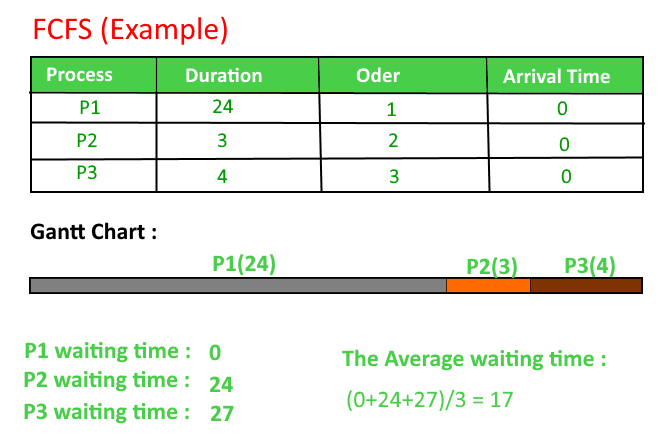
**Program for FCFS CPU Scheduling | Set 1**

Given n processes with their burst times, the task is to find average waiting time and average turn around time using FCFS scheduling algorithm.   
First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.   
In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.   
Here we are considering that arrival time for all processes is 0.

**How to compute below times in Round Robin using a program?**

1. Completion Time: Time at which process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
3. Waiting Time(W.T): Time Difference between turn around time and burst time.   
   Waiting Time = Turn Around Time – Burst Time

***In this post, we have assumed arrival times as 0, so turn around and completion times are same.***



**Implementation:**

1- Input the processes along with their burst time (bt).

2- Find waiting time (wt) for all processes.

3- As first process that comes need not to wait so

waiting time for process 1 will be 0 i.e. wt[0] = 0.

4- Find **waiting time** for all other processes i.e. for

process i ->

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

5- Find **turnaround time** = waiting\_time + burst\_time

for all processes.

6- Find **average waiting time** =

total\_waiting\_time / no\_of\_processes.

7- Similarly, find **average turnaround time** =

total\_turn\_around\_time / no\_of\_processes.

|  |
| --- |
| // Java program for implementation of FCFS  // scheduling    **import** java.text.ParseException;    **class** GFG {        // Function to find the waiting time for all      // processes  **static** **void** findWaitingTime(**int** processes[], **int** n,  **int** bt[], **int** wt[]) {          // waiting time for first process is 0          wt[0] = 0;            // calculating waiting time  **for** (**int** i = 1; i < n; i++) {              wt[i] = bt[i - 1] + wt[i - 1];          }      }        // Function to calculate turn around time  **static** **void** findTurnAroundTime(**int** processes[], **int** n,  **int** bt[], **int** wt[], **int** tat[]) {          // calculating turnaround time by adding          // bt[i] + wt[i]  **for** (**int** i = 0; i < n; i++) {              tat[i] = bt[i] + wt[i];          }      }        //Function to calculate average time  **static** **void** findavgTime(**int** processes[], **int** n, **int** bt[]) {  **int** wt[] = **new** **int**[n], tat[] = **new** **int**[n];  **int** total\_wt = 0, total\_tat = 0;            //Function to find waiting time of all processes          findWaitingTime(processes, n, bt, wt);            //Function to find turn around time for all processes          findTurnAroundTime(processes, n, bt, wt, tat);            //Display processes along with all details          System.out.printf("Processes Burst time Waiting"                         +" time Turn around time\n");            // Calculate total waiting time and total turn          // around time  **for** (**int** i = 0; i < n; i++) {              total\_wt = total\_wt + wt[i];              total\_tat = total\_tat + tat[i];              System.out.printf(" %d ", (i + 1));              System.out.printf("     %d ", bt[i]);              System.out.printf("     %d", wt[i]);              System.out.printf("     %d\n", tat[i]);          }  **float** s = (**float**)total\_wt /(**float**) n;  **int** t = total\_tat / n;          System.out.printf("Average waiting time = %f", s);          System.out.printf("\n");          System.out.printf("Average turn around time = %d ", t);      }        // Driver code  **public** **static** **void** main(String[] args) **throws** ParseException {          //process id's  **int** processes[] = {1, 2, 3};  **int** n = processes.length;            //Burst time of all processes  **int** burst\_time[] = {10, 5, 8};            findavgTime(processes, n, burst\_time);        }  }  // This code is contributed by 29ajaykumar |

**Output:**

Processes Burst time Waiting time Turn around time

1 10 0 10

2 5 10 15

3 8 15 23

Average waiting time = 8.33333

Average turn around time = 16

|  |
| --- |
| #include <bits/stdc++.h>  **using** **namespace** std;    **class** Process {  **private**:  **int** at;  **int** bt;  **int** ct;  **int** tat;  **int** wt;  **int** pid;    **public**:  **int**& operator[](string var)      {  **if** (var == "at")  **return** at;  **if** (var == "bt")  **return** bt;  **if** (var == "ct")  **return** ct;  **if** (var == "tat")  **return** tat;  **if** (var == "wt")  **return** wt;  **return** pid;      }    **void** update\_after\_ct()      {          tat = ct - at;          wt = tat - bt;      }    **void** display()      {  **printf**("%d\t%d\t%d\t%d\t%d\t%d\n", pid, at, bt, ct,                 tat, wt);      }  };    **float** average(vector<Process> P, string var)  {  **int** total = 0;  **for** (**auto** temp : P) {          total += temp[var];      }  **return** (**float**)total / P.size();  }    **int** main()  {      /\*      Input description.      First line contains an integer n      the next n lines contains 2 space separated integers      containing values for arrival time and burst time for      example:      2      0 3      1 2      \*/  **int** n;      cin >> n;  **int** counter = 0;      vector<Process> P(n);  **for** (Process& temp : P) {          temp["id"] = counter++;          cin >> temp["at"] >> temp["bt"];      }      sort(P.begin(), P.end(),           [](Process first, Process second) {  **return** first["at"] < second["at"];           });  **printf**("pid\tat\tbt\tct\ttat\twt\n");      P[0]["ct"] = P[0]["at"] + P[0]["bt"];      P[0].update\_after\_ct();      P[0].display();  **for** (**int** i = 1; i < P.size(); i++) {  **if** (P[i]["at"] < P[i - 1]["ct"]) {              P[i]["ct"] = P[i - 1]["ct"] + P[i]["bt"];          }  **else** {  **printf**("curr['at'] : %d, prev['ct'] : %d\n",                     P[i]["at"], P[i - 1]["ct"]);              P[i]["ct"] = P[i]["at"] + P[i]["bt"];          }          P[i].update\_after\_ct();          P[i].display();      }    **printf**("Average waiting time : %f\n", average(P, "wt"));  **return** 0;  } |

**Important Points:** 

1. Non-preemptive
2. Average Waiting Time is not optimal
3. Cannot utilize resources in parallel : Results in Convoy effect (Consider a situation when many IO bound processes are there and one CPU bound process. The IO bound processes have to wait for CPU bound process when CPU bound process acquires CPU. The IO bound process could have better taken CPU for some time, then used IO devices).