```
/* 1) Consider the following jobs submitted to a system:
        Process  Arrival time(ms)  CPU burst time(ms)   Priority
        Print                  0               7           3
        email                  2               3           2
        File transfer          2               8           1
        Web service            3               4           4
  a) Implement a scheduling algorithm that schedules all processes in FCFS for a fixed
     quantum of 4ms and switches till all processes finish their bursts.*/


//   FIFO queues processes in the order that they arrive in the ready queue.
//   The process that comes first will be executed first and next process starts only after
//     the previous gets fully executed.

//   But problem to be solved is FCFS AND each process gets fixed quantum

//   Hence the implementation should be CPU scheduling algorithm where each process is
//     assigned a fixed time slot in a cyclic way.
//   Switch occurs , a preemptive scheduling as processes are assigned CPU only for a fixed
//     slice of time at most - which is also known as Round Robin Scheduling

//  Completion Time  : Time at which process completes its execution

//  Turn Around Time : Time Difference between completion time and arrival time
//  Turn Around Time = Completion Time – Arrival Time

//  Waiting Time : Time Difference between turn around time and burst time.
//  Waiting Time = Turn Around Time – Burst Time

//  Limitations of program
//  Program works only when input is in ascending order with respect to arrival time AND any
//  new process is arriving but before all earlier processes complete

#include <stdio.h>
int main()
 {
  int n = 4 , remain = n ; // n is number of and remain is remaining process
  int processNo, elapsedTime, flag = 0, timeQuantum = 4; // time quantum = 4ms
  int totalWaitTime = 0, totalTurnAroundTime = 0;
  int arrivalTime[10]   = { 0, 2, 2, 3 }; // arrival time, array is zero indexed
  int burstTime[10]     = { 7, 3, 8, 4 }; // burst time
  int remainingTime[10] = { 7, 3, 8, 4 }; // reamining time

  printf("\n Process | Turnaround time | Waiting time\n");

  for( elapsedTime=0, processNo=0; remain!=0; ) // Process number - zero indexed
   {
    if( remainingTime[processNo] > 0 ) // If process not completed yet
     {
      if( remainingTime[processNo] <= timeQuantum )
       { // if remaining time of process is inbetween 0 and time quantum
         elapsedTime += remainingTime[processNo]; // add execution time to elapsed time
         remainingTime[processNo]=0; // Process has completed , remaining time = 0
         flag=1;   // Change state as process completed execution
         remain--; // Decrement remaining processes count
       }
      else // Remaining time is greater than time quantum for processNo
       {  // Process runs for time quantum and its remaining time decreases by time quantum
         remainingTime[processNo] -= timeQuantum; // remaining time - time quantum
         elapsedTime += timeQuantum;          // add execution time to elapsed time
       }
     }

    if( remainingTime[processNo]==0 && flag==1 ) //If any process has completed then print
     { //  Process | Turnaround time | Waiting time
      printf("  p[%d]\t |\t %d\t   |\t%d\n", processNo,
            elapsedTime - arrivalTime[processNo],
            elapsedTime - arrivalTime[processNo] - burstTime[processNo] );
      // Now update total waitingTime and turnAroundTime of the completed processes so far
      totalTurnAroundTime += elapsedTime - arrivalTime[processNo];
      totalWaitTime += elapsedTime - arrivalTime[processNo] - burstTime[processNo];
      flag=0; // Reset flag, can be used by process which will complete next
     }

    if( processNo == n-1 ) // If all processes have completed one round of execution
     processNo=0;          // then reinitialize index, zero indexed
    else if( arrivalTime[processNo+1] <= elapsedTime) // If next process has arrived
     processNo++;          // then update index to access next process
    else
     processNo=0;
   }

  printf("\n Average turnaround time = %f\n", totalTurnAroundTime * 1.0 / n);
  printf("\n Average waiting time    = %f", totalWaitTime * 1.0 / n);

  return 0;
 }
/*
Output :

 Process | Turnaround time | Waiting time
  p[1]      |      5      |   2
  p[3]      |      12     |   8
  p[0]      |      18     |   11
  p[2]      |      20     |   12

 Average turnaround time = 13.750000
 Average waiting time    = 8.250000
*/
```