###### Experiment Number: 05

###### TITLE: Scheduling Algorithms

# First Come First Serve

## Code

*#include* <iomanip>

*#include* <iostream>

*#define* WIDTH 20

using namespace std;

int main()

{

    int n, at[20], bt[20], wt[20], tat[20], i, j;

    cout << "Enter total number of processes(maximum 20):";

    cin >> n;

    cout << "\nEnter Process Burst Time\n";

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

    {

        cout << "P[" << i + 1 << "]:";

        cin >> bt[i];

    }

    wt[0] = 0; *// waiting time for first process is 0*

*// calculating waiting time*

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

    {

        wt[i] = 0;

*for* (j = 0; j < i; j++)

            wt[i] += bt[j];

    }

    cout << left << endl

         << setw(WIDTH) << "Process"

         << setw(WIDTH) << "Burst Time"

         << setw(WIDTH) << "Waiting Time"

         << "Turnaround Time";

    cout << endl

         << setw(4 \* WIDTH) << setfill('-')

         << "-";

*// calculating turnaround time*

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

    {

        tat[i] = bt[i] + wt[i];

        cout << setfill(' ') << left << endl

             << setw(WIDTH) << i + 1

             << setw(WIDTH) << bt[i]

             << setw(WIDTH) << wt[i]

             << tat[i];

    }

    cout << endl

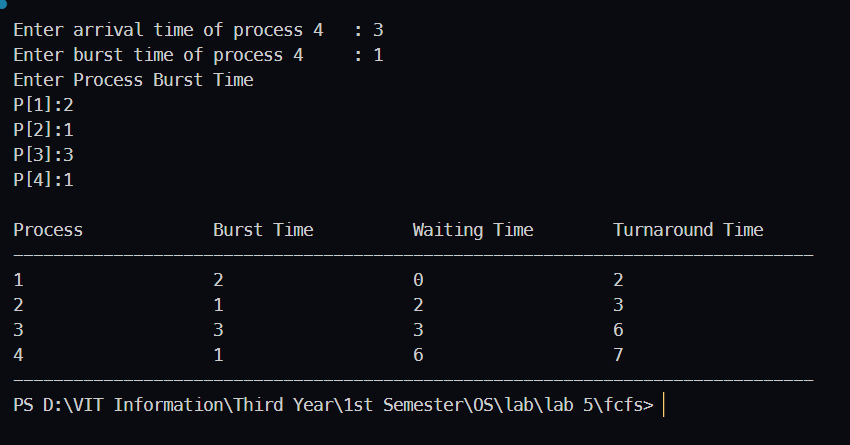
         << setw(4 \* WIDTH) << setfill('-')

         << "-" << endl;

*return* 0;

}

## Output



# Priority (Preemptive)

## Code

*#include* <algorithm>

*#include* <iomanip>

*#include* <iostream>

*#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[20];

    int burst\_remaining[20];

    int is\_completed[20];

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

    cout << setprecision(2) << fixed;

    cout << "Enter the number of processes max[20]: ";

    cin >> n;

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

    {

        cout << "Enter arrival time of process " << i + 1 << setw(5) << ": ";

        cin >> p[i].arrival\_time;

        cout << "Enter burst time of process " << i + 1 << setw(7) << ": ";

        cin >> p[i].burst\_time;

        cout << "Enter priority of the process " << i + 1 << setw(5) << ": ";

        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;

            }

            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;

                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);

    }

    cout << setw(90) << setfill('-')

         << "-"

         << endl;

    cout << left << setfill(' ')

         << setw(10) << "P"

         << setw(10) << "AT"

         << setw(10) << "BT"

         << setw(10) << "PRI"

         << setw(10) << "ST"

         << setw(10) << "CT"

         << setw(10) << "TAT"

         << setw(10) << "WT"

         << setw(10) << "RT"

         << endl;

    cout << setw(90) << setfill('-')

         << "-"

         << endl;

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

    {

        cout << left << setfill(' ')

             << setw(10) << p[i].pid

             << setw(10) << p[i].arrival\_time

             << setw(10) << p[i].burst\_time

             << setw(10) << p[i].priority

             << setw(10) << p[i].start\_time

             << setw(10) << p[i].completion\_time

             << setw(10) << p[i].turnaround\_time

             << setw(10) << p[i].waiting\_time

             << setw(10) << p[i].response\_time

             << endl;

    }

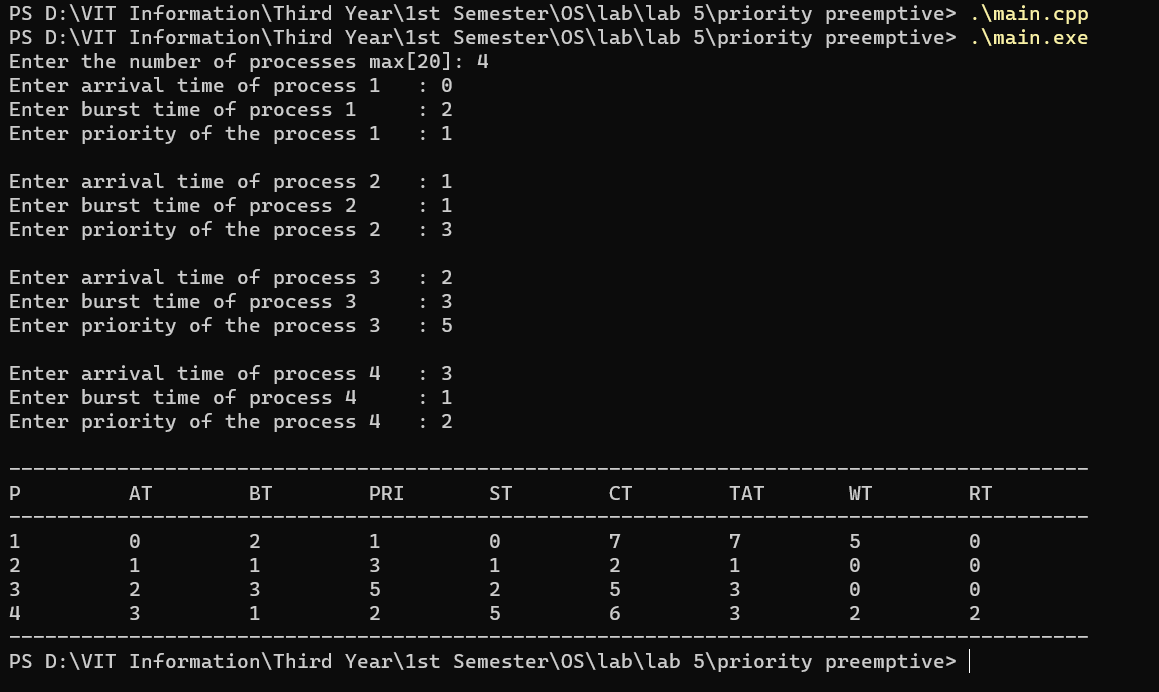
    cout << setw(90) << setfill('-')

         << "-"

         << endl;

}

## Output



# Priority Scheduling (Non-Preemptive)

## Code

*#include* <algorithm>

*#include* <iomanip>

*#include* <iostream>

*#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[20];

    int is\_completed[20];

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

    cout << setprecision(2) << fixed;

    cout << "Enter the number of processes max[20]: ";

    cin >> n;

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

    {

        cout << "Enter arrival time of process " << i + 1 << setw(5) << ": ";

        cin >> p[i].arrival\_time;

        cout << "Enter burst time of process " << i + 1 << setw(7) << ": ";

        cin >> p[i].burst\_time;

        cout << "Enter priority of the process " << i + 1 << setw(5) << ": ";

        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;

            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);

    }

    cout << setw(90) << setfill('-')

         << "-"

         << endl;

    cout << left << setfill(' ')

         << setw(10) << "P"

         << setw(10) << "AT"

         << setw(10) << "BT"

         << setw(10) << "PRI"

         << setw(10) << "ST"

         << setw(10) << "CT"

         << setw(10) << "TAT"

         << setw(10) << "WT"

         << setw(10) << "RT"

         << endl;

    cout << setw(90) << setfill('-')

         << "-"

         << endl;

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

    {

        cout << left << setfill(' ')

             << setw(10) << p[i].pid

             << setw(10) << p[i].arrival\_time

             << setw(10) << p[i].burst\_time

             << setw(10) << p[i].priority

             << setw(10) << p[i].start\_time

             << setw(10) << p[i].completion\_time

             << setw(10) << p[i].turnaround\_time

             << setw(10) << p[i].waiting\_time

             << setw(10) << p[i].response\_time

             << endl;

    }

    cout << setw(90) << setfill('-')

         << "-"

         << endl;

}

## Output

