Table of Contents

1. Introduction

In this project, a simulation of a control system for a robotic assembly line was implemented using multi-tasking/ multi-threading C++ program. It was required to design and develop this program using the concepts of concurrent programming. Silberschatz et al (2013) describe a concurrent system as a system where a computation can make progress without waiting for all other computations to complete—where more than one computation can make progress at "the same time. Concurrent programming is related to but distinct from parallel programming, though these concepts are frequently confused, and both can be described as "multiple processes executing during the same period of time". In parallel computing, execution literally occurs at the same instant, for example on separate processors of a multi-processor machine, with the goal of speeding up computations—parallel computing is impossible on a (single-core) single processor, as only one computation can occur at any instant. By contrast, concurrent computing consists of process lifetimes overlapping, but execution need not happen at the same instant. The goal here is to model processes in the outside world that happen concurrently (Schneider, 2012). Among the advantages of Concurrent programming is its increased application throughput, and high responsiveness for input/output, and

The communication mechanism for Concurrent programming is Inter-process communication (IPC), which is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system (Stevens et al, 2004). This allows a program to handle many user requests at the same time. Since even a single user request may result in multiple processes running in the operating system on the user's behalf, the processes need to communicate with each other. The IPC interfaces make this possible. Each IPC method has its own advantages and limitations so it is not unusual for a single program to use all of the IPC methods. Among IPC methods are pipes, message, queueing, semaphores, shared memory, and sockets.

2. System Analysis

The required control system is composed of a production Line A, which is a *producer* of the objects randomly in red, blue and yellow to feed them onto the *consumer* lines B, C, and D respectively. It was also required to control the speed and synchronisation of the produced objects via specified commands by the user.

In order to meet these requirements and even implement additional features, a simulation for our control system is performed via building six processes. The first process is to simulate the production line A, which produce objects in three different colours and put them in the Scheduler process. The scheduler process should then decide on the scheduling algorithm and then send these objects according to their colour into consumer lines B, C, and D. The consumer lines B, C, and D should simulate the execution of the process by decreasing their "Burst time" which is generated randomly by the production line A. The sixth process acts a control panel for the whole system, as it takes specified commands from the keyboard and then controls the production speed and synchronization of all system's lines.

3. System Design

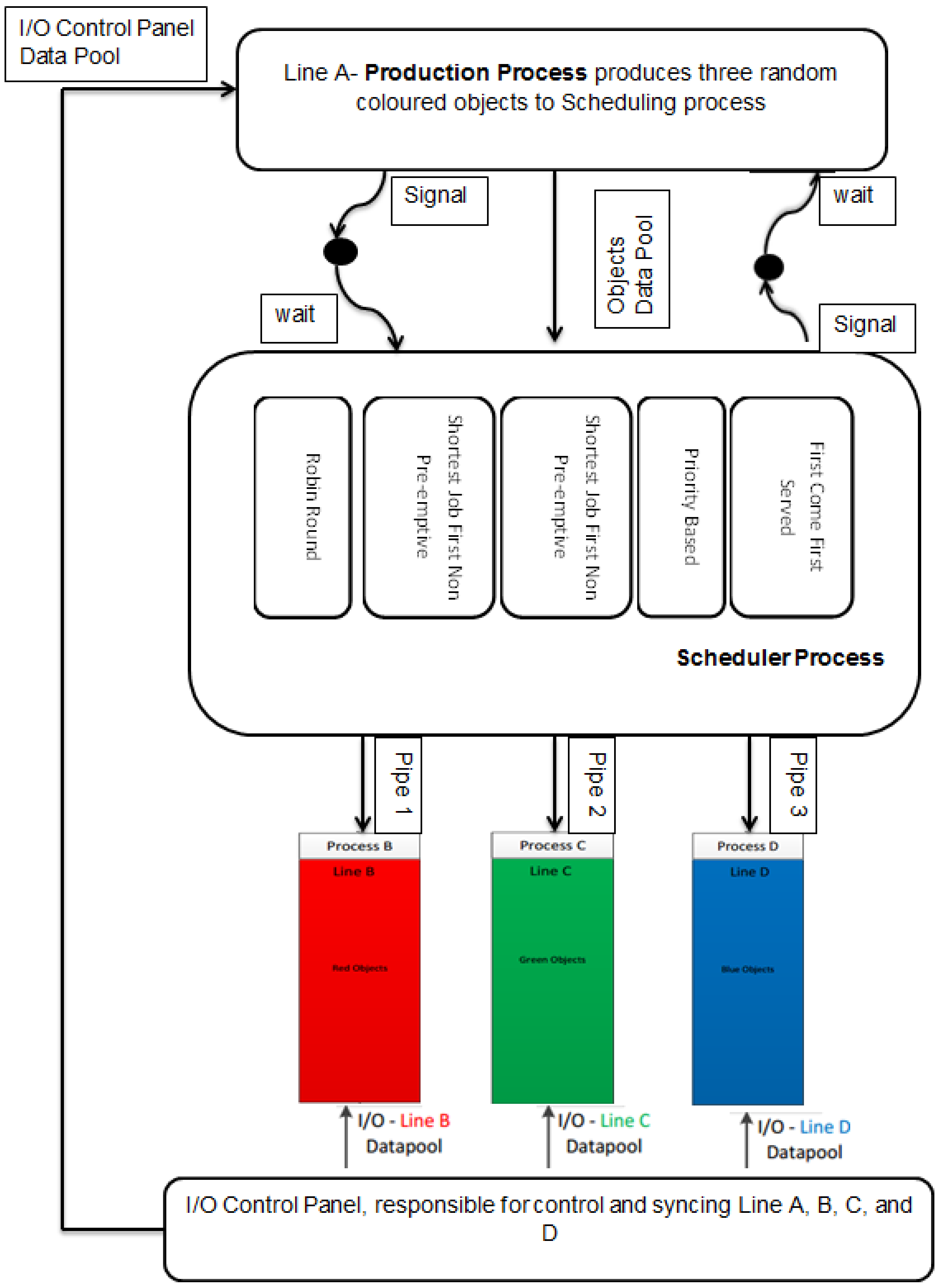


Figure 3. 1 Design for the Conrol System

In our design, the first process is to simulate the production line A (in the projects code it is Q7Parent.cpp) which produces object in three different colours and place them in the scheduler process. The scheduler process then receives these objects and sorts them according to the scheduling algorithm chosen by the user. Scheduler process can run five different algorithms; they are first one is First Come First Served, Priority based scheduling, Shortest Job First Non Pre-emptive, Shortest Job First Pre-emptive, and Robin Round. After objects are scheduled, they are sent to processes for lines B, C, and D according to their colours; where red objects are sent to Line B, green are sent to Line C, and Blue are sent to line D. In each of the processes for lines B, C, and D, objects execution is simulated by decreasing the burst/processing time, and finally all these processes are controlled and synchronized with Control Panel Process.

As shown in Figure 3.1, the first block is Production process which sends objects to the scheduler process via objects datapool. It also has two semaphores (Producer semaphore and Consumer semaphore) that will be used to control the read and write operation on jobs datapool.

The second block is the scheduler process which performs the following:

- ❖ Scheduler objects received from the production process according to the selected algorithm by the user.
- ❖ Implements First Come, First Served (FCFS), which is a simple scheduling algorithm, FCFS simply queues objects in the order that they arrive in the ready queue. In FCFS the throughput can be low, since long processes can hold the CPU. Also, the turnaround time, waiting time and response time can be high.
- ❖ Implements Priority Scheduling where the scheduler assigns a fixed priority rank to every object and the scheduler arranges the objects in the ready queue in order of their priority. Lower-priority objects get interrupted by incoming higher-priority objects. Waiting time and response time depend on the priority of the object. Higher-priority objects have smaller waiting and response times. Starvation of lower-priority objects is possible with large numbers of high-priority objects queuing for CPU time.
- ❖ Implements Shortest-Job-First (SJR) Non Pre-emptive Scheduling, where it associates with each object the length of its next CPU burst and uses these lengths to schedule the object with the shortest time. Once CPU is given to the object it cannot be preempted until completes its CPU burst.
- ❖ Implements Shortest-Job-First (SJR) Pre-emptive Scheduling, which is similar to its Non - Preemptive version, but the difference, is that if a new process arrives with CPU burst length less than remaining time of current executing.
- ❖ Implements Robin Round, in which each object is provided with a fixed time to execute called quantum. Once a object is executed for the given time period. Object is preempted and other Object executes for given time period. Context switching is used to save states of preempted Objects.

- Multilevel queue scheduling (not implemented), where multiple queues are maintained for Objects. Each queue can have its own scheduling algorithms. Priorities are assigned to each queue.

The processes for lines B, C, and D, receive objects via pipelines. Pipeline was chosen over data pool for communication between scheduler and processing lines to avoid two possible problems. One of them arises when the line process try to read object from the scheduler at the same time the scheduler is updating it, in this case the line process will read corrupted data that is not completely updated. The second problem is the synchronization problem between the scheduler process and other lines. The scheduler process can run faster than the other lines and in this case it will overwrite the previous object that has been stored in the data pool. These two problems can be solved by using semaphores that preventing the scheduler process from updating the pool at the same time the line process reading from it and vice versa. However, scheduler process will not be able to dispatch a new object until the previous one in the data pool is consumed by one of the lines processes. This will slow down the system and eventually the queue will become full and leading the producer thread to stop feeding objects.

Finally, The I/O control process is designed to accept input command from the keyboard and send it to the production process or one of the other line processes to respond to it. The communications between the I/O control process and the other processes is done via the data pool by creating a data pool with four fields one for each process. Every time a user enters a command from the keyboard, the I/O process checks that command and stores it in the field associated to that process that the command is intended to. The other processes also create a data pool and keep checking it continuously to see if there is a command in their fields. The probability of getting a synchronization problem by using a data pool to communicate with other process is small in this case because the processes checking frequency to the data pool is much faster than the I/O process since the I/O process works on the user speed that inputs the command through the keyboard. Just to prevent the situation that the processes keep reading the same command from the data pool, they clear their field directly after reading the command.

4. System Implementation

5. System Testing

5.1 Testing Aims

Software testing is an investigation conducted to provide customers with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects).

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- ❖ Meets the requirements that guided its design and development,
- ❖ Responds correctly to all kinds of inputs,
- ❖ Performs its functions within an acceptable time,
- ❖ Is sufficiently usable,
- ❖ Can be installed and run in its intended environments, and
- ❖ Achieves the general result its customers' desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects). The job of testing is an iterative process as when one bug is fixed; it can illuminate other, deeper bugs, or can even create new ones (Gelperin & Hetzel, 1998).

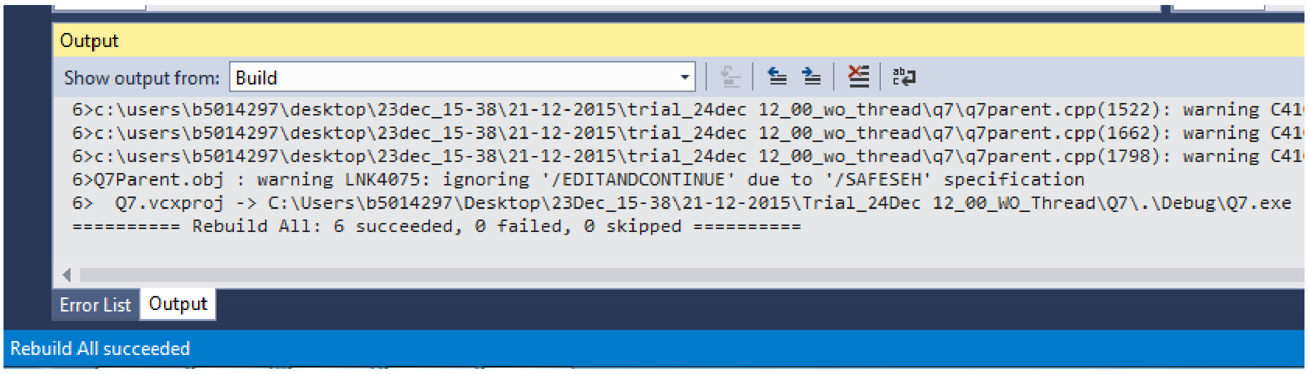
5.2 Tests Performed

Summary for the most important tests performed:

- ✓ Program building and running without any errors.
- ✓ Program accepts input from the user, and run according to user specifications.
- ✓ Program accepts manual and automatic entry for object production.
- ✓ Program creates three console windows corresponding to the three production lines for our control system.
- ✓ Program prints objects executed according to its colour specified.
- ✓ Program creates a console window as a control panel for our control system.
- ✓ Program executes each of the following scheduling algorithms: First Come First Served, Priority Based Algorithm, Shortest Job First Non Pre-emptive, Shortest Job First Pre-emptive, and Robin Round.
- ✓ Program can be scaled to more than three production lines

In the text below, results of program testing will be discussed in more details with screen shots.

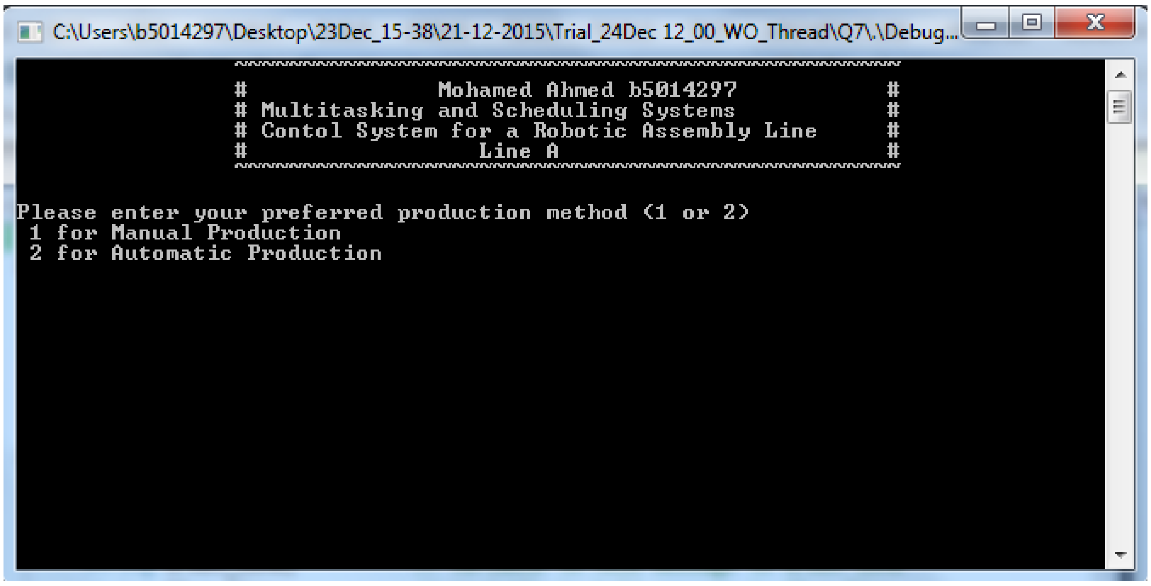
As shown in Figure 5.1, the program was compiled successfully.



```
6>c:\users\b5014297\desktop\23dec_15-38\21-12-2015\trial_24dec 12_00_wo_thread\q7\q7parent.cpp(1522): warning C41
6>c:\users\b5014297\desktop\23dec_15-38\21-12-2015\trial_24dec 12_00_wo_thread\q7\q7parent.cpp(1662): warning C41
6>c:\users\b5014297\desktop\23dec_15-38\21-12-2015\trial_24dec 12_00_wo_thread\q7\q7parent.cpp(1798): warning C41
6>Q7Parent.obj : warning LNK4075: ignoring '/EDITANDCONTINUE' due to '/SAFESEH' specification
6>  Q7.vcxproj -> C:\Users\b5014297\Desktop\23Dec_15-38\21-12-2015\Trial_24Dec 12_00_WO_Thread\Q7\.\Debug\Q7.exe
========== Rebuild All: 6 succeeded, 0 failed, 0 skipped ==========
```

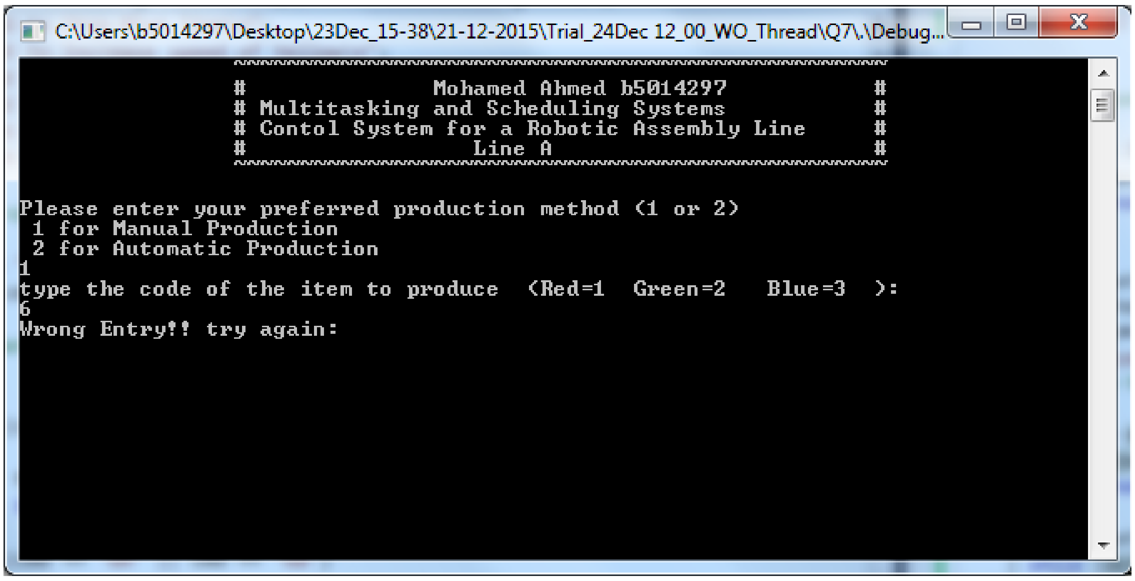
Rebuild All succeeded

Figure 5. 1 Successful Compilation of the project


C:\Users\b5014297\Desktop\23Dec_15-38\21-12-2015\Trial_24Dec 12_00_WO_Thread\Q7\.\Debug...

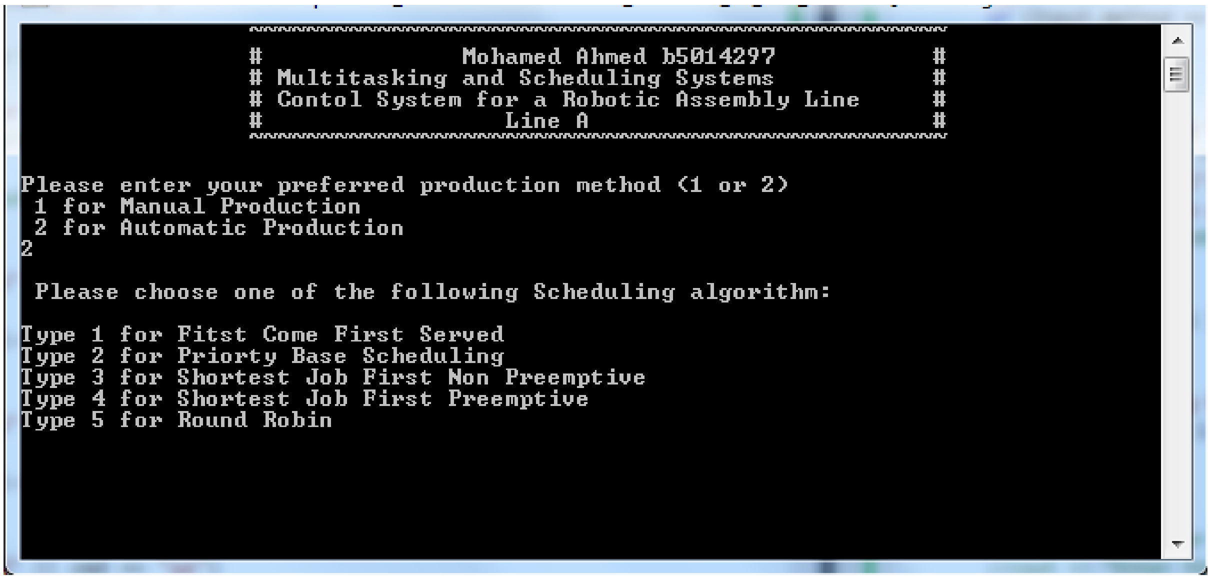
```
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                #              Mohamed Ahmed b5014297             #
                # Multitasking and Scheduling Systems             #
                # Contol System for a Robotic Assembly Line       #
                #                Line A                           #
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Please enter your preferred production method (1 or 2)
 1 for Manual Production
 2 for Automatic Production
```

Figure 5. 2: Production Line A, asking user to select production method


C:\Users\b5014297\Desktop\23Dec_15-38\21-12-2015\Trial_24Dec 12_00_WO_Thread\Q7\.\Debug...

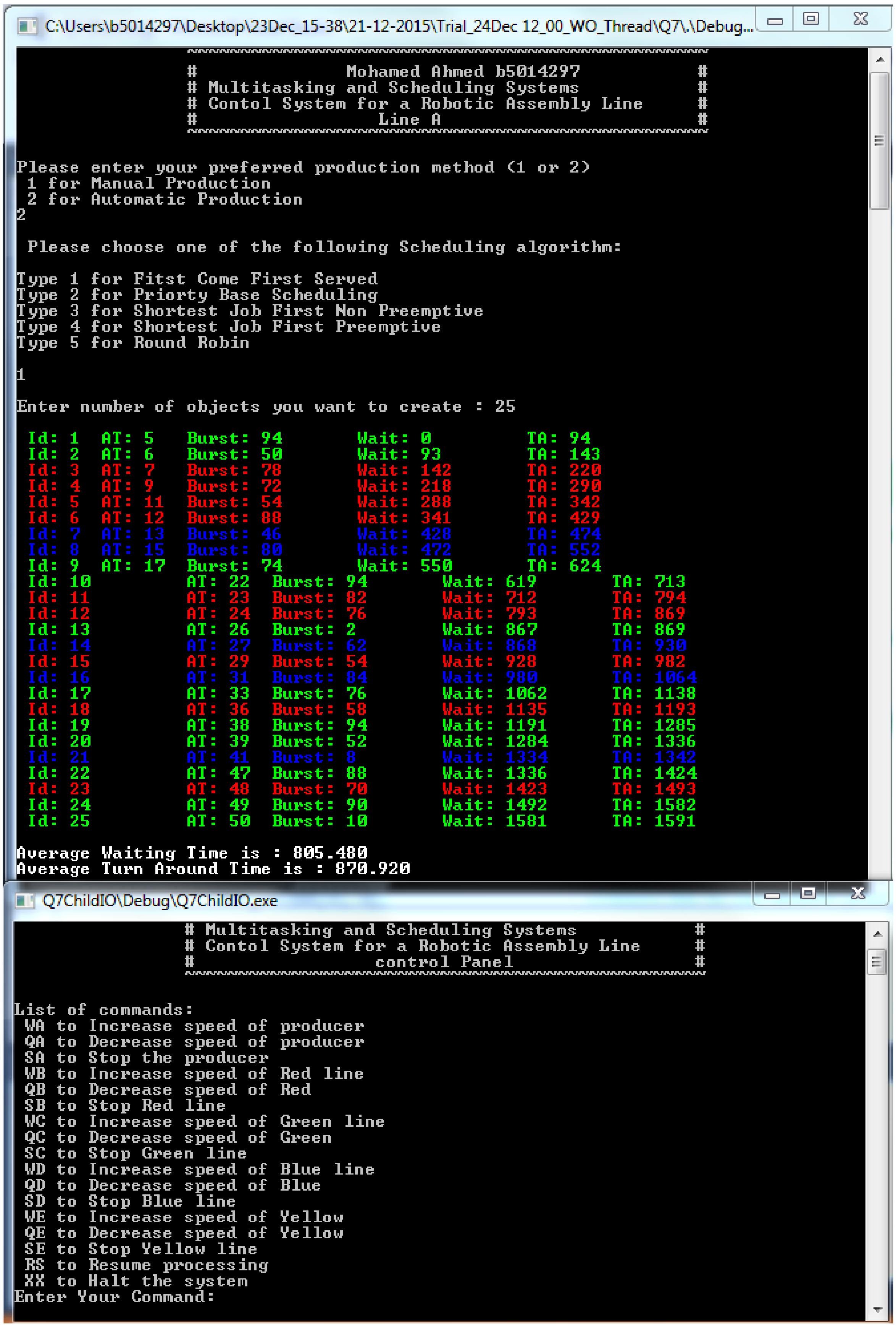
```
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                #               Mohamed Ahmed b5014297          #
                # Multitasking and Scheduling Systems           #
                # Contol System for a Robotic Assembly Line     #
                #                 Line A                        #
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Please enter your preferred production method (1 or 2)
 1 for Manual Production
 2 for Automatic Production
1
type the code of the item to produce  (Red=1  Green=2   Blue=3  ):
6
Wrong Entry!! try again:
```

Figure 5. 3 User choose manual entry, then attempted to enter invalid data



```
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                #               Mohamed Ahmed b5014297          #
                # Multitasking and Scheduling Systems           #
                # Contol System for a Robotic Assembly Line     #
                #                 Line A                        #
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Please enter your preferred production method (1 or 2)
 1 for Manual Production
 2 for Automatic Production
2

 Please choose one of the following Scheduling algorithm:

Type 1 for Fitst Come First Served
Type 2 for Priorty Base Scheduling
Type 3 for Shortest Job First Non Preemptive
Type 4 for Shortest Job First Preemptive
Type 5 for Round Robin
```

Figure 5. 4 List of Scheduling Algorithm to choose from

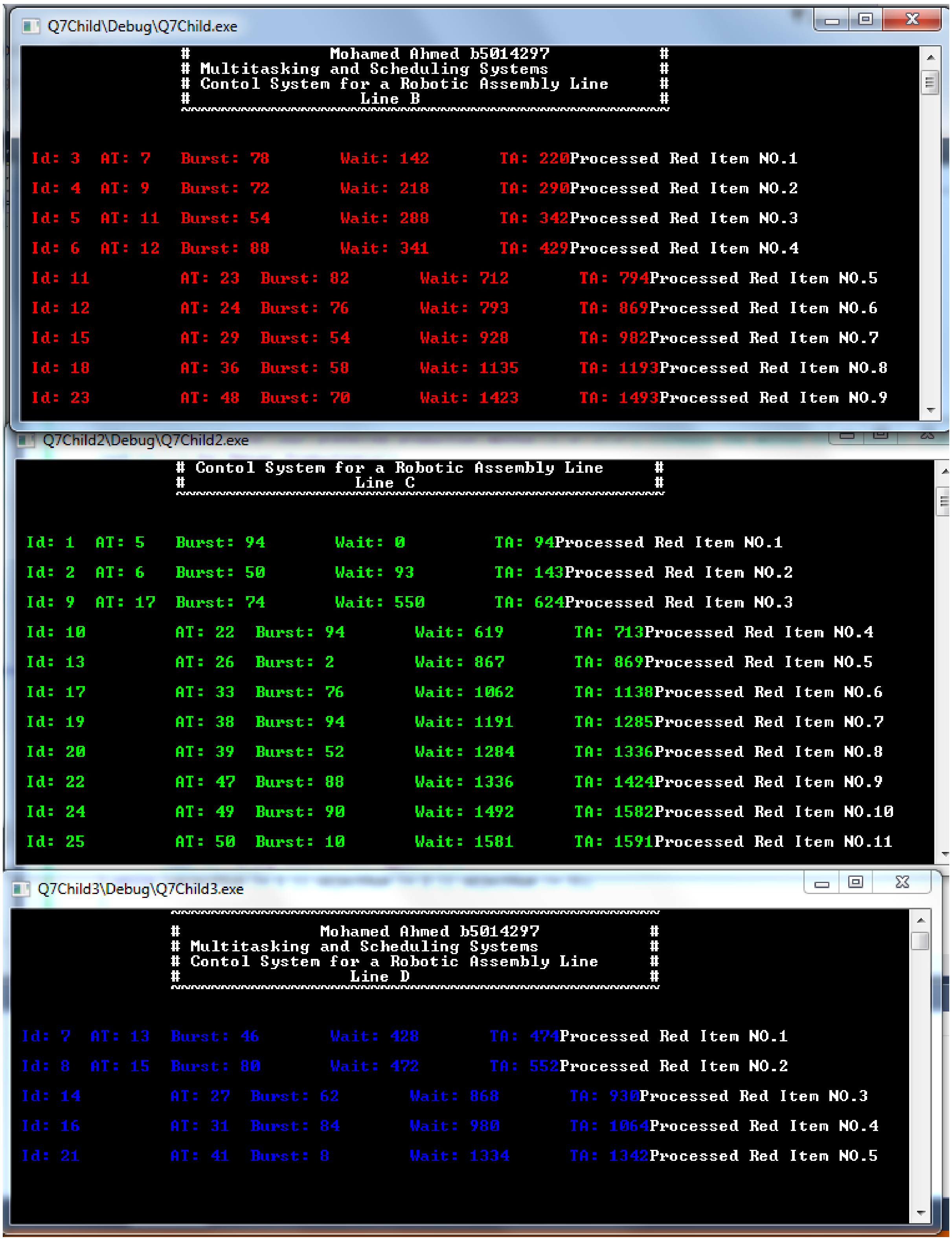

C:\Users\b5014297\Desktop\23Dec_15-38\21-12-2015\Trial_24Dec 12_00_WO_Thread\Q7\.\Debug...

```
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                #             Mohamed Ahmed b5014297             #
                # Multitasking and Scheduling Systems            #
                # Contol System for a Robotic Assembly Line      #
                #                 Line A                         #
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Please enter your preferred production method (1 or 2)
 1 for Manual Production
 2 for Automatic Production
2

 Please choose one of the following Scheduling algorithm:

Type 1 for Fitst Come First Served
Type 2 for Priorty Base Scheduling
Type 3 for Shortest Job First Non Preemptive
Type 4 for Shortest Job First Preemptive
Type 5 for Round Robin

1

Enter number of objects you want to create : 25

 Id: 1  AT: 5   Burst: 94       Wait: 0         TA: 94
 Id: 2  AT: 6   Burst: 50       Wait: 93        TA: 143
 Id: 3  AT: 7   Burst: 78       Wait: 142       TA: 220
 Id: 4  AT: 9   Burst: 72       Wait: 218       TA: 290
 Id: 5  AT: 11  Burst: 54       Wait: 288       TA: 342
 Id: 6  AT: 12  Burst: 88       Wait: 341       TA: 429
 Id: 7  AT: 13  Burst: 46       Wait: 428       TA: 474
 Id: 8  AT: 15  Burst: 80       Wait: 472       TA: 552
 Id: 9  AT: 17  Burst: 74       Wait: 550       TA: 624
 Id: 10         AT: 22  Burst: 94       Wait: 619       TA: 713
 Id: 11         AT: 23  Burst: 82       Wait: 712       TA: 794
 Id: 12         AT: 24  Burst: 76       Wait: 793       TA: 869
 Id: 13         AT: 26  Burst: 2        Wait: 867       TA: 869
 Id: 14         AT: 27  Burst: 62       Wait: 868       TA: 930
 Id: 15         AT: 29  Burst: 54       Wait: 928       TA: 982
 Id: 16         AT: 31  Burst: 84       Wait: 980       TA: 1064
 Id: 17         AT: 33  Burst: 76       Wait: 1062      TA: 1138
 Id: 18         AT: 36  Burst: 58       Wait: 1135      TA: 1193
 Id: 19         AT: 38  Burst: 94       Wait: 1191      TA: 1285
 Id: 20         AT: 39  Burst: 52       Wait: 1284      TA: 1336
 Id: 21         AT: 41  Burst: 8        Wait: 1334      TA: 1342
 Id: 22         AT: 47  Burst: 88       Wait: 1336      TA: 1424
 Id: 23         AT: 48  Burst: 70       Wait: 1423      TA: 1493
 Id: 24         AT: 49  Burst: 90       Wait: 1492      TA: 1582
 Id: 25         AT: 50  Burst: 10       Wait: 1581      TA: 1591

Average Waiting Time is : 805.480
Average Turn Around Time is : 870.920
```

Q7ChildIO\Debug\Q7ChildIO.exe

```
                # Multitasking and Scheduling Systems            #
                # Contol System for a Robotic Assembly Line      #
                #                 control Panel                  #
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

List of commands:
 WA to Increase speed of producer
 QA to Decrease speed of producer
 SA to Stop the producer
 WB to Increase speed of Red line
 QB to Decrease speed of Red
 SB to Stop Red line
 WC to Increase speed of Green line
 QC to Decrease speed of Green
 SC to Stop Green line
 WD to Increase speed of Blue line
 QD to Decrease speed of Blue
 SD to Stop Blue line
 WE to Increase speed of Yellow
 QE to Decrease speed of Yellow
 SE to Stop Yellow line
 RS to Resume processing
 XX to Halt the system
Enter Your Command:
```

Figure 5. 5: Simulation and results for running First Come First Served Algorithm


Q7Child\Debug\Q7Child.exe

```
                 #              Mohamed Ahmed b5014297              #
                 # Multitasking and Scheduling Systems              #
                 # Contol System for a Robotic Assembly Line        #
                 #                   Line B                         #
                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Id: 3   AT: 7   Burst: 78       Wait: 142       TA: 220Processed Red Item NO.1
Id: 4   AT: 9   Burst: 72       Wait: 218       TA: 290Processed Red Item NO.2
Id: 5   AT: 11  Burst: 54       Wait: 288       TA: 342Processed Red Item NO.3
Id: 6   AT: 12  Burst: 88       Wait: 341       TA: 429Processed Red Item NO.4
Id: 11          AT: 23  Burst: 82       Wait: 712       TA: 794Processed Red Item NO.5
Id: 12          AT: 24  Burst: 76       Wait: 793       TA: 869Processed Red Item NO.6
Id: 15          AT: 29  Burst: 54       Wait: 928       TA: 982Processed Red Item NO.7
Id: 18          AT: 36  Burst: 58       Wait: 1135      TA: 1193Processed Red Item NO.8
Id: 23          AT: 48  Burst: 70       Wait: 1423      TA: 1493Processed Red Item NO.9
```

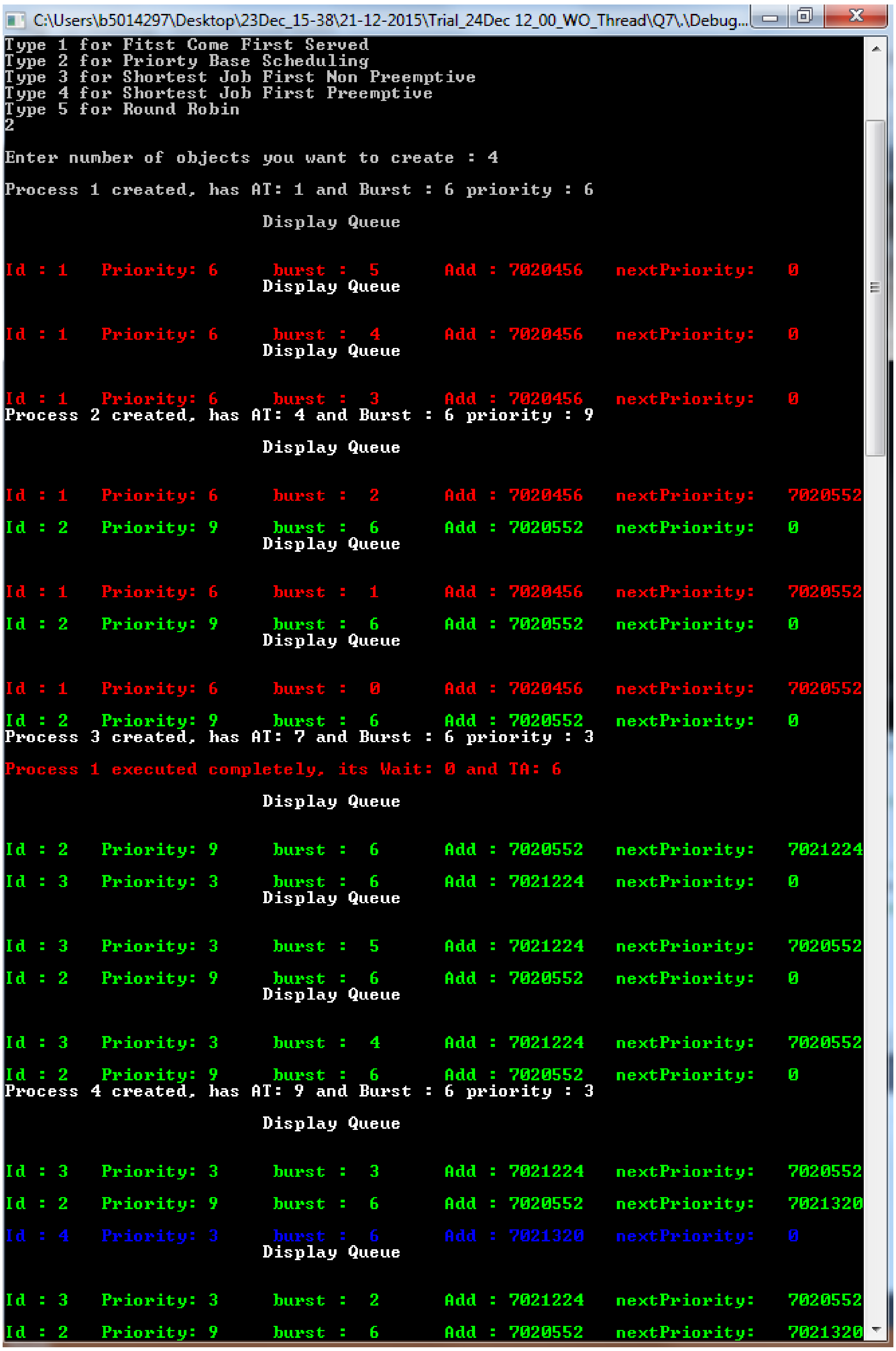
Q7Child2\Debug\Q7Child2.exe

```
                 # Contol System for a Robotic Assembly Line        #
                 #                   Line C                         #
                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Id: 1   AT: 5   Burst: 94       Wait: 0         TA: 94Processed Red Item NO.1
Id: 2   AT: 6   Burst: 50       Wait: 93        TA: 143Processed Red Item NO.2
Id: 9   AT: 17  Burst: 74       Wait: 550       TA: 624Processed Red Item NO.3
Id: 10          AT: 22  Burst: 94       Wait: 619       TA: 713Processed Red Item NO.4
Id: 13          AT: 26  Burst: 2        Wait: 867       TA: 869Processed Red Item NO.5
Id: 17          AT: 33  Burst: 76       Wait: 1062      TA: 1138Processed Red Item NO.6
Id: 19          AT: 38  Burst: 94       Wait: 1191      TA: 1285Processed Red Item NO.7
Id: 20          AT: 39  Burst: 52       Wait: 1284      TA: 1336Processed Red Item NO.8
Id: 22          AT: 47  Burst: 88       Wait: 1336      TA: 1424Processed Red Item NO.9
Id: 24          AT: 49  Burst: 90       Wait: 1492      TA: 1582Processed Red Item NO.10
Id: 25          AT: 50  Burst: 10       Wait: 1581      TA: 1591Processed Red Item NO.11
```

Q7Child3\Debug\Q7Child3.exe

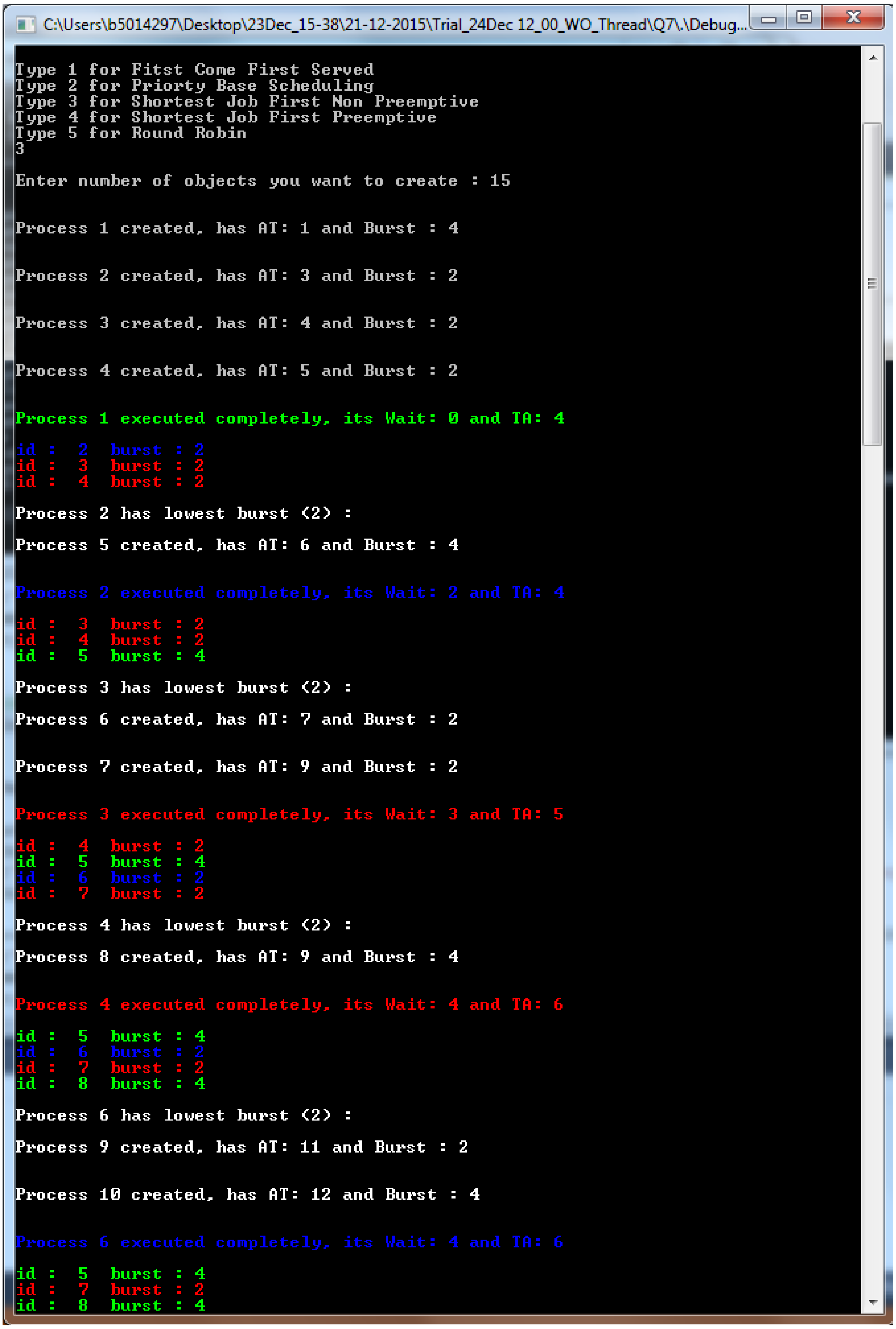
```
                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                 #              Mohamed Ahmed b5014297              #
                 # Multitasking and Scheduling Systems              #
                 # Contol System for a Robotic Assembly Line        #
                 #                   Line D                         #
                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Id: 7   AT: 13  Burst: 46       Wait: 428       TA: 474Processed Red Item NO.1
Id: 8   AT: 15  Burst: 80       Wait: 472       TA: 552Processed Red Item NO.2
Id: 14          AT: 27  Burst: 62       Wait: 868       TA: 930Processed Red Item NO.3
Id: 16          AT: 31  Burst: 84       Wait: 980       TA: 1064Processed Red Item NO.4
Id: 21          AT: 41  Burst: 8        Wait: 1334      TA: 1342Processed Red Item NO.5
```

Figure 5. 6: Print of FCFS on Line B, C, and D


C:\Users\b5014297\Desktop\23Dec_15-38\21-12-2015\Trial_24Dec 12_00_WO_Thread\Q7\.\Debug...

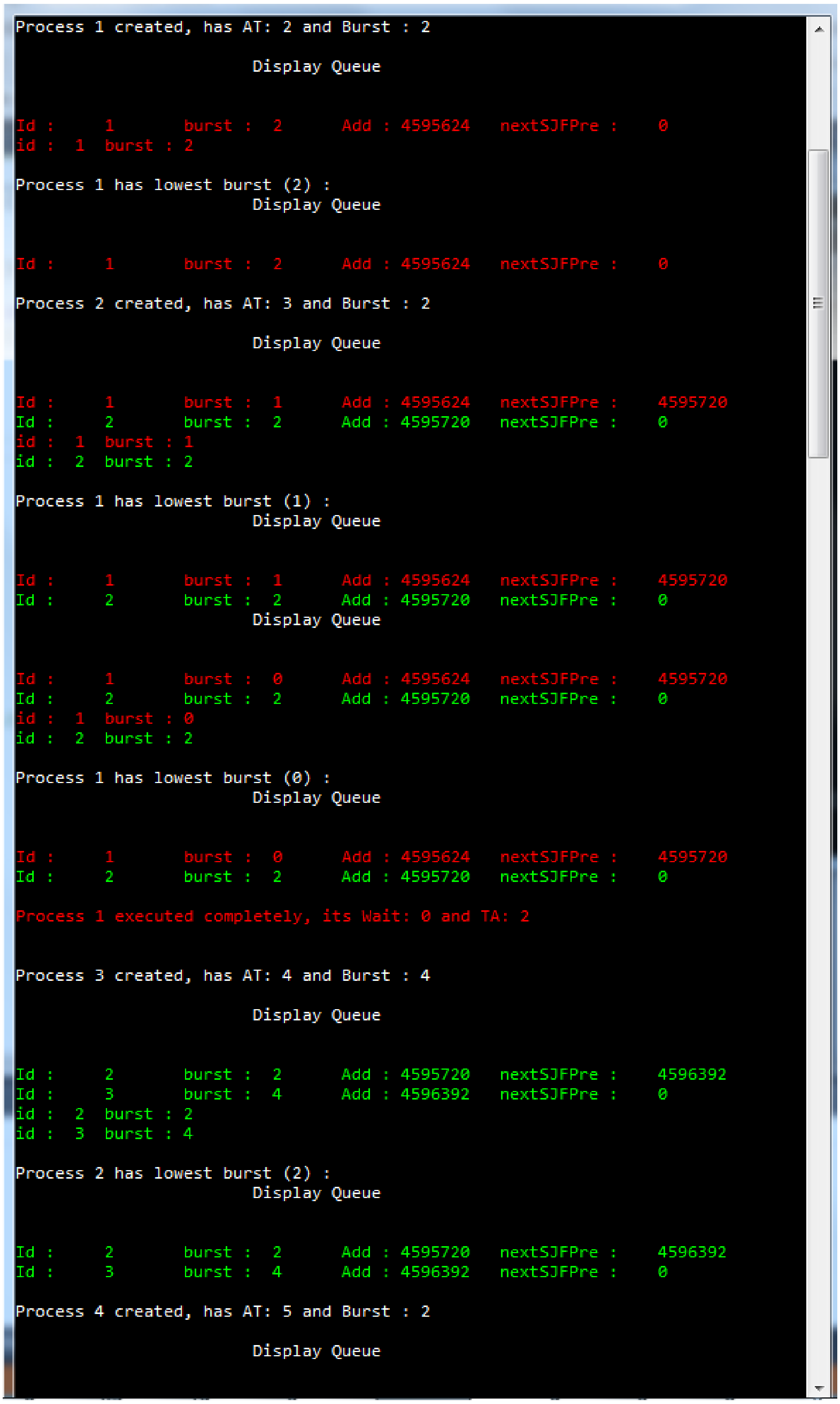
```
Type 1 for Fitst Come First Served
Type 2 for Priorty Base Scheduling
Type 3 for Shortest Job First Non Preemptive
Type 4 for Shortest Job First Preemptive
Type 5 for Round Robin
2

Enter number of objects you want to create : 4

Process 1 created, has AT: 1 and Burst : 6 priority : 6

                        Display Queue


Id : 1    Priority: 6     burst :  5      Add : 7020456    nextPriority:    0
                        Display Queue


Id : 1    Priority: 6     burst :  4      Add : 7020456    nextPriority:    0
                        Display Queue


Id : 1    Priority: 6     burst :  3      Add : 7020456    nextPriority:    0
Process 2 created, has AT: 4 and Burst : 6 priority : 9

                        Display Queue


Id : 1    Priority: 6     burst :  2      Add : 7020456    nextPriority:    7020552

Id : 2    Priority: 9     burst :  6      Add : 7020552    nextPriority:    0
                        Display Queue


Id : 1    Priority: 6     burst :  1      Add : 7020456    nextPriority:    7020552

Id : 2    Priority: 9     burst :  6      Add : 7020552    nextPriority:    0
                        Display Queue


Id : 1    Priority: 6     burst :  0      Add : 7020456    nextPriority:    7020552

Id : 2    Priority: 9     burst :  6      Add : 7020552    nextPriority:    0
Process 3 created, has AT: 7 and Burst : 6 priority : 3

Process 1 executed completely, its Wait: 0 and TA: 6

                        Display Queue


Id : 2    Priority: 9     burst :  6      Add : 7020552    nextPriority:    7021224

Id : 3    Priority: 3     burst :  6      Add : 7021224    nextPriority:    0
                        Display Queue


Id : 3    Priority: 3     burst :  5      Add : 7021224    nextPriority:    7020552

Id : 2    Priority: 9     burst :  6      Add : 7020552    nextPriority:    0
                        Display Queue


Id : 3    Priority: 3     burst :  4      Add : 7021224    nextPriority:    7020552

Id : 2    Priority: 9     burst :  6      Add : 7020552    nextPriority:    0
Process 4 created, has AT: 9 and Burst : 6 priority : 3

                        Display Queue


Id : 3    Priority: 3     burst :  3      Add : 7021224    nextPriority:    7020552

Id : 2    Priority: 9     burst :  6      Add : 7020552    nextPriority:    7021320

Id : 4    Priority: 3     burst :  6      Add : 7021320    nextPriority:    0
                        Display Queue


Id : 3    Priority: 3     burst :  2      Add : 7021224    nextPriority:    7020552

Id : 2    Priority: 9     burst :  6      Add : 7020552    nextPriority:    7021320
```

Figure 5. 7 Simulation and results for running Priority Based Queue



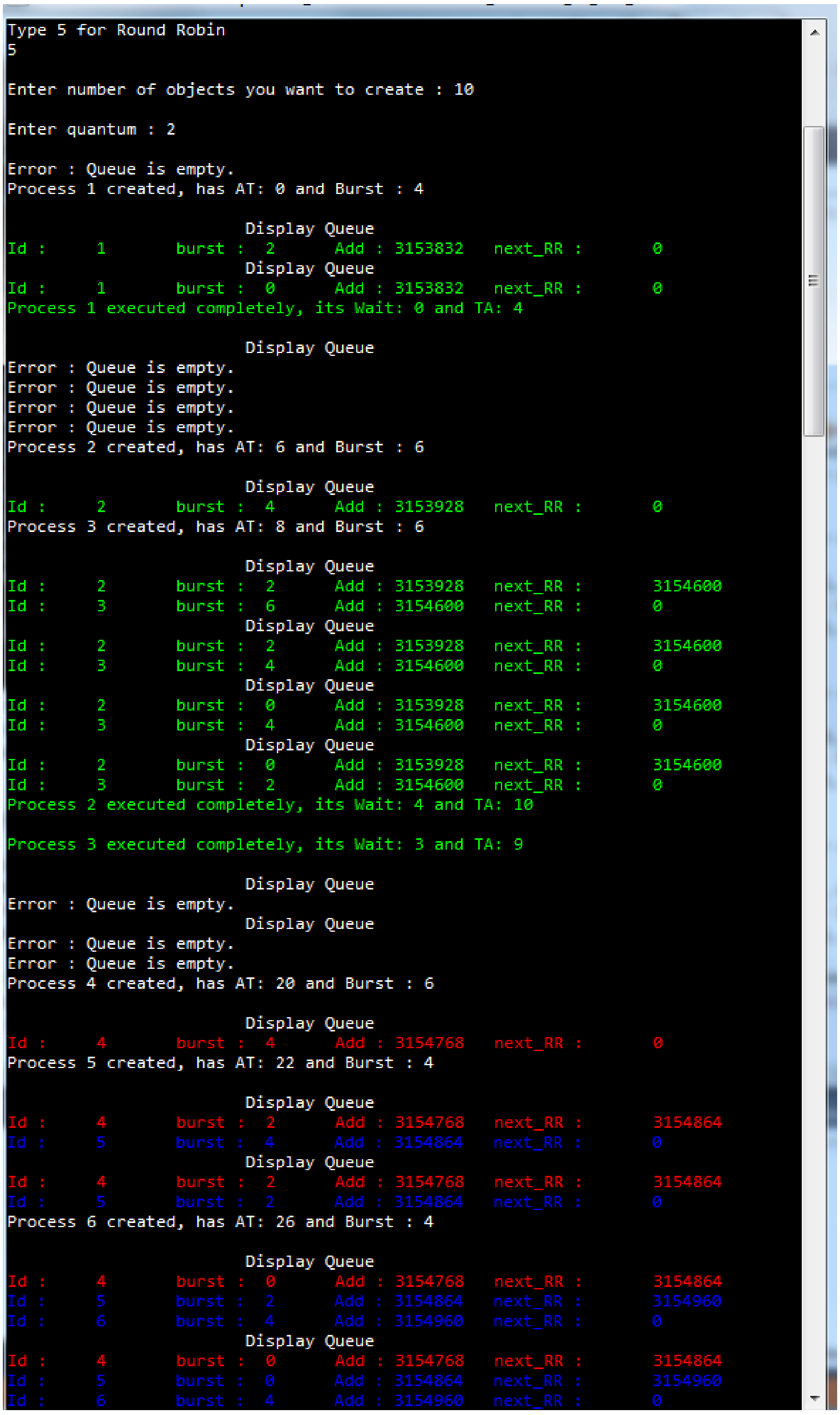
```
C:\Users\b5014297\Desktop\23Dec_15-38\21-12-2015\Trial_24Dec 12_00_WO_Thread\Q7\.\Debug...

Type 1 for Fitst Come First Served
Type 2 for Priorty Base Scheduling
Type 3 for Shortest Job First Non Preemptive
Type 4 for Shortest Job First Preemptive
Type 5 for Round Robin
3

Enter number of objects you want to create : 15


Process 1 created, has AT: 1 and Burst : 4


Process 2 created, has AT: 3 and Burst : 2


Process 3 created, has AT: 4 and Burst : 2


Process 4 created, has AT: 5 and Burst : 2


Process 1 executed completely, its Wait: 0 and TA: 4

id :  2  burst : 2
id :  3  burst : 2
id :  4  burst : 2

Process 2 has lowest burst (2) :

Process 5 created, has AT: 6 and Burst : 4


Process 2 executed completely, its Wait: 2 and TA: 4

id :  3  burst : 2
id :  4  burst : 2
id :  5  burst : 4

Process 3 has lowest burst (2) :

Process 6 created, has AT: 7 and Burst : 2


Process 7 created, has AT: 9 and Burst : 2


Process 3 executed completely, its Wait: 3 and TA: 5

id :  4  burst : 2
id :  5  burst : 4
id :  6  burst : 2
id :  7  burst : 2

Process 4 has lowest burst (2) :

Process 8 created, has AT: 9 and Burst : 4


Process 4 executed completely, its Wait: 4 and TA: 6

id :  5  burst : 4
id :  6  burst : 2
id :  7  burst : 2
id :  8  burst : 4

Process 6 has lowest burst (2) :

Process 9 created, has AT: 11 and Burst : 2


Process 10 created, has AT: 12 and Burst : 4


Process 6 executed completely, its Wait: 4 and TA: 6

id :  5  burst : 4
id :  7  burst : 2
id :  8  burst : 4
```

Figure 5. 8: Simulation and results for running Shortest Job First Non Pre-emptive



```
Process 1 created, has AT: 2 and Burst : 2

                        Display Queue


Id :     1       burst :  2      Add : 4595624   nextSJFPre :    0
id :  1  burst : 2

Process 1 has lowest burst (2) :
                        Display Queue


Id :     1       burst :  2      Add : 4595624   nextSJFPre :    0

Process 2 created, has AT: 3 and Burst : 2

                        Display Queue


Id :     1       burst :  1      Add : 4595624   nextSJFPre :    4595720
Id :     2       burst :  2      Add : 4595720   nextSJFPre :    0
id :  1  burst : 1
id :  2  burst : 2

Process 1 has lowest burst (1) :
                        Display Queue


Id :     1       burst :  1      Add : 4595624   nextSJFPre :    4595720
Id :     2       burst :  2      Add : 4595720   nextSJFPre :    0
                        Display Queue


Id :     1       burst :  0      Add : 4595624   nextSJFPre :    4595720
Id :     2       burst :  2      Add : 4595720   nextSJFPre :    0
id :  1  burst : 0
id :  2  burst : 2

Process 1 has lowest burst (0) :
                        Display Queue


Id :     1       burst :  0      Add : 4595624   nextSJFPre :    4595720
Id :     2       burst :  2      Add : 4595720   nextSJFPre :    0

Process 1 executed completely, its Wait: 0 and TA: 2


Process 3 created, has AT: 4 and Burst : 4

                        Display Queue


Id :     2       burst :  2      Add : 4595720   nextSJFPre :    4596392
Id :     3       burst :  4      Add : 4596392   nextSJFPre :    0
id :  2  burst : 2
id :  3  burst : 4

Process 2 has lowest burst (2) :
                        Display Queue


Id :     2       burst :  2      Add : 4595720   nextSJFPre :    4596392
Id :     3       burst :  4      Add : 4596392   nextSJFPre :    0

Process 4 created, has AT: 5 and Burst : 2

                        Display Queue
```

Figure 5. 9: Simulation and results for running Shortest Job First Pre-emptive



```
Type 5 for Round Robin
5

Enter number of objects you want to create : 10

Enter quantum : 2

Error : Queue is empty.
Process 1 created, has AT: 0 and Burst : 4

                        Display Queue
Id :     1        burst :  2       Add : 3153832    next_RR :        0
                        Display Queue
Id :     1        burst :  0       Add : 3153832    next_RR :        0
Process 1 executed completely, its Wait: 0 and TA: 4

                        Display Queue
Error : Queue is empty.
Error : Queue is empty.
Error : Queue is empty.
Error : Queue is empty.
Process 2 created, has AT: 6 and Burst : 6

                        Display Queue
Id :     2        burst :  4       Add : 3153928    next_RR :        0
Process 3 created, has AT: 8 and Burst : 6

                        Display Queue
Id :     2        burst :  2       Add : 3153928    next_RR :        3154600
Id :     3        burst :  6       Add : 3154600    next_RR :        0
                        Display Queue
Id :     2        burst :  2       Add : 3153928    next_RR :        3154600
Id :     3        burst :  4       Add : 3154600    next_RR :        0
                        Display Queue
Id :     2        burst :  0       Add : 3153928    next_RR :        3154600
Id :     3        burst :  4       Add : 3154600    next_RR :        0
                        Display Queue
Id :     2        burst :  0       Add : 3153928    next_RR :        3154600
Id :     3        burst :  2       Add : 3154600    next_RR :        0
Process 2 executed completely, its Wait: 4 and TA: 10

Process 3 executed completely, its Wait: 3 and TA: 9

                        Display Queue
Error : Queue is empty.
                        Display Queue
Error : Queue is empty.
Error : Queue is empty.
Process 4 created, has AT: 20 and Burst : 6

                        Display Queue
Id :     4        burst :  4       Add : 3154768    next_RR :        0
Process 5 created, has AT: 22 and Burst : 4

                        Display Queue
Id :     4        burst :  2       Add : 3154768    next_RR :        3154864
Id :     5        burst :  4       Add : 3154864    next_RR :        0
                        Display Queue
Id :     4        burst :  2       Add : 3154768    next_RR :        3154864
Id :     5        burst :  2       Add : 3154864    next_RR :        0
Process 6 created, has AT: 26 and Burst : 4

                        Display Queue
Id :     4        burst :  0       Add : 3154768    next_RR :        3154864
Id :     5        burst :  2       Add : 3154864    next_RR :        3154960
Id :     6        burst :  4       Add : 3154960    next_RR :        0
                        Display Queue
Id :     4        burst :  0       Add : 3154768    next_RR :        3154864
Id :     5        burst :  0       Add : 3154864    next_RR :        3154960
Id :     6        burst :  4       Add : 3154960    next_RR :        0
```

Figure 5. 10: Simulation and results for running Round Robin Algorithm

6. Conclusion

This report investigated the use of concurrent programming and inter-process communication (IPC) mechanisms to design and implement a simulation for control system production lines. The control system was implemented with the help of RT library that supports the implementation of concurrent processes and use of IPC mechanisms. Several IPCs mechanisms have been used in the implementation with justification in this report. This system enables the user to either enter object manually or automatically. It also implements First Come First Served, non-pre-emptive shortest job first, pre-emptive Priority, pre-emptive Shortest Job First, and Robin. It is also scalable to more than four production lines, as it is just needed to create a new process for the extra line, and accordingly creating associated communication channel to it.

Analysing the result of the implemented scheduling algorithms, it was found that the throughput for FCFS can be low, since long processes can hold the CPU. Also, the turnaround time, waiting time and response time can be high. Also, the lack of prioritization means that as long as every process eventually completes, there is no starvation. In an environment where some processes might not complete, there can be starvation. For pre-emptive SJF, the algorithm is designed for maximum throughput in most scenarios. Waiting time and response time increase as the process's computational requirements increase. Since turnaround time is based on waiting time plus processing time, longer processes are significantly affected by this. Overall waiting time is smaller than FCFS, however since no process has to wait for the termination of the longest process. Also, no particular attention is given to deadlines, the programmer can only attempt to make processes with deadlines as short as possible, and starvation is possible, especially in a busy system with many small processes being run. However, in priority pre-emptive scheduling, waiting time and response time depend on the priority of the object. Higher-priority objects have smaller waiting and response times. Starvation of lower-priority objects is possible with large numbers of high-priority objects queuing for CPU time. Finally Robin Round scheduling, has a balanced throughput between FCFS and SJF, shorter jobs are completed faster than in FCFS and longer processes are completed faster than in SJF. Also, it has good average response time; waiting time is dependent on number of processes, and not average process length. Because of high waiting times, deadlines are rarely met in a pure RR system. Moreover, starvation can never occur, since no priority is given. Order of time unit allocation is based upon process arrival time, similar to FCFS.

It is worth mentioning that approximately 70% of the time spent on this project was spent on trying to interface the RT Library with Visual Studio 2013. It was inevitable not to use RT Library and at the same time it had several issues when using it with VS 2013 (it was found that the RT Library was implemented in 2004). However, it was rewarding to finally make it working after spending significant time on it. Also it was an interesting learning experience that sharpened my debugging skills and how to look and find the missing information. On the other hand, this had adversely affected the time spent on implementing more features in the system, and on the quality of the report. Also, the initial design for the system which is discussed in this report has been changed in the implementation due to the time constraint.

References

GALVIN, Peter B., GAGNE, Greg and SILBERSCHATZ, Abraham (2013). Operating system concepts. John Wiley & Sons, Inc.

GELPERIN, David and HETZEL, Bill (1988). The growth of software testing. Communications of the ACM, 31 (6), 687-695.

KUMAR, Vivek (2012). Comparison of Manual and Automation Testing. International journal of research in science and technology (IJRST), , 2249-0604.

SCHNEIDER, Fred B. (2012). On concurrent programming. Springer Science & Business Media.

STEVENS, W. Richard, FENNER, Bill and RUDOFF, Andrew M. (2004). UNIX network programming. Addison-Wesley Professional. , 1.