PRATICAL FILE
OF
OPERATING SYSTEM

Session 2024–2025

DAYANAND COLLEGE

HISAR

Submitted to:

Mr. Karamvir Sir

Assistant Professor

(Department of Computer Science)

Submitted by:

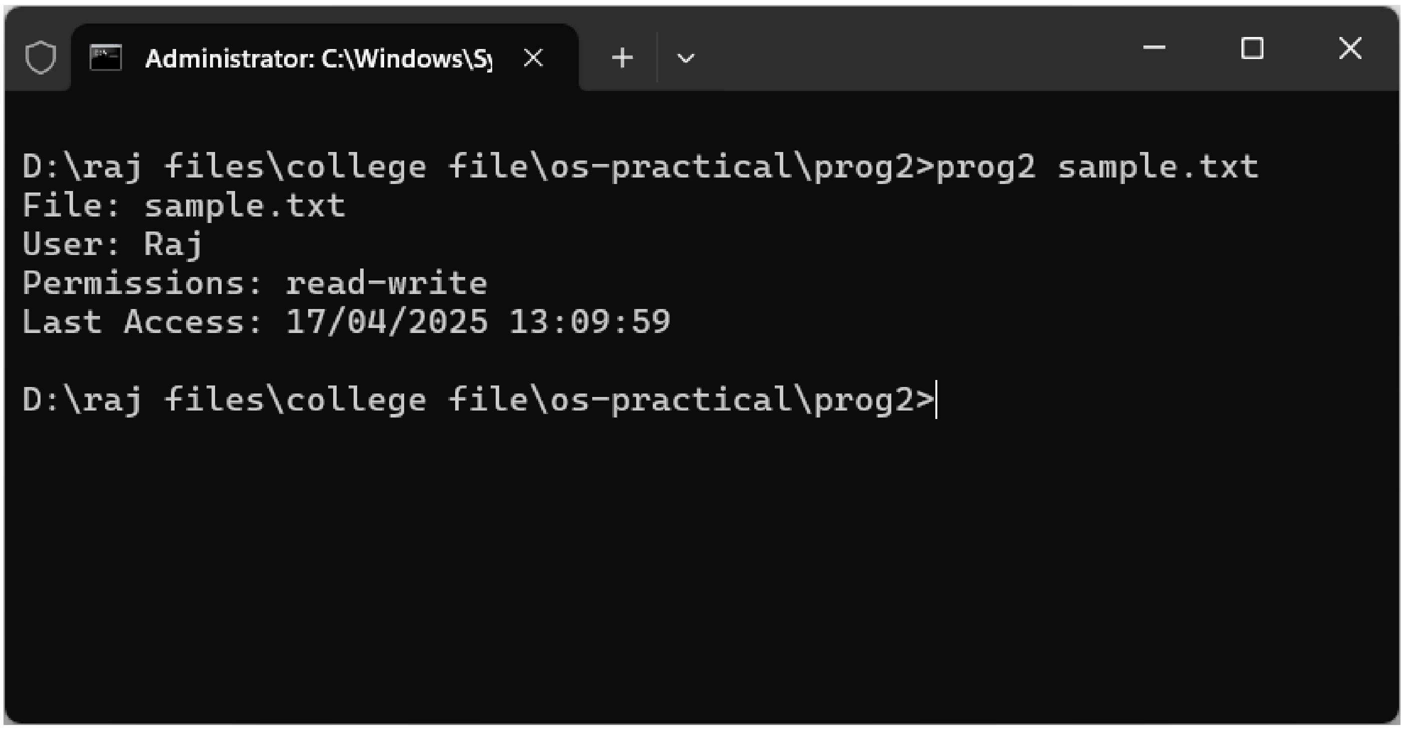
Voneeta

Class–BCA-2nd Sem.

Uni. Roll no.:

243042220114

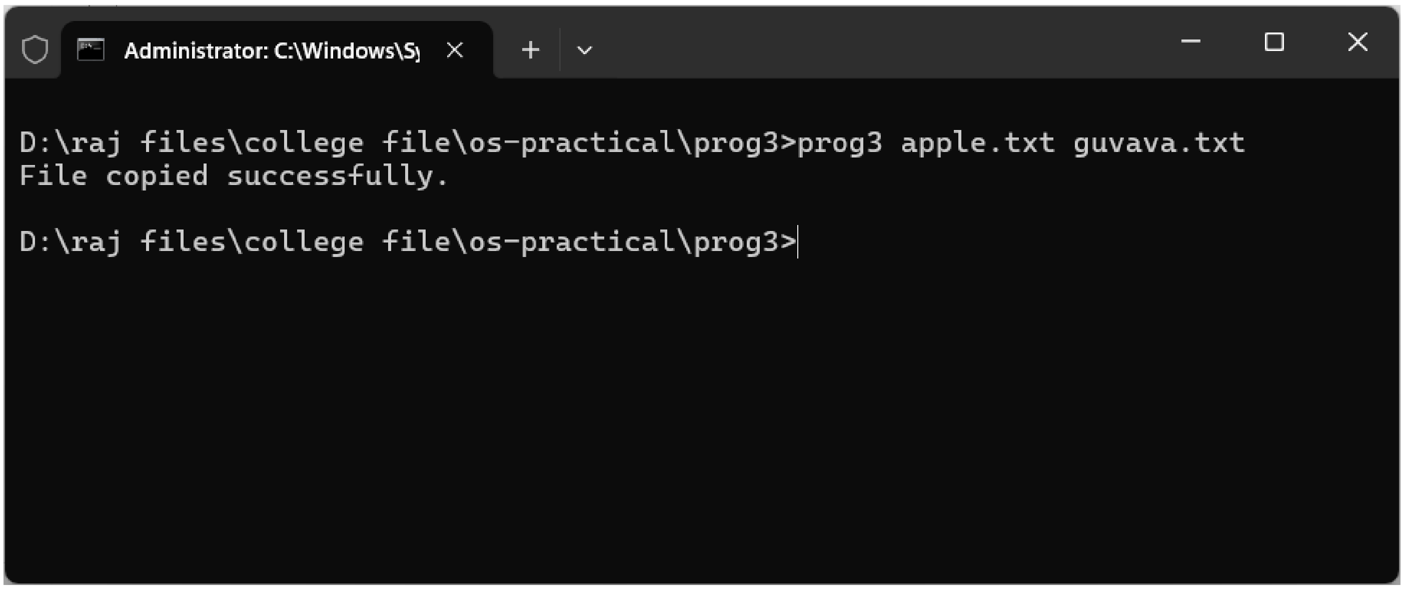
Index



```
D:\raj files\college file\os-practical\prog2>prog2 sample.txt
File: sample.txt
User: Raj
Permissions: read-write
Last Access: 17/04/2025 13:09:59

D:\raj files\college file\os-practical\prog2>
```

1. Write a program to print file details including all access, permissions, and file access time where file name is given as argument

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

void print_access_time(FILETIME ft) {
    SYSTEMTIME stUTC, stLocal;
    FileTimeToSystemTime(&ft, &stUTC);
    SystemTimeToTzSpecificLocalTime(NULL, &stUTC, &stLocal);

    printf("Last Access: %02d/%02d/%04d %02d:%02d:%02d\n",
        stLocal.wDay, stLocal.wMonth, stLocal.wYear,
        stLocal.wHour, stLocal.wMinute, stLocal.wSecond);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    // Get current user
    char username[256];
    DWORD size = sizeof(username);
    if (!GetUserNameA(username, &size)) {
        perror("GetUserName failed");
        return 1;
    }

    // Get file info
    WIN32_FILE_ATTRIBUTE_DATA fileAttr;
    if (!GetFileAttributesExA(argv[1], GetFileExInfoStandard, &fileAttr)) {
        printf("Cannot access file: %s\n", argv[1]);
        return 1;
    }

    printf("File: %s\n", argv[1]);
    printf("User: %s\n", username);
    printf("Permissions:       %s\n",        (fileAttr.dwFileAttributes        &
FILE_ATTRIBUTE_READONLY) ? "read-only" : "read-write");
    print_access_time(fileAttr.ftLastAccessTime);

    return 0;
}
```

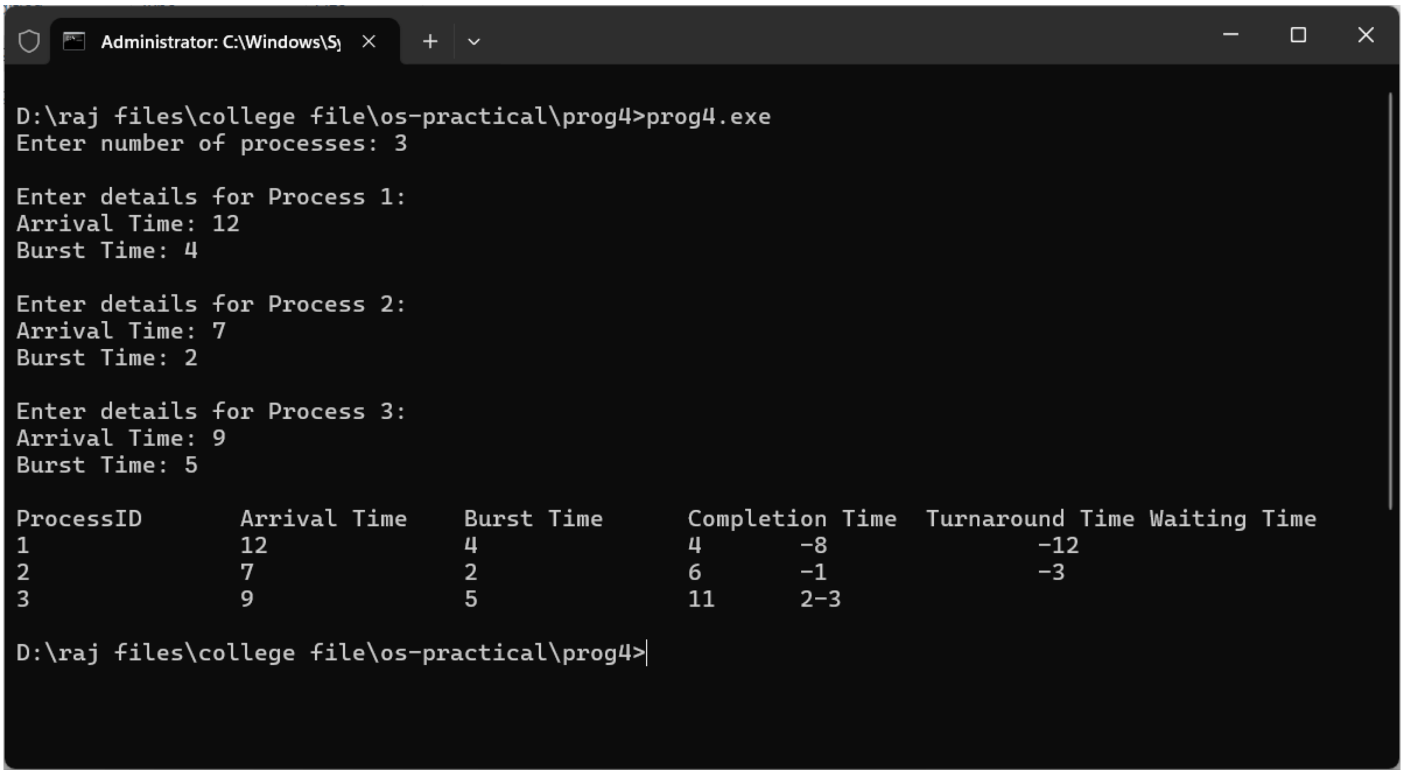


```
D:\raj files\college file\os-practical\prog3>prog3 apple.txt guvava.txt
File copied successfully.

D:\raj files\college file\os-practical\prog3>
```

2. Write a program to copy files using system calls.

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

#define BUF_SIZE 1024  // Buffer size for copying in chunks

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: %s <source_file> <destination_file>\n", argv[0]);
        return 1;
    }

    // Open the source file (read-only)
    HANDLE sourceFile = CreateFile(argv[1], GENERIC_READ, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
    if (sourceFile == INVALID_HANDLE_VALUE) {
        printf("Error opening source file: %lu\n", GetLastError());
        return 1;
    }

    // Create or open the destination file (write-only, create it if it doesn't
exist)
    HANDLE destFile = CreateFile(argv[2], GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);
    if (destFile == INVALID_HANDLE_VALUE) {
        printf("Error opening destination file: %lu\n", GetLastError());
        CloseHandle(sourceFile);
        return 1;
    }

    char buffer[BUF_SIZE];
    DWORD bytesRead, bytesWritten;

    // Read from the source file and write to the destination file
    while (ReadFile(sourceFile, buffer, BUF_SIZE, &bytesRead, NULL) && bytesRead
> 0) {
        if (!WriteFile(destFile, buffer, bytesRead, &bytesWritten, NULL)) {
            printf("Error writing to destination file: %lu\n", GetLastError());
            CloseHandle(sourceFile);
            CloseHandle(destFile);
            return 1;
        }
    }

    if (bytesRead == 0) {
        printf("File copied successfully.\n");
    } else {
        printf("Error reading from source file: %lu\n", GetLastError());
```

```
 }

    // Close both files
    CloseHandle(sourceFile);
    CloseHandle(destFile);

    return 0;
}
```

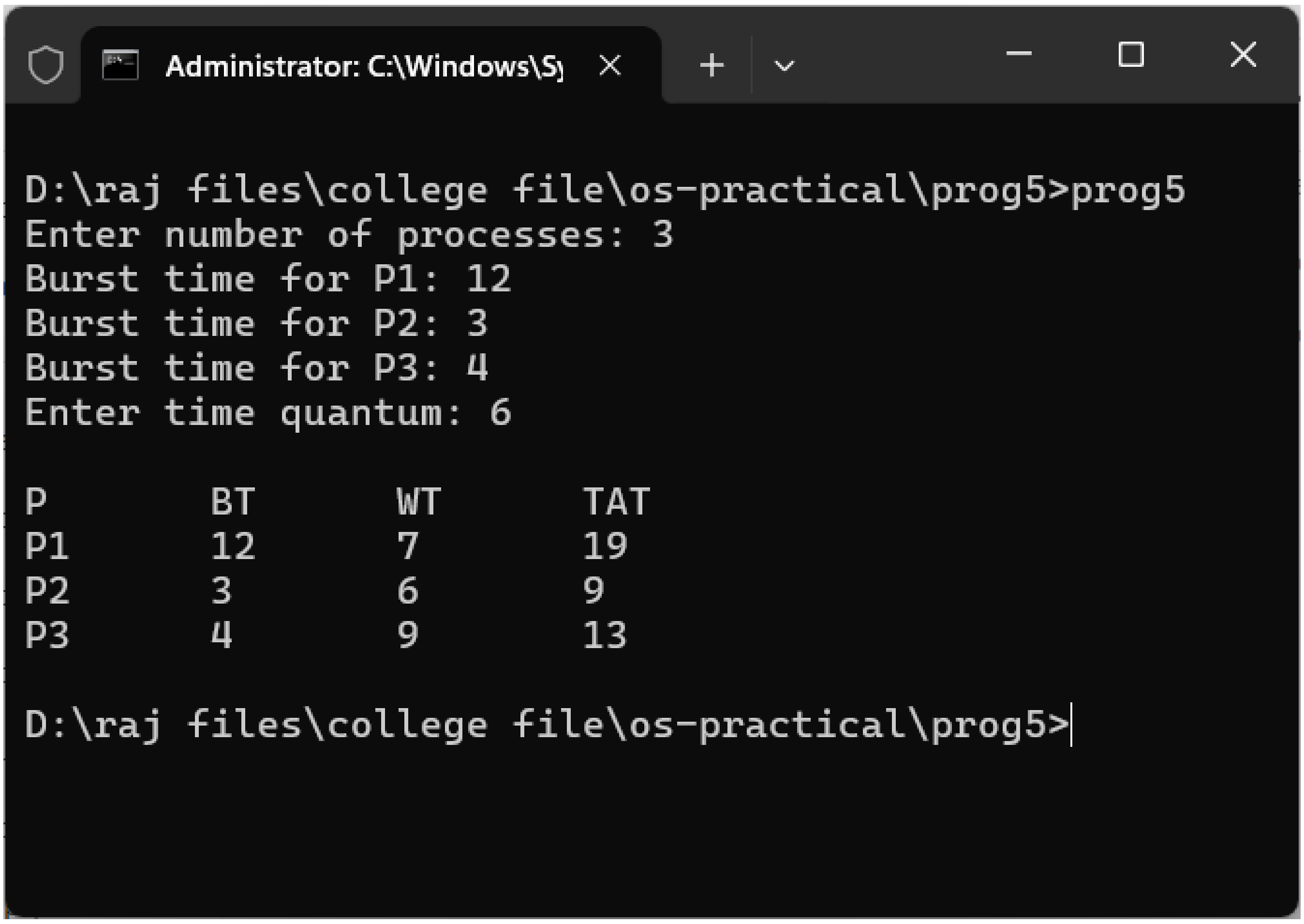


```
D:\raj files\college file\os-practical\prog4>prog4.exe
Enter number of processes: 3

Enter details for Process 1:
Arrival Time: 12
Burst Time: 4

Enter details for Process 2:
Arrival Time: 7
Burst Time: 2

Enter details for Process 3:
Arrival Time: 9
Burst Time: 5

ProcessID       Arrival Time    Burst Time      Completion Time  Turnaround Time Waiting Time
1               12              4               4       -8              -12
2               7               2               6       -1              -3
3               9               5               11      2-3

D:\raj files\college file\os-practical\prog4>
```

3. Write a program to implement the FCFS scheduling algorithm.

```
#include <stdio.h>

typedef struct {
    int process_id;    // Process ID
    int arrival_time;  // Arrival time
    int burst_time;    // Burst time (execution time)
    int completion_time; // Completion time
    int turn_around_time; // Turnaround time
    int waiting_time;    // Waiting time
} Process;

void calculate_times(Process processes[], int n) {
    int current_time = 0;

    // Calculate completion time, waiting time, and turnaround time
    for (int i = 0; i < n; i++) {
        // Completion time is current time + burst time
        processes[i].completion_time = current_time + processes[i].burst_time;

        // Turnaround time is completion time - arrival time
        processes[i].turn_around_time    =    processes[i].completion_time    -
processes[i].arrival_time;

        // Waiting time is turnaround time - burst time
        processes[i].waiting_time     =     processes[i].turn_around_time     -
processes[i].burst_time;

        // Update current time
        current_time = processes[i].completion_time;
    }
}

void print_table(Process processes[], int n) {
    printf("\nProcessID\tArrival  Time\tBurst  Time\tCompletion  Time\tTurnaround
Time\tWaiting Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
                processes[i].process_id,
                processes[i].arrival_time,
                processes[i].burst_time,
                processes[i].completion_time,
                processes[i].turn_around_time,
                processes[i].waiting_time);
    }
}
```

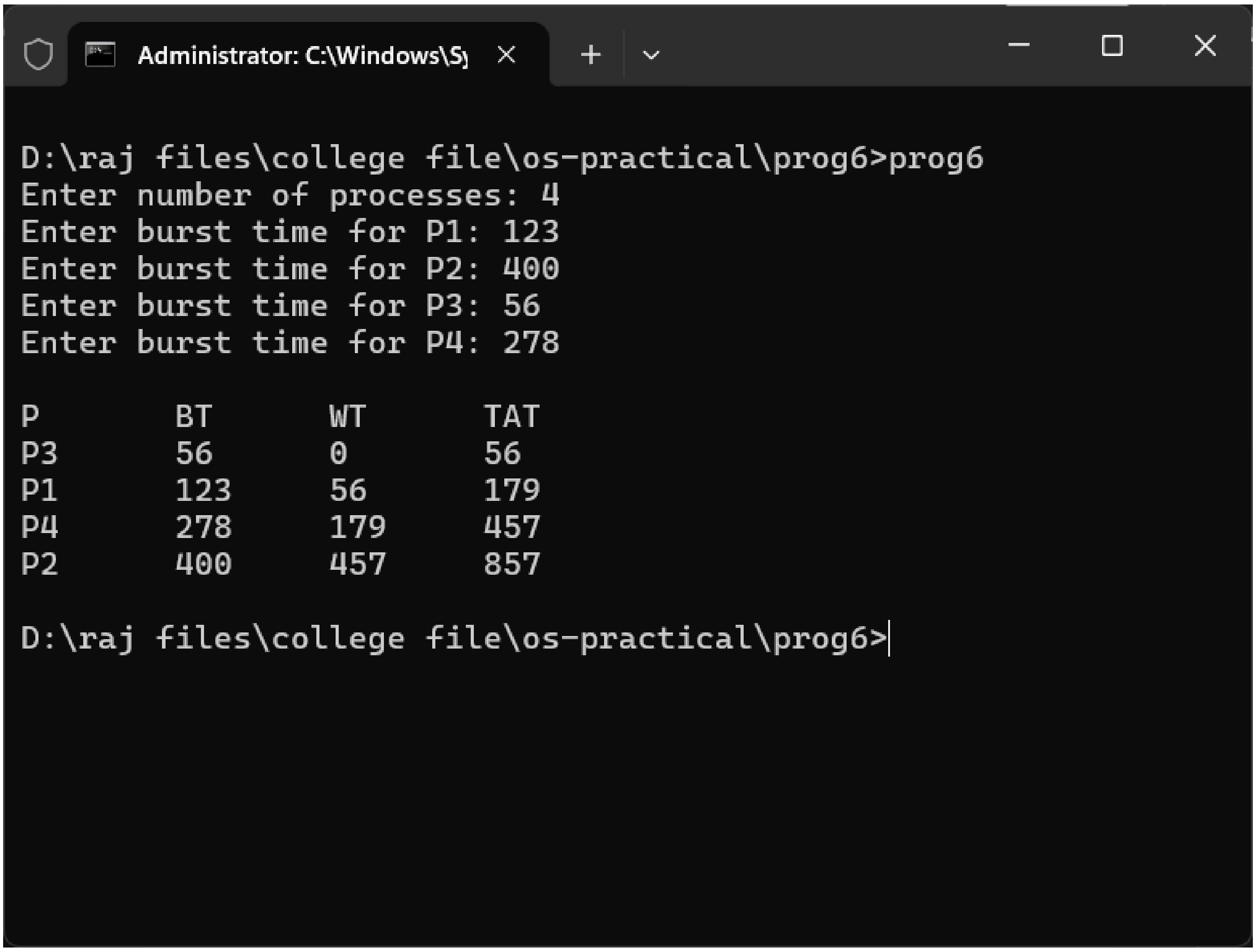
```c
int main() {
    int n;

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

    Process processes[n];

    // Get process details
    for (int i = 0; i < n; i++) {
        printf("\nEnter details for Process %d:\n", i + 1);
        processes[i].process_id = i + 1;
        printf("Arrival Time: ");
        scanf("%d", &processes[i].arrival_time);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burst_time);
    }

    // Calculate times (Completion, Turnaround, Waiting)
    calculate_times(processes, n);

    // Print the results in a table
    print_table(processes, n);

    return 0;
}
```



```
D:\raj files\college file\os-practical\prog5>prog5
Enter number of processes: 3
Burst time for P1: 12
Burst time for P2: 3
Burst time for P3: 4
Enter time quantum: 6

P       BT      WT      TAT
P1      12      7       19
P2      3       6       9
P3      4       9       13

D:\raj files\college file\os-practical\prog5>
```

4. Write a program to implement the Round Robin scheduling algorithm.

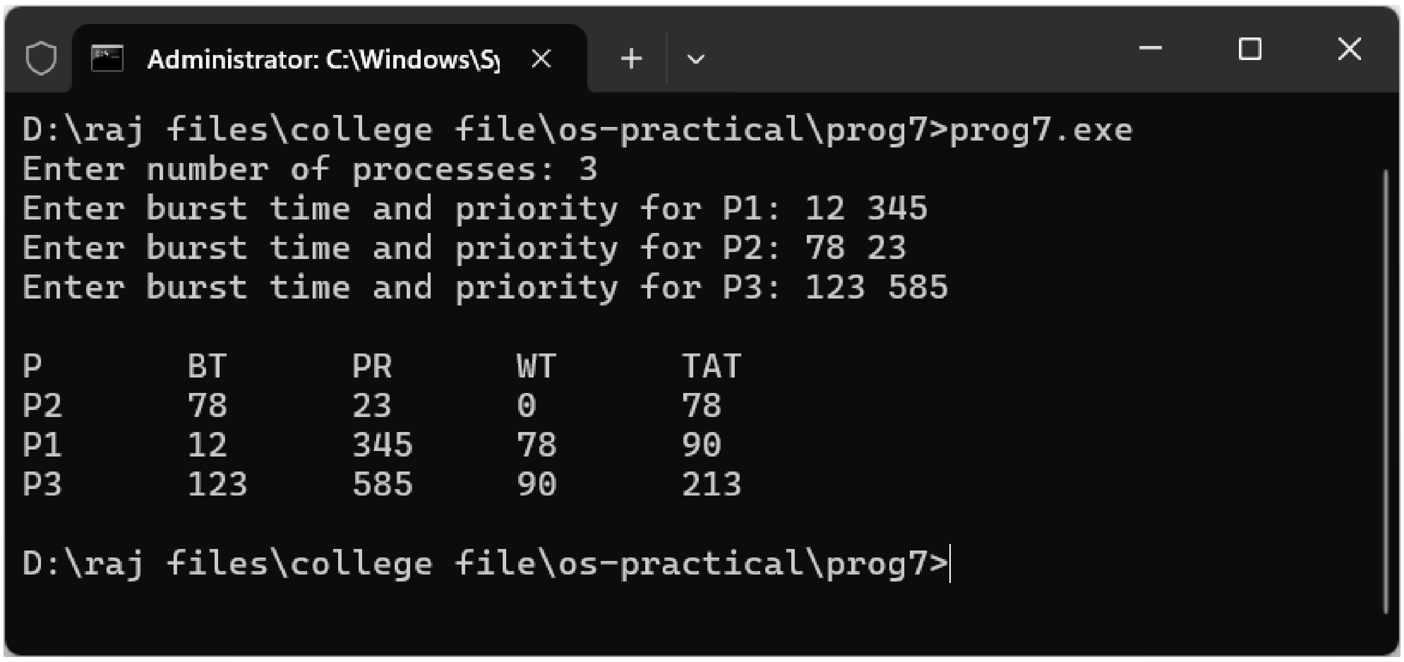
```
#include <stdio.h>

int main() {
    int n, tq, bt[100], rt[100], wt[100] = {0}, tat[100], time = 0, done;

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

    for (int i = 0; i < n; i++) {
        printf("Burst time for P%d: ", i + 1);
        scanf("%d", &bt[i]);
        rt[i] = bt[i];
    }

    printf("Enter time quantum: ");
    scanf("%d", &tq);

    do {
        done = 1;
        for (int i = 0; i < n; i++) {
            if (rt[i] > 0) {
                done = 0;
                int t = (rt[i] > tq) ? tq : rt[i];
                rt[i] -= t;
                time += t;
                if (rt[i] == 0)
                    wt[i] = time - bt[i];
            }
        }
    } while (!done);

    printf("\nP\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        printf("P%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    }

    return 0;
}
```



```
D:\raj files\college file\os-practical\prog6>prog6
Enter number of processes: 4
Enter burst time for P1: 123
Enter burst time for P2: 400
Enter burst time for P3: 56
Enter burst time for P4: 278

P       BT      WT      TAT
P3      56      0       56
P1      123     56      179
P4      278     179     457
P2      400     457     857

D:\raj files\college file\os-practical\prog6>
```

5. Write a program to implement the SJF scheduling algorithm.

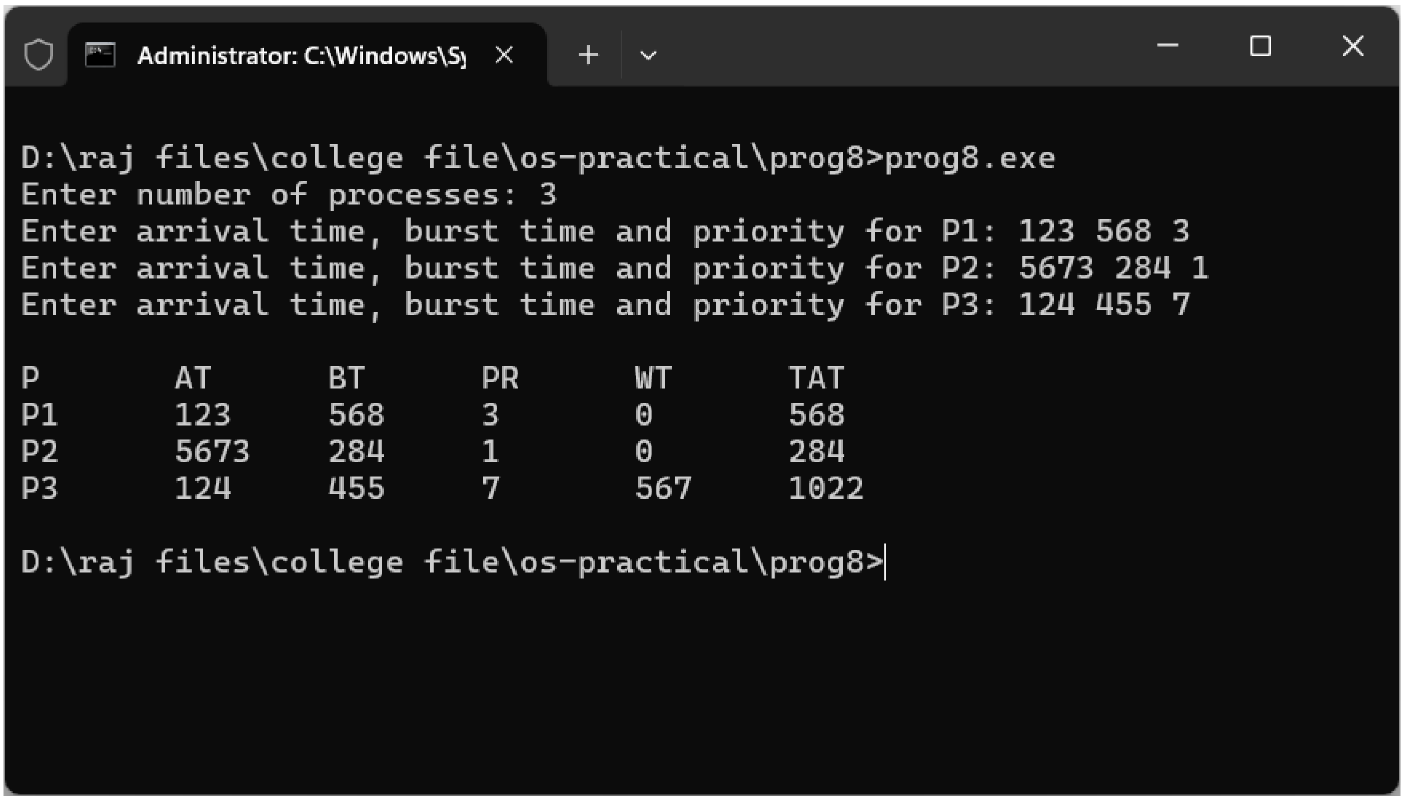
```
#include <stdio.h>

int main() {

    int n, bt[100], wt[100], tat[100], i, j;

    int p[100], temp;

    printf("Enter number of processes: ");

    scanf("%d", &n);

    for (i = 0; i < n; i++) {

        printf("Enter burst time for P%d: ", i + 1);

        scanf("%d", &bt[i]);

        p[i] = i + 1;  // Process ID

    }

    for (i = 0; i < n - 1; i++) {

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

            if (bt[i] > bt[j]) {

                temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;

                temp = p[i];  p[i] = p[j];  p[j] = temp;

            }

        }

    }

    wt[0] = 0;

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

        wt[i] = wt[i - 1] + bt[i - 1];

    for (i = 0; i < n; i++)

        tat[i] = bt[i] + wt[i];


    printf("\nP\tBT\tWT\tTAT\n");

    for (i = 0; i < n; i++)

        printf("P%d\t%d\t%d\t%d\n", p[i], bt[i], wt[i], tat[i]);


    return 0;

}
```



```
D:\raj files\college file\os-practical\prog7>prog7.exe
Enter number of processes: 3
Enter burst time and priority for P1: 12 345
Enter burst time and priority for P2: 78 23
Enter burst time and priority for P3: 123 585

P       BT      PR      WT      TAT
P2      78      23      0       78
P1      12      345     78      90
P3      123     585     90      213

D:\raj files\college file\os-practical\prog7>
```

6. Write a program to implement a non-preemptive priority based scheduling algorithm.

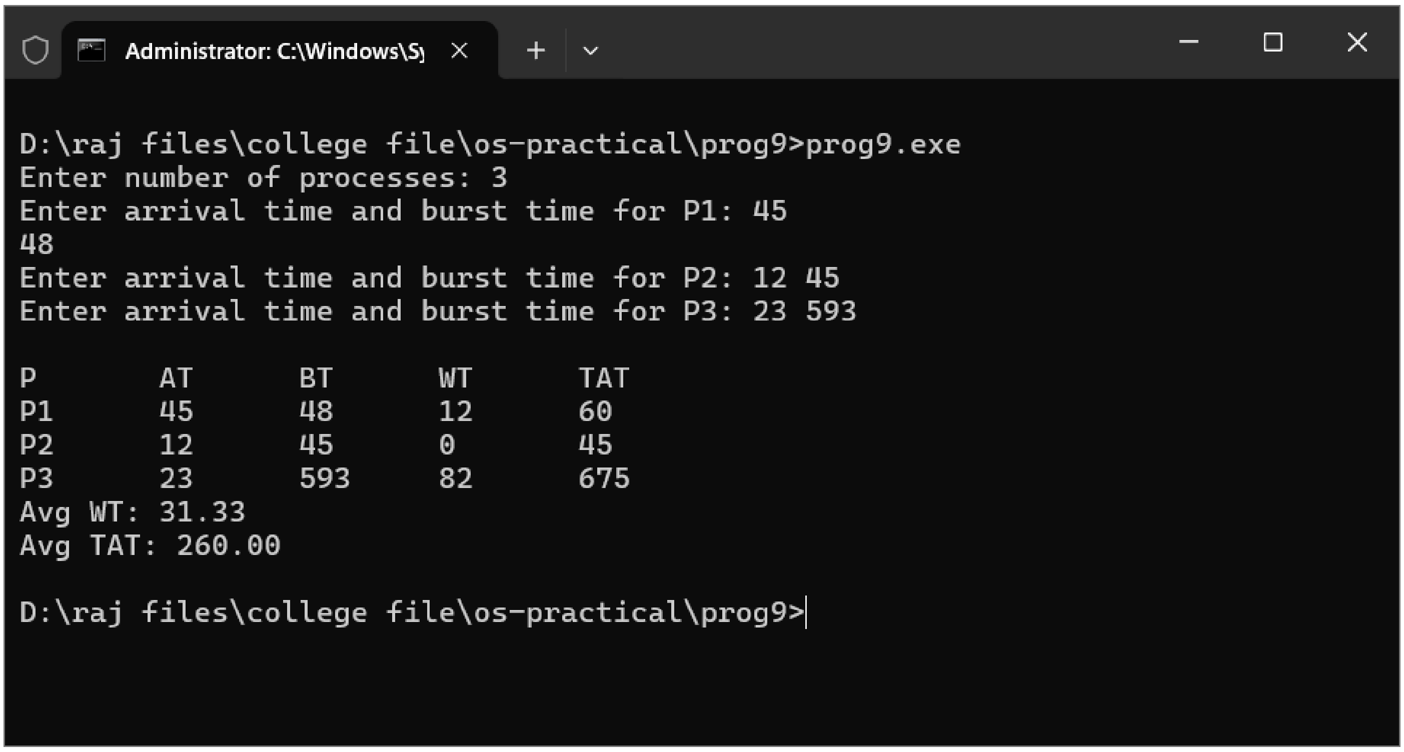
```c
#include <stdio.h>

int main() {

    int n, i, j, temp;

    int bt[100], p[100], pr[100], wt[100], tat[100];

    printf("Enter number of processes: ");

    scanf("%d", &n);

    for (i = 0; i < n; i++) {

        printf("Enter burst time and priority for P%d: ", i + 1);

        scanf("%d %d", &bt[i], &pr[i]);

        p[i] = i + 1;

    }

    for (i = 0; i < n - 1; i++) {

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

            if (pr[i] > pr[j]) {

                temp = pr[i]; pr[i] = pr[j]; pr[j] = temp;

                temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;

                temp = p[i];  p[i] = p[j];  p[j] = temp;

            }

        }

    }

    wt[0] = 0;

    for (i = 1; i < n; i++) wt[i] = wt[i - 1] + bt[i - 1];

    for (i = 0; i < n; i++) tat[i] = bt[i] + wt[i];

    printf("\nP\tBT\tPR\tWT\tTAT\n");

    for (i = 0; i < n; i++)

    printf("P%d\t%d\t%d\t%d\t%d\n", p[i], bt[i], pr[i], wt[i], tat[i]);


    return 0;

}
```



```
D:\raj files\college file\os-practical\prog8>prog8.exe
Enter number of processes: 3
Enter arrival time, burst time and priority for P1: 123 568 3
Enter arrival time, burst time and priority for P2: 5673 284 1
Enter arrival time, burst time and priority for P3: 124 455 7

P       AT      BT      PR      WT      TAT
P1      123     568     3       0       568
P2      5673    284     1       0       284
P3      124     455     7       567     1022

D:\raj files\college file\os-practical\prog8>
```

7. Write a program to implement a preemptive priority based scheduling algorithm.

```
#include <stdio.h>

int main() {

    int
n,i,t=0,completed=0,bt[100],pr[100],at[100],rt[100],wt[100]={0},tat[100],p[100]
,min_pr,idx;

    printf("Enter number of processes: ");

    scanf("%d",&n);

    for(i=0;i<n;i++){

        printf("Enter arrival time, burst time and priority for P%d: ",i+1);

        scanf("%d%d%d",&at[i],&bt[i],&pr[i]);

        rt[i]=bt[i];

        p[i]=i+1;}

    while(completed<n){

        min_pr=999,idx=-1;

        for(i=0;i<n;i++){

            if(at[i]<=t && rt[i]>0 && pr[i]<min_pr){

                min_pr=pr[i];

                idx=i;

            }}

        if(idx!=-1){

            rt[idx]--;

            if(rt[idx]==0){

                completed++;

                tat[idx]=t+1-at[idx];

                wt[idx]=tat[idx]-bt[idx];

            }}

        t++;}

    printf("\nP\tAT\tBT\tPR\tWT\tTAT\n");

    for(i=0;i<n;i++)

    printf("P%d\t%d\t%d\t%d\t%d\t%d\n",p[i],at[i],bt[i],pr[i],wt[i],tat[i]);
    return 0;}
```

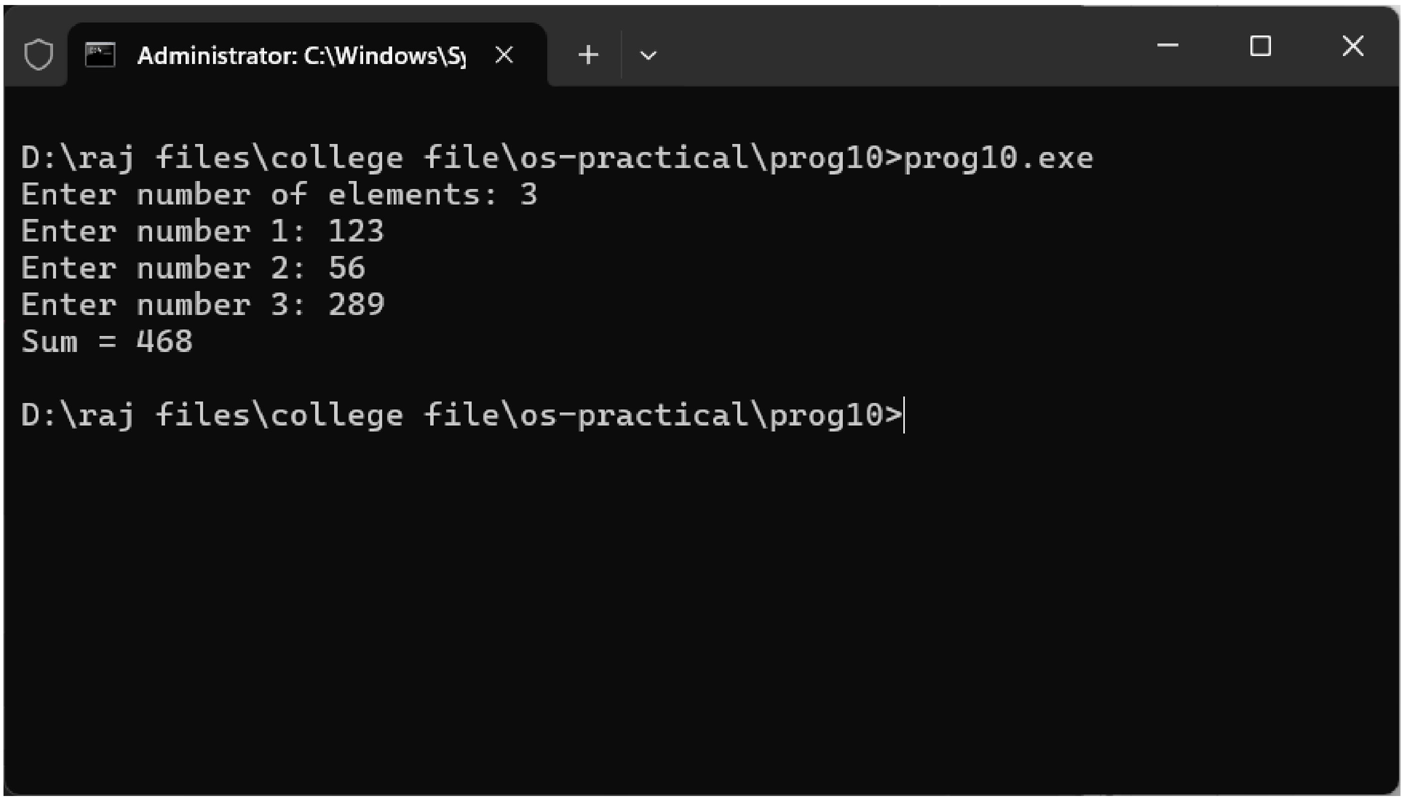

Administrator: C:\Windows\Sy

```
D:\raj files\college file\os-practical\prog9>prog9.exe
Enter number of processes: 3
Enter arrival time and burst time for P1: 45
48
Enter arrival time and burst time for P2: 12 45
Enter arrival time and burst time for P3: 23 593

P       AT      BT      WT      TAT
P1      45      48      12      60
P2      12      45      0       45
P3      23      593     82      675
Avg WT: 31.33
Avg TAT: 260.00

D:\raj files\college file\os-practical\prog9>
```

8. Write a program to implement the SRJF scheduling algorithm.

```
#include <stdio.h>

int main() {

    int n, at[100], bt[100], rt[100], wt[100], tat[100], t=0, completed=0,
shortest, min_rt=1e9, finish_time, i;

    float avg_wt=0, avg_tat=0;

    printf("Enter number of processes: ");

    scanf("%d",&n);

    for(i=0;i<n;i++){

        printf("Enter arrival time and burst time for P%d: ",i+1);

        scanf("%d%d",&at[i],&bt[i]);

        rt[i]=bt[i];

    }

    while(completed<n){

        shortest=-1;

        min_rt=1e9;

        for(i=0;i<n;i++){

            if(at[i]<=t && rt[i]>0 && rt[i]<min_rt){

                min_rt=rt[i];

                shortest=i;

            }

        }

        if(shortest==-1){

            t++;

            continue;

        }

        rt[shortest]--;

        if(rt[shortest]==0){

            completed++;

            finish_time = t+1;

            tat[shortest] = finish_time - at[shortest];

            wt[shortest] = tat[shortest] - bt[shortest];
```

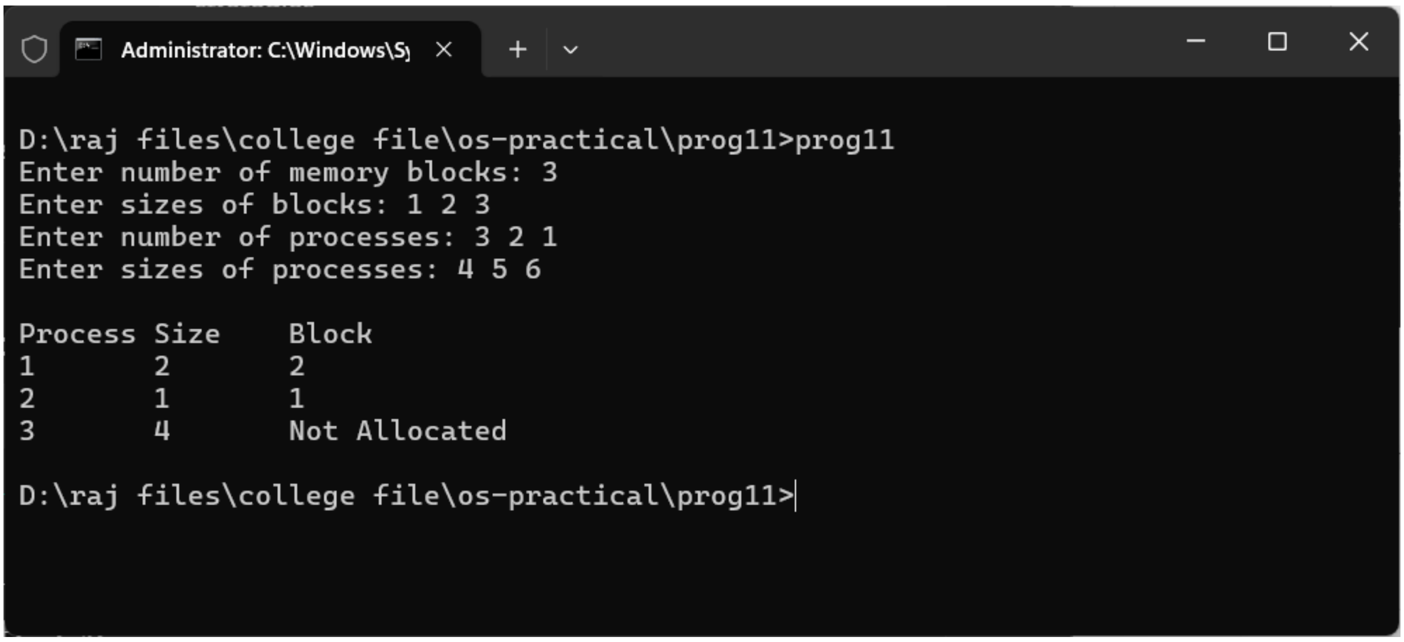
```
        }
        t++;
    }
    printf("\nP\tAT\tBT\tWT\tTAT\n");
    for(i=0;i<n;i++){
        avg_wt += wt[i];
        avg_tat += tat[i];
        printf("P%d\t%d\t%d\t%d\t%d\n",i+1,at[i],bt[i],wt[i],tat[i]);
    }
    printf("Avg WT: %.2f\nAvg TAT: %.2f\n", avg_wt/n, avg_tat/n);
    return 0;
}
```



```
D:\raj files\college file\os-practical\prog10>prog10.exe
Enter number of elements: 3
Enter number 1: 123
Enter number 2: 56
Enter number 3: 289
Sum = 468

D:\raj files\college file\os-practical\prog10>
```

9. Write a program to calculate the sum of n numbers using the thread library.

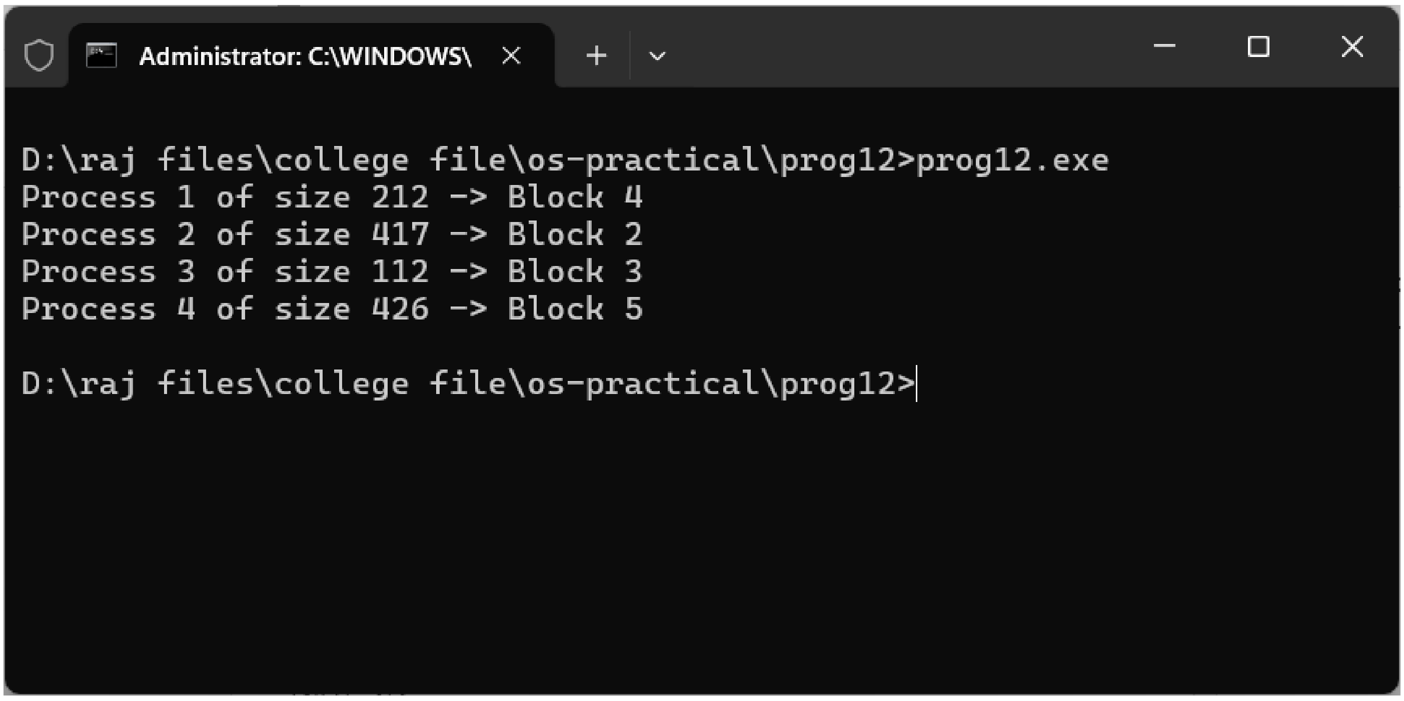
```
#include <stdio.h>
#include <pthread.h>

int numbers[100];
int n, sum = 0;

void* calculate_sum(void* arg) {
    for(int i = 0; i < n; i++) sum += numbers[i];
    return NULL;
}

int main() {
    pthread_t tid;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        printf("Enter number %d: ", i+1);
        scanf("%d", &numbers[i]);
    }
    pthread_create(&tid, NULL, calculate_sum, NULL);
    pthread_join(tid, NULL);
    printf("Sum = %d\n", sum);
    return 0;
}
```



```
D:\raj files\college file\os-practical\prog11>prog11
Enter number of memory blocks: 3
Enter sizes of blocks: 1 2 3
Enter number of processes: 3 2 1
Enter sizes of processes: 4 5 6

Process Size    Block
1       2       2
2       1       1
3       4       Not Allocated

D:\raj files\college file\os-practical\prog11>
```

10. Write a program to implement first-fit allocation strategies.

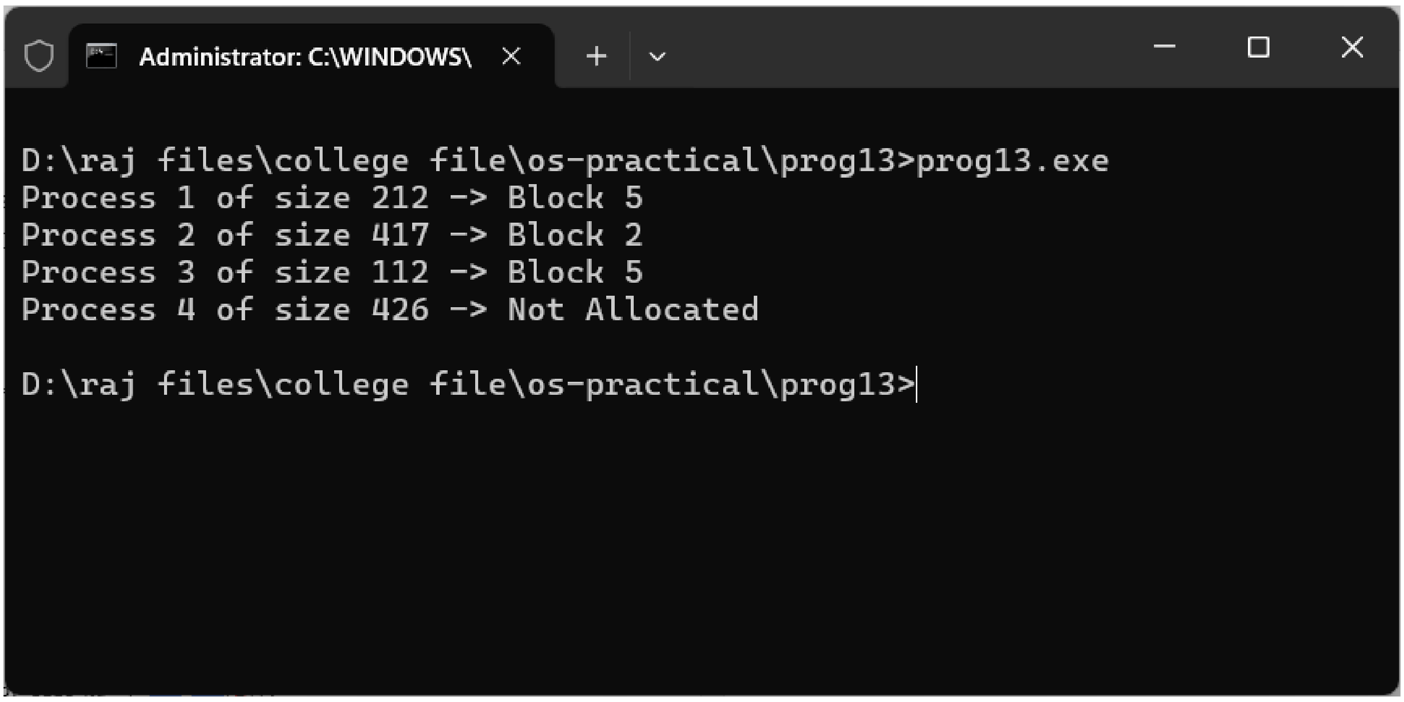
```
#include <stdio.h>
int main() {
    int m, p, i, j;
    int block[100], process[100], alloc[100];

    printf("Enter number of memory blocks: ");
    scanf("%d", &m);
    printf("Enter sizes of blocks: ");
    for(i = 0; i < m; i++) scanf("%d", &block[i]);

    printf("Enter number of processes: ");
    scanf("%d", &p);
    printf("Enter sizes of processes: ");
    for(i = 0; i < p; i++) scanf("%d", &process[i]);

    for(i = 0; i < p; i++) {
        alloc[i] = -1;
        for(j = 0; j < m; j++) {
            if(block[j] >= process[i]) {
                alloc[i] = j;
                block[j] -= process[i];
                break;
            }}
    }
    printf("\nProcess\tSize\tBlock\n");
    for(i = 0; i < p; i++) {
        printf("%d\t%d\t", i + 1, process[i]);
        if(alloc[i] != -1) printf("%d\n", alloc[i] + 1);
        else printf("Not Allocated\n");
    }
return 0;}
```



```
D:\raj files\college file\os-practical\prog12>prog12.exe
Process 1 of size 212 -> Block 4
Process 2 of size 417 -> Block 2
Process 3 of size 112 -> Block 3
Process 4 of size 426 -> Block 5

D:\raj files\college file\os-practical\prog12>
```

11. Write a program to implement best-fit allocation strategies.

```
#include<stdio.h>
int main(){
int b[]={100,500,200,300,600},p[]={212,417,112,426},i,j,bi=-1;
int n=5,m=4,alloc[4];
for(i=0;i<m;i++){
bi=-1;
for(j=0;j<n;j++)
if(b[j]>=p[i]&&(bi==-1||b[j]<b[bi]))
bi=j;
if(bi!=-1){
alloc[i]=bi;
b[bi]-=p[i];
}else alloc[i]=-1;
}
for(i=0;i<m;i++){
printf("Process %d of size %d -> ",i+1,p[i]);
if(alloc[i]!=-1)printf("Block %d\n",alloc[i]+1);
else printf("Not Allocated\n");
}
return 0;
}
```



```
D:\raj files\college file\os-practical\prog13>prog13.exe
Process 1 of size 212 -> Block 5
Process 2 of size 417 -> Block 2
Process 3 of size 112 -> Block 5
Process 4 of size 426 -> Not Allocated

D:\raj files\college file\os-practical\prog13>
```

12. Write a program to implement worst-fit allocation strategies.

```
#include<stdio.h>

int main(){

int b[]={100,500,200,300,600},p[]={212,417,112,426},i,j,bi;

int n=5,m=4,alloc[4];

for(i=0;i<m;i++){

bi=-1;

for(j=0;j<n;j++)

if(b[j]>=p[i]&&(bi==-1||b[j]>b[bi]))

bi=j;

if(bi!=-1){

alloc[i]=bi;

b[bi]-=p[i];

}else alloc[i]=-1;

}

for(i=0;i<m;i++){

printf("Process %d of size %d -> ",i+1,p[i]);

if(alloc[i]!=-1)printf("Block %d\n",alloc[i]+1);

else printf("Not Allocated\n");

}

return 0;

}
```