Fair Record with the following programs to be submitted on or before 25.7.22
Left Side-Program and output-Paste Printouts(should be clear)
Right side -Pgm no, Date, Question Aim, Algorithm, Result(all Handwritten)

1.Familiarization of Linux Commands.
2. Introduction to Shell programming
   a. To write a program to find whether a number is even or odd
   b. To write a program to find the biggest in two numbers.
   c. To Write a Program to Find Biggest In Three Numbers.
   d. To find a factorial of a number using shell script.
   e. To write a program to display the Fibonacci series.
   f. Print the given pattern
3.Write programs using the following system calls of Linux operating system : fork, getpid, exit, wait.

4. Program to implement opendir, readdir

5. Given the list of processes, their CPU burst times and arrival times, priority values display/print the Gantt chart for the given scheduling policies, compute and print the average waiting time and average turnaround time a)FCFS b) SJF c) Round Robin (pre-emptive) d) Priority
6.Implement programs for Inter Process Communication using Shared Memory
7. Write a C program to simulate producer-consumer problem using semaphores.

1.Familiarization of Linux Commands.
2. Introduction to Shell programming
8. Implement the banker's algorithm for deadlock avoidance.

9. Write a C program to simulate disk scheduling algorithms a) FCFS b) SCAN c) C-SCAN

Bankers Algorithm for Deadlock Avoidance

Let Work and Finish be vectors of length 'm' and 'n' respectively.
Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4....n
2) Find an i such that both
a) Finish[i] = false
b) Need$_i$ <= Work

if no such i exists goto step (4)
3) Work = Work + Allocation[i]
Finish[i] = true
goto step (2)
4) if Finish [i] = true for all i
then the system is in a safe state

```
#include <stdio.h>
int main()
{
   // P0, P1, P2, P3, P4 are the Process names here

   int n, m, i, j, k;
   n = 5; // Number of processes
   m = 3; // Number of resources
   int alloc[5][3] = { { 0, 1, 0 }, // P0                    // Allocation Matrix
                       { 2, 0, 0 }, // P1
                       { 3, 0, 2 }, // P2
                       { 2, 1, 1 }, // P3
                       { 0, 0, 2 } }; // P4

   int max[5][3] = { { 7, 5, 3 }, // P0                      // MAX Matrix
                     { 3, 2, 2 }, // P1
                     { 9, 0, 2 }, // P2
                     { 2, 2, 2 }, // P3
                     { 4, 3, 3 } }; // P4

   int avail[3] = { 3, 3, 2 }; // Available Resources
```

```
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 < 5; 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;

       for(int i=0;i<n;i++)
      {
       if(f[i]==0)
       {
         flag=0;
          printf("The following system is not safe");
         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", ans[n - 1]);
      }
    return (0);
}
```

Output

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
```

Producer Consumer Using Semaphores

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

int mutex=1,full=0,empty=3,x=0;

int main()
{
        int n;
        void producer();
        void consumer();
        int wait(int);
        int signal(int);
        printf("\n1.Producer\n2.Consumer\n3.Exit");
        while(1)
        {
                printf("\nEnter your choice:");
                scanf("%d",&n);
                switch(n)
                {
                        case 1:     if((mutex==1)&&(empty!=0))
                                            producer();
                                    else
                                            printf("Buffer is full!!");
                                    break;
                        case 2:     if((mutex==1)&&(full!=0))
                                            consumer();
```

```
                        else
                                printf("Buffer is empty!!");
                        break;
                case 3:
                        exit(0);
                        break;
                }
        }

        return 0;
}

int wait(int s)
{
        return (--s);
}

int signal(int s)
{
        return(++s);
}

void producer()
{
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
```

```
        printf("\nProducer produces the item %d",x);
        mutex=signal(mutex);
}

void consumer()
{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\nConsumer consumes item %d",x);
        x--;
        mutex=signal(mutex);
}
```

```
1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:1

Producer produces the item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3
```

Round Robin

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

void main()
{
   // initlialize the variable name
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10],
temp[10];
   float avg_wt, avg_tat;
   printf(" Total number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and
the Burst Time
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t");  // Accept arrival time
scanf("%d", &at[i]);
printf(" \nBurst time is: \t"); // Accept the Burst time
scanf("%d", &bt[i]);
temp[i] = bt[i]; // store the burst time in temp array
```

```
}
// Accept the Time qunat
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
// Display the process No, burst time, Turn Around Time and the waiting time
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
   sum = sum + temp[i];
   temp[i] = 0;
   count=1;
   }
   else if(temp[i] > 0)
   {
      temp[i] = temp[i] - quant;
      sum = sum + quant;
   }
   if(temp[i]==0 && count==1)
   {
      y--; //decrement the process no.
      printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
      wt = wt+sum-at[i]-bt[i];
      tat = tat+sum-at[i];
      count =0;
```

```
    }
    if(i==NOP-1)
    {
        i=0;
    }
    else if(at[i+1]<=sum)
    {
        i++;
    }
    else
    {
        i=0;
    }
}
// represents the average waiting time and Turn Around time
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f" , avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();
}
```

```
 Total number of process in the system: 4

 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:       0

Burst time is:  8

 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:       1

Burst time is:  5

 Enter the Arrival and Burst time of the Process[3]
 Arrival time is:       2

Burst time is:  10

 Enter the Arrival and Burst time of the Process[4]
 Arrival time is:       3

Burst time is:  11
Enter the Time Quantum for the process:         6

 Process No             Burst Time             TAT            Waiting Time

 Process No[2]          5              10             5
 Process No[1]          8              25             17
 Process No[3]          10             27             17
 Process No[4]          11             31             20
 Average Turn Around Time:      14.750000
 Average Waiting Time:  23.250000
```

Code to add Gantt Chart after process scheduling

```
#include <stdio.h>

int main()
{
   int a[6]={0,5,10,25,30,50};
   for(int i=1;i<6;i++)
   {
      printf("| P");
      printf("%d",i);
      int n=a[i]-a[i-1];
     // printf("%d",n);
      for(int j=0;j<n;j++)
      printf(" ");
      printf("--%d",a[i]);



   }
}
```

Program implementing opendir, closedir, readdir

```
#include<stdio.h>
#include<stdlib.h>
#include<dirent.h>
struct dirent *dptr;
int main(int argc,char *argv[])
{
char buff[256];
DIR *dirp;
printf("\n\nEnter directory name");
scanf("%s",buff);
if((dirp=opendir(buff))==NULL)
{
printf("Error");
exit(1);
}
while(dptr=readdir(dirp))
{
printf("%s\n",dptr->d_name);
}
closedir(dirp);
}
```

Output

```
Enter directory name/home/pg-45/Lab
```

```
dire.c
.
a.out
Untitled Document
..
test
processcrea.c
```

Program implementing fork(), getpid(), getppid(), exit, wait System calls

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
pid_t cpid;
int pid,status,exitch;
pid=fork();

if(pid==-1)
{
perror("error");
exit(0);
}
if(pid==0)
{
exit(0);
printf("child: My process id is %d\n",getpid());
```

```
printf("child: My parent process id is %d\n",getppid());

}
else
{

printf("Parent: My process id is %d\n",getpid());
printf("Parent: My parent process id is %d\n",getppid());
wait(NULL);
printf("Child Terminated");

}
}
```

```
 Output :
Parent: My process id is 9638
Parent: My parent process id is 7476
Child Terminated
```

Disk Scheduling

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 int ReadyQueue[100],i,n,TotalHeadMov=0,initial,c,size,move,j;
 printf("Enter the number of requests");
 scanf("%d",&n);
 printf("Enter the sequence of request");
 for(i=0;i<n;i++)
 {
 scanf("%d",&ReadyQueue[i]);
 }
 printf("Enter initial head position");
 scanf("%d",&initial);
 printf("Choice 1 : FCFS\nChoice 2: SCAN \nChoice 3: C-SCAN\nEnter
  you Choice ");
 scanf("%d",&c);
 switch(c)
 {
    case 1 :  for(i=0;i<n;i++)
              {
              TotalHeadMov=TotalHeadMov+abs(ReadyQueue[i]-initial);
              initial=ReadyQueue[i];
              }
```

```
        printf("Total Head Movement in FCFS Disk Scheduling is :
%d",TotalHeadMov);
        break;
  case 2 :  printf("Enter total disk size");
        scanf("%d",&size);
        printf("Enter head direction 1 for high and 0 for low");
        scanf("%d",&move);
        for(i=0;i<n;i++)
        {
        for(j=0;j<n-i-1;j++)
        {
          if(ReadyQueue[j]>ReadyQueue[j+1])
          {
          int temp;
          temp=ReadyQueue[j];
          ReadyQueue[j]=ReadyQueue[j+1];
          ReadyQueue[j+1]=temp;
          }
        }
        }
        if(move==1)
        {
          TotalHeadMov=size-1-initial+size-1-ReadyQueue[0];
        }
        else
        {
          TotalHeadMov=initial+ReadyQueue[n-1];
        }
```

```
        printf("Total Head Movement in SCAN Disk Scheduling is
%d",TotalHeadMov);
        break;
  case 3 :  printf("Enter total disk size");
        scanf("%d",&size);
        printf("Enter head direction 1 for high and 0 for low");
        scanf("%d",&move);
        for(i=0;i<n;i++)
        {
        for(j=0;j<n-i-1;j++)
        {
          if(ReadyQueue[j]>ReadyQueue[j+1])
          {
          int temp;
          temp=ReadyQueue[j];
          ReadyQueue[j]=ReadyQueue[j+1];
          ReadyQueue[j+1]=temp;
          }
        }
        }
        int index;
        for(i=0;i<n;i++)
        {
          if(initial<ReadyQueue[i])
          {
            index=i;
            break;
          }
```

```
            }
            if(move==1)
            {
               TotalHeadMov=size-1-initial+size-1+ReadyQueue[index-1];
            }
            else
            {
               TotalHeadMov=initial+size-1+size-1-ReadyQueue[index];
            }
            printf("Total Head Movement in C-SCAN Disk Scheduling is
  %d",TotalHeadMov);
            break;
        }
}
```