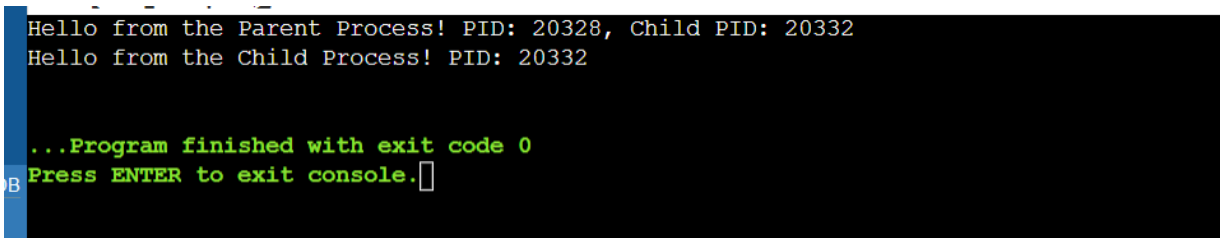
Week No-1 Practical Questions

1.**Objective**: Write a program demonstrates the creation of a child process using 'fork()'.

SOURCE CODE:

```
#include <stdio.h>
#include <unistd.h>
int main() {
 // ……..Create a child process……..
 pid_t pid = fork();
// ……Fork failed……..
 if (pid < 0) {
 printf("Fork failed!\n");
 return 1;
 } else if (pid == 0) {
 //……. This is the child process……
 printf("Hello from the Child Process! PID: %d\n", getpid());
 } else {
 // ……..This is the parent process……..
 printf("Hello from the Parent Process! PID: %d, Child PID: %d\n", getpid(), pid);
 }
 return 0;
}
```

OUTPUT:

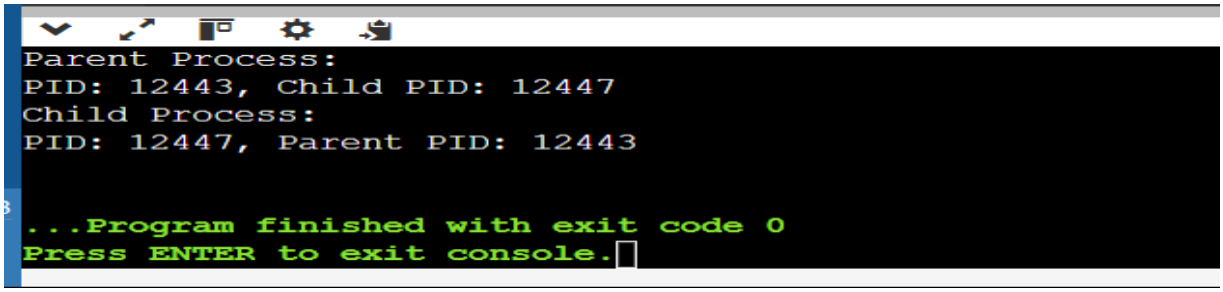


2.**Objective:** Write a program to print process Id's of parent and child process i.e. parent should print its own and its child process id while child process should print its own and its parent process id. (use getpid(), getppid())

SOURCE CODE:

```
#include <stdio.h>
#include <unistd.h>
int main() {
 // Create a child process
 pid_t pid = fork();
 if (pid < 0) {
 // Fork failed
 printf("Fork failed!\n");
 return 1;
 } else if (pid == 0) {
 // This is the child process
 printf("Child Process:\n");
 printf("PID: %d, Parent PID: %d\n", getpid(), getppid());
 } else {
 // This is the parent process
 printf("Parent Process:\n");
 printf("PID: %d, Child PID: %d\n", getpid(), pid);
 }
 return 0;}
```

OUTPUT:



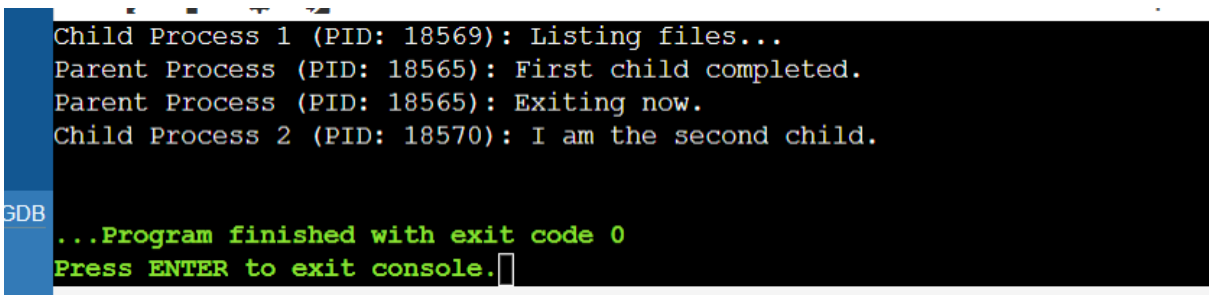
3. **Objective:** Write a program to create child process which will list all the files present in your system. Make sure that parent process waits until child has not completed its execution. (use wait(), exit()) What will happen if parent process dies before child process? Illustrate it by creating one more child of parent process.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main() {
 pid_t pid1, pid2;
 // Create the first child process
 pid1 = fork();
 if (pid1 < 0) {
 // Fork failed
 printf("Fork failed!\n");
 return 1;
 }
 else if (pid1 == 0) {
 // This is the first child process
 printf("Child Process 1 (PID: %d): Listing files...\n", getpid());
 exit(0);
}
else {
 // Parent process waits for the first child to complete
 wait(NULL);
 printf("Parent Process (PID: %d): First child completed.\n", getpid());
```

```
// Create the second child process
pid2 = fork();
if (pid2 < 0) {
// Fork failed
printf("Fork failed!\n");
return 1;
}
else if (pid2 == 0) {
// This is the second child process
printf("Child Process 2 (PID: %d): I am the second child.\n", getpid());
sleep(5); // Simulate some work
printf("Child Process 2 (PID: %d): Work done.\n", getpid());
exit(0);
}
else {
// Parent process dies before the second child finishes
printf("Parent Process (PID: %d): Exiting now.\n", getpid());
exit(0);
}
}
}
```

OUTPUT:



Week No-2 Practical Questions

1.**Objective**: Write a program to open a directory and list its contents using opendir(), readdir(), and closedir().

SOURCE CODE:

```
#include <stdio.h>
#include <dirent.h>
int main() {
    DIR *d;
    struct dirent *dir;
    d = opendir(".");
    if (d) {
        printf("Contents of directory:\n");
        while ((dir = readdir(d)) != NULL) {
            printf("%s\n", dir->d_name);
        }
        closedir(d);
    } else {
        perror("opendir");
    }
return 0;
}
```

OUTPUT:

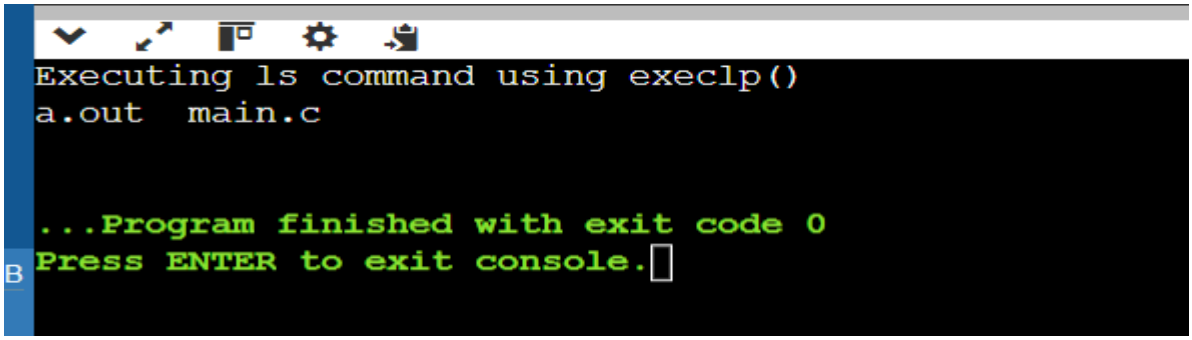
```
.
..
main.c
a.out

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

2.**Objective**: Write a program to show the working of the execlp() system call by executing the ls command.

SOURCE CODE:

```
#include <stdio.h>

#include <unistd.h>

int main() {

printf("Executing ls command using execlp()\n"); execlp("ls", "ls", NULL);

printf("This line will not be executed if execlp is successful\n");


return 0;

}
```

OUTPUT:



3.**Objective:** Write a program to read a file and store your details in that file. Your program should also create one more file and store your friend's details in that file. Once both files are created, print lines that are matching in both files.

SOURCE CODE:

```
#include <stdio.h>

#include <string.h>

#define MAX_LINE_LENGTH 256

void write_to_file(const char *filename, const char *content) {

 FILE *file = fopen(filename, "w");

if (file == NULL) {
```

```
perror("Unable to open file");
 return;
}
fprintf(file, "%s", content); fclose(file);
}
void find_matching_lines(const char *file1, const char *file2) {
 char line1[MAX_LINE_LENGTH], line2[MAX_LINE_LENGTH];
FILE *fp1 = fopen(file1, "r");
FILE *fp2 = fopen(file2, "r");

if (fp1 == NULL || fp2 == NULL) {
perror("Error opening files");
return;
}
while (fgets(line1, MAX_LINE_LENGTH, fp1) != NULL) {
 rewind(fp2);
while (fgets(line2, MAX_LINE_LENGTH, fp2) != NULL) {
if (strcmp(line1, line2) == 0) {
printf("Matching line: %s", line1);
}
}
}
fclose(fp1);
fclose(fp2);
}
int main() {
const char *my_details = "Name: John\nAge: 25\nCity: New York\n"; const char
*friend_details = "Name: Jane\nAge: 25\nCity: New York\n";
```

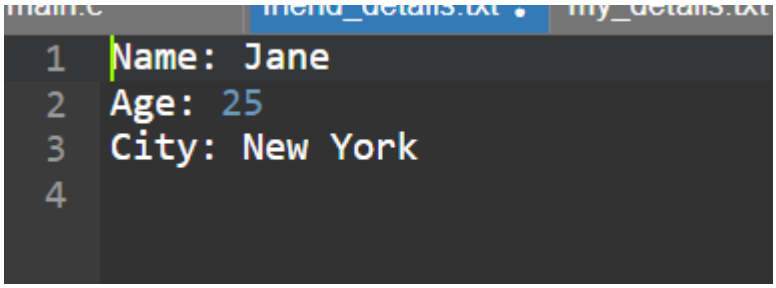
write_to_file("my_details.txt", my_details); write_to_file("friend_details.txt", friend_details);

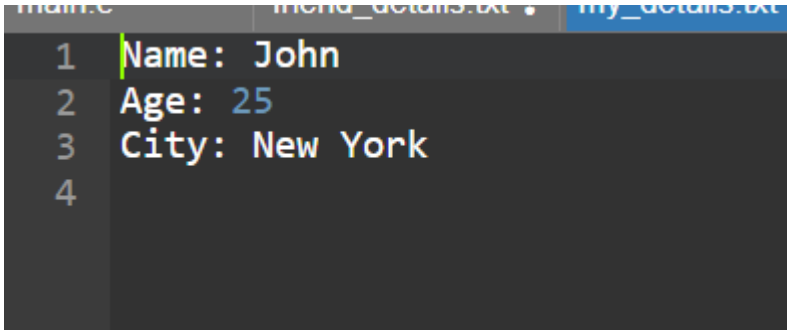
printf("Matching lines in both files:\n"); find_matching_lines("my_details.txt", "friend_details.txt");

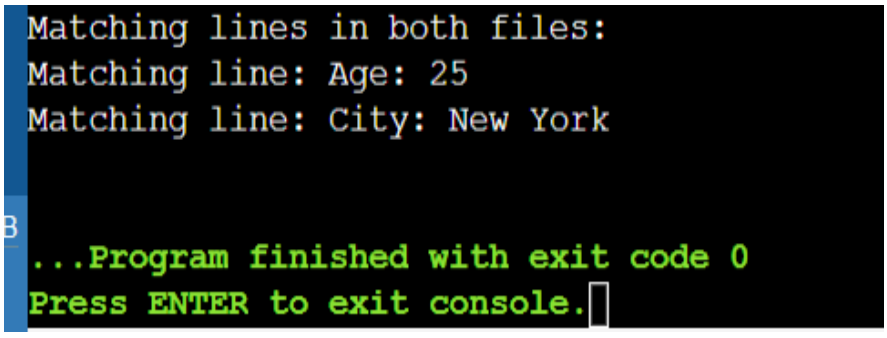
return 0;

}

OUTPUT:







Week No-3 Practical Questions

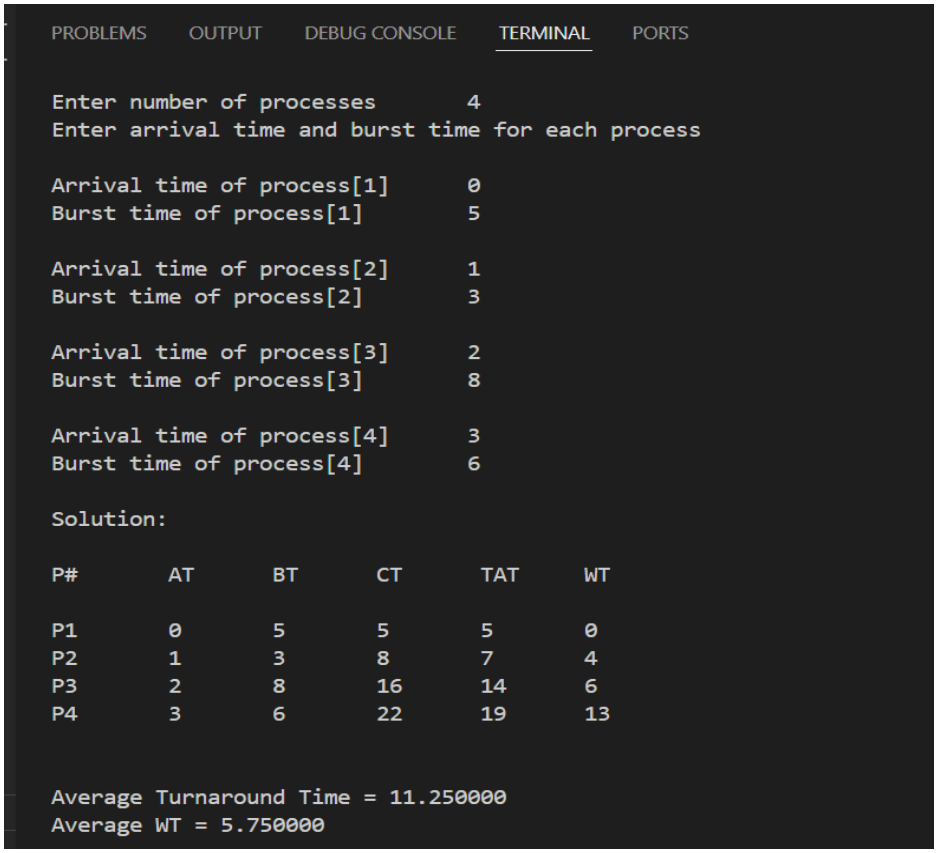
1.Objective : Write a C program to implement FCFS Scheduling Algorithm.

SOURCE CODE:

```
#include<stdio.h>
int main(){
   int bt[10]={0},at[10]={0},tat[10]={0},wt[10]={0},ct[10]={0};

   int n,sum=0;
   float totalTAT=0,totalWT=0;

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

   printf("Enter arrival time and burst time for each process\n\n");
      for(int i=0;i<n;i++){

      printf("Arrival time of process[%d] ",i+1);
      scanf("%d",&at[i]);

      printf("Burst time of process[%d]   ",i+1);
      scanf("%d",&bt[i]);

      printf("\n");
   }

   //calculate completion time of processes

   for(int j=0;j<n;j++){
      sum+=bt[j];
      ct[j]+=sum;
   }

   //calculate turnaround time and waiting times

   for(int k=0;k<n;k++){
      tat[k]=ct[k]-at[k];
      totalTAT+=tat[k];
   }

   for(int k=0;k<n;k++){
      wt[k]=tat[k]-bt[k];
```

```
        totalWT+=wt[k];
    }

    printf("Solution: \n\n");
    printf("P#\t AT\t BT\t CT\t TAT\t WT\t\n\n");

    for(int i=0;i<n;i++){
        printf("P%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("\n\nAverage Turnaround Time = %f\n",totalTAT/n);
    printf("Average WT = %f\n\n",totalWT/n);

    return 0;
}
```

OUTPUT



2.Objective : Write a C program to implement SJF Non-pre-emptive Scheduling Algorithm.

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int n, i, j, temp, total = 0, currentTime = 0, completed = 0, minIndex;
 int at[101], bt[101], process[101], wt[101], tat[101], ct[101], isCompleted[101] = {0};
    float avgWt = 0, avgTat = 0;

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

    printf("Enter the arrival times and burst times:\n");
    for (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]);
        process[i] = i + 1;  // Store process ID
    }

    // Initialize all W.T, TAT, and C.T to 0
    for (i = 0; i < n; i++) {
        wt[i] = 0;
        tat[i] = 0;
        ct[i] = 0;
```

```
}

// Start scheduling
while (completed != n) {
    // Find the process with the smallest B.T among the processes that have arrived
    minIndex = -1;
    for (i = 0; i < n; i++) {
        if (at[i] <= currentTime && !isCompleted[i]) {
            if (minIndex == -1 || bt[i] < bt[minIndex]) {
                minIndex = i;
            }
        }
    }

    // If a process is found, execute it
    if (minIndex != -1) {
        currentTime += bt[minIndex];
        ct[minIndex] = currentTime;  // Completion time
        tat[minIndex] = ct[minIndex] - at[minIndex];  // Turnaround time
        wt[minIndex] = tat[minIndex] - bt[minIndex];  // W.T
        isCompleted[minIndex] = 1;  // Mark process as completed
        completed++;
    } else {
        // If no process is available, move the time forward (idle time)
        currentTime++;
    }
}
```

```
// Cal. total W.t and TAT time
for (i = 0; i < n; i++) {
    total += wt[i];
}
avgWt = (float)total / n;

total = 0;
for (i = 0; i < n; i++) {
    total += tat[i];
}
avgTat = (float)total / n;

// Gantt Chart
printf("\nGantt Chart:\n");
printf("-----------------------------------------------\n|");
currentTime = 0;
for (i = 0; i < n; i++) {
    if (isCompleted[i]) {
        printf(" P%d |", process[i]);
    }
}
printf("\n-----------------------------------------------\n");

// Show C.T markers below the chart
currentTime = 0;
printf("0");
for (i = 0; i < n; i++) {
    if (isCompleted[i]) {
        currentTime = ct[i];
```

```
            printf("    %d", currentTime);
        }
    }


    // Print
    printf("\n\nProcess\t    Arrival Time\tBurst Time\tWaiting Time\tTurnaround Time\tCompletion Time");
    for (i = 0; i < n; i++) {
  printf("\nP%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d", process[i], at[i], bt[i], wt[i], tat[i],ct[i]);
    }
    printf("\n\nAverage Waiting Time: %.2f", avgWt);
    printf("\nAverage Turnaround Time: %.2f\n", avgTat);
    return 0;
}
```

OUTPUT :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\OEPRATING_C> cd "d:\OEPRATING_C\weeek1\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRu
nerFile }
Enter the number of processes: 4
Enter the arrival times and burst times:
Process[1] Arrival Time: 0
Process[1] Burst Time: 5
Process[2] Arrival Time: 1
Process[2] Burst Time: 3
Process[3] Arrival Time: 2
Process[3] Burst Time: 8
Process[4] Arrival Time: 3
Process[4] Burst Time: 6

Gantt Chart:
-----------------------------------------------
| P1 | P2 | P3 | P4 |
-----------------------------------------------
0    5    8    22    14

Process     Arrival Time        Burst Time      Waiting Time    Turnaround Time Completion Time
P1              0               5               0               5               5
P2              1               3               4               7               8
P3              2               8               12              20              22
P4              3               6               5               11              14

Average Waiting Time: 5.25
Average Turnaround Time: 10.75
PS D:\OEPRATING_C\weeek1>
```

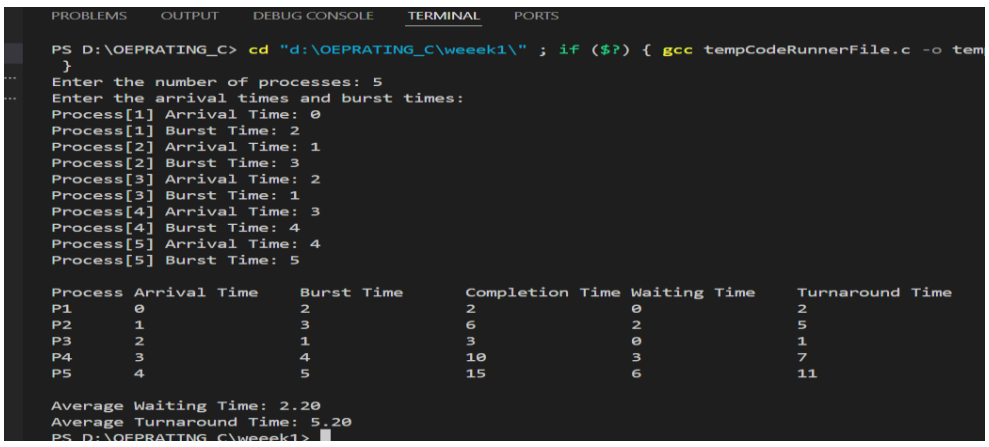
3.Objective : Write a C program to implement SJF pre-emptive Scheduling Algorithm.

SOURCE CODE:

```
#include <stdio.h>

int main() {

    int n, i, currentTime = 0, completed = 0, shortest = 0, finishTime;
    int at[101], bt[101], remainingBt[101], wt[101], tat[101], ct[101];
    int min_time = 9999, is_completed = 0;
    float totalWt = 0, totalTat = 0;

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

    printf("Enter the arrival times and burst times:\n");
    for (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]);
        remainingBt[i] = bt[i];  // Initialize remaining B.T
    }
    int complete = 0, min = 9999, shortestProcess = -1;
    int finish = 0, isAnyProcessRunning = 0;

    // Iterate over all processes until all are completed
    while (completed != n) {
        // Find the process with the shortest remaining B.T
        for (i = 0; i < n; i++) {
            if (at[i] <= currentTime && remainingBt[i] < min && remainingBt[i] > 0) {
                min = remainingBt[i];
                shortestProcess = i;
                isAnyProcessRunning = 1;
            }
        }

        if (!isAnyProcessRunning) {
            currentTime++;
            continue;
        }

        // Reduce the remaining burst time of the selected process
        remainingBt[shortestProcess]--;
```

```
        // If a process gets completely executed
        if (remainingBt[shortestProcess] == 0) {
            completed++;
            isAnyProcessRunning = 0;
            finishTime = currentTime + 1;

            // Calculate completion, WT, and TAT
            ct[shortestProcess] = finishTime;
       tat[shortestProcess] = ct[shortestProcess] - at[shortestProcess]; // TAT = C.T -A.T
        wt[shortestProcess] = tat[shortestProcess] - bt[shortestProcess]; // WT = TAT-BT

            // Accumulate wt and TAT for avg calc.
            totalWt += wt[shortestProcess];
            totalTat += tat[shortestProcess];

            if (wt[shortestProcess] < 0) {
                wt[shortestProcess] = 0; }

            min = 9999; // Reset min for next process   }
        currentTime++;
    }
    printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tWaiting
Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, at[i], bt[i], ct[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f", totalWt / n);
    printf("\nAverage Turnaround Time: %.2f\n", totalTat / n);

    return 0;
}
```

OUTPUT:



Week No-4 Practical Questions

Objective : Write a C program to implement Priority Scheduling Algorithm**.**

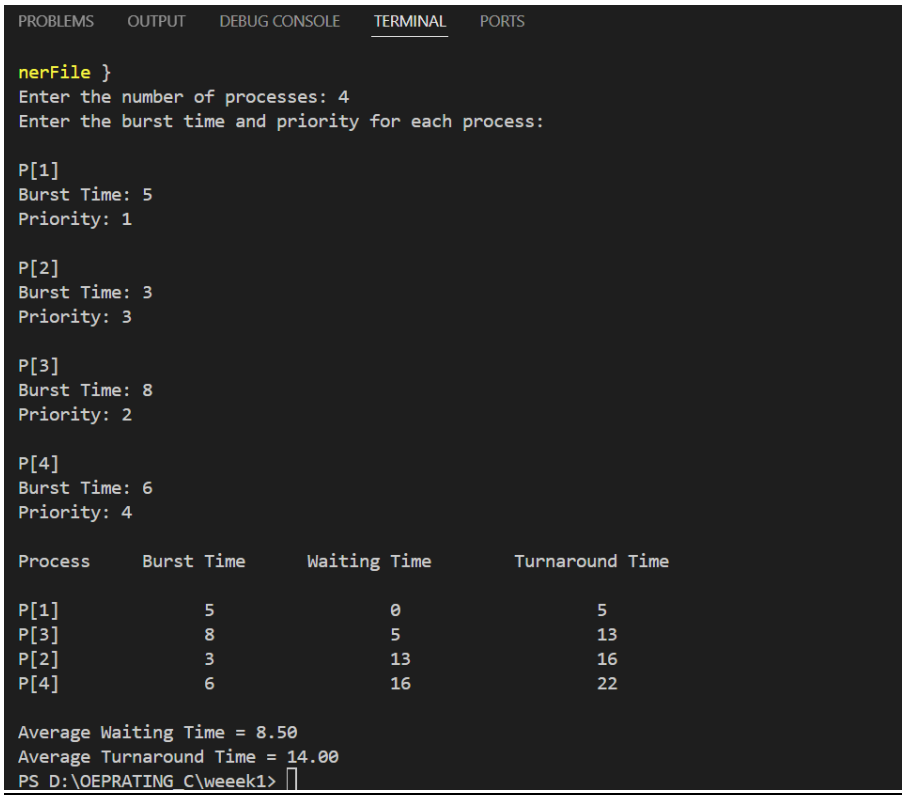
SOURCE CODE:

```
#include<stdio.h>
int main() {
    int p[101], bt[101], wt[101], tat[101], pr[101], i, j, n, total = 0, pos, temp;
    float avgWt, avgTat;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter the burst time and priority for each process:\n");
    for (i = 0; i < n; i++) {
        printf("\nP[%d]\n", i + 1);
        printf("Burst Time: ");
        scanf("%d", &bt[i]);
        printf("Priority: ");
        scanf("%d", &pr[i]);
        p[i] = i + 1;
    }
    // Sort processes based on priority (lower value means higher priority)
    for (i = 0; i < n; i++) {
        pos = i;
        for (j = i + 1; j < n; j++) {
            if (pr[j] < pr[pos]) {
                pos = j;
            }
        }
```

```
    // Swap priority
    temp = pr[i];
    pr[i] = pr[pos];
    pr[pos] = temp;

    // Swap burst time
    temp = bt[i];
    bt[i] = bt[pos];
    bt[pos] = temp;

    // Swap process number
    temp = p[i];
    p[i] = p[pos];
    p[pos] = temp;
}
// Calculate W.T for each process
wt[0] = 0;  // W.T for the first process is 0
total = 0;
for (i = 1; i < n; i++) {
    wt[i] = 0;
    for (j = 0; j < i; j++) {
        wt[i] += bt[j];
    }
    total += wt[i];
}
// Cal. avg. WT
avgWt = (float)total / n;

// Cal. TAT for each process
```

```
    total = 0;
    printf("\nProcess\t    Burst Time\t    Waiting Time\tTurnaround Time\n");

    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];  // TAT = B.T + W.T
        total += tat[i];
        // Print process Details
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d", p[i], bt[i], wt[i], tat[i]);
    }
    // Cal. avg TAT
    avgTat = (float)total / n;
    printf("\n\nAverage Waiting Time = %.2f", avgWt);
    printf("\nAverage Turnaround Time = %.2f\n", avgTat);
    return 0;
}
```

OUTPUT::



2. Objective :: Write a C program to implement Round Robin Scheduling.

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
int main(){
	int n,i,qt,count=0,temp,f=0,bt[101],wt[101],tat[101],rt[101],ct[101];
	float avgWt=0,avgTat=0;

	printf("Enter number of process: ");
	scanf("%d",&n);
	 printf("Enter the burst time\n");
	for(i=0;i<n;++i){
		scanf("%d",&bt[i]);
		rt[i]=bt[i];
	}
	printf("Time quantum: ");
	scanf("%d",&qt);
	while(1)
	{
		for(i=0,count=0;i<n;++i){
			temp=qt;
			if(rt[i]==0){
				count++;
				continue;
			}
			if(rt[i]>qt){
			rt[i]-=qt;
			}
			else
```

```
                    if(rt[i]>=0){
                        temp=rt[i];
                        rt[i]=0;
                    }
                f+=temp;
                tat[i]=f;
            }
            if(n==count){
                break;
            }
    }
    printf("\nprocess\t\tBurst time\t\tTurnaround time\t\tWaiting time");
    for(i=0;i<n;++i){
            wt[i]=tat[i]-bt[i];
            avgWt+=wt[i];
            avgTat+=tat[i];
            printf("\n%d\t\t%d\t\t\t\t%d\t\t\t%d",i+1,bt[i],tat[i],wt[i]);
    }
    avgWt/=n;
    avgTat/=n;
    printf("\nAverage waiting time=%f",avgWt);
    printf("\nAverage turnaround time=%f",avgTat);}
```

OUTPUT: