }
    return 0;
}
```

Output:

Code:

```
#include
#define MAX_NODES 100

int RAG[MAX_NODES][MAX_NODES], WFG[MAX_NODES][MAX_NODES], visited[MAX_NODES],
path[MAX_NODES];
int n, path_len;

void initialize() {
    int i, j;
    for(i = 0; i < n; i++) {
```

```
        visited[i] = 0; // initialize visited array to 0
        path[i] = 0; // initialize path array to 0
        for(j = 0; j < n; j++) {
            WFG[i][j] = 0; // initialize WFG matrix to 0
        }
    }
}

int dfs(int node, int start) {
    visited[node] = 1; // mark node as visited
    path[path_len++] = node; // add node to current path
    int i;
    for(i = 0; i < n; i++) {
        if(RAG[node][i] != 0) { // check if there is an edge from node to i
            if(!visited[i]) { // if i has not been visited, add an edge to WFG
                WFG[start][i] = 1;
                if(dfs(i, start)) { // continue dfs from i
                    return 1; // return 1 if cycle is detected
                }
            } else {
                int j;
                for(j = 0; j < path_len; j++) { // check if i is in the current path
                    if(path[j] == i) {
                        int k;
                        for(k = j; k < path_len; k++) {
                            WFG[path[k]][i] = 1; // add edges to WFG to represent cycle
                        }
                        return 1; // return 1 if cycle is detected
                    }
                }
            }
        }
    }
    path_len--; // remove node from current path
    visited[node] = 2; // mark node as completed
    return 0; // return 0 if no cycle is detected
}

void convertRAGtoWFG() {
    int i;
    for(i = 0; i < n; i++) {
        path_len = 0; // reset path length
        if(dfs(i, i)) { // start dfs from each node
            printf("Cycle detected in the wait-for graph.\n");
            return;
        }
    }
    printf("No cycle detected in the wait-for graph.\n");
}

int main() {
    printf("Enter the number of nodes in the graph: ");
```

```
    scanf("%d", &n);
    int i, j;
    printf("Enter the RAG as an adjacency matrix: \n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            scanf("%d", &RAG[i][j]);
        }
    }
    initialize();
    convertRAGtoWFG();
    printf("The wait-for graph is: \n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            printf("%d ", WFG[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

Experiment - 10

1. Implementation of FORK and JOIN construct.
a. Program where parent process counts number of vowels in the given sentence and child process will count number of words in the same sentence.
b. Program where parent process sorts array elements in descending order and child process sorts array elements in ascending order.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/wait.h>

int vowelCount(char *s)
{
    int num_vowels = 0;

    for (int i = 0; s[i] != '\0'; i++)
    {
        if (s[i] == 'a' || s[i] == 'e' || s[i] == 'i' ||
            s[i] == 'o' || s[i] == 'u' || s[i] == 'A' ||
            s[i] == 'E' || s[i] == 'I' || s[i] == 'O' || s[i] == 'U')
        {
            num_vowels++;
        }
    }

    return num_vowels;
}

int wordCount(char *s)
{
    int num_words = 0;
    char *word = strtok(s, " \n");

    while (word != NULL)
    {
        num_words++;
        word = strtok(NULL, " \n");
    }

    return num_words;
```

```
}

int main()
{
    char s[100];
    printf("Enter a sentence: ");
    fgets(s, sizeof(s), stdin);

    pid_t process_id = fork();

    if (process_id < 0)
    {
        fprintf(stderr, "Fork has been failed");
        return 1;
    }
    else if (process_id == 0)
    { // child process
        int num_words = wordCount(s);
        exit(num_words);
    }
    else
    { // parent process
        int num_vowels = vowelCount(s);

        int status;
        wait(&status);

        printf("Number of vowels: %d\n", num_vowels);
        printf("Number of words: %d\n", WEXITSTATUS(status));

        return 0;
    }
}
```

Output:

Code:

```
#include
#include
#include
#include

// this function will sort the array in descending order(Bubble sort)
void descendingSort(int arr[], int n)
{
        int i, j, temp;
        for (i = 0; i < n - 1; i++)
```

```
	{
		for (j = 0; j < n - i - 1; j++)
		{
			if (arr[j] < arr[j + 1])
			{
				temp = arr[j];
				arr[j] = arr[j + 1];
				arr[j + 1] = temp;
			}
		}
	}
}

// this function will sort the array in ascending order(Bubble sort)

void ascendingSort(int arr[], int n)
{
	int i, j, temp;
	for (i = 0; i < n - 1; i++)
	{
		for (j = 0; j < n - i - 1; j++)
		{
			if (arr[j] > arr[j + 1])
			{
				temp = arr[j];
				arr[j] = arr[j + 1];
				arr[j + 1] = temp;
			}
		}
	}
}

int main()
{
	int num; // total number of elements in the array

	printf("Enter elements in array: ");
	scanf("%d", &num);
	int arr[n];
	printf("Enter the elements of the array: "); //elements in the array
	for (int i = 0; i < num; i++)
	{
		scanf("%d", &arr[i]);
	}
	int pid = fork();
	if (pid == 0)
	{
		// Child process sorts the array in ascending order
		ascendingSort(arr, num);
		printf("Ascending order: ");
		for (int i = 0; i < num; i++)
		{
```

```
                printf("%d ", arr[i]);
            }
            printf("\n");
        }
        else
        {
            // Parent process sorts the array in descending order
            wait(NULL);
            descendingSort(arr, num);
            printf("Descending order: ");
            for (int i = 0; i < num; i++)
            {
                printf("%d ", arr[i]);
            }
            printf("\n");
        }
        return 0;
}
```

output:

Experiment - 11

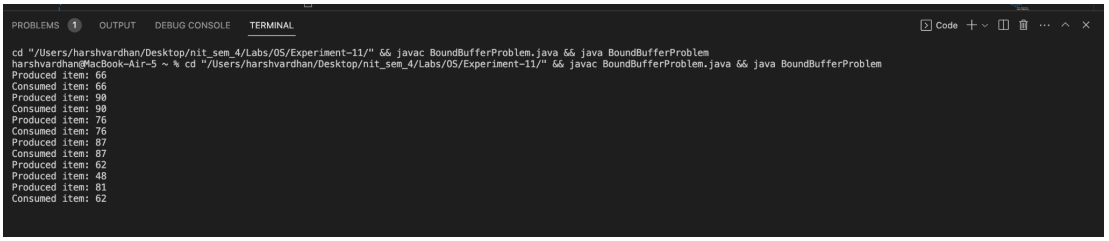
1. Implementation of semaphores for concurrency. Implementation of Inter Process Communication techniques :-
a. Bound Buffer
b. Reader-Writer
c. Dining-Philosopher

Code:

```
import java.util.concurrent.Semaphore;



public class BoundBufferProblem {


    private static final int BUFFER_SIZE = 5; // total no of peaple eating
    private static int[] buffer = new int[BUFFER_SIZE]; // array containing them

    private static int numItems = 0; // number of items
    private static int in = 0; // variable for in use person
    private static int out = 0; // for not in use persons
    private static Semaphore mutex = new Semaphore(1);
    private static Semaphore full = new Semaphore(0);
    private static Semaphore empty = new Semaphore(BUFFER_SIZE);

    public static void main(String[] args) {
        // thread is created for produces which is process in our case
        Thread producer = new Thread(() -> {
            while (true) {
                try {
                    empty.acquire();
                    mutex.acquire();
                    int item = (int) (Math.random() * 100);
                    buffer[in] = item;
                    in = (in + 1) % BUFFER_SIZE;
                    numItems++;
                    System.out.println("Produced item: " + item);
                    mutex.release();
                    full.release();
                    Thread.sleep((long) (Math.random() * 1000));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
```

```
        });
        producer.start();
        // thread for consumer is created which is CPU in our case
        Thread consumer = new Thread(() -> {
            while (true) {
                try {
                    full.acquire();
                    mutex.acquire();
                    int item = buffer[out];
                    out = (out + 1) % BUFFER_SIZE;
                    numItems--;
                    System.out.println("Consumed item: " + item);
                    mutex.release();
                    empty.release();
                    Thread.sleep((long) (Math.random() * 1000));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        consumer.start();
    }
}
```

output :



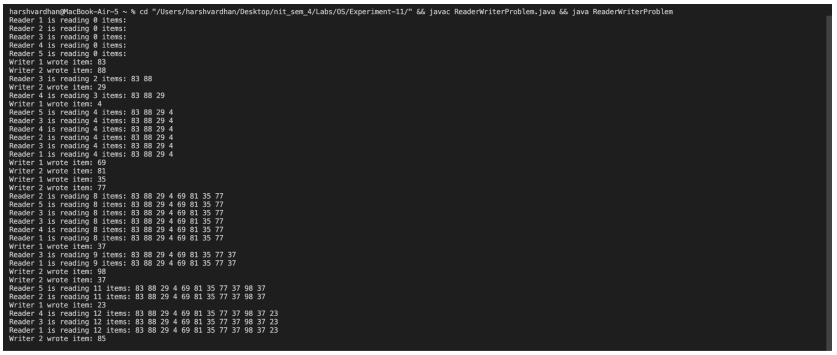
Code:

```
import java.util.concurrent.Semaphore;



public class ReaderWriterProblem {
    private static final int BUFFER_SIZE = 100;

    private static final int NUM_READERS = 5;
    private static final int NUM_WRITERS = 2;
    private static int[] buffer = new int[BUFFER_SIZE];
    private static int numItems = 0;

    // Create three semaphores: mutex (for readers/writers access to numItems), writeMutex (for writers
to access buffer), and readMutex (for readers to access buffer)
    private static Semaphore mutex = new Semaphore(1);
    private static Semaphore writeMutex = new Semaphore(1);
    private static Semaphore readMutex = new Semaphore(NUM_READERS);
```

```
public static void main(String[] args) {
    Thread[] readers = new Thread[NUM_READERS];
    Thread[] writers = new Thread[NUM_WRITERS];

    // Create threads for readers
    for (int i = 0; i < NUM_READERS; i++) {
        int id = i + 1;
        readers[i] = new Thread(() -> {
            while (true) {
                try {
                    // Acquire the read mutex
                    readMutex.acquire();

                    // Acquire the mutex to access numItems
                    mutex.acquire();

                    // Read items from buffer and print them out
                    System.out.print("Reader " + id + " is reading " + numItems + " items: ");
                    for (int j = 0; j < numItems; j++) {
                        System.out.print(buffer[j] + " ");
                    }
                    System.out.println();

                    // Release the mutex to access numItems
                    mutex.release();

                    // Release the read mutex
                    readMutex.release();

                    // Sleep for a random amount of time (up to 1 second)
                    Thread.sleep((long)(Math.random() * 1000));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        readers[i].start();
    }

    // Create threads for writers
    for (int i = 0; i < NUM_WRITERS; i++) {
        int id = i + 1;
        writers[i] = new Thread(() -> {
            while (true) {
                try {
                    // Acquire the write mutex
                    writeMutex.acquire();

                    // Acquire the mutex to access numItems
                    mutex.acquire();

                    // Generate a random item and add it to the buffer
```

```
                int item = (int)(Math.random() * 100);
                buffer[numItems] = item;
                numItems++;

                // Print out that a writer wrote an item
                System.out.println("Writer " + id + " wrote item: " + item);

                // Release the mutex to access numItems
                mutex.release();

                // Release the write mutex
                writeMutex.release();

                // Sleep for a random amount of time (up to 1 second)
                Thread.sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    });
    writers[i].start();
  }
 }
}
```

output :



Code:

```
import java.util.concurrent.Semaphore;



public class DiningPhilosophers {
   private static final int NUM_PHILOSOPHERS = 5; // total number of phiosphers

   private static final Semaphore[] forks = new Semaphore[NUM_PHILOSOPHERS]; // forks created
   private static final Semaphore maxDiners = new Semaphore(NUM_PHILOSOPHERS - 1); // max
number that can eat at a time

   public static void main(String[] args) {
      // creates semaphores  equal to number of philosphers
      for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
```

```
            forks[i] = new Semaphore(1);
        }
        // creates left and right fork for all philosphers
        for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
            int id = i;
            int leftFork = i;
            int rightFork = (i + 1) % NUM_PHILOSOPHERS;
        // threads are created
            Thread philosopher = new Thread(() -> {
                while (true) {
                    try {
                        // Philosopher is thinking
                        Thread.sleep((long)(Math.random() * 1000));

                        // Philosopher is hungry
                        maxDiners.acquire();
                        forks[leftFork].acquire();
                        forks[rightFork].acquire();

                        // Philosopher is eating
                        System.out.println("Philosopher " + id + " is eating");
                        Thread.sleep((long)(Math.random() * 1000));

                        // Philosopher is done eating
                        forks[leftFork].release();
                        forks[rightFork].release();
                        maxDiners.release();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });
            philosopher.start();
        }
    }
}
```

output :

```
cd "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-11/" && javac DiningPhilosophers.java && java DiningPhilosophers
harshvardhan@MacBook-Air-5 ~ % cd "/Users/harshvardhan/Desktop/nit_sem_4/Labs/OS/Experiment-11/" && javac DiningPhilosophers.java && java DiningPhilosophers
Philosopher 2 is eating
Philosopher 3 is eating
Philosopher 1 is eating
Philosopher 4 is eating
Philosopher 2 is eating
Philosopher 0 is eating
Philosopher 3 is eating
Philosopher 4 is eating
Philosopher 2 is eating
Philosopher 1 is eating
Philosopher 3 is eating
Philosopher 0 is eating
Philosopher 0 is eating
Philosopher 2 is eating
Philosopher 4 is eating
Philosopher 1 is eating
Philosopher 3 is eating
Philosopher 0 is eating
Philosopher 2 is eating
Philosopher 1 is eating
Philosopher 4 is eating
Philosopher 3 is eating
Philosopher 0 is eating
Philosopher 2 is eating
Philosopher 4 is eating
```