LIST OF PROGRAMS

Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.

Program:

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

 // make two process which run same

 // program after this instruction

    fork();

    printf("Hello world!\n");

    return 0;

}
```

2. Identify the system calls to copy the content of one file to another and illustrate the same using a C program.

Program:

```
#include <stdio.h>
#include <stdlib.h> // For exit()
int main()
{
        FILE *fptr1, *fptr2;
        char filename[100], c;
        printf("Enter the filename to open for reading \n");
scanf("%s", filename);
// Open one file for reading
```

```
fptr1 = fopen(filename, "r");
if (fptr1 == NULL)
{
printf("Cannot open file %s \n", filename);
exit(0);
}
printf("Enter the filename to open for writing \n");
scanf("%s", filename);
// Open another file for writing
fptr2 = fopen(filename, "w");
if (fptr2 == NULL)
{
printf("Cannot open file %s \n", filename);
exit(0);
}
// Read contents from file
c = fgetc(fptr1);
while (c != EOF)
{
fputc(c, fptr2);
c = fgetc(fptr1);
}
printf("\nContents copied to %s", filename);
fclose(fptr1);
fclose(fptr2);
return 0;
}
```

3. Design a CPU scheduling program with C using First Come First Served technique with the following considerations.

a. All processes are activated at time 0.

b. Assume that no process waits on I/O devices.

Program:// C program for implementation of FCFS

```
#include<stdio.h>
void findWaitingTime(int processes[], int n,int bt[], int wt[])
{
// waiting time for first process is 0
```

```
wt[0] = 0;
for (int i = 1; i < n ; i++ )
        wt[i] = bt[i-1] + wt[i-1] ;
}
// Function to calculate turn around time
void findTurnAroundTime( int processes[], int n,
                        int bt[], int wt[], int tat[])
{
// calculating turnaround time by adding
// bt[i] + wt[i]
for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}
void findavgTime( int processes[], int n, int bt[])
{
int wt[n], tat[n], total_wt = 0, total_tat = 0;
findWaitingTime(processes, n, bt, wt);
findTurnAroundTime(processes, n, bt, wt, tat);
printf("Processes Burst time Waiting time Turn around time\n");
for (int i=0; i<n; i++)
{
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf(" %d ",(i+1));
        printf("   %d ", bt[i] );
        printf("   %d",wt[i] );
        printf("   %d\n",tat[i] );
}
int s=(float)total_wt / (float)n;
int t=(float)total_tat / (float)n;
printf("Average waiting time = %d",s);
printf("\n");
printf("Average turn around time = %d ",t);
}
int main()
{
int processes[] = { 1, 2, 3};
```

```
        int n = sizeof processes / sizeof processes[0];
        //Burst time of all processes
        int burst_time[] = {10, 5, 8};
        findavgTime(processes, n, burst_time);
        return 0;
        }
```

4. Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.

Program:

```
#include<stdio.h>
 int main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Enter number of process:");
   scanf("%d",&n);

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

  //sorting of burst times
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(bt[j]<bt[pos])
            pos=j;
      }

      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;

      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
   }

   wt[0]=0;


   for(i=1;i<n;i++)
   {
```

```
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=(float)total/n;
    total=0;

    printf("nProcesst    Burst Time    tWaiting TimetTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("np%dtt  %dtt    %dttt%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=(float)total/n;
    printf("nnAverage Waiting Time=%f",avg_wt);
    printf("nAverage Turnaround Time=%fn",avg_tat);
}
```

5. Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.

Program

```
#include<stdio.h>
 // structure representing a structure
struct priority_scheduling {
 // name of the process
  char process_name;
 // time required for execution
  int burst_time;
  // waiting time of a process
  int waiting_time;
  // total time of execution
  int turn_around_time;
  // priority of the process
  int priority;
};
int main() {
  int number_of_process;
  int total = 0;
```

```
struct priority_scheduling temp_process;
int ASCII_number = 65;
int position;
// average waiting time of the process
float average_waiting_time;
// average turnaround time of the process
float average_turnaround_time;
printf("Enter the total number of Processes: ");
// get the total number of the process as input
scanf("%d", & number_of_process);
// initializing the structure array
struct priority_scheduling process[number_of_process];
printf("\nPlease Enter the  Burst Time and Priority of each process:\n");
// get burst time and priority of all process
for (int i = 0; i < number_of_process; i++) {
  // assign names consecutively using ASCII number
  process[i].process_name = (char) ASCII_number;
  printf("\nEnter the details of the process %c \n", process[i].process_name);
  printf("Enter the burst time: ");
  scanf("%d", & process[i].burst_time);
  printf("Enter the priority: ");
  scanf("%d", & process[i].priority);
  // increment the ASCII number to get the next alphabet
  ASCII_number++;
}
// swap process according to high priority
for (int i = 0; i < number_of_process; i++) {
  position = I;
  for (int j = i + 1; j < number_of_process; j++) {
    // check if priority is higher for swapping
    if (process[j].priority > process[position].priority)
      position = j;
  }
  // swapping of lower priority process with the higher priority process
  temp_process = process[i];
  process[i] = process[position];
  process[position] = temp_process;}
```

```
  process[0].waiting_time = 0;
  for (int i = 1; i < number_of_process; i++) {
    process[i].waiting_time = 0;
    for (int j = 0; j < i; j++) {
          process[i].waiting_time += process[j].burst_time;
    }
    // calculate total waiting time
    total += process[i].waiting_time;
  average_waiting_time = (float) total / (float) number_of_process;
  total = 0;
  printf("\n\nProcess_name \t Burst Time \t Waiting Time \t  Turnaround Time\n");
  printf("-------------------------------------------------------------\n");
  for (int i = 0; i < number_of_process; i++) {
    process[i].turn_around_time = process[i].burst_time + process[i].waiting_time;
    total += process[i].turn_around_time;
       printf("\t    %c  \t\t    %d  \t\t %d  \t\t %d", process[i].process_name, process[i].burst_time,
          process[i].waiting_time, process[i].turn_around_time);
    printf("\n-------------------------------------------------------------\n");
  }
  average_turnaround_time = (float) total / (float) number_of_process;
  printf("\n\n Average Waiting Time : %f", average_waiting_time);
  printf("\n Average Turnaround Time: %f\n", average_turnaround_time);
  return 0;
}
```