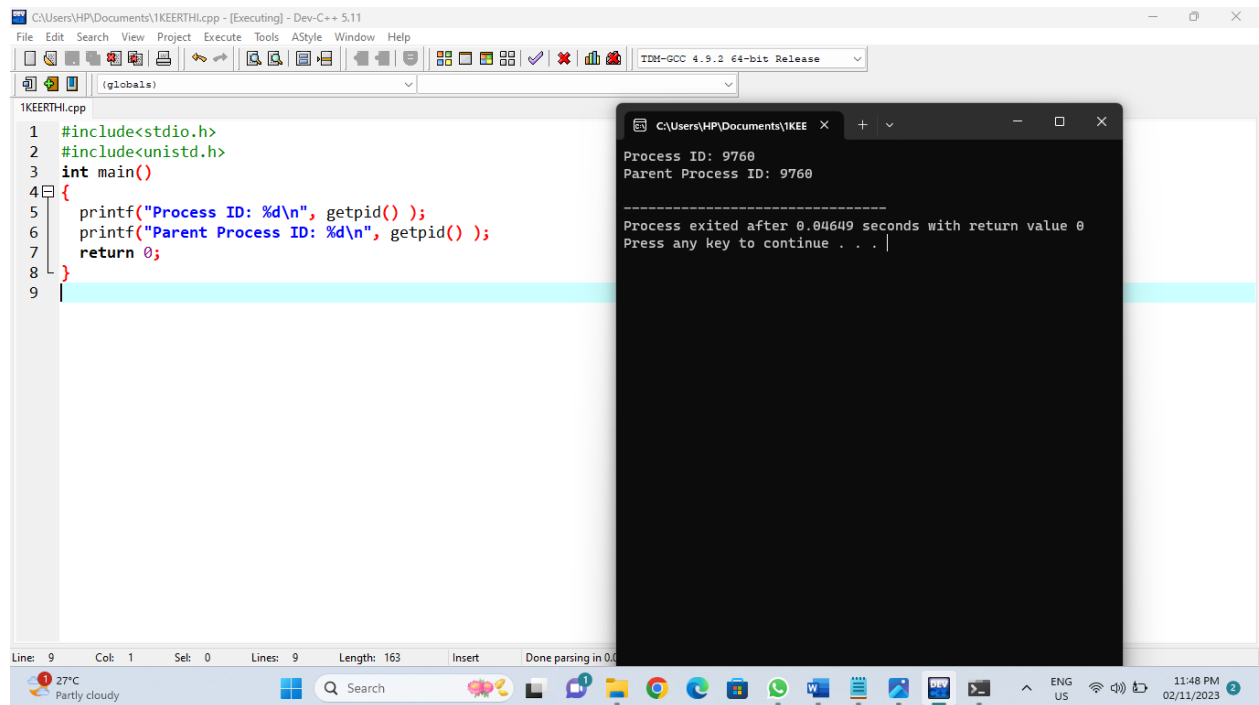


Practical programs

1. 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



The screenshot shows a Dev-C++ IDE window titled "C:\Users\HP\Documents\1KEERTHI.cpp - [Executing] - Dev-C++ 5.11". The code in the editor is as follows:

```
1 #include<stdio.h>
2 #include<unistd.h>
3 int main()
4 {
5     printf("Process ID: %d\n", getpid() );
6     printf("Parent Process ID: %d\n", getppid() );
7     return 0;
8 }
9
```

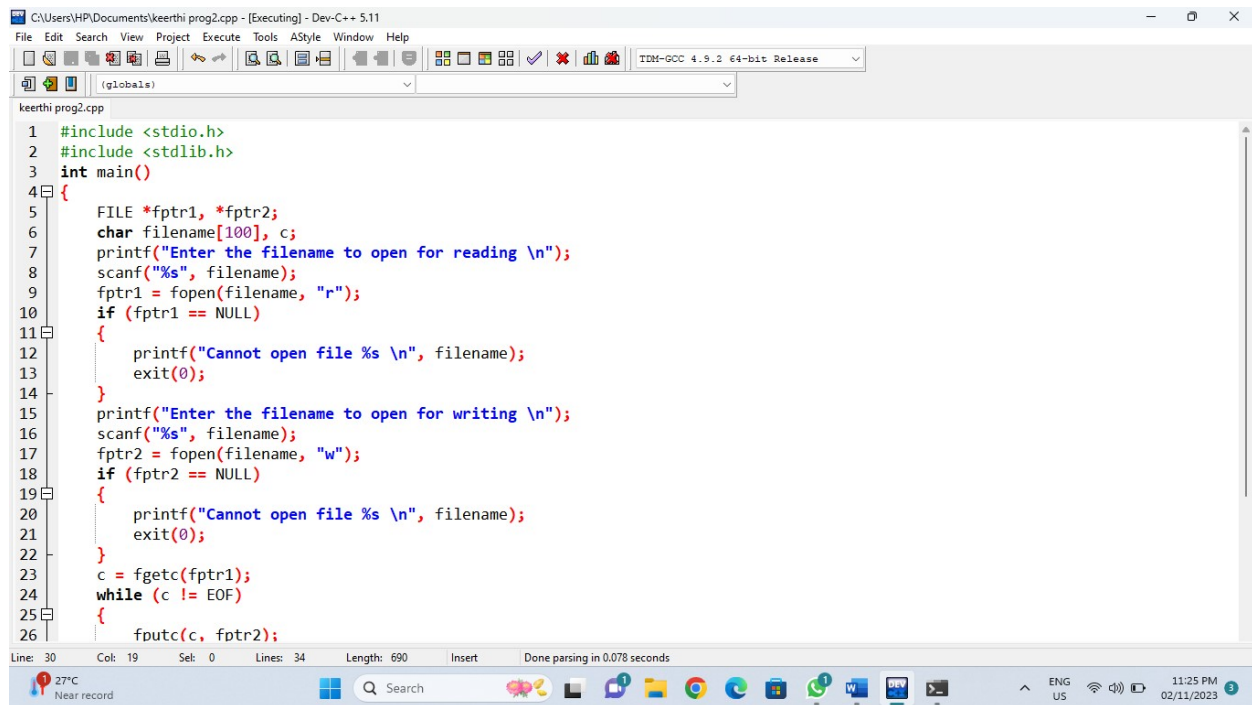
The program is running, and a terminal window is open showing the output:

```
Process ID: 9760
Parent Process ID: 9760

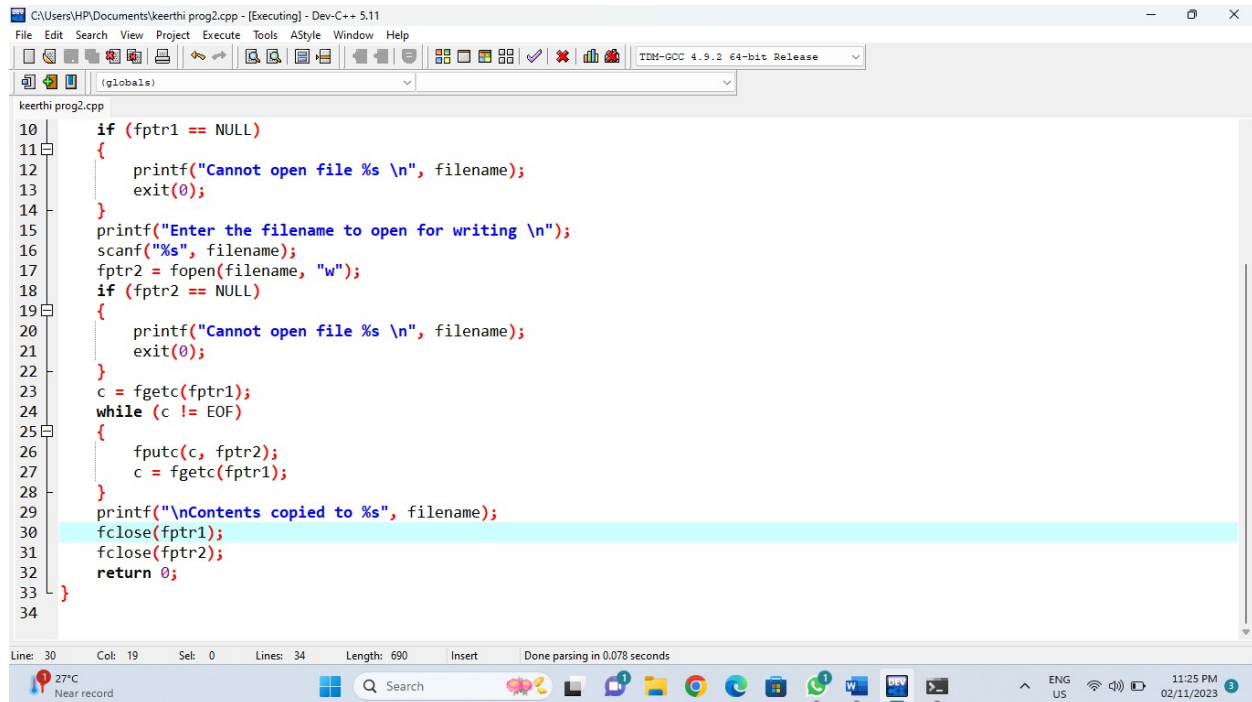
-----
Process exited after 0.04649 seconds with return value 0
Press any key to continue . . .
```

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

Practical programs

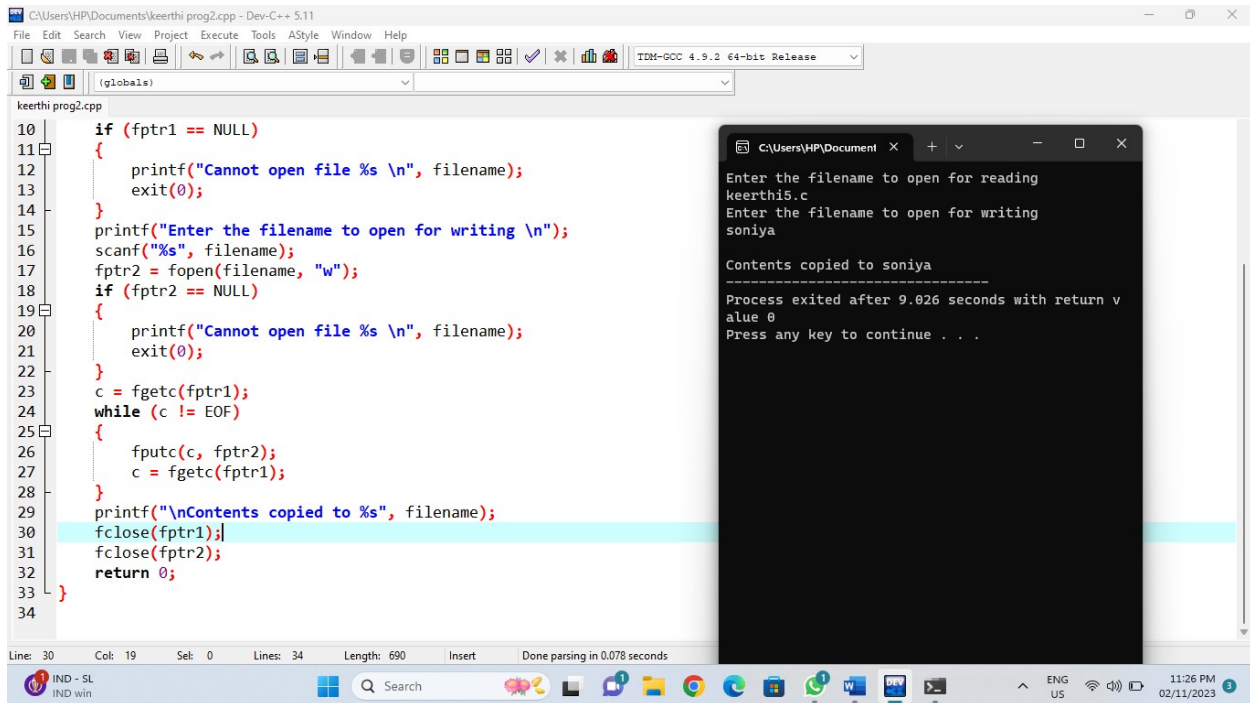


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     FILE *fptr1, *fptr2;
6     char filename[100], c;
7     printf("Enter the filename to open for reading \n");
8     scanf("%s", filename);
9     fptr1 = fopen(filename, "r");
10    if (fptr1 == NULL)
11    {
12        printf("Cannot open file %s \n", filename);
13        exit(0);
14    }
15    printf("Enter the filename to open for writing \n");
16    scanf("%s", filename);
17    fptr2 = fopen(filename, "w");
18    if (fptr2 == NULL)
19    {
20        printf("Cannot open file %s \n", filename);
21        exit(0);
22    }
23    c = fgetc(fptr1);
24    while (c != EOF)
25    {
26        fputc(c, fptr2);
```



```
26        fputc(c, fptr2);
27        c = fgetc(fptr1);
28    }
29    printf("\nContents copied to %s", filename);
30    fclose(fptr1);
31    fclose(fptr2);
32    return 0;
33 }
34
```

Practical programs



The screenshot shows a C++ IDE with a file named `keerthi prog2.cpp`. The code implements a program that reads from a file and writes its contents to another file. The execution output shows the program running successfully, copying the contents of `keerthi5.c` to `soniya`.

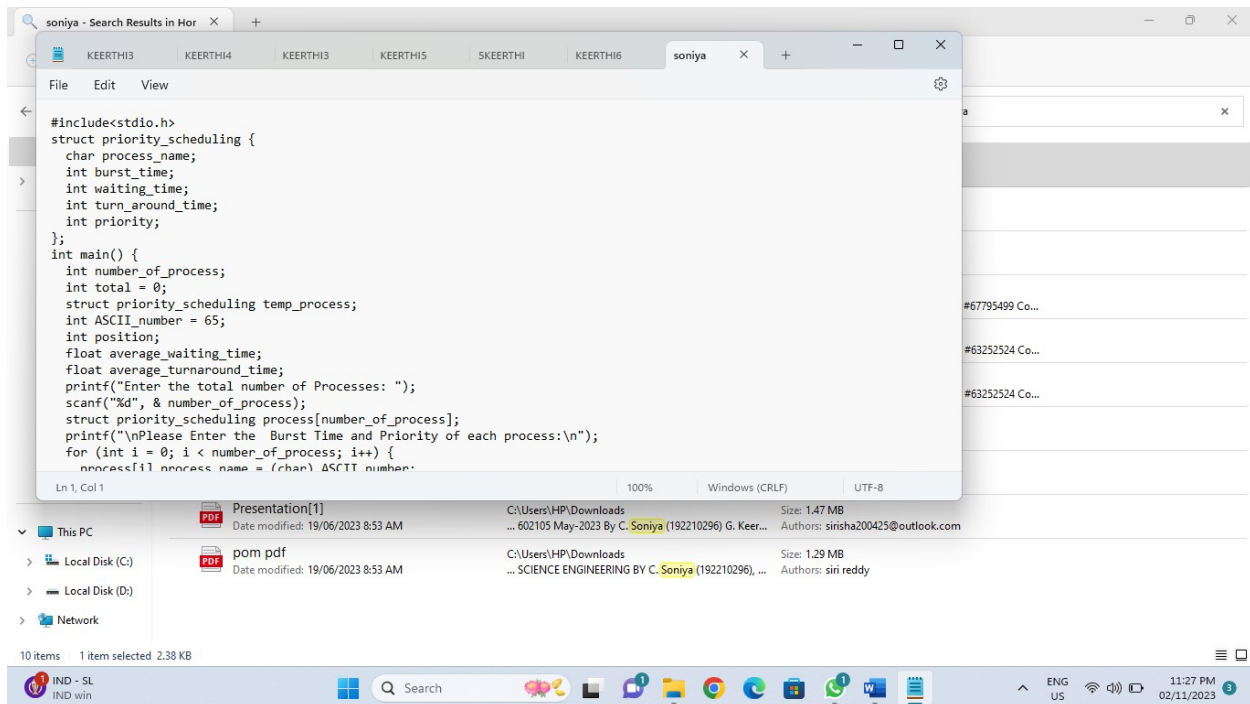
```
10 if (fptr1 == NULL)
11 {
12     printf("Cannot open file %s \n", filename);
13     exit(0);
14 }
15 printf("Enter the filename to open for writing \n");
16 scanf("%s", filename);
17 fptr2 = fopen(filename, "w");
18 if (fptr2 == NULL)
19 {
20     printf("Cannot open file %s \n", filename);
21     exit(0);
22 }
23 c = fgetc(fptr1);
24 while (c != EOF)
25 {
26     fputc(c, fptr2);
27     c = fgetc(fptr1);
28 }
29 printf("\nContents copied to %s", filename);
30 fclose(fptr1);
31 fclose(fptr2);
32 return 0;
33 }
34
```

Execution Output:

```
Enter the filename to open for reading
keerthi5.c
Enter the filename to open for writing
soniya

Contents copied to soniya

Process exited after 9.826 seconds with return value 0
Press any key to continue . . .
```



The screenshot shows a C++ IDE with a file named `soniya`. The code implements a priority scheduling algorithm. The execution output shows the program running successfully, displaying the total number of processes and the burst time and priority of each process.

```
#include<stdio.h>
struct priority_scheduling {
    char process_name;
    int burst_time;
    int waiting_time;
    int turn_around_time;
    int priority;
};
int main() {
    int number_of_process;
    int total = 0;
    struct priority_scheduling temp_process;
    int ASCII_number = 65;
    int position;
    float average_waiting_time;
    float average_turnaround_time;
    printf("Enter the total number of Processes: ");
    scanf("%d", &number_of_process);
    struct priority_scheduling process[number_of_process];
    printf("\nPlease Enter the Burst Time and Priority of each process:\n");
    for (int i = 0; i < number_of_process; i++) {
        process[i].process_name = (char) ASCII_number;
```

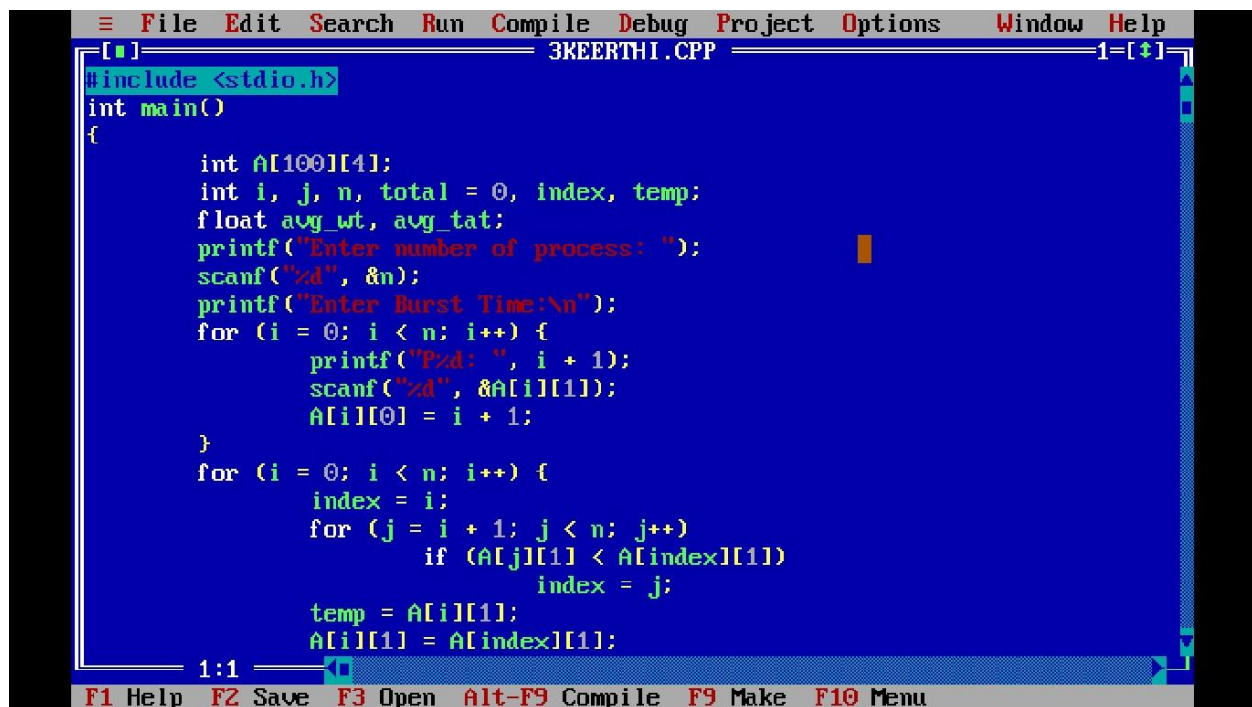
Execution Output:

```
Enter the total number of Processes: 3
Please Enter the Burst Time and Priority of each process:
1 10 1
2 5 2
3 8 3
```

Practical programs

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

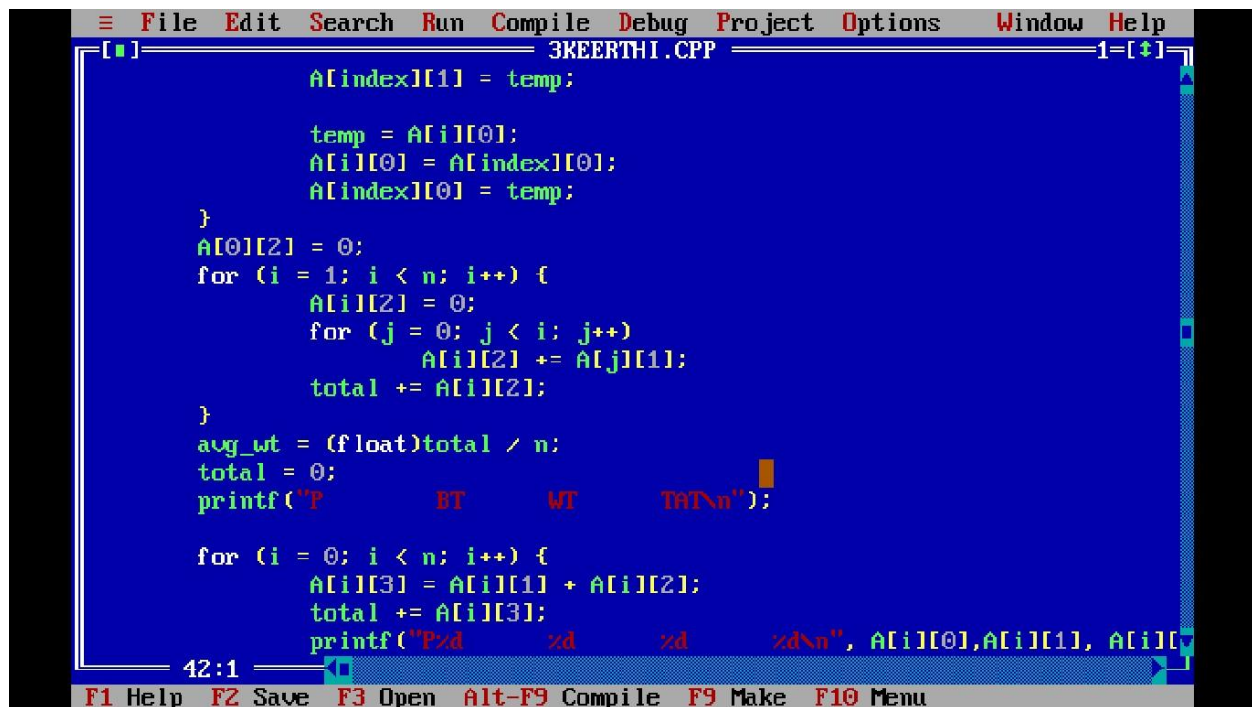
- All processes are activated at time 0.
- Assume that no process waits on I/O devices



The image shows a screenshot of a C program titled "3KEERTHI.CPP" running in a text-based IDE. The program implements a First Come First Served (FCFS) CPU scheduling algorithm. It starts by including the standard input/output library (`<stdio.h>`). The `main` function begins by declaring an array `A` of size `100x11` to store process details, and variables for `i`, `j`, `n`, `total`, `index`, and `temp`. It prompts the user to enter the number of processes (`n`) and then enters a loop to collect the process ID (`P`) and burst time (`B`) for each process. The burst times are stored in the first column of the array `A`. Another loop then finds the minimum burst time among the remaining processes to determine the next process to be scheduled, storing its index and burst time in `index` and `temp` respectively. The array `A` is then updated to reflect the scheduling of the process.

```
#include <stdio.h>
int main()
{
    int A[100][11];
    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");
    for (i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d", &A[i][1]);
        A[i][0] = i + 1;
    }
    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];
    }
}
```

Practical programs

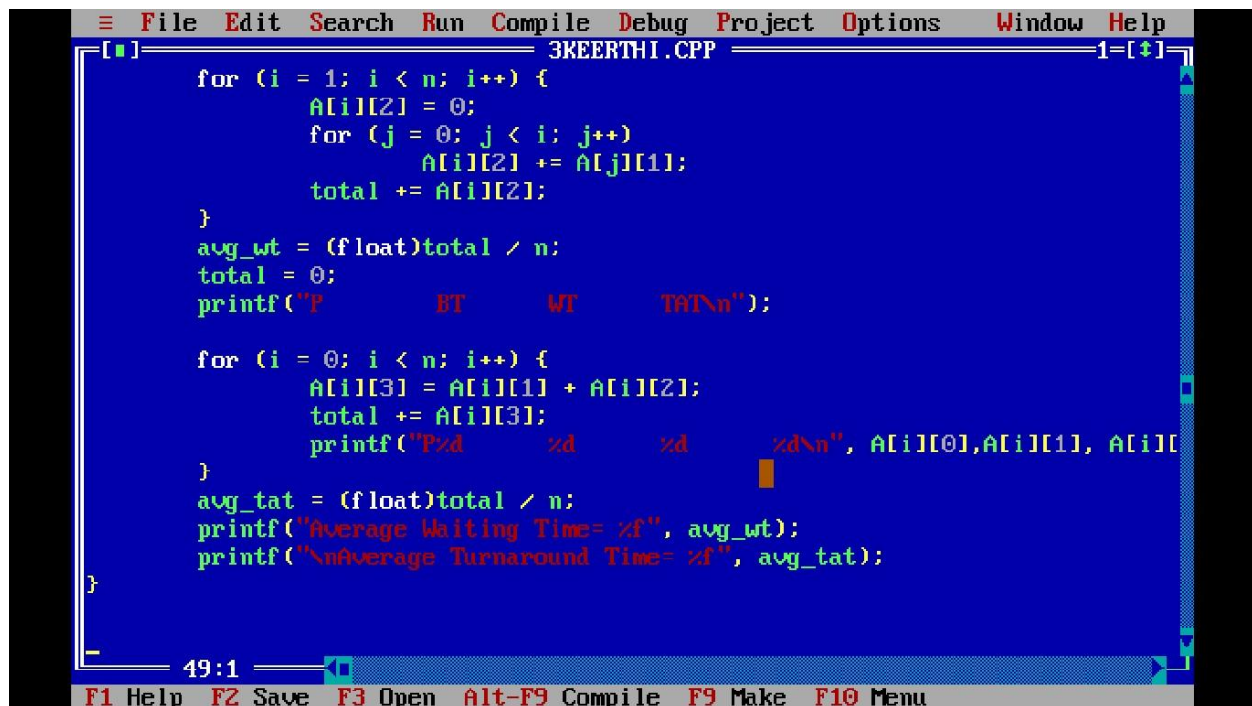


```
File Edit Search Run Compile Debug Project Options Window Help
3KEERTHI.CPP 1=1+1
A[index][1] = temp;

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]);
}
42:1
```

F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu



```
File Edit Search Run Compile Debug Project Options Window Help
3KEERTHI.CPP 1=1+1
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");

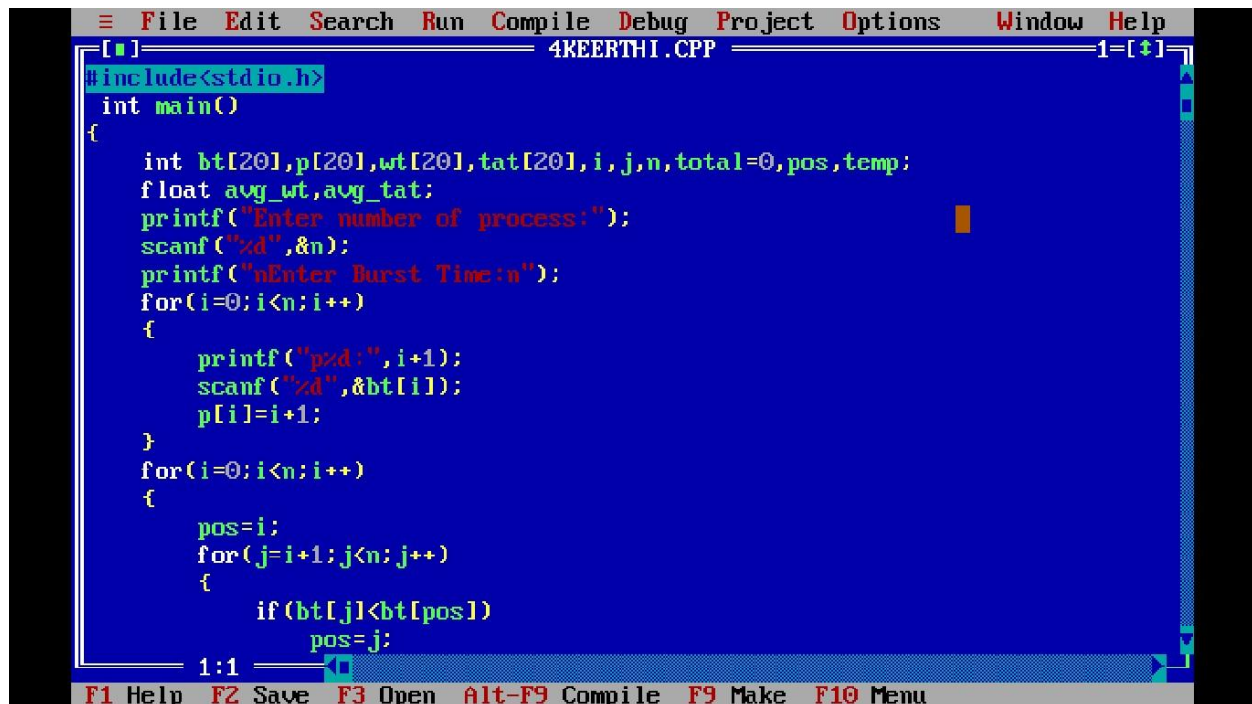
for (i = 0; i < n; i++) {
    A[i][3] = A[i][1] + A[i][2];
    total += A[i][3];
    printf("P%d      %d      %d      %d\n", A[i][0], A[i][1], A[i][2], A[i][3]);
}
avg_tat = (float)total / n;
printf("Average Waiting Time= %f", avg_wt);
printf("\nAverage Turnaround Time= %f", avg_tat);
}
49:1
```

F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

Practical programs

```
C:\TURBOC3\BIN>TC
Enter number of process: 3
Enter Burst Time:
P1: 1
P2: 2
P3: 3
P      BT      WT      TAT
P1     1       0       1
P2     2       1       3
P3     3       3       6
Average Waiting Time= 1.333333
Average Turnaround Time= 3.333333Enter number of process:
```

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



```
#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;
    }
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }
        total+=bt[pos];
        wt[pos]=total-bt[pos];
        tat[i]=wt[i]+bt[i];
    }
    avg_wt=(float)total/n;
    avg_tat=(float)total/n;
    printf("\n\n");
    printf("P\t\tBT\t\tWT\t\tTAT\n");
    for(i=0;i<n;i++)
    {
        printf("P%d\t\t%d\t\t%d\t\t%d\n",p[i],bt[i],wt[i],tat[i]);
    }
    printf("\nAverage Waiting Time= %.2f",avg_wt);
    printf("\nAverage Turnaround Time= %.2f",avg_tat);
}
```

Practical programs

```

File Edit Search Run Compile Debug Project Options Window Help
4KEERTHI.CPP 1=[+]
}
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("nProcess\t\t Burst Time\t\t tWaiting Time\tTurnaround Time");
42:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```

```
File Edit Search Run Compile Debug Project Options Window Help
[.] 4KEERTHI.CPP 1-[+]
wt[i]=0;
for(j=0;j<i;j++)
    wt[i]+=bt[j];

total+=wt[i];
}
avg_wt=(float)total/n;
total=0;
printf("\nProcess\t Burst Time\t Waiting Time\t Turnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    total+=tat[i];
    printf("p%d\t %d\t\t %d\t\t %d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;
printf("\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f",avg_tat);
}
```

F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

Practical programs

```

≡ File Edit Search Run Compile Debug Project Options Window Help
Help 2
[.] 4KEERTHI.CPP 3-[↑]
    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 Time\tTurnaround Time");
for(i=0; i<n; i++)
{
    tat[i]=bt[i]+wt[i];
    total+=tat[i];
    printf("opzdt\t zdt\t zdt\tz\t",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;
printf("\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
54:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```

```
C:\TURBOC3\BIN>TC
Enter number of process:3
Enter Burst Time:np1:6
p2:7
p3:8
nProcesst    Burst Time    tWaiting TimeTurnaround Timenp1tt    6tt    0ttt6np2tt
7tt    6ttt13np3tt    8tt    13ttt21nnAverage Waiting Time=6.333333nAverage Turn
around Time=13.333333nEnter number of process:
```

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

Practical programs

The screenshot shows a C program in a Dev-C++ IDE. The program implements a priority scheduling algorithm. It prompts the user to enter the number of processes, then for each process, it asks for the process name, burst time, and priority. It then calculates the waiting and turnaround times for each process and displays a table of results.

```
#include<stdio.h>
1 struct priority_scheduling {
2     char process_name;
3     int burst_time;
4     int waiting_time;
5     int turn_around_time;
6     int priority;
7 }
8
9 int main() {
10     int number_of_processes;
11     int total = 0;
12     struct priority_scheduling temp_process;
13     int ASCII_number = 0;
14     int position;
15     float average_waiting_time;
16     float average_turnaround_time;
17     printf("Enter the total number of processes: ");
18     scanf("%d", &number_of_processes);
19     struct priority_scheduling process[number_of_processes];
20     printf("Please Enter the Burst Time and Priority of each process:\n");
21     for (int i = 0; i < number_of_processes; i++) {
22         process[i].process_name = (char) ASCII_number;
23         printf("Enter the details of the process %c:\n", process[i].process_name);
24         printf("Enter the burst time: ");
25         scanf("%d", &process[i].burst_time);
26         printf("Enter the priority: ");
27         scanf("%d", &process[i].priority);
28         ASCII_number++;
29     }
30     for (int i = 0; i < number_of_processes; i++) {
31         position = i;
32         for (int j = i + 1; j < number_of_processes; j++) {
33             if (process[j].priority < process[position].priority)
34                 position = j;
35         }
36         temp_process = process[i];
37         process[i] = process[position];
38         process[position] = temp_process;
39     }
40     process[0].waiting_time = 0;
41     for (int i = 1; i < number_of_processes; i++) {
42         process[i].waiting_time = 0;
43         for (int j = 0; j < i; j++) {
44             process[i].waiting_time += process[j].burst_time;
45         }
46     }
47     total = 0;
48     for (int i = 0; i < number_of_processes; i++) {
49         process[i].turn_around_time = process[i].waiting_time + process[i].burst_time;
50         total += process[i].turn_around_time;
51     }
52     average_waiting_time = total / number_of_processes;
53     average_turnaround_time = total / number_of_processes;
54     printf("Average Waiting Time : %.5f\n", average_waiting_time);
55     printf("Average Turnaround Time : %.5f\n", average_turnaround_time);
56     printf("Process exited after 29.82 seconds with return value 0\n");
57     printf("Press any key to continue . . . ");
58 }
```

Process_name	Burst Time	Waiting Time	Turnaround Time
A	4	0	4
B	2	4	6
C	3	6	9
D	5	9	14

Average Waiting Time : 4.750000
Average Turnaround Time : 8.750000
Process exited after 29.82 seconds with return value 0
Press any key to continue . . .

6. Construct a C program to implement pre-emptive priority scheduling algorithm.

The screenshot shows a C program in a Dev-C++ IDE. The program implements a pre-emptive priority scheduling algorithm. It prompts the user to enter the number of processes, then for each process, it asks for the arrival time, burst time, and priority. It then calculates the waiting and turnaround times for each process and displays a table of results.

```
#include<stdio.h>
2 struct process
3 {
4     int WT,AT,BT,TAT,PT;
5 };
6
7 struct process a[10];
8
9 int main()
10 {
11     int n,temp[10],t,count=0,short_p;
12     float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;
13     printf("Enter the number of the process\n");
14     scanf("%d",&n);
15     printf("Enter the arrival time , burst time and priority of the process\n");
16     printf("AT BT PT\n");
17     for(int i=0;i<n;i++)
18     {
19         scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);
20         temp[i]=a[i].BT;
21     }
22     a[9].PT=10000;
23     for(t=0;count!=n;t++)
24     {
25         // Find the process with the shortest burst time
26         int min_BT = 1000000000;
27         int min_PT = 1000000000;
28         int min_index = -1;
29         for(int i=0;i<n;i++)
30         {
31             if(a[i].BT < min_BT || (a[i].BT == min_BT & a[i].PT < min_PT))
32             {
33                 min_BT = a[i].BT;
34                 min_PT = a[i].PT;
35                 min_index = i;
36             }
37         }
38         // Calculate waiting time
39         for(int j=0;j<n;j++)
40         {
41             if(a[j].PT < a[min_index].PT || (a[j].PT == a[min_index].PT & a[j].BT < a[min_index].BT))
42                 continue;
43             a[min_index].waiting_time += a[j].BT;
44         }
45         // Calculate turnaround time
46         a[min_index].turn_around_time = a[min_index].waiting_time + a[min_index].BT;
47         // Update total waiting and turnaround times
48         total_WT += a[min_index].waiting_time;
49         total_TAT += a[min_index].turn_around_time;
50         // Increment count and reset waiting time
51         count++;
52         a[min_index].waiting_time = 0;
53     }
54     Avg_WT = total_WT / n;
55     Avg_TAT = total_TAT / n;
56     printf("Avg waiting time of the process is %.6f\n", Avg_WT);
57     printf("Avg turn around time of the process is %.6f\n", Avg_TAT);
58     printf("Process exited after 40.7 seconds with return value 0\n");
59     printf("Press any key to continue . . . ");
60 }
```

ID	WT	TAT
1	4	10
2	10	17
3	0	4

Avg waiting time of the process is 4.666667
Avg turn around time of the process is 10.333333
Process exited after 40.7 seconds with return value 0
Press any key to continue . . .

Practical programs

7. Construct a C program to implement non-preemptive SJF algorithm.

```

≡ File Edit Search Run Compile Debug Project Options Window Help
[.] 7KEERTHI.CPP 1-[+]
#include<stdio.h>

int main() {
    int time, burst_time[10], at[10], sum_burst_time = 0, smallest, n, i;
    int sumt = 0, sumw = 0;
    printf("enter the no of processes : ");
    scanf("%d", & n);
    for (i = 0; i < n; i++) {
        printf("the arrival time for process P%d : ", i + 1);
        scanf("%d", & at[i]);
        printf("the burst time for process P%d : ", i + 1);
        scanf("%d", & burst_time[i]);
        sum_burst_time += burst_time[i];
    }
    burst_time[9] = 9999;
    for (time = 0; time < sum_burst_time;) {
        smallest = 9;
        for (i = 0; i < n; i++) {
            if (at[i] <= time && burst_time[i] > 0 && burst_time[i] < burst_time[smallest])
                smallest = i;
        }
    }
}
1:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```

```

≡ File Edit Search Run Compile Debug Project Options Window Help
[.] 7KEERTHI.CPP 1=[.]
scanf("%d", & burst_time[i]);
sum_burst_time += burst_time[i];
}
burst_time[9] = 9999;
for (time = 0; time < sum_burst_time;) {
    smallest = 9;
    for (i = 0; i < n; i++) {
        if (at[i] <= time && burst_time[i] > 0 && burst_time[i] < burst_time[smallest])
            smallest = i;
    }
    printf("P%d\tI\tS\tD\tI\tS\tD\tI\tS\tD\tS\n", smallest + 1, time + burst_time[smallest]);
    sumt += time + burst_time[smallest] - at[smallest];
    sumw += time - at[smallest];
    time += burst_time[smallest];
    burst_time[smallest] = 0;
}
printf("\n\n average waiting time = %f", sumw * 1.0 / n);
printf("\n\n average turnaround time = %f", sumt * 1.0 / n);
return 0;
}
32:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```

Practical programs

```
C:\TURBOC3\BIN>TC
enter the no of processes : 2
the arrival time for process P1 : 1
the burst time for process P1 : 2
the arrival time for process P2 : 4
the burst time for process P2 : 5
P[10] : 9999 : 0

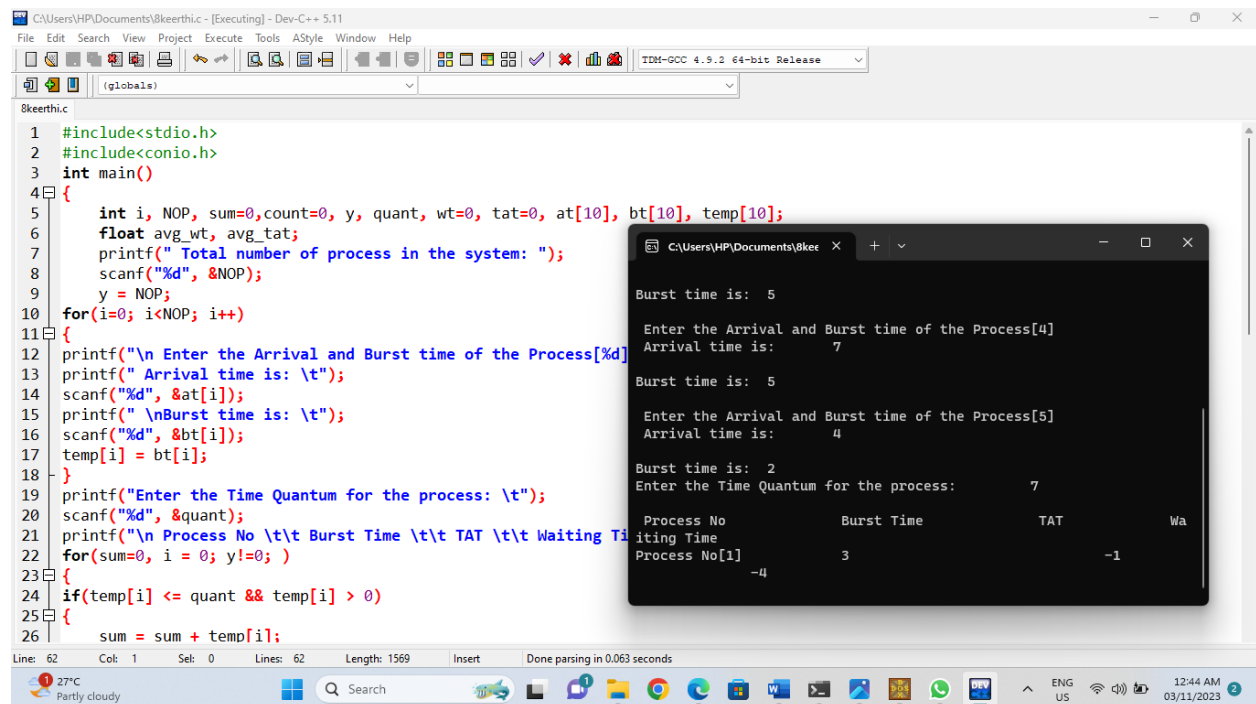
average waiting time = 0.000000

average turnaround time = 4999.500000
enter the no of processes : 1
the arrival time for process P1 : 1
the burst time for process P1 : 2
P[10] : 8290 : -1709

average waiting time = -1709.000000

average turnaround time = 8290.000000
enter the no of processes : _
```

8. Construct a C program to simulate Round Robin scheduling algorithm with C.



The screenshot shows a C program in Dev-C++ titled "8keerthi.c". The program simulates Round Robin scheduling. It prompts the user for the number of processes (NOP), then for each process, it asks for arrival time (at) and burst time (bt). It also asks for a time quantum. The program then calculates and displays the waiting time (Wa) and turnaround time (TAT) for each process. A table is shown with columns for Process No, Burst Time, TAT, and Wa. The output shows Process No[1] with a burst time of 3, TAT of -1, and Wa of -4.

```
1 #include<stdio.h>
2 #include<conio.h>
3 int main()
4 {
5     int i, NOP, sum=0, count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
6     float avg_wt, avg_tat;
7     printf(" Total number of process in the system: ");
8     scanf("%d", &NOP);
9     y = NOP;
10    for(i=0; i<NOP; i++)
11    {
12        printf("\n Enter the Arrival and Burst time of the Process[%d]", i);
13        printf(" Arrival time is: \t");
14        scanf("%d", &at[i]);
15        printf(" \nBurst time is: \t");
16        scanf("%d", &bt[i]);
17        temp[i] = bt[i];
18    }
19    printf("Enter the Time Quantum for the process: \t");
20    scanf("%d", &quant);
21    printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time");
22    for(sum=0, i = 0; y!=0; )
23    {
24        if(temp[i] <= quant && temp[i] > 0)
25        {
26            sum = sum + temp[i];
```

Practical programs