

Programiz

C Online Compiler

Ae Pr

Amp up your storytelling with All Apps.

Try now

Adobe

```
main.c
1 #include <stdio.h>
2 void findWaitingTime(int processes[], int n, int burst_time[], int wait_time[])
3 {
4     wait_time[0] = 0;
5     for (int i = 1; i < n; i++) {
6         wait_time[i] = burst_time[i - 1] + wait_time[i - 1];
7     }
8     void findTurnAroundTime(int processes[], int n, int burst_time[], int
9         wait_time[], int tat[]){
10    for (int i = 0; i < n; i++) {
11        tat[i] = burst_time[i] + wait_time[i];
12    }
13    void findavgTime(int processes[], int n, int burst_time[]) {
14        int wait_time[n], tat[n];
15        float total_wt = 0, total_tat = 0;
16        findWaitingTime(processes, n, burst_time, wait_time);
17        findTurnAroundTime(processes, n, burst_time, wait_time, tat);
18
19        printf("Processes Burst time Waiting time Turn around time\n");
20        for (int i = 0; i < n; i++) {
21            total_wt += wait_time[i];
22            total_tat += tat[i];
23            printf("%d \t\t %d \t\t %d \t\t %d\n", i+1, burst_time[i], wait_time[i],
24                tat[i]);
25        }
26    }
27}
```

Run

Output

Processes	Burst time	Waiting time	Turn around time
1	21	0	21
2	3	21	24
3	6	24	30
4	2	30	32

Average waiting time = 18.75
Average turn around time = 26.75

==== Code Execution Successful ===

2024/05/06 15:20

main.c

```

11 }
12 }
13 void findavgTime(int processes[], int n, int burst_time[]) {
14 int wait_time[n], tat[n];
15 float total_wt = 0, total_tat = 0;
16 findWaitingTime(processes, n, burst_time, wait_time);
17 findTurnAroundTime(processes, n, burst_time, wait_time, tat);
18
19 printf("Processes Burst time Waiting time Turn around time\n");
20 for (int i = 0; i < n; i++) {
21 total_wt += wait_time[i];
22 total_tat += tat[i];
23 printf("%d \t %d \t %d \t %d\n", i+1, burst_time[i], wait_time[i],
tat[i]); }
24
25 printf("Average waiting time = %.2f\n", total_wt / n);
26 printf("Average turn around time = %.2f\n", total_tat / n);
27 }
28 int main() {
29 int processes[] = {1, 2, 3, 4};
30 int n = sizeof processes / sizeof processes[0];
31 int burst_time[] = {21, 3, 6, 2};
32 findavgTime(processes, n, burst_time);
33 return 0;
34 }
35

```

2024/05/06 15:20

Output

```

/tmp/FVe9uhOGt0.o
Processes Burst time Waiting time Turn around time
1      21        0       21
2      3         21      24
3      6         24      30
4      2         30      32
Average waiting time = 18.75
Average turn around time = 26.75
== Code Execution Successful ==

```

main.c

```
1 #include <stdio.h>
2 void swap(int *xp, int *yp) {
3     int temp = *xp;
4     *xp = *yp;
5     *yp = temp;
6 }
7 void sortProcessByBurst(int n, int burst[], int process[]) {
8     for (int i = 0; i < n-1; i++) {
9         for (int j = 0; j < n-i-1; j++) {
10            if (burst[j] > burst[j+1]) {
11                swap(&burst[j], &burst[j+1]);
12                swap(&process[j], &process[j+1]);
13            }
14        }
15    void calculateTimes(int processes[], int n, int burst_time[]) {
16        int wait_time[n], tat[n], total_wt = 0, total_tat = 0;
17
18        // Sort processes by burst time
19        sortProcessByBurst(n, burst_time, processes);
20
21        // Calculate waiting times and turn-around times
22        wait_time[0] = 0;
23        tat[0] = burst_time[0];
24
25        for (int i = 1; i < n; i++) {
26            wait_time[i] = wait_time[i-1] + burst_time[i-1];
```



Run

Output

```
/tmp/BHVqW2vD3b.o
Processes Burst time Waiting time Turn around time
1      3      0      3
2      6      3      9
3      7      9      16
4      8      16      24
Average waiting time = 7.00
Average turn around time = 13.00
```

```
== Code Execution Successful ==
```

2024/05/06 15:24

Programiz
C Online Compiler

Premium Coding
Courses by Programiz

Run

Output

```
main.c
24
25- for (int i = 1; i < n; i++) {
26     wait_time[i] = wait_time[i-1] + burst_time[i-1];
27     tat[i] = wait_time[i] + burst_time[i];
28 }
29
30 printf("Processes Burst time Waiting time Turn around time\n");
31
32- for (int i = 0; i < n; i++) {
33     total_wt += wait_time[i];
34     total_tat += tat[i];
35     printf("%d \t %d \t %d \t %d \n", i+1, burst_time[i], wait_time[i],
36             tat[i]);
37
38     printf("Average waiting time = %.2f\n", (float)total_wt / (float)n);
39     printf("Average turn around time = %.2f\n", (float)total_tat / (float)n);
40
41     int main() {
42         int processes[] = {1, 2, 3, 4};
43         int n = sizeof processes / sizeof processes[0];
44         int burst_time[] = {6, 8, 7, 3};
45
46         calculateTimes(processes, n, burst_time);
47     }
48 }
```

2024/05/06 15:24

Processes	Burst time	Waiting time	Turn around time
1	3	0	3
2	6	3	9
3	7	9	16
4	8	16	24

Average waiting time = 7.00
Average turn around time = 13.00

== Code Execution Successful ==

BSE midcap 1019

Online Compiler

Explore Adobe Creative Cloud All Apps, the ultimate toolkit for design, photo editing, and more.

Try for free

main.c

```
25
26
27 void findTurnAroundTime(Process processes[], int n, int waitTime[], int
    turnAroundTime[]) { for (int i = 0; i < n; i++) {
28     turnAroundTime[i] = processes[i].burstTime + waitTime[i];
29 }
30 }
31
32 void findAvgTime(Process processes[], int n) {
33     int waitTime[n], turnAroundTime[n], total_wt = 0, total_tat = 0;
34
35     sortProcessesByPriority(processes, n);
36     findWaitingTime(processes, n, waitTime);
37     findTurnAroundTime(processes, n, waitTime, turnAroundTime);
38
39     printf("Processes Burst time Priority Waiting time Turn around time\n");
40
41     for (int i = 0; i < n; i++) {
42         total_wt += waitTime[i];
43         total_tat += turnAroundTime[i];
44         printf("%d %d %d %d %d %d\n", processes[i].id, processes[i]
            .burstTime, processes[i].priority, waitTime[i], turnAroundTime[i]);
45     }
46
47     printf("Average waiting time = %.2f\n", (float)total_wt / n);
48     printf("Average turn around time = %.2f\n", (float)total_tat / n);
49 }
```

Output

```
^ /tmp/X00nxopB9W.o
Processes Burst time Priority Waiting time Turn around time
2      5      0      0      5
3      8      1      5      13
1     10      2     13      23
Average waiting time = 6.00
Average turn around time = 13.67
==== Code Execution Successful ===
```

40°C
Mostly sunny

ENG IN 06-05-2023

Programiz
C Online Compiler

**Push the boundaries
with All Apps.**

Try now **Adobe**

main.c

```
34
35 sortProcessesByPriority(processes, n);
36 findWaitingTime(processes, n, waitTime);
37 findTurnAroundTime(processes, n, waitTime, turnAroundTime);
38
39 printf("Processes Burst time Priority Waiting time Turn around time\n");
40
41 for (int i = 0; i < n; i++) {
42 total_wt += waitTime[i];
43 total_tat += turnAroundTime[i];
44 printf("%d \t %d \t %d \t %d \t %d\n", processes[i].id, processes[i]
        .burstTime, processes[i].priority, waitTime[i], turnAroundTime[i]);
45 }
46
47 printf("Average waiting time = %.2f\n", (float)total_wt / n);
48 printf("Average turn around time = %.2f\n", (float)total_tat / n);
49 }
50
51 int main() {
52 Process processes[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}};
53 int n = sizeof(processes) / sizeof(processes[0]);
54
55 findAvgTime(processes, n);
56 return 0;
57 }
58
```

Output

Processes	Burst time	Priority	Waiting time	Turn around time
2	5	0	0	5
3	8	1	5	13
1	10		2	13
				23

Average waiting time = 6.00
Average turn around time = 13.67

== Code Execution Successful ==

40°C Mostly sunny 2024/05/06 15:28 ENG IN 06-05-2

Programiz
C Online Compiler

Programiz PRO

Premium Coding
Courses by Programiz

[Learn More](#)

main.c

```
1 #include <stdio.h>
2 * typedef struct {
3     int id;
4     int burstTime;
5 } Process;
6
7 * void findWaitingTime(Process processes[], int n, int quantum) {
8     int rem_bt[n];
9     for (int i = 0; i < n; i++)
10    rem_bt[i] = processes[i].burstTime;
11
12    int t = 0; // Current time
13
14    while (1) {
15        int done = 1;
16
17        for (int i = 0; i < n; i++) {
18            if (rem_bt[i] > 0) {
19                done = 0; // There is a pending process
20
21
22            if (rem_bt[i] > quantum) {
23                t += quantum;
24
25                rem_bt[i] -= quantum;
26            } else {
```

Run

Output

Processes	Burst time	Waiting time	Turn around time
1	60	60	120
2	14	14	28
3	20	20	40

Average waiting time = 31.33
Average turn around time = 62.67

==== Code Execution Successful ===

2024/05/06 15:31

Programiz

Online Compiler

Programiz PRO

Premium Coding Courses by Programiz

Learn More

main.c

```
25    rem_bt[i] -= quantum;
26 } else {
27     t += rem_bt[i];
28     processes[i].burstTime = t; // Store waiting time
29     rem_bt[i] = 0;
30 }
31 }
32 }
33 if (done == 1)
34 break;
35 }
36 }
37 void findTurnAroundTime(Process processes[], int n) {
38     for (int i = 0; i < n; i++) {
39
40         processes[i].burstTime += processes[i].burstTime;
41     }
42 void findAvgTime(Process processes[], int n, int quantum) {
43     findWaitingTime(processes, n, quantum);
44     findTurnAroundTime(processes, n);
45
46     int total_wt = 0, total_tat = 0;
47     printf("Processes Burst time Waiting time Turn around time\n");
48
49     for (int i = 0; i < n; i++) {
50         total_wt += processes[i].burstTime;
51
52         processes[i].tat = processes[i].burstTime + total_wt;
53
54         total_tat += processes[i].tat;
55     }
56
57     float avg_wt = (float)total_wt / n;
58     float avg_tat = (float)total_tat / n;
59
60     printf("Average waiting time = %f\n", avg_wt);
61     printf("Average turn around time = %f\n", avg_tat);
62 }
```

Output

```
^ /tmp/sLRvTVNh2G.o
Processes Burst time Waiting time Turn around time
1      60        60       120
2      14        14       28
3      20        20       40
Average waiting time = 31.33
Average turn around time = 62.67

==== Code Execution Successful ===
```

38°C
Mostly sunny

2024/05/06 15:31

ENGLISH IN

Programiz

C Online Compiler

Program

```
main.c
41 }
42 void findAvgTime(Process processes[], int n, int quantum) {
43     findWaitingTime(processes, n, quantum);
44     findTurnAroundTime(processes, n);
45
46     int total_wt = 0, total_tat = 0;
47     printf("Processes Burst time Waiting time Turn around time\n");
48
49     for (int i = 0; i < n; i++) {
50         total_wt += processes[i].burstTime;
51         total_tat += processes[i].burstTime * processes[i].burstTime;
52         printf("%d %d %d %d\n", i+1, processes[i].burstTime,
53               processes[i].burstTime, processes[i].burstTime + processes[i].burstTime);
54
55     printf("Average waiting time = %.2f\n", (float)total_wt / n);
56     printf("Average turn around time = %.2f\n", (float)total_tat / n);
57 }
58 Process processes[] = {{1, 24}, {2, 3}, {3, 3}};
59 int n = sizeof(processes) / sizeof(processes[0]);
60 int quantum = 4;
61 findAvgTime(processes, n, quantum);
62 return 0;
63 }
64
```

Run Output

```
* /tmp/sLRvTVNh2G.o
Processes Burst time Waiting time Turn around time
1      60      60      120
2      14      14      28
3      20      20      40
Average waiting time = 31.33
Average turn around time = 62.67

*** Code Execution Successful ***
```

38°C Mostly sunny

ENG IN 06-05-2018

The screenshot shows a web-based C compiler interface from Programiz. The code in the editor is for a process scheduling algorithm. It defines a `Process` struct with fields for id, burstTime, priority, arrivalTime, waitingTime, and turnaroundTime. It includes utility functions for comparing processes based on arrival time, burst time, and priority. The output window shows the results of running the program, which calculates waiting and turnaround times for processes under FCFS and SJF scheduling policies.

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 typedef struct {
4     int id;
5     int burstTime;
6     int priority;
7     int arrivalTime;
8     int waitingTime;
9     int turnaroundTime;
10 } Process;
11
12 // Utility function to sort processes by arrival time
13 int compareArrival(const void *a, const void *b) {
14     Process *p1 = (Process *)a;
15     Process *p2 = (Process *)b;
16     return p1->arrivalTime - p2->arrivalTime;
17 }
18 // Utility function to sort processes by burst time for SJF
19 int compareBurstTime(const void *a, const void *b) {
20     Process *p1 = (Process *)a;
21     Process *p2 = (Process *)b;
22     return p1->burstTime - p2->burstTime;
23 }
24
25 // Utility function to sort processes by priority for Priority Scheduling
26 int comparePriority(const void *a, const void *b) {
```

Output

```
/tmp/kF3MsS7LtH.o
FCFS:
Average Waiting Time = 9.50
Average Turnaround Time = 15.50
SJF:
```

The screenshot shows a computer monitor displaying the Programiz Online Compiler. The interface has a header with the Programiz logo and "C Online Compiler". Below the header is a banner for "Premium Coding Courses by Programiz" with a "Learn More" button. The main area shows a code editor with a file named "main.c" containing C code for process scheduling. The code includes functions for comparing priority, calculating waiting and turnaround times, and implementing the FCFS algorithm. The output window shows the results for the FCFS algorithm: Average Waiting Time = 9.50 and Average Turnaround Time = 15.50. The system tray at the bottom shows various icons and the date/time 2024/05/06 15:43.

```
main.c
25 // Utility function to sort processes by priority for Priority Scheduling
26 int comparePriority(const void *a, const void *b) {
27     Process *p1 = (Process *)a;
28     Process *p2 = (Process *)b;
29     return p1->priority - p2->priority;
30 }
31
32 // Function to calculate waiting time and turnaround time
33 void calculateMetrics(Process p[], int n) {
34     int total_wt = 0, total_tat = 0;
35     for (int i = 0; i < n; i++) {
36         p[i].turnaroundTime = p[i].burstTime + p[i].waitingTime;
37         total_wt += p[i].waitingTime;
38         total_tat += p[i].turnaroundTime;
39     }
40
41     printf("Average Waiting Time = %.2f\n", (float)total_wt / n);
42     printf("Average Turnaround Time = %.2f\n", (float)total_tat / n);
43 }
44 // First Come First Serve Algorithm
45 void FCFS(Process p[], int n) {
46     printf("FCFS:\n");
47     qsort(p, n, sizeof(Process), compareArrival);
48
49     int currentTime = 0;
50     for (int i = 0; i < n; i++) {
```

BSE smicap
-1.02%

Output

```
/tmp/kF3MsS7LTH.o
FCFS:
Average Waiting Time = 9.50
Average Turnaround Time = 15.50
SJF:
```

Programiz
C Online Compiler

Programiz PRO Premium Coding Courses by Programiz Learn More

main.c

```
50+ for (int i = 0; i < n; i++) {  
51 if (currentTime < p[i].arrivalTime)  
52 currentTime = p[i].arrivalTime;  
53 p[i].waitingTime = currentTime - p[i].arrivalTime; currentTime += p[i]  
    .burstTime;  
54 }  
55  
56 calculateMetrics(p, n);  
57 }  
58  
59 // Shortest Job First Algorithm (non-preemptive)  
60 void SJF(Process p[], int n) {  
61 printf("SJF:\n");  
62 qsort(p, n, sizeof(Process), compareArrival);  
63  
64 int completed = 0, currentTime = 0, minIndex = -1;  
65 int remaining = n;  
66 while (completed < n) {  
67 // Find process with shortest burst time among the arrived processes  
68 int minBurst = 999999;  
69 for (int i = 0; i < n; i++) {  
70 if (p[i].arrivalTime <= currentTime && p[i].burstTime < minBurst && p[i]  
    .waitingTime == 0) { minBurst = p[i].burstTime;  
71 minIndex = i;  
72 }  
73 }
```

Output

```
^ /tmp/kF3MsS7Lth.o  
FCFS:  
Average Waiting Time = 9.50  
Average Turnaround Time = 15.50  
SJF:
```

38°C Mostly sunny 2024/05/06 15:43 ENG IN 15:47 06-05-2024

The screenshot shows a web-based C compiler interface from Programiz. The code editor contains a C program named `main.c` which implements a priority scheduling algorithm. The output window displays the results for FCFS and SJF scheduling methods.

```
main.c
73 }
74
75 if (minIndex == -1) {
76 currentTime++;
77 } else {
78 p[minIndex].waitingTime = currentTime - p[minIndex].arrivalTime;
79 currentTime += p[minIndex].burstTime;
80 completed++;
81 p[minIndex].waitingTime = (p[minIndex].waitingTime < 0) ? 0 : p[minIndex]
82 .waitingTime; minIndex = -1;
83 }
84
85 calculateMetrics(p, n);
86 }
87
88 // Priority Scheduling Algorithm (non-preemptive)
89
90 void PriorityScheduling(Process p[], int n) {
91     printf("Priority:\n");
92     qsort(p, n, sizeof(Process), comparePriority);
93
94     int currentTime = 0;
95     for (int i = 0; i < n; i++) {
96         if (currentTime < p[i].arrivalTime)
97             currentTime = p[i].arrivalTime;
98         for (int j = 0; j < p[i].burstTime; j++)
99             currentTime++;
100    }
101 }
```

Output:

```
/tmp/kF3Ms57Lth.o
FCFS:
Average Waiting Time = 9.50
Average Turnaround Time = 15.50
SJF:
```

2024/05/06 15:43

Online C++ Compiler | Online C Compiler | programiz.com/c-programming/online-compiler/

Programiz
C Online Compiler

main.c

```
96 if (currentTime < p[i].arrivalTime)
97 currentTime = p[i].arrivalTime;
98 p[i].waitingTime = currentTime - p[i].arrivalTime;
99 currentTime += p[i].burstTime;
100 }
101
102 calculateMetrics(p, n);
103 }
104
105 // Round Robin Algorithm
106 void RoundRobin(Process p[], int n, int quantum) {
107     printf("Round Robin:\n");
108     qsort(p, n, sizeof(Process), compareArrival);
109
110     int rem_bt[n];
111     for (int i = 0; i < n; i++) {
112         rem_bt[i] = p[i].burstTime;
113     }
114
115     int t = 0;
116     while (1) {
117         int done = 1;
118         for (int i = 0; i < n; i++) {
119             if (rem_bt[i] > 0) {
120                 done = 0;
121                 if (rem_bt[i] > quantum) {
```

Premium Coding Courses by Programiz [Learn More](#)

Run Output

/tmp/kF3MsS7LtH.o
FCFS:
Average Waiting Time = 9.50
Average Turnaround Time = 15.50
SJF:

38°C Mostly sunny 2024/05/06 15:44 ENG IN

The screenshot shows a C Online Compiler interface on a computer screen. The main window displays a C program named 'main.c' with line numbers 122 through 145. The code implements a First-Come-First-Served (FCFS) scheduling algorithm for processes. The output window shows the results of the execution, including the generated executable file path, FCFS metrics, and SJF metrics.

```
main.c
122
123     t += quantum;
124     rem_bt[i] -= quantum;
125 } else {
126     t += rem_bt[i];
127     p[i].waitingTime = t - p[i].burstTime - p[i].arrivalTime;  rem_bt[i] = 0;
128 }
129 }
130 }
131
132 if (done == 1)
133 break;
134
135
136 }
137 calculateMetrics(p, n);
138 }
139
140 int main() {
141     Process processes[] = {{1, 6, 2, 1}, {2, 8, 3, 1}, {3, 7, 1, 2}, {4, 3, 4,
142     3}};
143     int n = sizeof(processes) / sizeof(processes[0]);
144     int quantum = 4;
145     FCFS(processes, n);
146 }
```

Output

```
/tmp/kF3MsS7Lth.o
FCFS:
Average Waiting Time = 9.50
Average Turnaround Time = 15.50
SJF:
```

Programiz
Online Compiler

See where creativity can take you.
Explore Adobe Creative Cloud All Apps, the ultimate toolkit for design, photo editing, and more.

Try for free

main.c

```
1 #include <stdio.h>
2 typedef struct {
3     int id;
4     int burstTime;
5     int priority;
6 } Process;
7
8 void sortProcessesByPriority(Process processes[], int n) {
9     for (int i = 0; i < n; i++) {
10        for (int j = 0; j < n - i - 1; j++) {
11            if (processes[j].priority > processes[j + 1].priority) {
12                Process temp = processes[j];
13                processes[j] = processes[j + 1];
14                processes[j + 1] = temp;
15            }
16        }
17    }
18}
19 void findWaitingTime(Process processes[], int n, int waitTime[]) {
20    waitTime[0] = 0;
21
22    for (int i = 1; i < n; i++) {
23        waitTime[i] = processes[i - 1].burstTime + waitTime[i - 1];
24    }
25}
26
```

Run

Output

```
/tmp/X00nxopB9W.o
Processes Burst time Priority Waiting time Turn around time
2      5       0       0       5
3      8       1       5      13      23
1     10      2      13      23
Average waiting time = 6.00
Average turn around time = 13.67

== Code Execution Successful ==
```

2024/05/06 15:27

The screenshot shows a C Online Compiler interface on Programiz. The code editor displays 'main.c' with the following content:

```
main.c
127     p[i].waitingTime = t - p[i].burstTime - p[i].arrivalTime; rem_bt[i] = 0;
128 }
129 }
130 }
131
132 if (done == 1)
133 break;
134
135
136 }
137 calculateMetrics(p, n);
138 }
139
140 int main() {
141     Process processes[] = {{1, 6, 2, 1}, {2, 8, 3, 1}, {3, 7, 1, 2}, {4, 3, 4,
142     3}};
143     int n = sizeof(processes) / sizeof(processes[0]);
144     int quantum = 4;
145
146     FCFS(processes, n);
147     SJF(processes, n);
148     PriorityScheduling(processes, n);
149     RoundRobin(processes, n, quantum);
150     return 0;
151 }
```

The output window shows the results of the execution:

```
Output
/tmp/kF3MsS7Lth.o
FCFS:
Average Waiting Time = 9.50
Average Turnaround Time = 15.50
SJF:
```