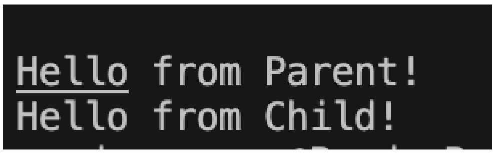
PRACTICAL -7

AIM: Write a C program to create a child process.

Code:

```
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
// Check if the return value of fork() is 0
if (fork() == 0)
{
printf("Hello from Child!\n");
}
// If the return value of fork() is non-zero
else
{
printf("Hello from Parent!\n");
}
}
int main()
{
forkexample();
return 0;
}
```

Output:



```
Hello from Parent!
Hello from Child!
```

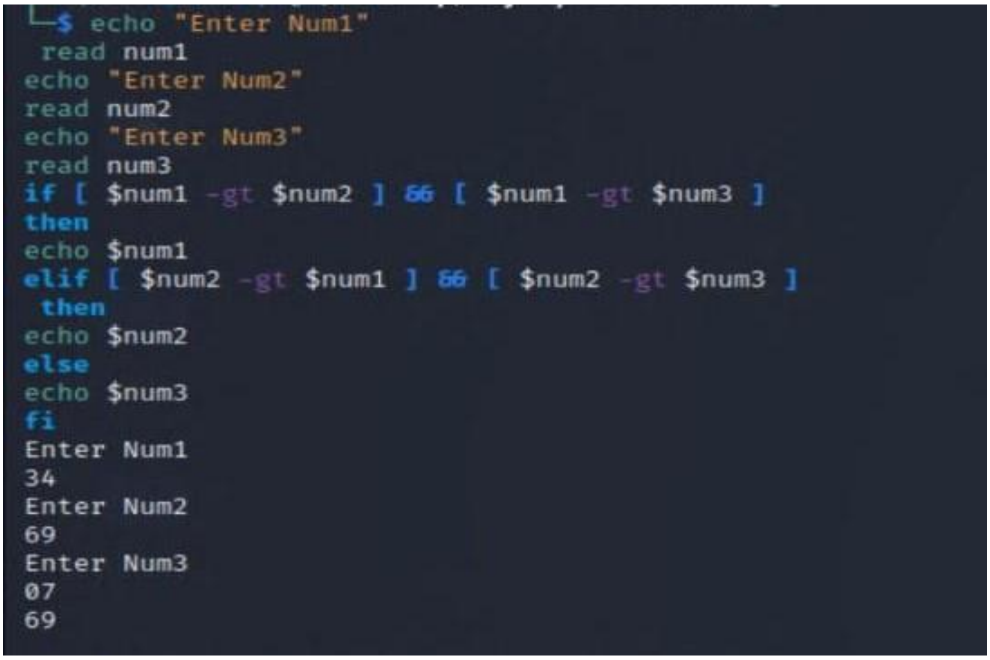
PRACTICAL- 8

AIM: Finding out biggest number from given three numbers supplied as command line arguments.

Input:

```
echo "Enter Num1" read
 num1
echo "Enter Num2" read
num2
echo "Enter Num3" read
num3
if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ] then
echo $num1
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
 then
echo $num2
else
echo $num3 fi
```

Output:



```
└─$ echo "Enter Num1"
  read num1
echo "Enter Num2"
read num2
echo "Enter Num3"
read num3
if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
echo $num1
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
 then
echo $num2
else
echo $num3
fi
Enter Num1
34
Enter Num2
69
Enter Num3
07
69
```

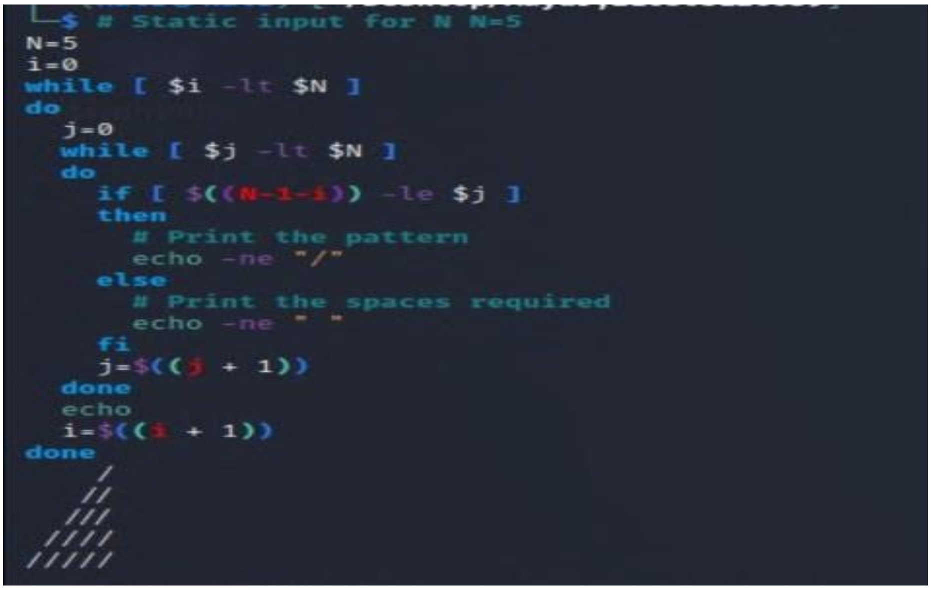
PRACTICAL- 9

AIM: Printing the patterns using for loop

Input:

```
# Static input for N N=5
N=5
i=0
while [ $i -lt $N ] do
  j=0
  while [ $j -lt $N ]
  do
    if [ $((N-1-i)) -le $j ]
    then
      # Print the pattern echo
      -ne "/"
    else
      # Print the spaces required
      echo -ne " "
    fi
    j=$((j + 1))
  done
  echo
  i=$((i + 1))
done
```

Output:



```
└─$ # Static input for N N=5
N=5
i=0
while [ $i -lt $N ]
do
  j=0
  while [ $j -lt $N ]
  do
    if [ $((N-1-i)) -le $j ]
    then
      # Print the pattern
      echo -ne "/"
    else
      # Print the spaces required
      echo -ne " "
    fi
    j=$((j + 1))
  done
  echo
  i=$((i + 1))
done
    /
   //
  ///
 ////
/////
```

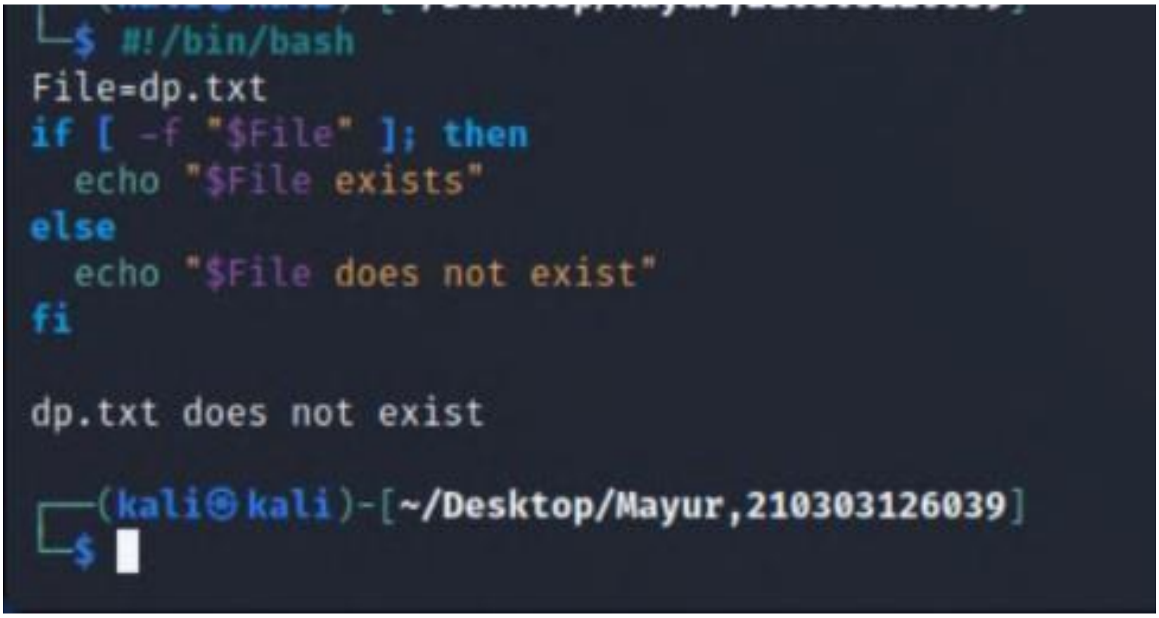
PRACTICAL-10

AIM: Shell script to determine whether given file exist or not.

Input:

```
#!/bin/bash
File=dp.txt
if [ -f "$File" ]; then
echo "$File exists"
else echo "$File does not exist"
fi
```

Output:



```
└─$ #!/bin/bash
File=dp.txt
if [ -f "$File" ]; then
  echo "$File exists"
else
  echo "$File does not exist"
fi

dp.txt does not exist

┌──(kali㉿kali)-[~/Desktop/Mayur,210303126039]
└─$ █
```

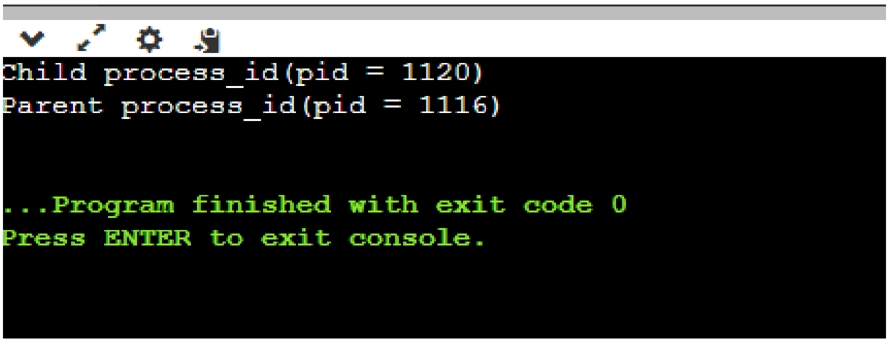
PRACTICAL-11

AIM: Write a program for process creation using C (Use of gcc compiler)

Code :

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
int main(void){
    int pid = fork();
    if(pid= =0){
      printf("Child process_id(pid = %d) \n",getpid());
    }
    else if(pid>0){
        int status;
        wait (&status);
        printf("Parent process_id(pid = %d) \n",getpid());
    }
    else{
        printf("fork");
    }
    return 0;
}
```

<u>Output</u>:



```
Child process_id(pid = 1120)
Parent process_id(pid = 1116)


...Program finished with exit code 0
Press ENTER to exit console.
```

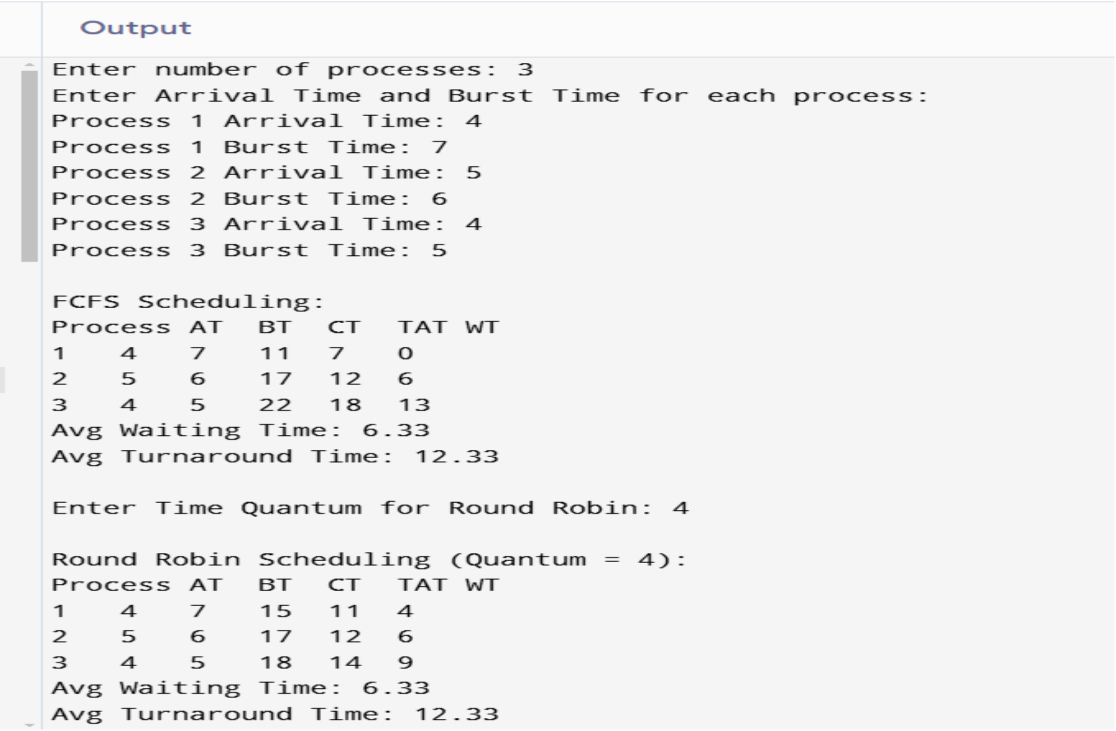
PRACTICAL-12

AIM : Implementation of FCFS Algorithm.

Code:

```
#include <stdio.h>
// FCFS Scheduling
   void fcfs(int n, int at[], int bt[]) {
    int ct[n], tat[n], wt[n], total_wt = 0, total_tat = 0;
    ct[0] = at[0] + bt[0];
    for (int i = 1; i < n; i++)
     ct[i] = (ct[i - 1] > at[i]) ? ct[i - 1] + bt[i] : at[i] + bt[i];
  for (int i = 0; i < n; i++) {
        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt[i];
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("\nFCFS Scheduling:\n");
    printf("Process\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], ct[i], tat[i], wt[i]);

    printf("Avg Waiting Time: %.2f\n", (float)total_wt / n);
    printf("Avg Turnaround Time: %.2f\n", (float)total_tat / n);
}

// Round Robin Scheduling
void roundRobin(int n, int at[], int bt[], int quantum) {
    int remaining_bt[n], ct[n], tat[n], wt[n], total_wt = 0, total_tat = 0, time = 0, done = 0;
  for (int i = 0; i < n; i++) remaining_bt[i] = bt[i];
  while (done < n) {
        for (int i = 0; i < n; i++) {
            if (remaining_bt[i] > 0) {
                int exec_time = (remaining_bt[i] > quantum) ? quantum : remaining_bt[i];
                time += exec_time;
                remaining_bt[i] -= exec_time;
                if (remaining_bt[i] == 0) {
                    ct[i] = time;
                    tat[i] = ct[i] - at[i];
                    wt[i] = tat[i] - bt[i];
                    total_wt += wt[i];
                    total_tat += tat[i];
                    done++;
                }
            }
        }
    }

    printf("\nRound Robin Scheduling (Quantum = %d):\n", quantum);
    printf("Process\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
   printf("%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], ct[i], tat[i], wt[i]);
     printf("Avg Waiting Time: %.2f\n", (float)total_wt / n);
    printf("Avg Turnaround Time: %.2f\n", (float)total_tat / n);
}
int main() {
    int n, quantum;
    printf("Enter number of processes: ");
    scanf("%d", &n);
     int at[n], bt[n];
 printf("Enter Arrival Time and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d Arrival Time: ", i + 1);
        scanf("%d", &at[i]);
        printf("Process %d Burst Time: ", i + 1);
        scanf("%d", &bt[i]);
    }
 fcfs(n, at, bt);
 printf("\nEnter Time Quantum for Round Robin: ");
   scanf("%d", &quantum);
  roundRobin(n, at, bt, quantum);  // Run Round Robin
  return 0;
}
```

Output:



```
Enter number of processes: 3
Enter Arrival Time and Burst Time for each process:
Process 1 Arrival Time: 4
Process 1 Burst Time: 7
Process 2 Arrival Time: 5
Process 2 Burst Time: 6
Process 3 Arrival Time: 4
Process 3 Burst Time: 5

FCFS Scheduling:
Process AT  BT  CT  TAT WT
1   4   7   11  7   0
2   5   6   17  12  6
3   4   5   22  18  13
Avg Waiting Time: 6.33
Avg Turnaround Time: 12.33

Enter Time Quantum for Round Robin: 4

Round Robin Scheduling (Quantum = 4):
Process AT  BT  CT  TAT WT
1   4   7   15  11  4
2   5   6   17  12  6
3   4   5   18  14  9
Avg Waiting Time: 6.33
Avg Turnaround Time: 12.33
```

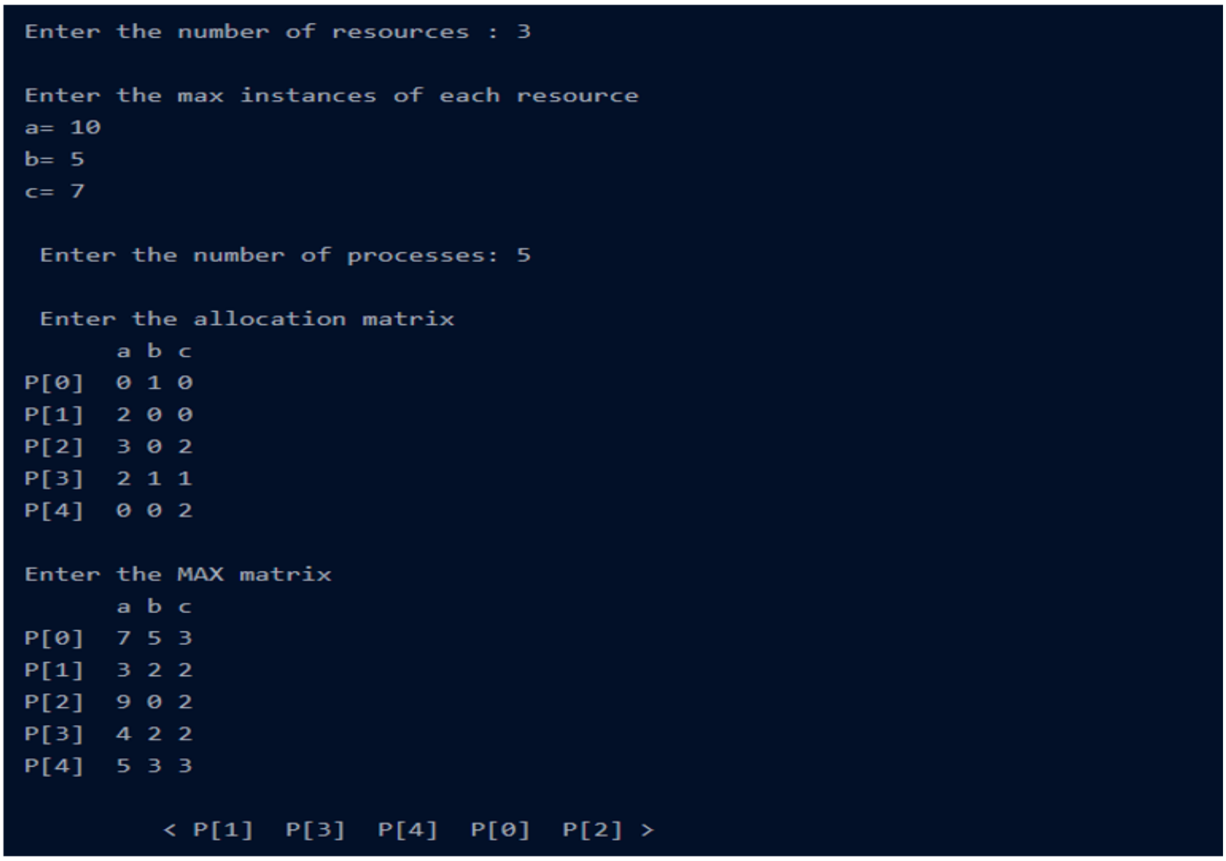
Practical -13

AIM: Implementation of Banker’s Algorithm

CODE:

```
#include <stdio.h>
int isSafe(int n, int m, int alloc[][m], int max[][m], int avail[]) {
    int need[n][m], safeSeq[n], finish[n];
    int work[m];

    // Calculate Need matrix: Need[i][j] = Max[i][j] - Alloc[i][j]
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];

    // Initialize Work = Available resources
    for (int i = 0; i < m; i++)
        work[i] = avail[i];

    // Initialize Finish array to false (0)
    for (int i = 0; i < n; i++)
        finish[i] = 0;

    int count = 0;
    while (count < n) {
        int found = 0;
        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                int flag = 1;
                for (int j = 0; j < m; j++) {
                    if (need[i][j] > work[j]) {
                        flag = 0;
                        break;
                    }
                }
                if (flag) {
                    // Allocate resources
                    for (int j = 0; j < m; j++)
                        work[j] += alloc[i][j];
                    safeSeq[count++] = i;
                    finish[i] = 1;
                    found = 1;
                }
            }
        }
        if (!found) {
            printf("System is in an unsafe state!\n");
            return 0;
        }
    }

    // Safe Sequence found
    printf("System is in a safe state.\nSafe Sequence: ");
    for (int i = 0; i < n; i++)
        printf("P%d ", safeSeq[i]);
    printf("\n");

    return 1;
}

// Main function
int main() {
    int n, m;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("Enter number of resource types: ");
    scanf("%d", &m);

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

    // Input Allocation matrix
    printf("Enter Allocation Matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);

    // Input Max matrix
    printf("Enter Maximum Matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &max[i][j]);
    // Input Available resources
    printf("Enter Available resources:\n");
    for (int i = 0; i < m; i++)
        scanf("%d", &avail[i]);

    // Check safe state
    isSafe(n, m, alloc, max, avail);

    return 0;
}
```

Output:



```
Enter the number of resources : 3

Enter the max instances of each resource
a= 10
b= 5
c= 7

 Enter the number of processes: 5

 Enter the allocation matrix
      a b c
P[0]  0 1 0
P[1]  2 0 0
P[2]  3 0 2
P[3]  2 1 1
P[4]  0 0 2

Enter the MAX matrix
      a b c
P[0]  7 5 3
P[1]  3 2 2
P[2]  9 0 2
P[3]  4 2 2
P[4]  5 3 3

          < P[1]  P[3]  P[4]  P[0]  P[2] >
```