**Assignment-04: Scheduling Algorithms**

**Name : Kunal Sachin Kharat**

**Roll no: 33**

**Class : CSAIML-A**

**Batch : 2**

**1.FCFS (First Come First Serve):**

#include <iostream>

#include <algorithm>

#include <iomanip>

using namespace std;

struct process {

    int pid;

    int arrival\_time;

    int burst\_time;

    int start\_time;

    int completion\_time;

    int turnaround\_time;

    int waiting\_time;

    int response\_time;

};

bool compareArrival(process p1, process p2)

{

    return p1.arrival\_time < p2.arrival\_time;

}

bool compareID(process p1, process p2)

{

    return p1.pid < p2.pid;

}

int main() {

    int n;

    struct process p[100];

    float avg\_turnaround\_time;

    float avg\_waiting\_time;

    float avg\_response\_time;

    float cpu\_utilisation;

    int total\_turnaround\_time = 0;

    int total\_waiting\_time = 0;

    int total\_response\_time = 0;

    int total\_idle\_time = 0;

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";

    cin>>n;

    for(int i = 0; i < n; i++) {

        cout<<"Enter arrival time of process "<<i+1<<": ";

        cin>>p[i].arrival\_time;

        cout<<"Enter burst time of process "<<i+1<<": ";

        cin>>p[i].burst\_time;

        p[i].pid = i+1;

        cout<<endl;

    }

    sort(p,p+n,compareArrival);

    for(int i = 0; i < n; i++) {

        p[i].start\_time = (i == 0)?p[i].arrival\_time:max(p[i-1].completion\_time,p[i].arrival\_time);

        p[i].completion\_time = p[i].start\_time + p[i].burst\_time;

        p[i].turnaround\_time = p[i].completion\_time - p[i].arrival\_time;

        p[i].waiting\_time = p[i].turnaround\_time - p[i].burst\_time;

        p[i].response\_time = p[i].start\_time - p[i].arrival\_time;

        total\_turnaround\_time += p[i].turnaround\_time;

        total\_waiting\_time += p[i].waiting\_time;

        total\_response\_time += p[i].response\_time;

        total\_idle\_time += (i == 0)?(p[i].arrival\_time):(p[i].start\_time - p[i-1].completion\_time);

    }

    avg\_turnaround\_time = (float) total\_turnaround\_time / n;

    avg\_waiting\_time = (float) total\_waiting\_time / n;

    avg\_response\_time = (float) total\_response\_time / n;

    cpu\_utilisation = ((p[n-1].completion\_time - total\_idle\_time) / (float) p[n-1].completion\_time)\*100;

    sort(p,p+n,compareID);

    cout<<endl;

    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {

        cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

    }

    cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

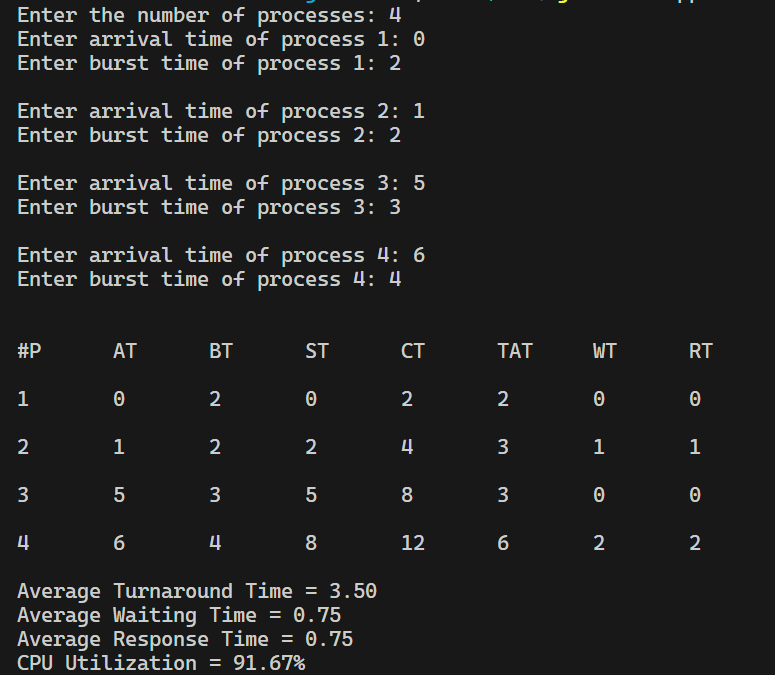
    cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

    cout<<"Average Response Time = "<<avg\_response\_time<<endl;

    cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;

}

**Output:**



**2. SJF(Shortest Job First):**

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <string.h>

using namespace std;

struct process {

    int pid;

    int arrival\_time;

    int burst\_time;

    int start\_time;

    int completion\_time;

    int turnaround\_time;

    int waiting\_time;

    int response\_time;

};

int main() {

    int n;

    struct process p[100];

    float avg\_turnaround\_time;

    float avg\_waiting\_time;

    float avg\_response\_time;

    float cpu\_utilisation;

    int total\_turnaround\_time = 0;

    int total\_waiting\_time = 0;

    int total\_response\_time = 0;

    int total\_idle\_time = 0;

    float throughput;

    int is\_completed[100];

    memset(is\_completed,0,sizeof(is\_completed));

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";

    cin>>n;

    for(int i = 0; i < n; i++) {

        cout<<"Enter arrival time of process "<<i+1<<": ";

        cin>>p[i].arrival\_time;

        cout<<"Enter burst time of process "<<i+1<<": ";

        cin>>p[i].burst\_time;

        p[i].pid = i+1;

        cout<<endl;

    }

    int current\_time = 0;

    int completed = 0;

    int prev = 0;

    while(completed != n) {

        int idx = -1;

        int mn = INT\_FAST32\_MAX;

        for(int i = 0; i < n; i++) {

            if(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {

                if(p[i].burst\_time < mn) {

                    mn = p[i].burst\_time;

                    idx = i;

                }

                if(p[i].burst\_time == mn) {

                    if(p[i].arrival\_time < p[idx].arrival\_time) {

                        mn = p[i].burst\_time;

                        idx = i;

                    }

                }

            }

        }

        if(idx != -1) {

            p[idx].start\_time = current\_time;

            p[idx].completion\_time = p[idx].start\_time + p[idx].burst\_time;

            p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

            p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

            p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;

            total\_turnaround\_time += p[idx].turnaround\_time;

            total\_waiting\_time += p[idx].waiting\_time;

            total\_response\_time += p[idx].response\_time;

            total\_idle\_time += p[idx].start\_time - prev;

            is\_completed[idx] = 1;

            completed++;

            current\_time = p[idx].completion\_time;

            prev = current\_time;

        }

        else {

            current\_time++;

        }

    }

    int min\_arrival\_time = INT\_FAST32\_MAX;

    int max\_completion\_time = -1;

    for(int i = 0; i < n; i++) {

        min\_arrival\_time = min(min\_arrival\_time,p[i].arrival\_time);

        max\_completion\_time = max(max\_completion\_time,p[i].completion\_time);

    }

    avg\_turnaround\_time = (float) total\_turnaround\_time / n;

    avg\_waiting\_time = (float) total\_waiting\_time / n;

    avg\_response\_time = (float) total\_response\_time / n;

    cpu\_utilisation = ((max\_completion\_time - total\_idle\_time) / (float) max\_completion\_time )\*100;

    cout<<endl<<endl;

    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {

        cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

    }

    cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

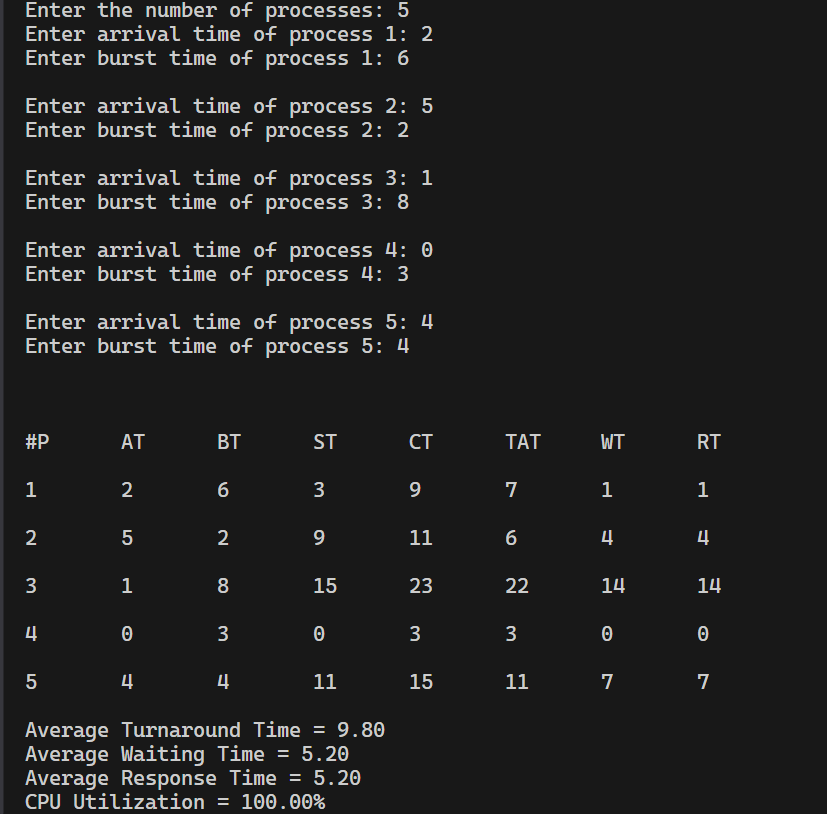
    cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

    cout<<"Average Response Time = "<<avg\_response\_time<<endl;

    cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;

}

**Output:**



**3.SRTF(Shortest Remaining Time First):**

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <string.h>

using namespace std;

struct process {

    int pid;

    int arrival\_time;

    int burst\_time;

    int start\_time;

    int completion\_time;

    int turnaround\_time;

    int waiting\_time;

    int response\_time;

};

int main() {

    int n;

    struct process p[100];

    float avg\_turnaround\_time;

    float avg\_waiting\_time;

    float avg\_response\_time;

    float cpu\_utilisation;

    int total\_turnaround\_time = 0;

    int total\_waiting\_time = 0;

    int total\_response\_time = 0;

    int total\_idle\_time = 0;

    int burst\_remaining[100];

    int is\_completed[100];

    memset(is\_completed,0,sizeof(is\_completed));

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";

    cin>>n;

    for(int i = 0; i < n; i++) {

        cout<<"Enter arrival time of process "<<i+1<<": ";

        cin>>p[i].arrival\_time;

        cout<<"Enter burst time of process "<<i+1<<": ";

        cin>>p[i].burst\_time;

        p[i].pid = i+1;

        burst\_remaining[i] = p[i].burst\_time;

        cout<<endl;

    }

    int current\_time = 0;

    int completed = 0;

    int prev = 0;

    while(completed != n) {

        int idx = -1;

        int mn = INT\_FAST32\_MAX;

        for(int i = 0; i < n; i++) {

            if(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {

                if(burst\_remaining[i] < mn) {

                    mn = burst\_remaining[i];

                    idx = i;

                }

                if(burst\_remaining[i] == mn) {

                    if(p[i].arrival\_time < p[idx].arrival\_time) {

                        mn = burst\_remaining[i];

                        idx = i;

                    }

                }

            }

        }

        if(idx != -1) {

        if(burst\_remaining[idx] == p[idx].burst\_time) {  // Checking whether the process has arrived first   time

                p[idx].start\_time = current\_time;

                total\_idle\_time += p[idx].start\_time - prev;

            }

            burst\_remaining[idx] -= 1;

            current\_time++;

            prev = current\_time;

            if(burst\_remaining[idx] == 0) {

                p[idx].completion\_time = current\_time;

                p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

                p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

                p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;

                total\_turnaround\_time += p[idx].turnaround\_time;

                total\_waiting\_time += p[idx].waiting\_time;

                total\_response\_time += p[idx].response\_time;

                is\_completed[idx] = 1;

                completed++;

            }

        }

        else {

             current\_time++;

        }

    }

    int min\_arrival\_time = 10000000;

    int max\_completion\_time = -1;

    for(int i = 0; i < n; i++) {

        min\_arrival\_time = min(min\_arrival\_time,p[i].arrival\_time);

        max\_completion\_time = max(max\_completion\_time,p[i].completion\_time);

    }

    avg\_turnaround\_time = (float) total\_turnaround\_time / n;

    avg\_waiting\_time = (float) total\_waiting\_time / n;

    avg\_response\_time = (float) total\_response\_time / n;

    cpu\_utilisation = ((max\_completion\_time - total\_idle\_time) / (float) max\_completion\_time )\*100;

    cout<<endl<<endl;

    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {

        cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

    }

    cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

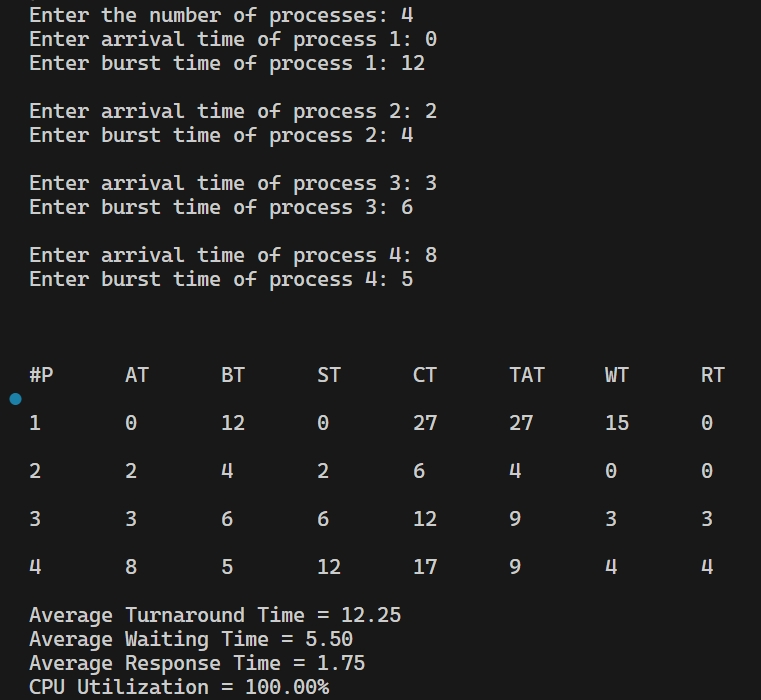
    cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

    cout<<"Average Response Time = "<<avg\_response\_time<<endl;

    cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;

}

**Output:**



**4.Round Robin :**

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <queue>

#include <string.h>

using namespace std;

struct process {

    int pid;

    int arrival\_time;

    int burst\_time;

    int start\_time;

    int completion\_time;

    int turnaround\_time;

    int waiting\_time;

    int response\_time;

};

bool compare1(process p1, process p2)

{

    return p1.arrival\_time < p2.arrival\_time;

}

bool compare2(process p1, process p2)

{

    return p1.pid < p2.pid;

}

int main() {

    int n;

    int tq;

    struct process p[100];

    float avg\_turnaround\_time;

    float avg\_waiting\_time;

    float avg\_response\_time;

    float cpu\_utilisation;

    int total\_turnaround\_time = 0;

    int total\_waiting\_time = 0;

    int total\_response\_time = 0;

    int total\_idle\_time = 0;

    float throughput;

    int burst\_remaining[100];

    int idx;

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";

    cin>>n;

    cout<<"Enter time quantum: ";

    cin>>tq;

    for(int i = 0; i < n; i++) {

        cout<<"Enter arrival time of process "<<i+1<<": ";

        cin>>p[i].arrival\_time;

        cout<<"Enter burst time of process "<<i+1<<": ";

        cin>>p[i].burst\_time;

        burst\_remaining[i] = p[i].burst\_time;

        p[i].pid = i+1;

        cout<<endl;

    }

    sort(p,p+n,compare1);

    queue<int> q;

    int current\_time = 0;

    q.push(0);

    int completed = 0;

    int mark[100];

    memset(mark,0,sizeof(mark));

    mark[0] = 1;

    while(completed != n) {

        idx = q.front();

        q.pop();

        if(burst\_remaining[idx] == p[idx].burst\_time) {

            p[idx].start\_time = max(current\_time,p[idx].arrival\_time);

            total\_idle\_time += p[idx].start\_time - current\_time;

            current\_time = p[idx].start\_time;

        }

        if(burst\_remaining[idx]-tq > 0) {

            burst\_remaining[idx] -= tq;

            current\_time += tq;

        }

        else {

            current\_time += burst\_remaining[idx];

            burst\_remaining[idx] = 0;

            completed++;

            p[idx].completion\_time = current\_time;

            p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

            p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

            p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;

            total\_turnaround\_time += p[idx].turnaround\_time;

            total\_waiting\_time += p[idx].waiting\_time;

            total\_response\_time += p[idx].response\_time;

        }

        for(int i = 1; i < n; i++) {

            if(burst\_remaining[i] > 0 && p[i].arrival\_time <= current\_time && mark[i] == 0) {

                q.push(i);

                mark[i] = 1;

            }

        }

        if(burst\_remaining[idx] > 0) {

            q.push(idx);

        }

        if(q.empty()) {

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

                if(burst\_remaining[i] > 0) {

                    q.push(i);

                    mark[i] = 1;

                    break;

                }

            }

        }

    }

    avg\_turnaround\_time = (float) total\_turnaround\_time / n;

    avg\_waiting\_time = (float) total\_waiting\_time / n;

    avg\_response\_time = (float) total\_response\_time / n;

    cpu\_utilisation = ((p[n-1].completion\_time - total\_idle\_time) / (float) p[n-1].completion\_time)\*100;

    sort(p,p+n,compare2);

    cout<<endl;

    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {

        cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

    }

    cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

    cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

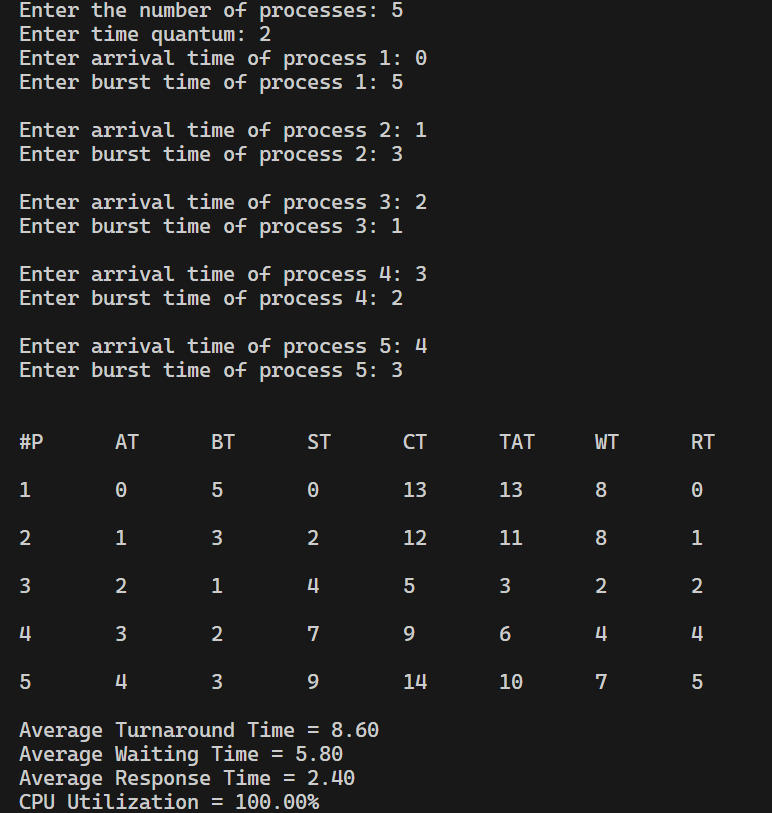
    cout<<"Average Response Time = "<<avg\_response\_time<<endl;

    cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;

}

s

**Output:**



**5. Priority-Non preemptive**

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <string.h>

using namespace std;

struct process {

    int pid;

    int arrival\_time;

    int burst\_time;

    int priority;

    int start\_time;

    int completion\_time;

    int turnaround\_time;

    int waiting\_time;

    int response\_time;

};

int main() {

    int n;

    struct process p[100];

    float avg\_turnaround\_time;

    float avg\_waiting\_time;

    float avg\_response\_time;

    float cpu\_utilisation;

    int total\_turnaround\_time = 0;

    int total\_waiting\_time = 0;

    int total\_response\_time = 0;

    int total\_idle\_time = 0;

    float throughput;

    int is\_completed[100];

    memset(is\_completed,0,sizeof(is\_completed));

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";

    cin>>n;

    for(int i = 0; i < n; i++) {

        cout<<"Enter arrival time of process "<<i+1<<": ";

        cin>>p[i].arrival\_time;

        cout<<"Enter burst time of process "<<i+1<<": ";

        cin>>p[i].burst\_time;

        cout<<"Enter priority of the process "<<i+1<<": ";

        cin>>p[i].priority;

        p[i].pid = i+1;

        cout<<endl;

    }

    int current\_time = 0;

    int completed = 0;

    int prev = 0;

    while(completed != n) {

        int idx = -1;

        int mx = -1;

        for(int i = 0; i < n; i++) {

            if(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {

                if(p[i].priority > mx) {

                    mx = p[i].priority;

                    idx = i;

                }

                if(p[i].priority == mx) {

                    if(p[i].arrival\_time < p[idx].arrival\_time) {

                        mx = p[i].priority;

                        idx = i;

                    }

                }

            }

        }

        if(idx != -1) {

            p[idx].start\_time = current\_time;

            p[idx].completion\_time = p[idx].start\_time + p[idx].burst\_time;

            p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

            p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

            p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;

            total\_turnaround\_time += p[idx].turnaround\_time;

            total\_waiting\_time += p[idx].waiting\_time;

            total\_response\_time += p[idx].response\_time;

            total\_idle\_time += p[idx].start\_time - prev;

            is\_completed[idx] = 1;

            completed++;

            current\_time = p[idx].completion\_time;

            prev = current\_time;

        }

        else {

            current\_time++;

        }

    }

    int min\_arrival\_time = 10000000;

    int max\_completion\_time = -1;

    for(int i = 0; i < n; i++) {

        min\_arrival\_time = min(min\_arrival\_time,p[i].arrival\_time);

        max\_completion\_time = max(max\_completion\_time,p[i].completion\_time);

    }

    avg\_turnaround\_time = (float) total\_turnaround\_time / n;

    avg\_waiting\_time = (float) total\_waiting\_time / n;

    avg\_response\_time = (float) total\_response\_time / n;

    cpu\_utilisation = ((max\_completion\_time - total\_idle\_time) / (float) max\_completion\_time )\*100;

    cout<<endl<<endl;

    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {

        cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].priority<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

    }

    cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

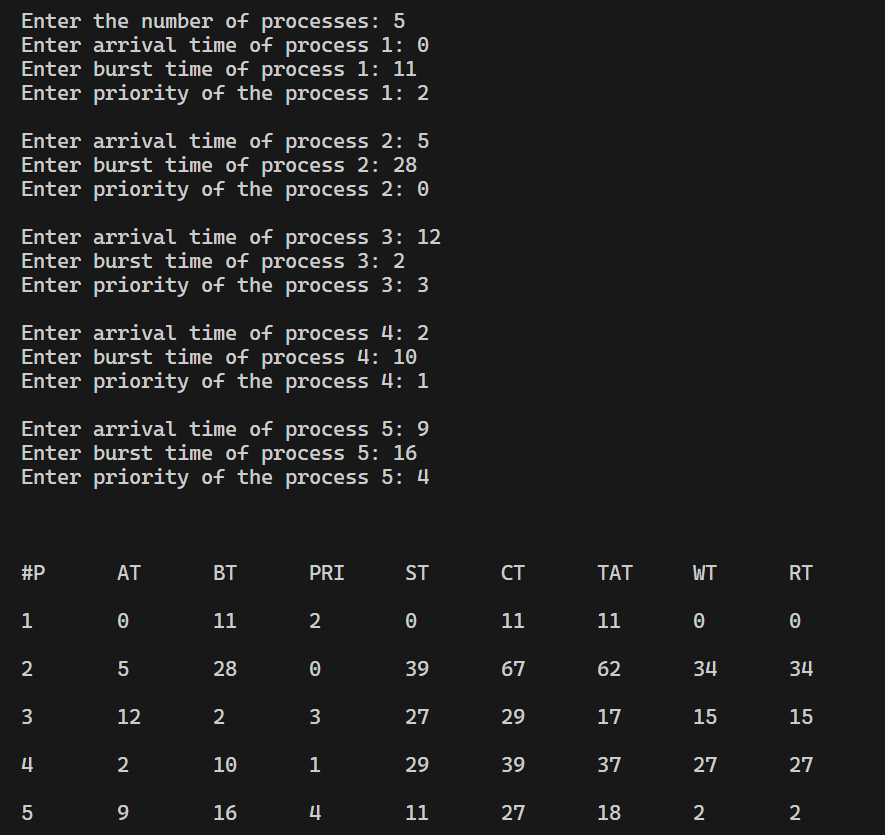
    cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

    cout<<"Average Response Time = "<<avg\_response\_time<<endl;

    cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;

}

**Output:**

****

****

**6.  Priority-Preemptive**

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <string.h>

using namespace std;

struct process {

    int pid;

    int arrival\_time;

    int burst\_time;

    int priority;

    int start\_time;

    int completion\_time;

    int turnaround\_time;

    int waiting\_time;

    int response\_time;

};

int main() {

    int n;

    struct process p[100];

    float avg\_turnaround\_time;

    float avg\_waiting\_time;

    float avg\_response\_time;

    float cpu\_utilisation;

    int total\_turnaround\_time = 0;

    int total\_waiting\_time = 0;

    int total\_response\_time = 0;

    int total\_idle\_time = 0;

    float throughput;

    int burst\_remaining[100];

    int is\_completed[100];

    memset(is\_completed,0,sizeof(is\_completed));

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";

    cin>>n;

    for(int i = 0; i < n; i++) {

        cout<<"Enter arrival time of process "<<i+1<<": ";

        cin>>p[i].arrival\_time;

        cout<<"Enter burst time of process "<<i+1<<": ";

        cin>>p[i].burst\_time;

        cout<<"Enter priority of the process "<<i+1<<": ";

        cin>>p[i].priority;

        p[i].pid = i+1;

        burst\_remaining[i] = p[i].burst\_time;

        cout<<endl;

    }

    int current\_time = 0;

    int completed = 0;

    int prev = 0;

    while(completed != n) {

        int idx = -1;

        int mx = -1;

        for(int i = 0; i < n; i++) {

            if(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {

                if(p[i].priority > mx) {

                    mx = p[i].priority;

                    idx = i;

                }

                if(p[i].priority == mx) {

                    if(p[i].arrival\_time < p[idx].arrival\_time) {

                        mx = p[i].priority;

                        idx = i;

                    }

                }

            }

        }

        if(idx != -1) {

            if(burst\_remaining[idx] == p[idx].burst\_time) {

                p[idx].start\_time = current\_time;

                total\_idle\_time += p[idx].start\_time - prev;

            }

            burst\_remaining[idx] -= 1;

            current\_time++;

            prev = current\_time;

            if(burst\_remaining[idx] == 0) {

                p[idx].completion\_time = current\_time;

                p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

                p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

                p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;

                total\_turnaround\_time += p[idx].turnaround\_time;

                total\_waiting\_time += p[idx].waiting\_time;

                total\_response\_time += p[idx].response\_time;

                is\_completed[idx] = 1;

                completed++;

            }

        }

        else {

             current\_time++;

        }

    }

    int min\_arrival\_time = 10000000;

    int max\_completion\_time = -1;

    for(int i = 0; i < n; i++) {

        min\_arrival\_time = min(min\_arrival\_time,p[i].arrival\_time);

        max\_completion\_time = max(max\_completion\_time,p[i].completion\_time);

    }

    avg\_turnaround\_time = (float) total\_turnaround\_time / n;

    avg\_waiting\_time = (float) total\_waiting\_time / n;

    avg\_response\_time = (float) total\_response\_time / n;

    cpu\_utilisation = ((max\_completion\_time - total\_idle\_time) / (float) max\_completion\_time )\*100;

    cout<<endl<<endl;

    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {

        cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].priority<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

    }

    cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

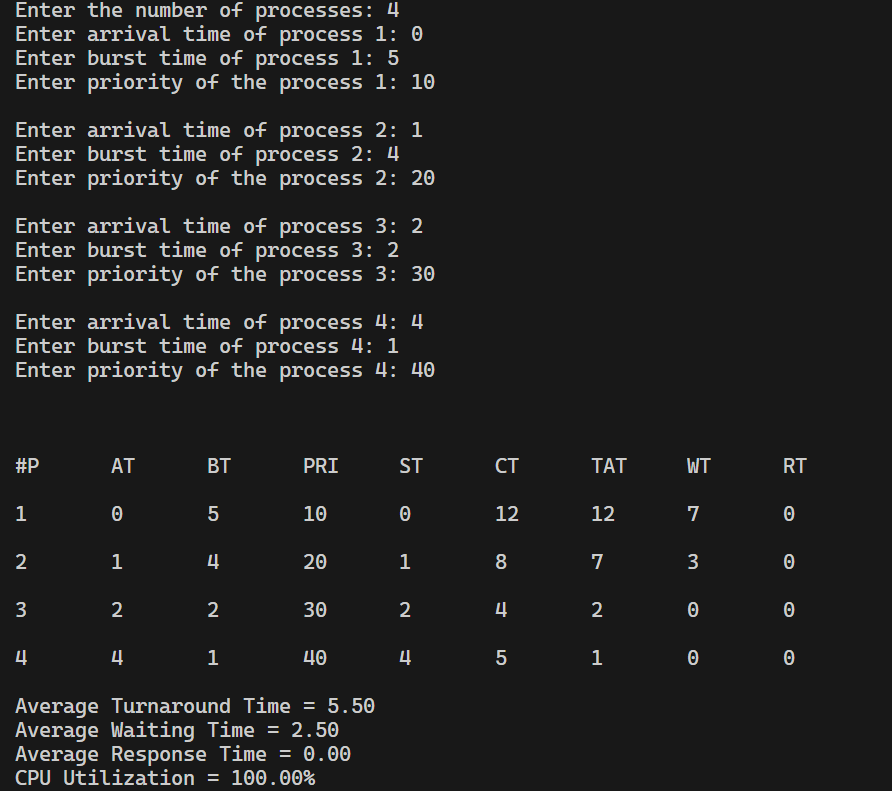
    cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

    cout<<"Average Response Time = "<<avg\_response\_time<<endl;

    cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;

}

**Output:**

****