



ASSIGNMENT 3



ID	NAME
18010371	الاء احمد حسن احمد عبد المجيد
18010652	ردينه محمد حسن عبد العال
18010307	اسراء حسن عبد الشهيد عبد الله
18010440	ايه حسام صلاح الدين بكر مصطفى
15010473	اسماء جمال عبد الحليم مبروك ناجى

FCFS Code :

```
1 #include <stdio.h>
2 #include <windows.h>
3 #include <stdlib.h>
4 #include <ctype.h>
5 #define _CRT_SECURE_NO_WARNINGS
6 #include <inttypes.h>
7 #include <iostream>
8 #include <stdio.h>
9 #include <list>
10 #include <vector>
11 #include <bits/stdc++.h>
12 #include <string.h>
13 using namespace std;
14
15
16 #define INITIALIZE 0
17 #define MAX 100
18 #define DEBUG 1
19 #define EXIT -1
20
21 #define onlyONE 1
22 #define RR 1
23 #define FCFS 0
24
25 // ----- FCFS Structs process nodes data types -----
-----
26 struct processNode
27 {
28     unsigned int id;
29     unsigned int cpu;
30     unsigned int io;
31     unsigned int arrivalTime;
32     unsigned int turnaroundTIME;
33
34     //to sorting according to the arrival time and processNode id
35     bool operator <(const struct processNode & processNodeObj) const
36     {
37
38         if (arrivalTime < processNodeObj.arrivalTime)
39         {
40             return true;
41         }
42         else if (arrivalTime == processNodeObj.arrivalTime && cpu!=0 &&
processNodeObj.cpu!=0)
43         {
44             return (id < processNodeObj.id);
45         }
46         else if (arrivalTime == processNodeObj.arrivalTime && cpu!=0 &&
processNodeObj.cpu==0)
47         {
48             return true;
49         }
50         else
51         {
52             return false;
53         }
54     }
55 };
56
57 typedef struct processNode Process;
58
59 vector<Process> processARR(MAX); //same as an array of process nodes .. as a
dynamic array.. stores the elements at memory locations which are next to each other
60 list<Process> readyLIST,readyQueue,queueCopy, blockLIST; // same as doubly linked lists ..
stores the
elements at memory locations which are not next to each other with head and tail pointers
61
62
```

```

63 //----- main driver
64 //-----
64 int main()
65 {
66
67 // ----- data types definitions and declarations -----
68
69 int i = INITIALIZE; //iterations counter
70 int notBusyTick = INITIALIZE;
71 int scheduler = EXIT;
72 int quantum = INITIALIZE;
73 int processCounter = INITIALIZE;
74 int flag=INITIALIZE;
75 int idleSystem=INITIALIZE;
76 char singleline[MAX];
77 float cpuUtilize = INITIALIZE;
78 unsigned int tick = INITIALIZE;
79 int runningDuplicatePRINTING= EXIT;
80
81 //----- input from user and initializing the process array
from input file
82
83 FILE *inputFile;
84 FILE *outputFile;
85
86 inputFile = fopen("inFile.txt", "r");
87 outputFile = fopen("outFile.txt", "w");
88
89 if(inputFile == NULL){
90 printf("File Not Found\n");
91 return -1;
92 }
93
94 while (!feof(inputFile))
95 {
96 fgets(singleline, MAX, inputFile);
97 sscanf(singleline, "%d %d %d", &processARR[i].id, &processARR[i].cpu, &processARR[i].io, &
processARR[i].arrivalTime);
98 processARR[i].turnaroundTIME = 0;
99 processCounter++;
100 i++;
101 }
102 int remainingPROCESSES = processCounter;
103 fclose(inputFile);
104 #if DEBUG == 0
105 printf("lines: %d", processCounter);
106 for(int i= 0 ; i < processCounter ; i++)
107 {
108 printf("id: %d, cpu: %d, io: %d, arrival: %d\n", processARR[i].id, processARR[i].cpu,
processARR[i].io, processARR[i].arrivalTime);
109 }
110 #endif
111 printf("Hello Dude, I am your scheduling system, Choose What u wanna do:\n0:FCFS \n1:RR
\n-1:exit \n");
112 scanf("%d", &scheduler);
113 system("cls"); //to avoid a system exit error with -ve status instead of exit with status 0
114
115 if(scheduler == EXIT )
116 {
117 return 0;
118 }
119
120 if(scheduler == RR){
121 printf("Enter your desirable quantum length: \n");
122 scanf("%d", &quantum);
123 system("cls");
124 }

```

```

125 ///----- operations and processing
the input data
-----
126 while(1)
127 {
128 fprintf(outputFile,"%d: ",tick);
129 fflush(outputFile); // I just had to call fflush() after the call to fprintf()
because my version of the code block program refused to write any thing ..otherwise the output
was an empty text
file
130 // i found here the same programming issue and its answer:
https://stackoverflow.com/questions/33763921/c-program-not-writing-to-text-file
131
132 // traverse processARR to check if process as arrived and add them to queue
133 for(i = 0 ; i < processCounter ; i++)
134 {
135 // check for arrival
136 if(processARR[i].arrivalTime == tick)
137 {
138 readyLIST.push_back(processARR[i]);
139 flag ++;
140 idleSystem++;
141
142 }
143 }
144 //if
145 if ( flag ==0 && tick==0)
146 {
147 fprintf(outputFile,"No process has arrived yet.\t");
148 fflush(outputFile);
149 }
150
151
152
153
154 //For Debugging purposes only:
155 #if DEBUG==0
156 fprintf(outputFile,"Hello Asmaa \n\n");
157 fflush(outputFile); // I just had to call fflush() after the
call to fprintf() because my version of the code block program refused to write any thing
..otherwise the output
was an empty text file
158 /*
159 //-----currently, there is no need for these next few lines ,so i
comment it for now -----
160 // if the arrival time of all processes < current tick
161 //int n = sizeof(processARR) / sizeof(processARR[0]);
162 unsigned int tickCopy= tick;
163 if(readyLIST.size()==0 )
164 {
165 int n=processCounter - (int) tickCopy; //type
casting for the tick copy
166 // sort the process array in increasing order of
arrival time
167 sort(processARR.begin() , processARR.end() ,
compareArrivalTime);
168 for(i = 0 ; i < n && processARR[i].arrivalTime !=
tickCopy ; i++)
169 {
170
171 fprintf(outputFile,"some processes haven't
arrived yet.\t");
172 fflush(outputFile);
173 readyLIST.push_back(processARR[i]);
174 tickCopy++;
175 }
176
177
178 */
179 #endif // DEBUG

```

```

180
//-----
-----
181
182 if(blockLIST.size() > 0)
183 {
184 for(auto it = blockLIST.begin(); it != blockLIST.end(); ++it) //i2 for this
element ==> it;
185
186 {
187 // decrement io_block and check if io request handled
188 if(it->io <= 0)
189 {
190 for(int i = 0 ; i < processCounter ; i++)
191 {
192 if(processARR[i].id == it->id)
193 {
194 it->cpu = processARR[i].cpu;
195 }
196 }
197 readyLIST.push_back(*it);
198 it = blockLIST.erase(it);
199 --it;
200 }else
201 {
202 fprintf(outputFile, "process %d: blocked\t", it->id);
203 fflush(outputFile); // I just had to call fflush() after the call to
fprintf() because my version of the code block program refused to write any thing ..otherwise the
output was an
empty text file
204 // i found here the same programming issue and its
answer: https://stackoverflow.com/questions/33763921/c-program-not-writing-to-text-file
205
206 it->io--; //decrease the i/o delay of this iterator element
207 idleSystem++;
208 }
209 }
210 }
211
212 // sort the processes in increasing order of arrival time and id
213 readyLIST.sort();
214 //it==> iterator
acts like a pointer indicates the items inside the list
215 for(auto it = readyLIST.begin(); it != readyLIST.end(); ++it) //auto ==> this
keyword so the type of the "it" declared variable will be automatically deducted from its
initializer.
216 {
217 readyQueue.push_back(*it);
218 it = readyLIST.erase(it);
219 it--;
220
221 // DEBUG
222 #if DEBUG == 0
223 // printf("\n\n\n %d \n \n \n \n", *it);
224
225 // if ( (it->cpu) ==0 ) //If this process does not need a CPU usage
, and, it ONLY needs i/o usage
226 // readyQueue.pop_back(); //then pop it from the ready queue of the
CPU as a waiting process for the CPU to finish
227 #endif
228 }
229 if(readyQueue.size() <= 0 && remainingPROCESSES > 0)
230 {
231 notBusyTick++; //if this one is an Empty processor tick one, with no thing to
update or do
232
233 }
234 //-----FCFS scheduling system
-----

```

```

235 if(scheduler == FCFS)
236 {
237     if(readyQueue.size() > 0)
238     {
239         if(readyQueue.front().cpu > 0)
240         {
241             fprintf(outputFile,"process %d: running \t",readyQueue.front().id);
242             fflush(outputFile); // I just had to call fflush() after the call to
// because my version of the code block program refused to write any thing ..otherwise the
// output was an
// empty text file
243 // i found here the same programming issue and its
// answer: https://stackoverflow.com/questions/33763921/c-program-not-writing-to-text-file
244
245
246         runningDuplicatePRINTING=readyQueue.front().id; //to solve the
// repeated issue in printing the ready and the running
247         readyQueue.front().cpu--;
248         idleSystem++;
249
250         #if DEBUG == 0
251             printf("id: %d, cpu: %d, io: %d, arrival: %d\n",readyQueue.front().
id,readyQueue.front().cpu,readyQueue.front().io,readyQueue.front().arrivalTime);
252         #endif
253     }
254     if(readyQueue.front().cpu <= 0 && readyQueue.front().io > 0)
255     {
256         blockLIST.push_back(readyQueue.front());
257         readyQueue.pop_front();
258
259
260         #if DEBUG == 0
261             printf("id from ready: %d,id from blocked: %d , readyQueue size:
%d\n",readyQueue.front().id,blockLIST.front().id,readyQueue.size());
262
263         #endif
264     }
265     if(readyQueue.front().cpu <= 0 && readyQueue.front().io <= 0)
266     {
267         for(int i = 0 ; i < processCounter ; i++)
268         {
269             if(readyQueue.front().id == processARR[i].id)
270             {
271                 processARR[i].turnaroundTIME = tick - processARR[i].
arrivalTime + 1; // or ready queue>=0 in if condition
272             }
273         }
274
275         remainingPROCESSES--;
276         if (readyQueue.size()) // this "if condition" purpose is to fix and
// debug test 0 in the pdf file of assignment 3
277             readyQueue.pop_front();
278
279
280
281         #if DEBUG == 0
282
283             printf("remaining %d processes \t the id of the current process
in the readyQueue front:%d process in the blockQueue front:%d \n
",remainingPROCESSES,readyQueue.front().id,
blockLIST.front().id);
284         #endif // DEBUG
285
286     }
287 }
288
289 }
290
291
292

```

```

293 /*
294 //-----Round Robin scheduling
system
-----

295
296 else if(scheduler == RR)
297 {
298
299 }
300 */
301 //----- finishing the final operations
and printing
the output -----
302
303 if (tick !=0 && remainingPROCESSES == onlyONE)
304 {
305 //to solve an error in empty linee in the printing file
306 if (readyQueue.front().cpu== INITIALIZE && readyQueue.front().io==INITIALIZE)
307 {
308 remainingPROCESSES--;
309 readyQueue.pop_front();
310 }
311 }
312
313 //int tot=0;
314 if(remainingPROCESSES <= INITIALIZE)
315 {
316 #if DEBUG==0
317
318 //to make sure or to make indicator that all process finishes there i/o and
cpu
319 for (int i =0 ; i<processCounter ;i++)
320 {
321 tot= tot + processARR[i].cpu + processARR[i].io;
322 }
323
324 if (tot==INITIALIZE)
325 #endif
326
327
328
329
330 break;
331
332 }
333
334 #if DEBUG == 0
335 //tot=0;
336
337 /*
338 queueCopy= readyQueue; // just to keep the data
339 for(auto j = queueCopy.begin() ; j == queueCopy.end() ;
j++)
340 {
341 //if the current process finished its cpu and io
time
342 if( tick!=0 && (j->cpu) <=INITIALIZE && (j->io) <=
INITIALIZE)
343 {
344 j= queueCopy.erase(j);
345 j--;
346 remainingPROCESSES--;
347 }
348
349 }
350 readyQueue=queueCopy;
351 */
352 #endif
353 //ready states printing of the processes
354 for(i = 0 ; i < processCounter ; i++)

```

```

355 {
356 //if arrival time == tick and it hasn't been printed before in the running
states
357 if( processARR[i].arrivalTime == tick && i!=runningDuplicatePRINTING )
358 {
359
360 fprintf(outputFile,"process %d: ready\t",processARR[i].id);
361 fflush(outputFile); // I just had to call fflush() after the call to
fprintf() because my version of the code block program refused to write any thing ..otherwise the
output was an
empty text file
362 // i found here the same programming issue and its
answer: https://stackoverflow.com/questions/33763921/c-program-not-writing-to-text-file
363 }
364 }
365
366
367 // to solve the empty printing line in the output file if the the i/o, CPU and the
whole sys are idle for this current time tick
368 if(idleSystem==0 && tick!=0)
369 {
370 fprintf(outputFile,"your whole system is idle for now.");
371 fflush(outputFile); // I just had to call fflush() after the call to
fprintf() because my version of the code block program refused to write any thing ..otherwise the
output was an
empty text file
372 // i found here the same programming issue and its
answer: https://stackoverflow.com/questions/33763921/c-program-not-writing-to-text-file
373
374 }
375 idleSystem=0; // getting ready for the next while (1) looping
376
377 tick++;
378 fprintf(outputFile,"\n");
379 fflush(outputFile); // I just had to call fflush() after the call to fprintf()
because my version of the code block program refused to write any thing ..otherwise the output
was an empty text
file
380 // i found here the same programming issue and its answer:
https://stackoverflow.com/questions/33763921/c-program-not-writing-to-text-file
381
382 #if DEBUG==0
383 if(readyQueue.front().cpu <= 0 && readyQueue.front().io > 0 && remainingPROCESSES
)
384 {
385 continue;
386 }
387 #endif
388 }
389
390 fprintf(outputFile,"\n\n");
391 fflush(outputFile); // I just had to call fflush() after the call to fprintf() because my
version of the code block program refused to write any thing ..otherwise the output was an empty
text file
392 // i found here the same programming issue and its answer:
https://stackoverflow.com/questions/33763921/c-program-not-writing-to-text-file
393
394
395 //-----calculating the final outputs
and exit
-----
396
397 fprintf(outputFile,"Finishing time: %d\n",tick);
398 fflush(outputFile);
399
400 cpuUtilize = (float)(tick + 1 - notBusyTick)/(tick+1);
401 fprintf(outputFile,"CPU utilization : %.3f\n",cpuUtilize);
402 fflush(outputFile);
403
404 for(int i =0 ; i < processCounter; i++)

```

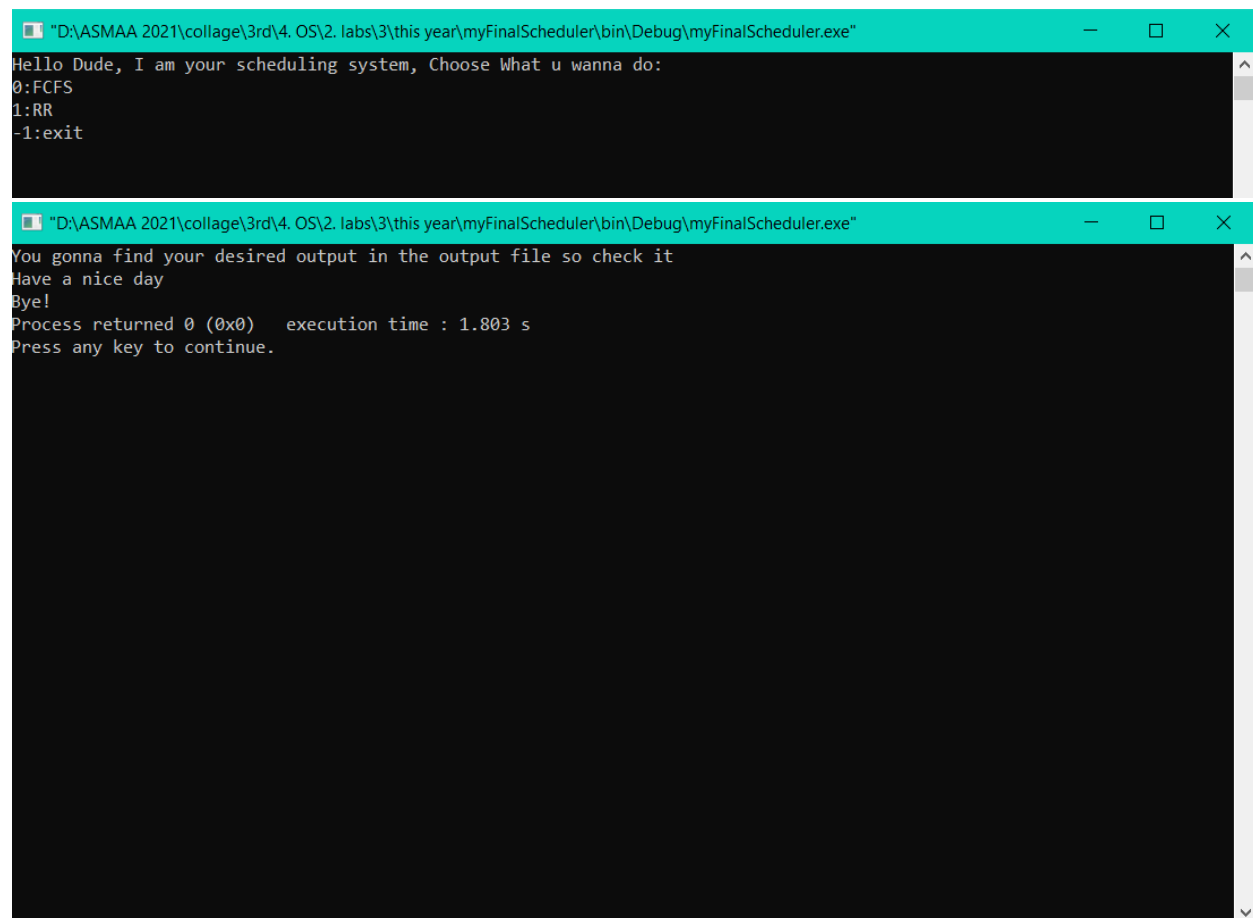


```

405 {
406 if( processARR[i].cpu ==INITIALIZE && processARR[i].io!=INITIALIZE)
407 {
408 processARR[i].turnaroundTIME = processARR[i].io + processARR[i].arrivalTime;
409 fprintf(outputFile,"Turnaround time of Process %d: %d\n",processARR[i].id,processARR[i]
].turnaroundTIME );
410
411 }else
412 {
413 fprintf(outputFile,"Turnaround time of Process %d: %d\n",processARR[i].id,processARR[i]
].turnaroundTIME);
414 fflush(outputFile);
415 }
416
417 }
418 fclose(inputFile);
419 fclose(outputFile);
420 printf("You gonna find your desired output in the output file so check it\nHave a nice
day\nBye!");
421 return 0;
422 }

```

OUTPUT SCREENSHOTS FROM THE CODE:



TEST CASE 1:

The image shows two Notepad windows. The left window, titled 'outFile - Notepad', displays the output of a test case. It lists 28 steps of process execution for three processes (0, 1, 2, 3), showing their states (idle, running, blocked, ready) and their relative positions. At the bottom, it provides summary statistics: Finishing time: 28, CPU utilization: 0.93, and turnaround times for each process. The right window, titled 'inFile - Notepad', shows a snippet of C++ code. The code includes a loop that checks if a process is ready to run and then updates its state and position in an array. The code is partially visible, showing the beginning of a loop and a function call.

```
outFile - Notepad
File Edit Format View Help
0: No process has arrived yet.
1: your whole system is idle for now.
2: process 1: running    process 3: ready
3: process 1: running    process 0: ready    process 2: ready
4: process 1: blocked    process 3: running
5: process 1: blocked    process 3: running
6: process 1: blocked    process 3: running
7: process 1: blocked    process 3: running
8: process 3: running
9: process 3: running
10: process 3: running
11: process 3: running
12: process 3: running
13: process 3: blocked    process 2: running
14: process 3: blocked    process 2: running
15: process 3: blocked    process 2: blocked
16: process 2: blocked    process 0: blocked    process 1: running
17: process 2: blocked    process 0: blocked    process 1: running
18: process 2: blocked    process 0: blocked    process 3: running
19: process 0: blocked    process 3: running
20: process 3: running
21: process 3: running
22: process 3: running
23: process 3: running
24: process 3: running
25: process 3: running
26: process 3: running
27: process 2: running
28: process 2: running

Finishing time: 28
CPU utilization : 0.93
Turnaround time of Process 0: 7
Turnaround time of Process 1: 16
Turnaround time of Process 2: 26
Turnaround time of Process 3: 25

inFile - Notepad
File Edit Format View Help
0 0 4 3
1 2 4 2
2 2 4 3
3 9 3 2

check if id
processCounte
d == it->id
essARR[i].c
t):
t):

process %d: b
I just ha
i found he
decrease th

CppCheck/Ver
Ln 1, Col 1    100%    Windows (CRLF)    UTF-8
C:\Program Files\PolySpace\R2020a\bin\Cr\Program Files\PolySpace\R2020a\poly\bin\C\Use
pData\Local\Programs\Python\Python39\Scripts
\3rd\4. OS\2. labe\3\this year\myFinalScheduler\bin\Debug\myFinalScheduler.exe" (in D:\ASMAA
-----
Target is up to date.
Process terminated with status 0 (0 minute(s), 3 second(s))
```

TEST CASE 2:

The screenshot shows the Code::Blocks IDE interface. The top menu bar includes options like Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DoxyBlocks, Settings, and Help. Below the menu is a toolbar with icons for file operations and building. Two Notepad windows are open. The foreground window, 'outFile - Notepad', contains the following text:

```

17: process 2: running
18: process 2: blocked   process 0: running
19: process 2: blocked   process 0: running
20: process 0: running
21: process 0: running
22: process 0: running
23: process 0: running
24: process 1: running
25: process 1: running
26: process 1: running
27: process 1: running
28: process 3: running
29: process 3: running
30: process 3: running
31: process 3: running
32: process 2: running
33: process 2: running
34: process 2: running
35: process 2: running

Finishing time: 35
CPU utilization : 1.00
Turnaround time of Process 0: 24
Turnaround time of Process 1: 28
Turnaround time of Process 2: 34
Turnaround time of Process 3: 32

```

The background window, 'inFile - Notepad', shows a portion of a C++ source file with the following code:

```

if (is...)
    cout<<endl;
t->id...
[i], c...

//d: b...
st ha...
nd he...

se th...

```

TEST CASE 3:

[illegible]

outFile - Notepad

```
File Edit Format View Help
13: process 0: blocked    process 3: blocked    process 1: running
14: process 0: blocked    process 1: blocked    process 5: running
15: process 1: blocked    process 5: running
16: process 1: blocked    process 5: blocked
17: process 1: blocked    process 5: blocked
18: process 5: blocked    process 4: blocked    process 2: running
19: process 5: blocked    process 4: blocked    process 2: running
20: process 4: blocked    process 2: running
21: process 2: running
22: process 6: running
23: process 6: running
24: process 7: running
25: process 7: running
26: your whole system is idle for now.
27: your whole system is idle for now.
28: process 1: running
29: process 1: running
30: process 5: running
31: process 5: running

Finishing time: 31
CPU utilization : 0.97
Turnaround time of Process 0: 6
Turnaround time of Process 1: 27
Turnaround time of Process 2: 21
Turnaround time of Process 3: 4
Turnaround time of Process 4: 7
Turnaround time of Process 5: 29
Turnaround time of Process 6: 23
Turnaround time of Process 7: 24
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

inFile - Notepad

```
File Edit Format View Help
0 0 4 2
1 2 4 3
2 5 5 1
3 0 2 2
4 0 4 3
5 2 4 3
6 2 3 1
7 2 3 2
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

```
...=INITIALIZE)
...processARR[i].arrivalTime;
...&id\n",processARR[i].id,proc
...&id\n",processARR[i].id,proc
...
...ile so check it\nHave a nice
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

A description of the overall organization of your code and the major functions:

First Come First Serve (FCFS):

First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which requests the CPU first get the CPU allocation first. This is managed with a FIFO queue. The full form of FCFS is First Come First Serve.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue and, when the CPU becomes free, it should be assigned to the process at the beginning of the queue.

Characteristics of FCFS method

- It supports non-preemptive and pre-emptive scheduling algorithm.
- Jobs are always executed on a first-come, first-serve basis.
- It is easy to implement and use.
- This method is poor in performance, and the general wait time is quite high.

Example of FCFS scheduling

A real-life example of the FCFS method is buying a movie ticket on the ticket counter. In this scheduling algorithm, a person is served according to the queue manner. The person who arrives first in the queue first buys the ticket and then the next one. This will continue until the last person in the queue purchases the ticket. Using this algorithm, the CPU process works in a similar manner.

Advantages of FCFS

Here, are pros/benefits of using FCFS scheduling algorithm:

- The simplest form of a CPU scheduling algorithm
- Easy to program
- First come first served

Disadvantages of FCFS

Here, are cons/ drawbacks of using FCFS scheduling algorithm:

- It is a Non-Preemptive CPU scheduling algorithm, so after the process has been allocated to the CPU, it will never release the CPU until it finishes executing.
- The Average Waiting Time is high.
- Short processes that are at the back of the queue have to wait for the long process at the front to finish.
- Not an ideal technique for time-sharing systems.
- Because of its simplicity, FCFS is not very efficient.

Summary:

- Definition: FCFS is an operating system scheduling algorithm that automatically executes queued requests and processes by order of their arrival
- It supports non-preemptive and pre-emptive scheduling algorithm.
- FCFS stands for First Come First Serve
- A real-life example of the FCFS method is buying a movie ticket on the ticket counter.
- It is the simplest form of a CPU scheduling algorithm
- It is a Non-Preemptive CPU scheduling algorithm, so after the process has been allocated to the CPU, it will never release the CPU until it finishes executing.

The struct for processes to hold its information named process node where there is variable that holds :

1. Process id
2. Cpu time
3. i/o time

4. arrival time
5. turn-around time

and to start sorting according to the arrival time stored in the process node and its id

Main function:

Declaration of variables

Getting input from user and initializing the process array from input file

Starting processing on the input data got from the file

Check if the process time =0 and put it in the ready list to start preparing the schedule

Passing to the if condition as it checks if it hasn't passed to the loop. It means that there is no process has arrived and there will be an error to be found.

After that it will continue processing and preparing the schedule by putting first from the block to the ready. There is also a check condition as it checks if the process is still blocked, it will show in the output file that it is blocked.

After this, the ready list is getting sorted.

The ready queue is being filled and the removing from the ready list.

It starts to form the fcfs as it sees if it needs cpu time and take a cycle. This cycle is being subtracted from the cpu time later.

The process put in the ready if it has finished its cpu time but needs i/o time. It is being transferred from the ready to blocked list

After finishing its cpu and i/o time. The turnaround time is being calculated.

Round Robin (RR):

```
int n=0,n_io=0;
int clock=0,q=2;
int top=-1,front=0;
int top_io=-1,front_io=0;
int que [100],block[100];
int count=0, ff_r=-1,ff_cpu=-1,ff_clock=-1,ff_io=-1,ff_io2=-1,ff=-1,ff_id=-1,ff_for=-1;
```

Declaration of the variables needed like :

- **n** for num of process
- **clock** to calculate the total time of all the processes
- **top_io** and **front_io** for the queue
- **Que** for the ready queue to run the processes and **block** to put the processes has no cpu
- **Some flags** for avoiding repetitions in printing

```
struct processes
{
    int ID;           ///name of the process
    int cpu;          ///CPU time
    int at;           ///arrival time
    int IO;           ///IO time
    int cpu_c;        ///finished time of the process
    int c;            ///2nd usage of CPU time
    int flag;         ///no of times of using cpu
    int atc;          ///first arrival time
};
struct processes queue_array[10];
```

- The struct for processes to hold its information
- Making array of structs (for ready queue)


```

//sort elements in the queue according to their arrival time
void sort()
{
    int i,j,t;
    for(i=front; i<top; i++)
    {
        for(j=i+1; j<=top; j++)
        {
            if(queue_array[que[i]].at > queue_array[que[j]].at)
            {
                t = que[i];
                que[i] = que[j];
                que[j] = t;
            }
            //if two processes arrived at the same time put the smallest ID first in the Queue
            else if (queue_array[que[i]].at == queue_array[que[j]].at)
            {
                if (queue_array[que[i]].ID > queue_array[que[j]].ID)
                {
                    t = que[i];
                    que[i] = que[j];
                    que[j] = t;
                }
            }
        }
    }
}

```

Making sort function :

- To put the processes in the ready queue as the lowest arrival coming first
- To see if two processes came at the same time putting the process with the lowest id first

```

//get input from file
void inputs(FILE *fp)
{
    char chr = 'a';
    chr =getc(fp);
    while (chr!=EOF)
    {
        queue_array[n].ID=chr-'0';

        chr =getc(fp); //white space
        chr =getc(fp);

        queue_array[n].cpu=chr-'0';
        queue_array[n].c=chr-'0';

        chr =getc(fp);
        chr =getc(fp);

        queue_array[n].IO=chr-'0';
        chr =getc(fp);
        chr =getc(fp);

        queue_array[n].at=chr-'0';
        queue_array[n].atc=chr-'0';
        queue_array[n].cpu_c=0;
        queue_array[n].flag=2;
        chr =getc(fp); //new line
    }
}

```

```

    ///in case the process doesn't use the CPU put it in the block queue
    if ( queue_array[n].cpu == 0)
    {    ///finished time of this process will be the time which it finished IO
        queue_array[n].cpu_c= queue_array[n].at+ queue_array[n].IO ;
        block[++top_io]=queue_array[n].ID;
    }
    else
        que[++top]=queue_array[n].ID; //insert the pro in the que
    n++;
    chr =getc(fp);
}
}

```

Inputs function

- to read the information of each process from the input file
- sub from '0' to convert char to int
- checking if the process has 0 cpu to put it in the block queue to calculate its finishing time as the io time only (plus it's arrival time)

```

//////////////////////////////////////print the output//////////////////////////////////////
void print ()
{
    FILE *fptr;
    fptr = fopen("output.txt","a");
    if (ff_clock != clock)
    {   fprintf(fptr,"%d ",clock);
        printf("%d",clock );
        ff_clock=clock;
    }
    ///run the process if its arrived while clock is greater than or equal it
    if (queue_array[que[front]].at<=clock && ff_cpu!=clock)
    {   fprintf(fptr," %d:running ",queue_array[que[front]].ID);
        printf(" %d:running ",queue_array[que[front]].ID);
        ff_cpu=clock;
        ///count++;
    }
    ///the process is blocked when it finishes the first usage of CPU
    else if (queue_array[que[front]].at > clock && queue_array[que[front]].at != queue_array[que[front]].atc
    |&& ff_io!=clock)
    {   fprintf(fptr," %d:block ",queue_array[que[front]].ID);
        printf(" %d:block ",queue_array[que[front]].ID);
        ff_io=clock;
    }
    for (int i=front_io ; i<=top_io ; i++)
    {   ///print the blocked processes which doesn't use the CPU
        if (queue_array[block[front_io]].at+queue_array[block[front_io]].IO > clock
        && clock> queue_array[que[0]].atc &&ff_io2!=clock)
    }
}

```

```

{ fprintf(fp, " %d:block ", queue_array[block[i]].ID);
  printf(" %d:block ", queue_array[block[i]].ID);
  ff_io2=clock;
}
}
if (ff_for != clock)
{
  for (int i=front+1 ; i<=top ; i++)
  { //print the ready process if when it arrives there is another process with smaller ID use the CPU
    if (queue_array[que[i]].at <= clock && (ff_r!=clock || ff_id!= queue_array[que[i]].ID) )
    { fprintf(fp, " %d:ready ", queue_array[que[i]].ID);
      printf(" %d:ready ", queue_array[que[i]].ID);

      ff_r=clock;
      ff_id= queue_array[que[i]].ID;
    }
    //the process is blocked when it finishes the first usage of CPU
    else if (queue_array[que[i]].at > clock && queue_array[que[i]].at != queue_array[que[i]].atc && ff_io!=clock)
    { fprintf(fp, " %d:block ", queue_array[que[i]].ID);
      printf(" %d:block ", queue_array[que[i]].ID);
      ff_io=clock;
    }
  }
  ff_for = clock;
}
if (ff != clock)
{ fprintf(fp, "\n");
  printf("\n");
  ff=clock;
}
}

```

Print function

- print **running (for quantum)** if its arrival time greater than or equal to the clock (and it is in its right sorted position in the queue)
 - the ff_cpu is just a flag to prevent repetition in printing
- print **block** when it finishes its first cpu and doing io
 - the ff_io is just a flag to prevent repetition in printing
- then making for loop to check that if there is a process doesn't cpu and it is in the block queue as **block**
 - the ff_io2 is just a flag to prevent repetition in printing
- now to print the **ready** case there is a for loop to check if the process with the smallest id which we sorted previously running so the one before it will be **ready**
 - the ff_for is just a flag to prevent repetition in printing
- and at the same loop check if the process is ready but it finishes its first cpu so it will be **block** doing io
 - the ff is just a flag to prevent repetition in printing

```

int main()
{
    //taking the file name////////////////////////////////////
    char file[100];
    char in;
    printf("Enter the inputs File name : ");
    int i=0;
    scanf("%c",&in);
    while(in!='\n')
    {
        file[i]=in;
        scanf("%c",&in);
        i++;
    }

    //read from the file
    FILE *fp= fopen(file, "r",stdin);
    inputs(fp);
    //sort the input processes according to arrival time
    sort();
    print();
}

```

in main

- we start taking the file name from user and then read it and store it using inputs function which we created above
- then sort the process using sort function which we created above
- using print function to check and print the running process after sorting and the ready one if it exists

```

//while queue of processes isn't empty
while (top>=front )
{
    print(); //to print the output

    if (queue_array[que[front]].at<=clock) //check for arrived process to run it
    {
        if (queue_array[que[front]].cpu-q>=0) //if the process use CPU with time greater than or equal to q
        { //increase the clock by quantum time
            clock+=q;
            count+=q; //increase no of usage of CPU
        }
        //if the process use CPU with time smaller the quantum time
        else
        {
            //increase the clock by CPU time
            clock+=(queue_array[que[front]].cpu);
            count+=queue_array[que[front]].cpu; //increase no of usage of CPU
        }

        queue_array[que[front]].cpu-=q;
        //if the process is done its first usage of CPU
        if ( queue_array[que[front]].cpu<=0 && queue_array[que[front]].flag==2)
        {
            queue_array[que[front]].flag--; //decrease the flag of no of usage CPU
            queue_array[que[front]].cpu= queue_array[que[front]].c; //store the CPU time for 2nd usage
            queue_array[que[front]].at=clock+queue_array[que[front]].IO; //update the arrived time of the process to '
            que[++top]=queue_array[que[front]].ID; //push the process at the end of the queue

            front++; //increase the front pointer of the queue
            sort(); //sort the processes according to their arrival time
        }
    }
}

```

Now if there is a process in the queue :

- We call the print function to keep printing the updates
- Check if $cpu > q$

- increasing clock with the q ,increasing count with q too to count the running cycles of cpu in all the program
- if not
 - increasing clock with the cpu time ,increasing count with cpu time too to count the running cycles of cpu in all the program
- at the end we sub the quantum from the cpu time of the process as its running
- now check if the process finished its first cpu , we will change some values :
 - decreases the cpu flag by 1(as we declared with 2 in the first of the program as the process using cpu twice with same period before and after doing the io)
 - store the cpu time all again in the struct to make it all again
 - update the arrival time of the process for the second cpu and put it at the end of the queue , then sort again as its arrival time

```

//if the process is done its 2nd usage of CPU
else if ( queue_array[que[front]].cpu<=0&&queue_array[que[front]].flag==1)
{
    //finishing time of the process
    queue_array[que[front]].cpu_c=clock;
    front++;
    //remove the process from the queue
}
//if the process was in ready queue
else
{
    queue_array[que[front]].at=clock;
    que[++top]=queue_array[que[front]].ID;
    front++;
    sort();
    print();
    //update its arrived time
    //push the process at the end of the queue
    //increase the front pointer of the queue
    //sort the processes according to their arrival time
    //for print the output
}
}
//if there aren't any running process
else
{
    print();
    clock++;
}
}

```

- Now check if the process finished its 2nd cpu usage (cpu flag now equal 1)
 - so we knew that the process finished and we can calculate the finished time of the process

- If not doing same updates as above and put it at the end of the queue , then sort
- There is a case if there is no running process (block) , we will increase the clock to go to the next cycle and make all the checks to know which running now

```

'
///print at output file
FILE *fptr;
fptr = fopen("output.txt","a");
///print finish time,CPU Utilization and Turnaround time
fprintf(fptr,"\nFinishing time: %d",clock);
printf("\nFinishing time: %d",clock);
fprintf(fptr,"\nCPU Utilization:%.3f\n", (float)count/(float)clock);
printf("\nCPU Utilization:%.3f\n", (float)count/(float)clock);
for (int i=0 ; i< n; i++)
{   fprintf(fptr,"Turnaround time of Process %d:",queue_array[i].ID);
    printf("Turnaround time of Process %d:",queue_array[i].ID);
    fprintf(fptr," %d\n",queue_array[i].cpu_c-queue_array[i].atc);
    printf(" %d\n",queue_array[i].cpu_c-queue_array[i].atc);
}
}

```

- Here we print the output file and make the calculation of
 -**cpu utilization** = num of running cycles / total time of the program
 -**turnaround time** = finish time of the process – the arrival time of the process

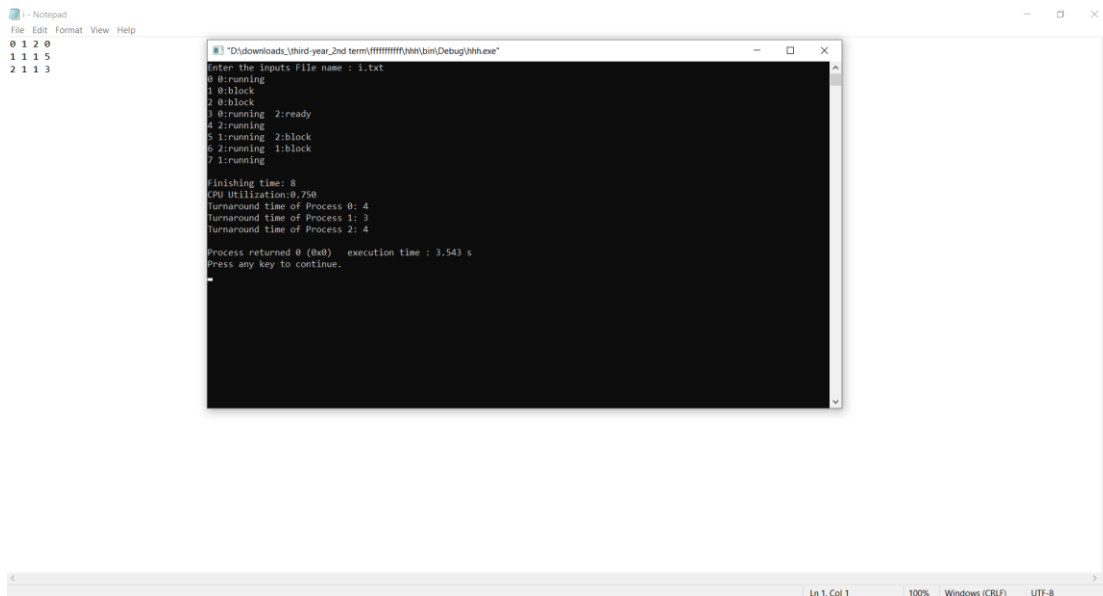
Screenshots:

✚ Screenshot for First Come First Serve (FCFS):

✚ Screenshot for Round Robin (RR) :

Assignment test case

(q=1)

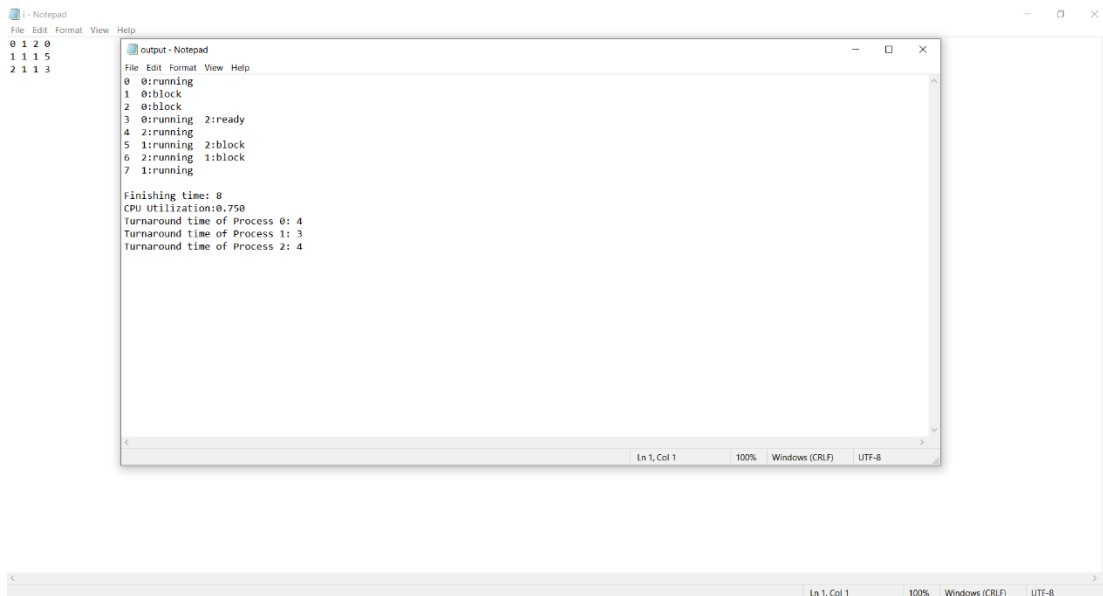


```
File Edit Format View Help
0 1 2 0
1 1 1 5
2 1 1 3

"D:\downloads_3third-year_2nd term\ffffff\hhh\bin\Debug\hhh.exe"
Enter the inputs File name : 1.txt
0:running
1:0:block
2:0:block
3:0:running 2:ready
4:2:running
5:1:running 2:block
6:2:running 1:block
7:1:running

Finishing time: 8
CPU Utilization:0.750
Turnaround time of Process 0: 4
Turnaround time of Process 1: 3
Turnaround time of Process 2: 4

Process returned 0 (0x0)   execution time : 3.543 s
Press any key to continue.
```



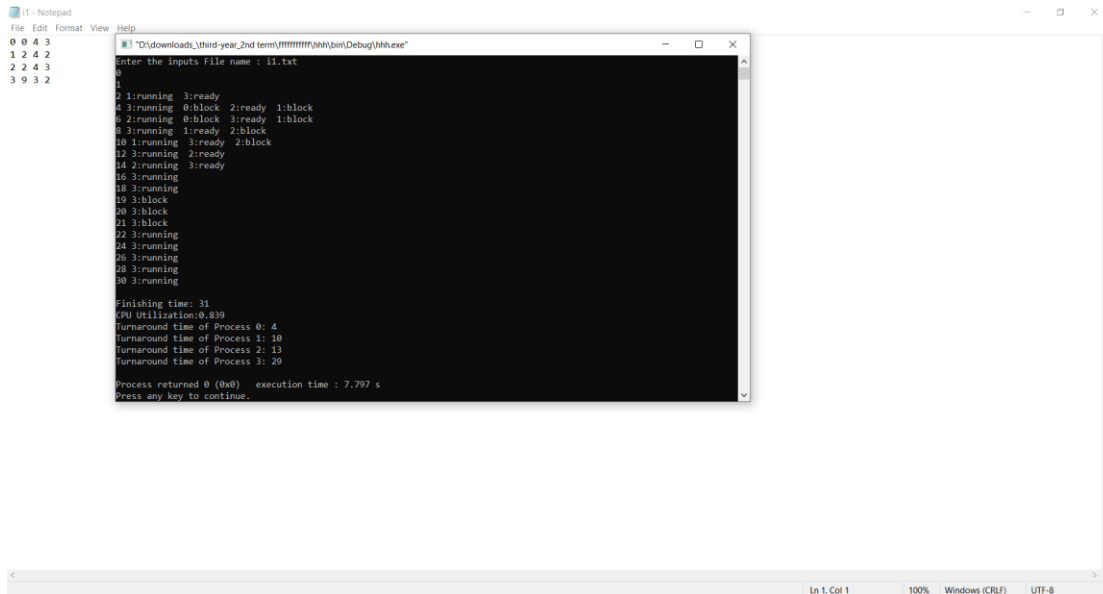
```
File Edit Format View Help
0 1 2 0
1 1 1 5
2 1 1 3

output - Notepad
File Edit Format View Help
0:running
1:0:block
2:0:block
3:0:running 2:ready
4:2:running
5:1:running 2:block
6:2:running 1:block
7:1:running

Finishing time: 8
CPU Utilization:0.750
Turnaround time of Process 0: 4
Turnaround time of Process 1: 3
Turnaround time of Process 2: 4
```

Test case 1

(q=2)

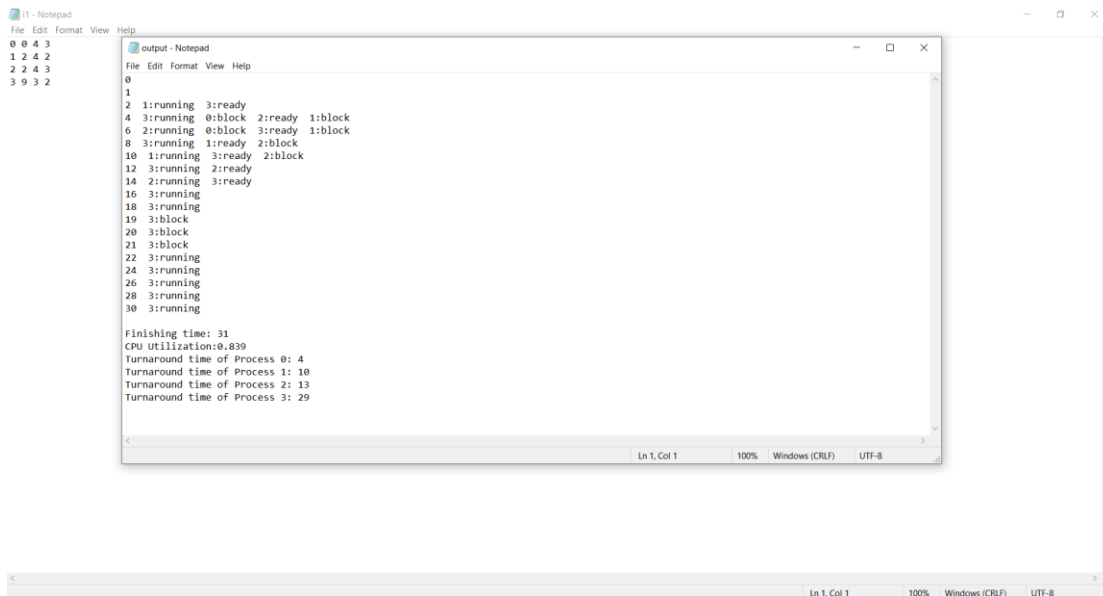


```
File Edit Format View Help
0 0 4 3
1 2 4 2
2 2 4 3
3 9 3 2

"D:\downloads\3rd year_2nd term\mmmmmm\hh\bin\Debug\hh.exe"
Enter the inputs File name : ll.txt
0
1
2 1:running 3:ready
4 3:running 0:block 2:ready 1:block
6 2:running 0:block 3:ready 1:block
8 3:running 1:ready 2:block
10 1:running 3:ready 2:block
12 3:running 2:ready
14 2:running 3:ready
16 3:running
18 3:running
19 3:block
20 3:block
21 3:block
22 3:running
24 3:running
26 3:running
28 3:running
30 3:running

Finishing time: 31
CPU Utilization:0.839
Turnaround time of Process 0: 4
Turnaround time of Process 1: 10
Turnaround time of Process 2: 13
Turnaround time of Process 3: 29

Process returned 0 (0x0)   execution time : 7.797 s
Press any key to continue.
```



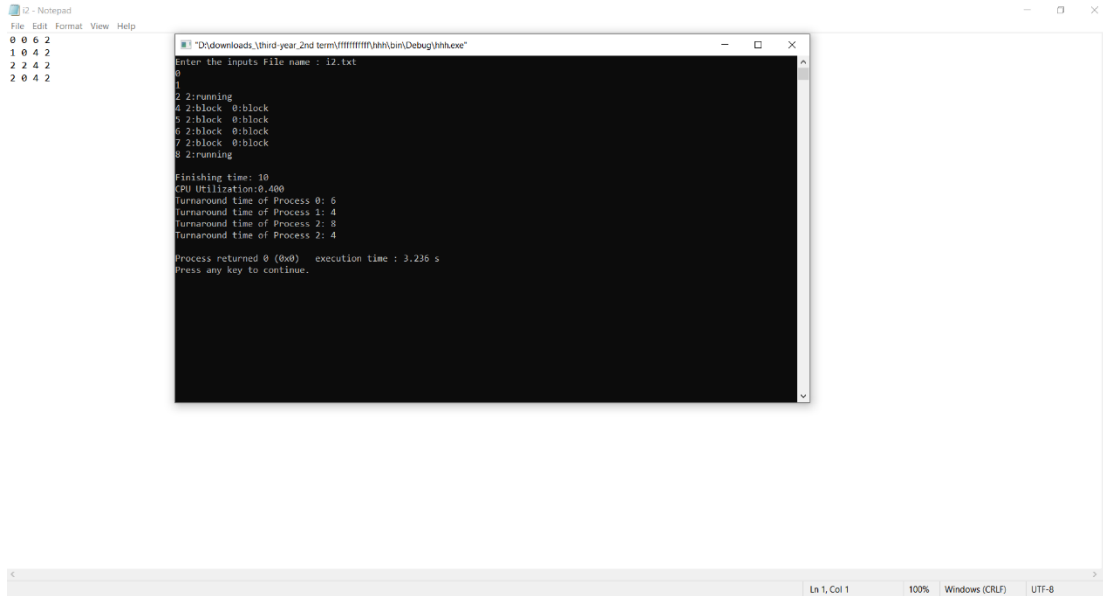
```
File Edit Format View Help
0 0 4 3
1 2 4 2
2 2 4 3
3 9 3 2

output - Notepad
File Edit Format View Help
0
1
2 1:running 3:ready
4 3:running 0:block 2:ready 1:block
6 2:running 0:block 3:ready 1:block
8 3:running 1:ready 2:block
10 1:running 3:ready 2:block
12 3:running 2:ready
14 2:running 3:ready
16 3:running
18 3:running
19 3:block
20 3:block
21 3:block
22 3:running
24 3:running
26 3:running
28 3:running
30 3:running

Finishing time: 31
CPU Utilization:0.839
Turnaround time of Process 0: 4
Turnaround time of Process 1: 10
Turnaround time of Process 2: 13
Turnaround time of Process 3: 29
```

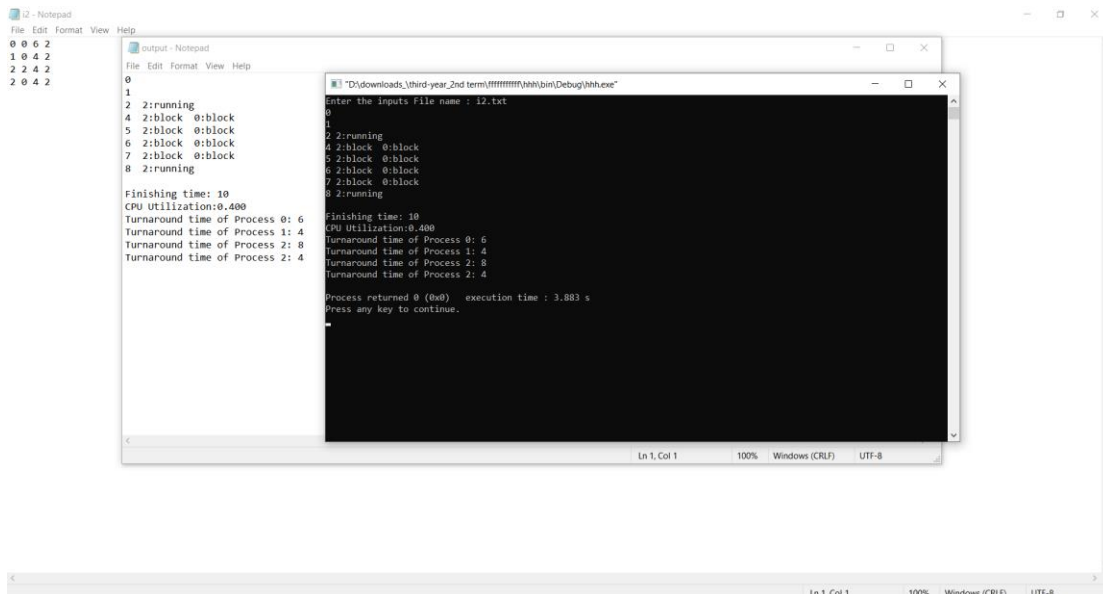

Test case 2

(q=2)



```
0 0 6 2
1 0 4 2
2 2 4 2
2 0 4 2

"D:\downloads\_third-year_2nd term\ifmfmfmfm\bin\Debug\hh.exe"
Enter the inputs File name : 12.txt
0
1
2 2:running
4 2:block 0:block
5 2:block 0:block
6 2:block 0:block
7 2:block 0:block
8 2:running
Finishing time: 10
CPU Utilization:0.400
Turnaround time of Process 0: 6
Turnaround time of Process 1: 4
Turnaround time of Process 2: 8
Turnaround time of Process 2: 4
Process returned 0 (0x0) execution time : 3.236 s
Press any key to continue.
```

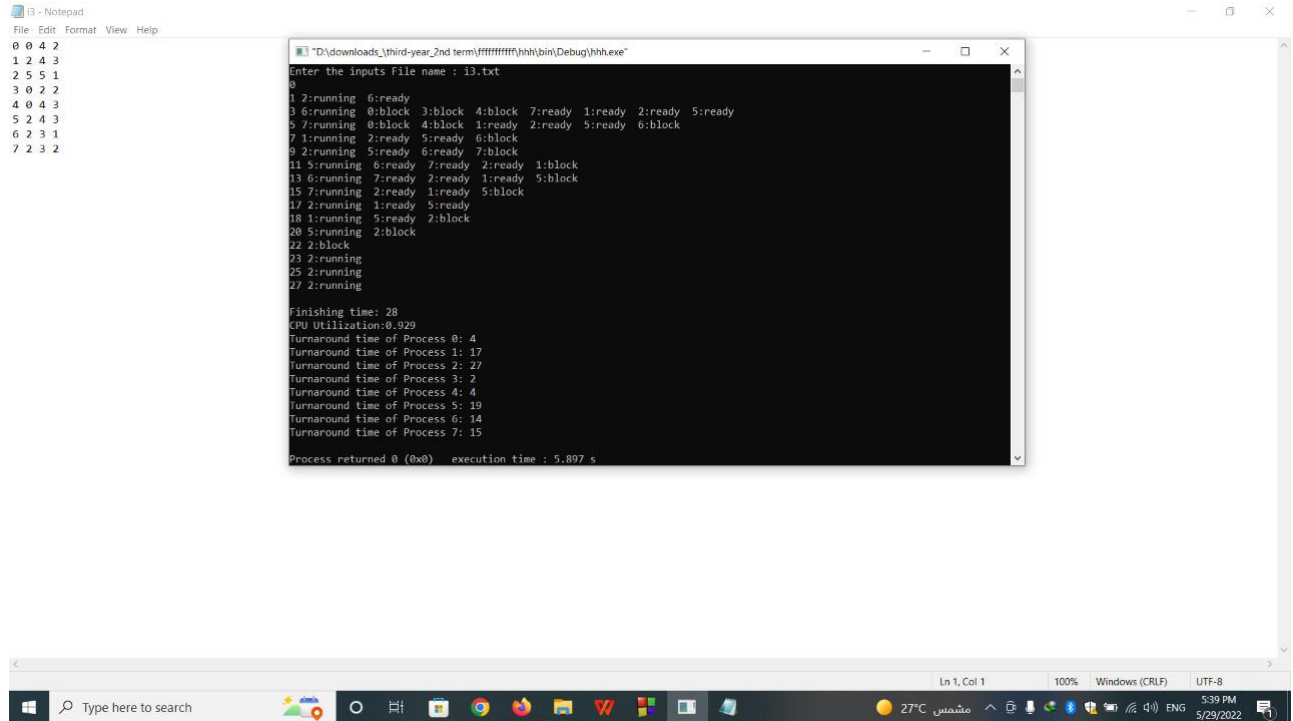


```
0 0 6 2
1 0 4 2
2 2 4 2
2 0 4 2

output - Notepad
"D:\downloads\_third-year_2nd term\ifmfmfmfm\bin\Debug\hh.exe"
Enter the inputs File name : 12.txt
0
1
2 2:running
4 2:block 0:block
5 2:block 0:block
6 2:block 0:block
7 2:block 0:block
8 2:running
Finishing time: 10
CPU Utilization:0.400
Turnaround time of Process 0: 6
Turnaround time of Process 1: 4
Turnaround time of Process 2: 8
Turnaround time of Process 2: 4
Process returned 0 (0x0) execution time : 3.883 s
Press any key to continue.
```

Test case 3

(q=2)



```
0 0 4 2
1 2 4 3
2 5 5 1
3 0 2 2
4 0 4 3
5 2 4 3
6 2 3 1
7 2 3 2
```

```
"D:\downloads\_third-year_2nd term\ffffffh\bin\Debug\hhh.exe"
Enter the inputs file name : i3.txt
0
1 2:running 6:ready
3 6:running 0:block 3:block 4:block 7:ready 1:ready 2:ready 5:ready
5 7:running 0:block 4:block 1:ready 2:ready 5:ready 6:block
7 1:running 2:ready 5:ready 6:block
9 2:running 5:ready 6:ready 7:block
11 5:running 6:ready 7:ready 2:ready 1:block
13 6:running 7:ready 2:ready 1:ready 5:block
15 7:running 2:ready 1:ready 5:block
17 2:running 1:ready 5:ready
18 1:running 5:ready 2:block
20 5:running 2:block
22 2:block
23 2:running
25 2:running
27 2:running

Finishing time: 28
CPU Utilization:8.929
Turnaround time of Process 0: 4
Turnaround time of Process 1: 17
Turnaround time of Process 2: 27
Turnaround time of Process 3: 2
Turnaround time of Process 4: 4
Turnaround time of Process 5: 19
Turnaround time of Process 6: 14
Turnaround time of Process 7: 15
Process returned 0 (0x0) execution time : 5.897 s
```

0 0 4 2
1 2 4 3
2 5 5 1
3 0 2 2
4 0 4 3
5 2 4 3
6 2 3 1
7 2 3 2

```
output - Notepad
File Edit Format View Help
0
1 2:running 6:ready
3 6:running 0:block 3:block 4:block 7:ready 1:ready 2:ready 5:ready
5 7:running 0:block 4:block 1:ready 2:ready 5:ready 6:block
7 1:running 2:ready 5:ready 6:block
9 2:running 5:ready 6:ready 7:block
11 5:running 6:ready 7:ready 2:ready 1:block
13 6:running 7:ready 2:ready 1:ready 5:block
15 7:running 2:ready 1:ready 5:block
17 2:running 1:ready 5:ready
18 1:running 5:ready 2:block
20 5:running 2:block
22 2:block
23 2:running
25 2:running
27 2:running

Finishing time: 28
CPU Utilization:0.929
Turnaround time of Process 0: 4
Turnaround time of Process 1: 17
Turnaround time of Process 2: 27
Turnaround time of Process 3: 2
Turnaround time of Process 4: 4
Turnaround time of Process 5: 19
Turnaround time of Process 6: 14
Turnaround time of Process 7: 15
```