1Q Consider the following 4 source codes:

**Source Code 1:**

program My\_Pgm1

i = 1

for n = 1 to 10

x = i + n

next

end

**Source Code 2:**

program My\_Pgm2

i = 10

for n = 1 to 8

x = i + n

next

end

**Source Code 3**:

program My\_Pgm3

i = 10

for n = 1 to 15

x = i + n

next

end

**Source Code 4**:

program My\_Pgm3

i = 5

for n = 1 to 15

x = i + n

next

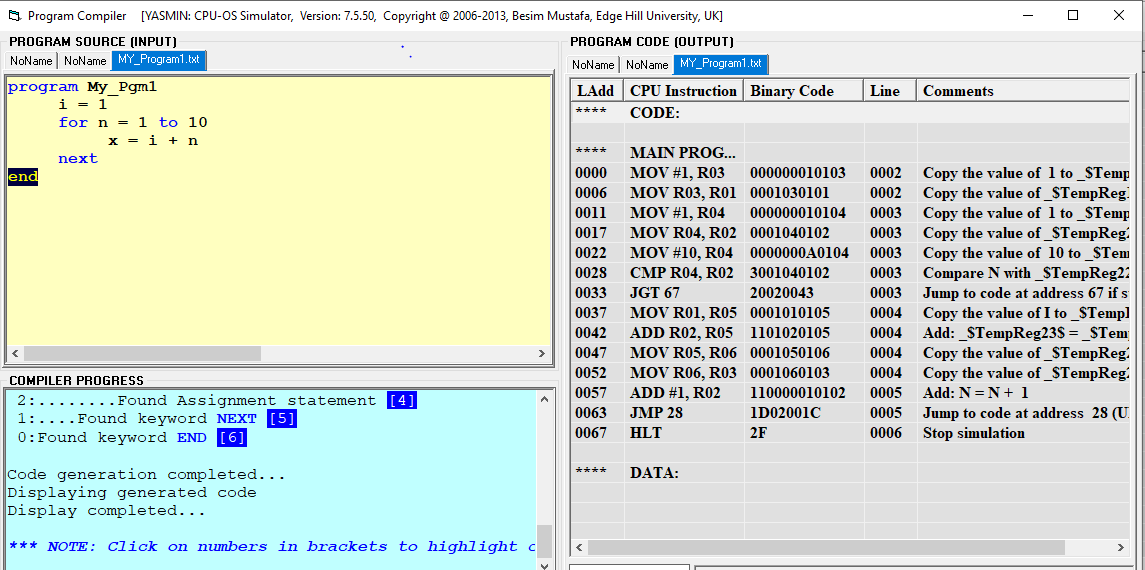
end

Create 4 processes P1, P2, P3 and P4 from source codes 1, 2, 3 and 4 respectively with following properties. Fill up the following table:

|  |  |  |
| --- | --- | --- |
| Scheduling Algorithm : FCFS | | |
| Process | Arrival Time | Waiting time |
| P1 | 0 |  |
| P2 | 0 |  |
| P3 | 0 |  |
| P4 | 0 |  |
| Average waiting time | |  |
| Scheduling Algorithm : Round Robin with time quantum 5 | | |
| Process | Arrival Time | Waiting time |
| P1 | 0 |  |
| P2 | 0 |  |
| P3 | 0 |  |
| P4 | 0 |  |
| Average waiting time | |  |
| Scheduling Algorithm : Round Robin with time quantum 10 | | |
| Process | Arrival Time | Waiting time |
| P1 | 0 |  |
| P2 | 0 |  |
| P3 | 0 |  |
| P4 | 0 |  |
| Average waiting time | |  |
| Out of three cases, which one is better and why? | | |

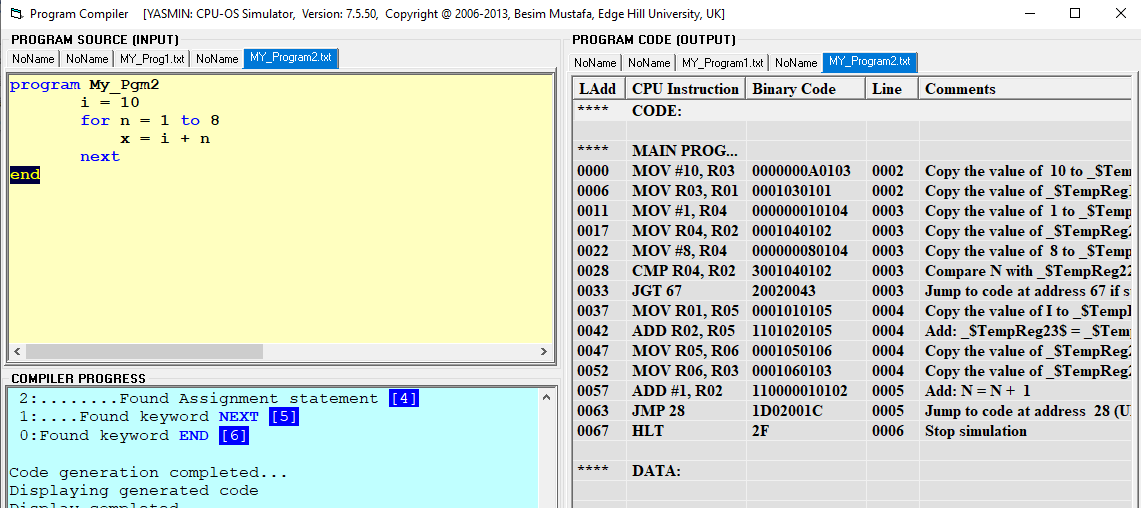
Solution:-

Loading and compiling program 1



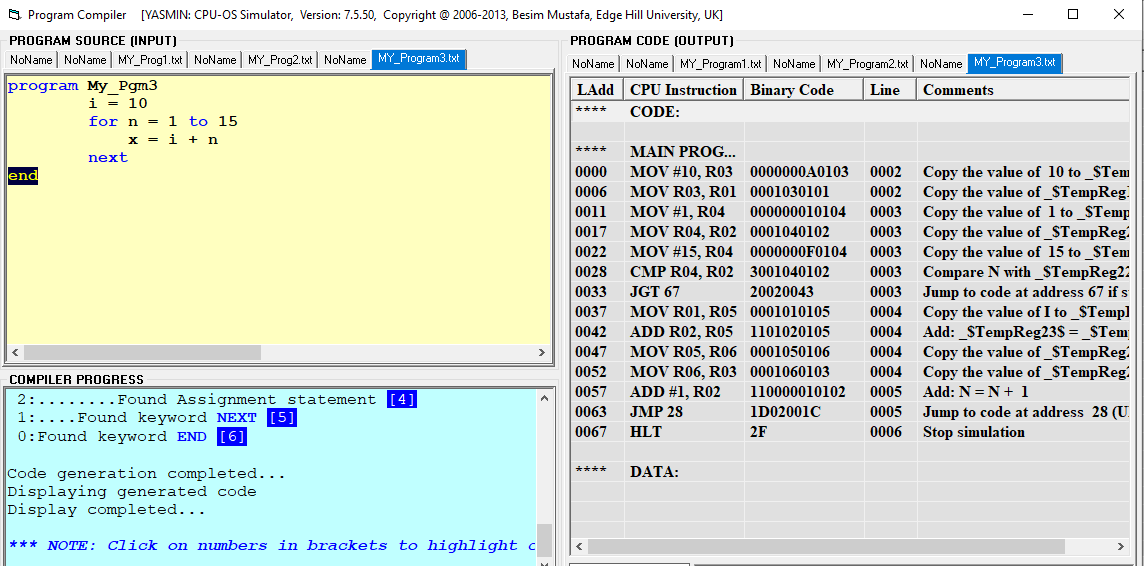
BITS ID: 2019WA15586

Loading and compiling program 2:



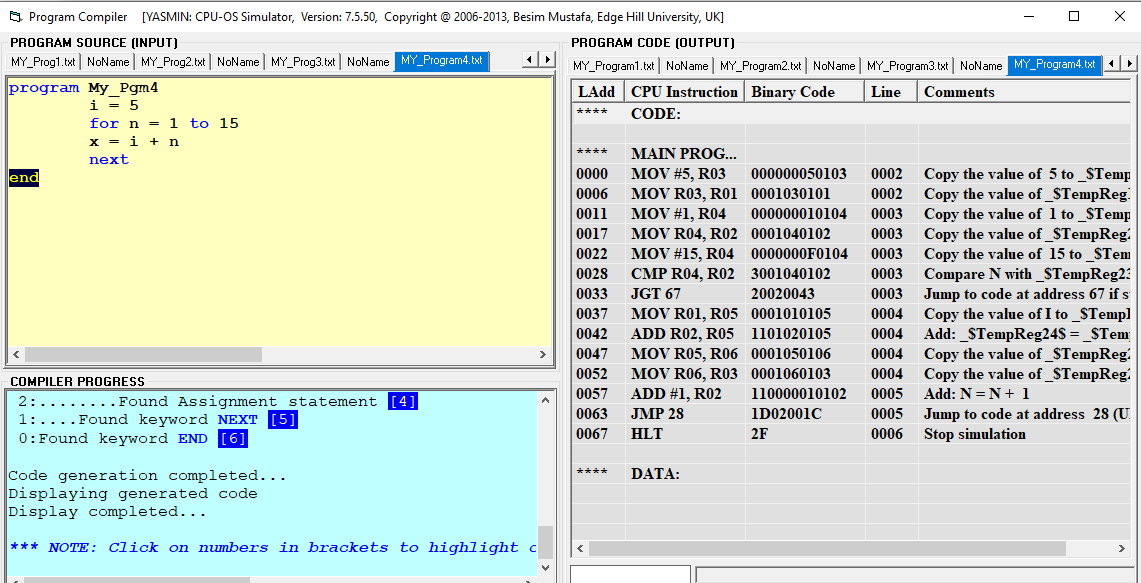
BITS ID: 2019WA15586

Loading and compiling program 3



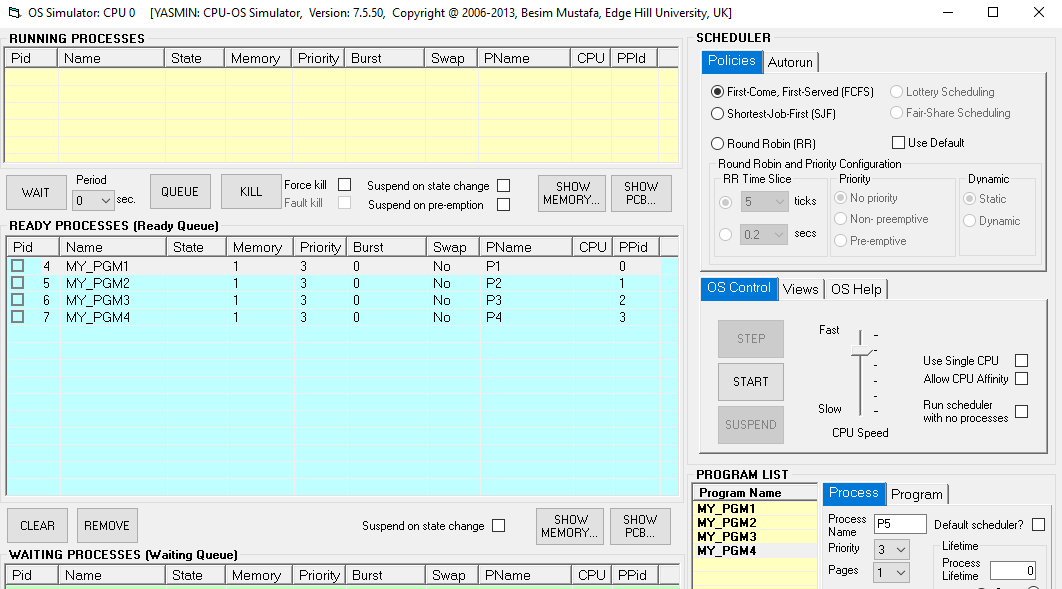
BITS ID: 2019WA15586

Loading and compiling program 4



BITS ID: 2019WA15586

Below snippet shows the program process created

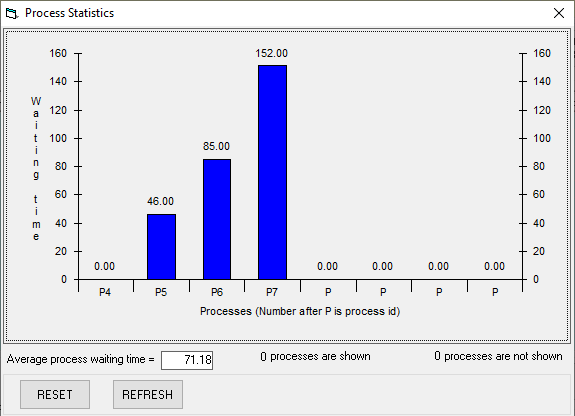


BITS ID: 2019WA15586

**Output of waiting time when scheduled as FCFS(First come , First Served)**

|  |  |  |
| --- | --- | --- |
| Scheduling Algorithm : FCFS | | |
| Process | Arrival Time | Waiting time |
| P1 | 0 | 0 |
| P2 | 0 | 46.00 |
| P3 | 0 | 85.00 |
| P4 | 0 | 152.00 |
| Average waiting time: | | 71.18 |

Output

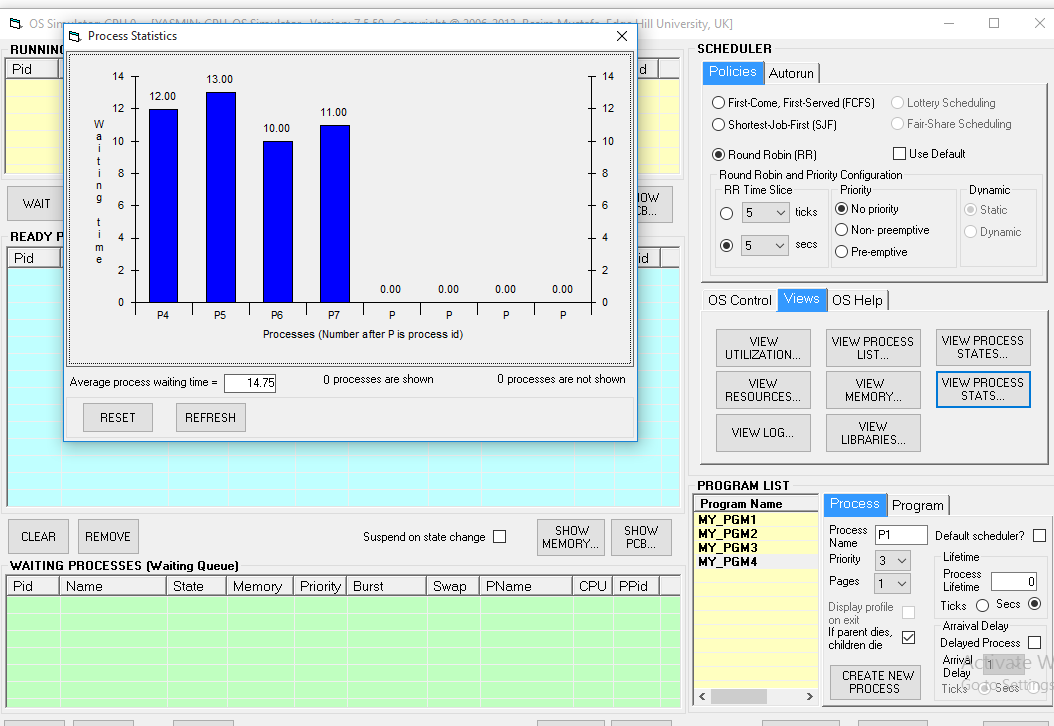


BITS ID: 2019WA15586

ii) Scheduling using Round Robin method

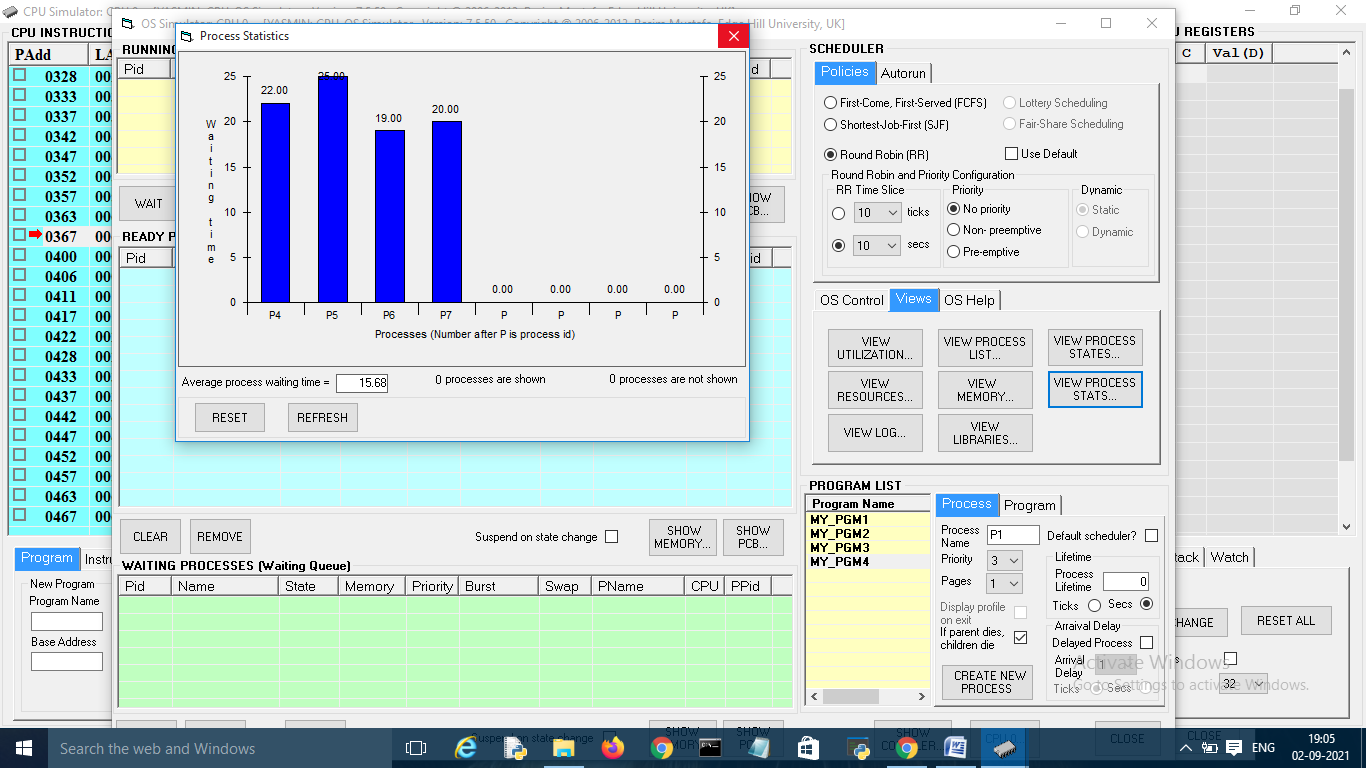
BITS ID: 2019WA15586

Output

****

|  |  |  |
| --- | --- | --- |
| Scheduling Algorithm : Round Robin with time quantum 5 | | |
| Process | Arrival Time | Waiting time |
| P1 | 0 | 12 |
| P2 | 0 | 13 |
| P3 | 0 | 10 |
| P4 | 0 | 11 |
| Average waiting time | | 14.75 |

3) Round Robin with time quantum 10

****

Output

BITS ID: 2019WA15586

|  |  |  |
| --- | --- | --- |
| Scheduling Algorithm : Round Robin with time quantum 10 | | |
| Process | Arrival Time | Waiting time |
| P1 | 0 | 22.00 |
| P2 | 0 | 20.00 |
| P3 | 0 | 19.00 |
| P4 | 0 | 20.00 |
| Average waiting time | | 15.68 |

Among three Round Robin with quantum 5 is better which takes less waiting time to execute all jobs.

**5Q** Consider the following source code

program CriticalRegion

var g integer

sub thread1 as thread

writeln("In thread1")

g = 0

for n = 1 to 5

g = g + 1

next

writeln("thread1 g = ", g)

writeln("Exiting thread1")

end sub

sub thread2 as thread

writeln("In thread2")

g = 0

for n = 1 to 15

g = g + 1

next

writeln("thread2 g = ", g)

writeln("Exiting thread2")

end sub

writeln("In main")

call thread2

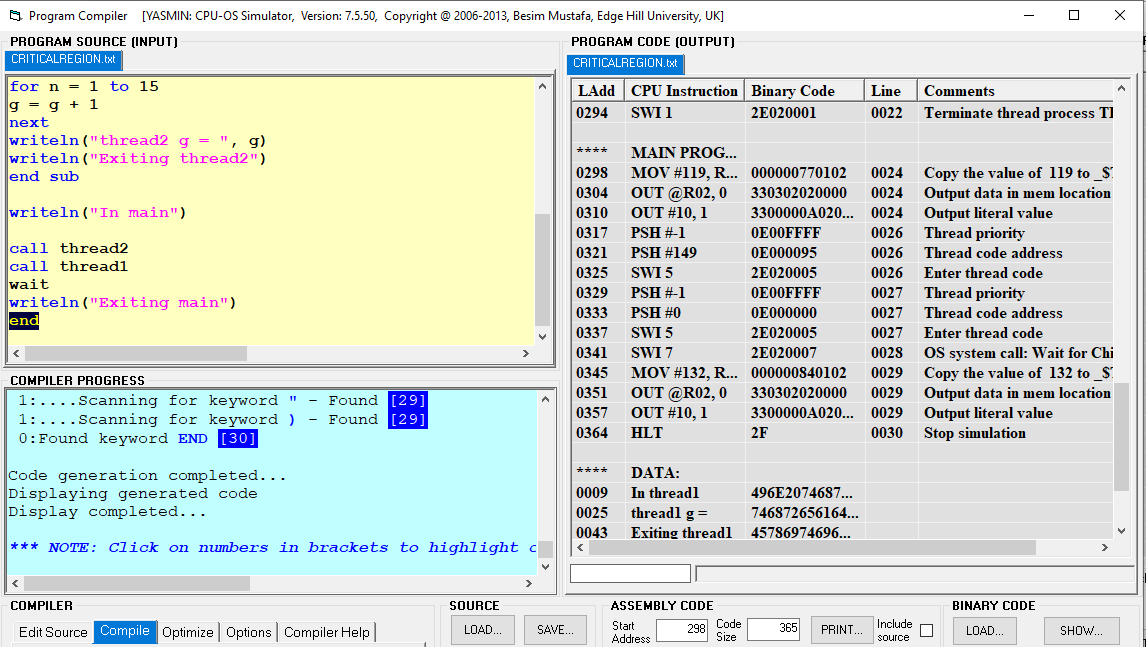
call thread1

wait

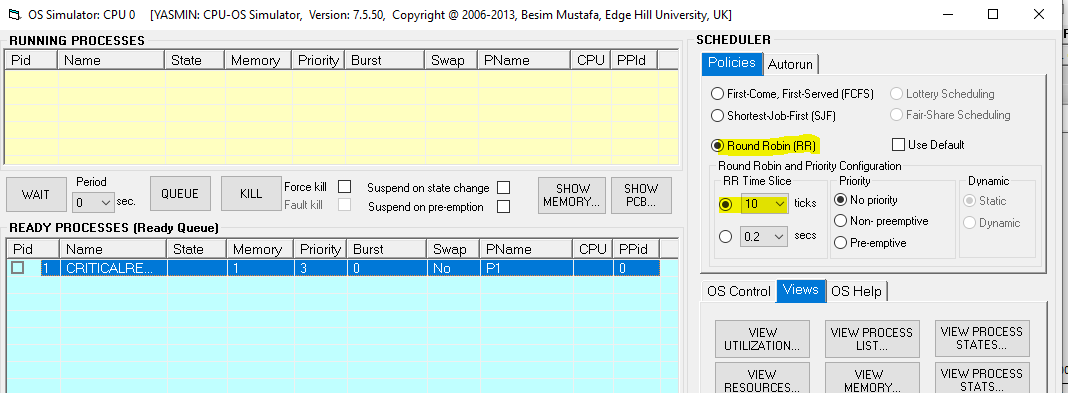
writeln("Exiting main")

end

Program loaded and compiled

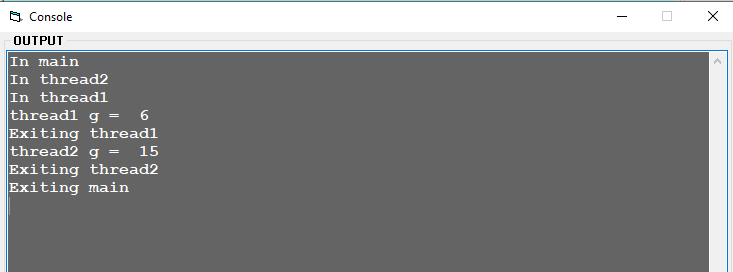


BITS ID: 2019WA15586



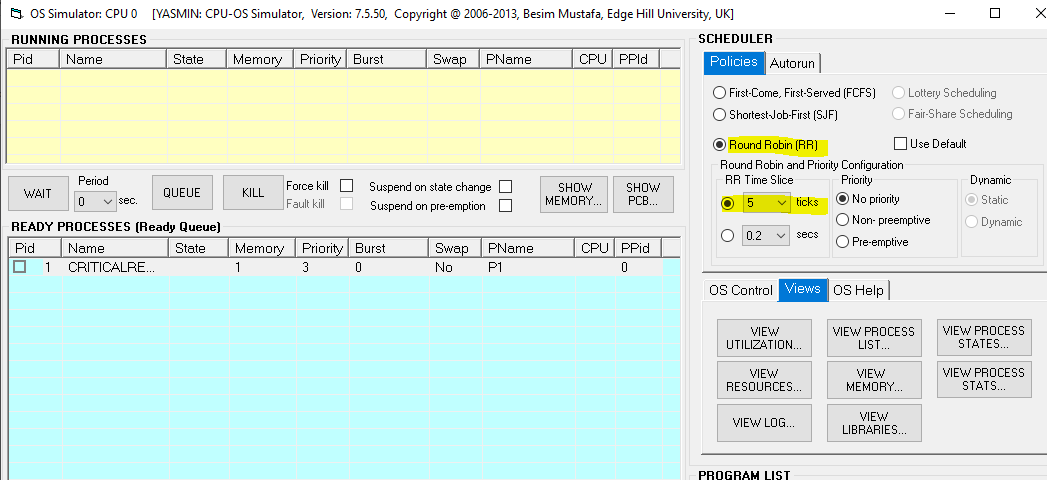
Set time quantum for RR scheduling algorithm to 10 ticks and start the program. When the program stops, note down the value of g printed in sub thread1 and sub thread2 routines.

Below are the values



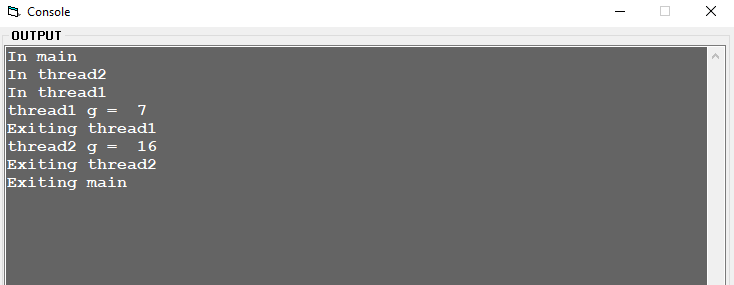
BITS ID: 2019WA15586

B) Set time quantum for RR scheduling algorithm to 5 ticks and start the program. When the program stops, note down the value of g printed in sub thread1 and sub thread2 routines.



BITS ID: 2019WA15586

Output



BITS ID: 2019WA15586

1. Is there any difference in values printed in **a** and **b**? If so, what is the reason behind it?

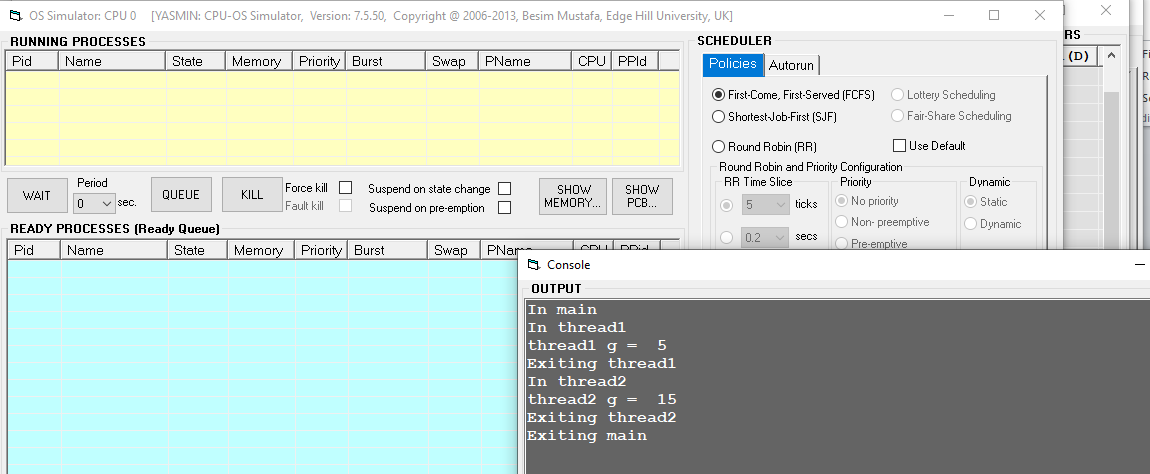
**Ans**: Yes there is difference in values printed in a and b .

Values got when executed a) 6 and 15

Values got when executed b) 7 and 16

The reason behind it is time slice taken for executing “a” is 10 ticks and time slice taken for executing “b” is 5 ticks.

1. What is the value of g printed if thread1 and thread2 executes in sequential manner?



BITS ID: 2019WA15586

7) Consider the following source code

program ThreadTest

sub thread1 as thread

fori = 1 to 10

writeln("In thread1: ", i)

next

end sub

sub thread2 as thread

fori = 1 to 10

writeln("In thread1: ", i)

next

end sub

call thread1

call thread2

writeln("In main")

do

loop

end

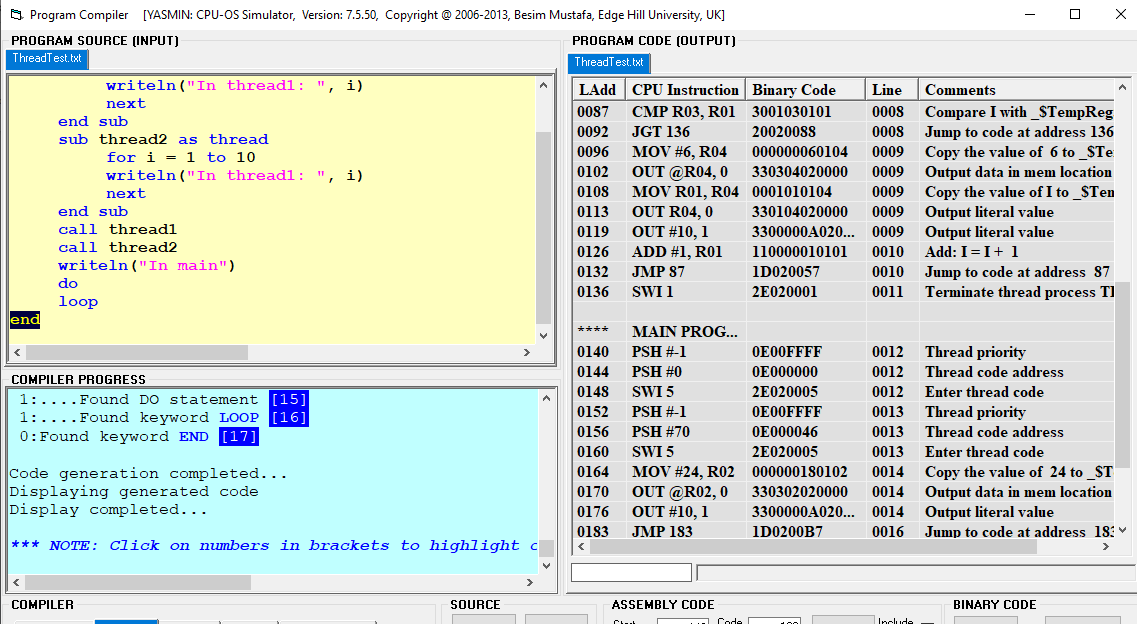
Using CPU-OS simulator ,compile the above source and load it in main memory. Create a single process, choose RR scheduling algorithm with time quantum of 3 ticks. Run the Process.

Answer the following questions:

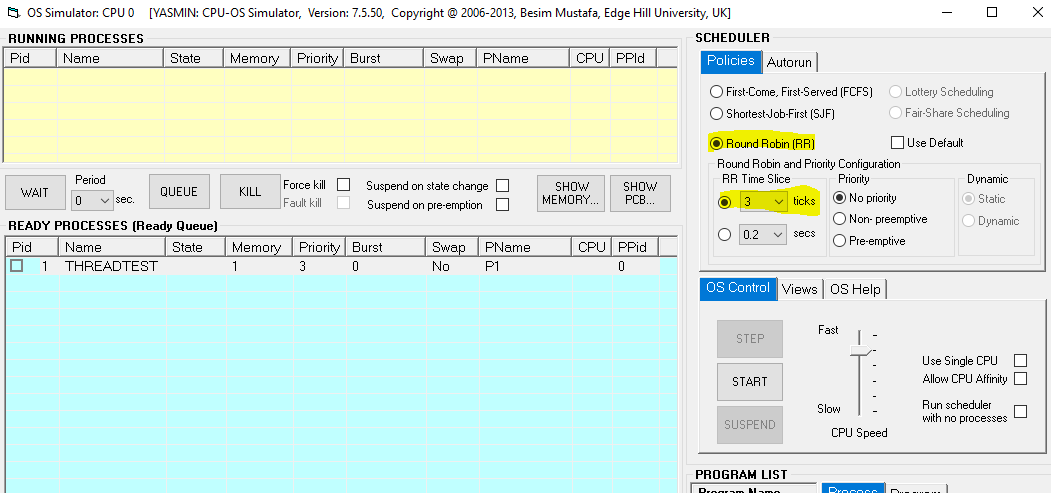
1. How many processes and how many threads are created?
2. Identify the name of the processes and threads.
3. What is the PID and PPID of the processes and threads created?

Represent the parent and child relationship using tree representation

Program loaded and compiled:

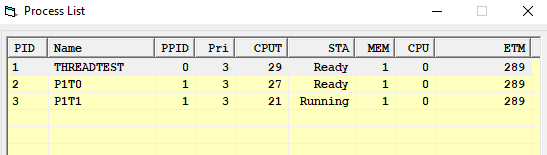


Created process Instance with Round Robin 3 quantum ticks



1. How many processes and how many threads are created?

Total 3 process instances and 1 Thread has been created





1. Identify the name of the processes and threads.

**Name of the Processes: THREADTEST, P1T0 AND P1T1**

**Name Of the Thread: Thread 1 and Thread 1**

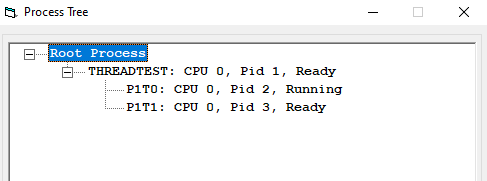
1. What is the PID and PPID of the processes and threads created?

**PID PPID**

1. **0**
2. **1**
3. **1**

**Threads Created : Thread 1 and Thread 1**

Represent the parent and child relationship using tree representation



THREADSET is parent tree and P1T0 and P1T1 are the childs for THREADSET .

9) In the compiler window, enter the following source code in the compiler’s source editor window

program Loop

i = 0

for n = 1 to 40

i = i + 1

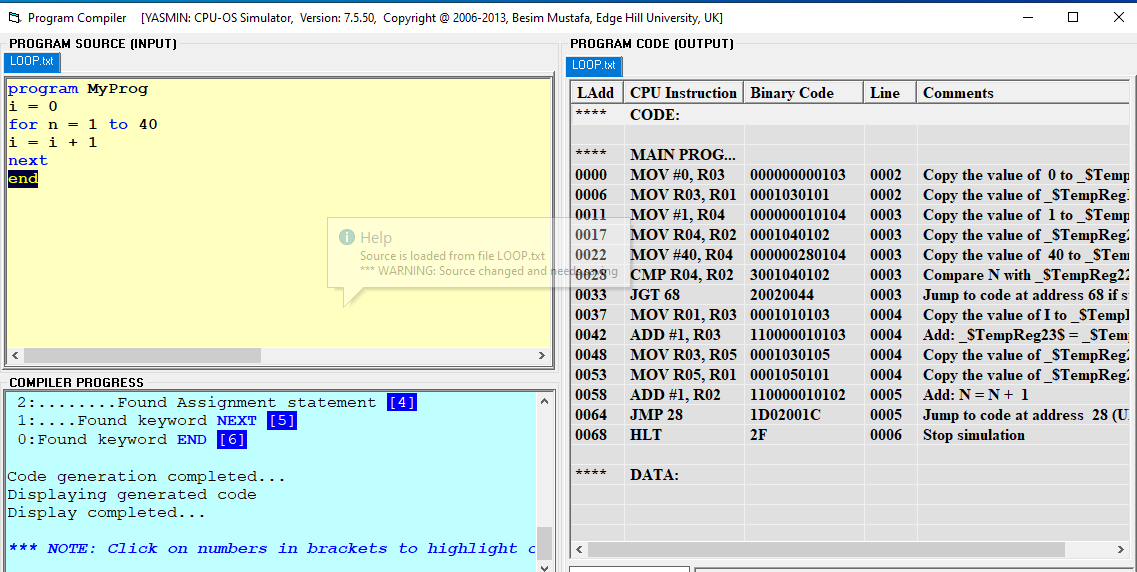
next

end

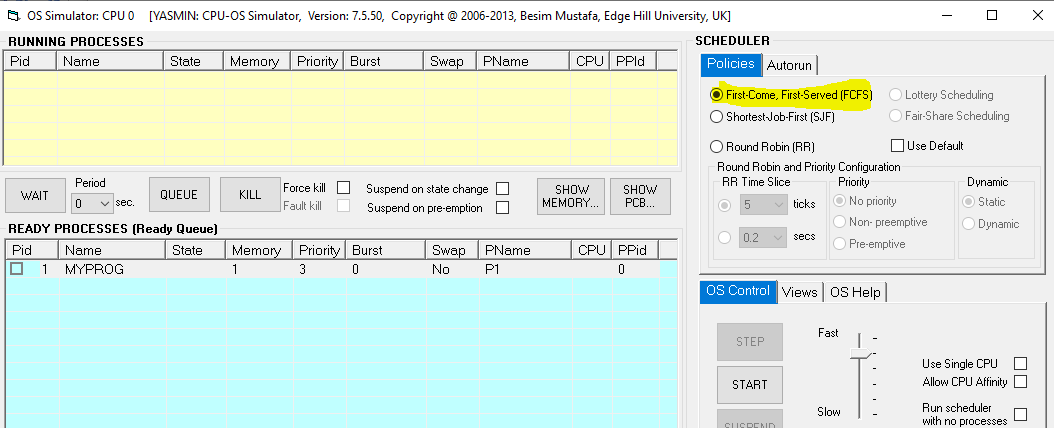
a) Select different scheduling policies and run the processes in the OS simulator.

b) Explain the differences between pre‐emptive and non‐pre‐emptive scheduling.

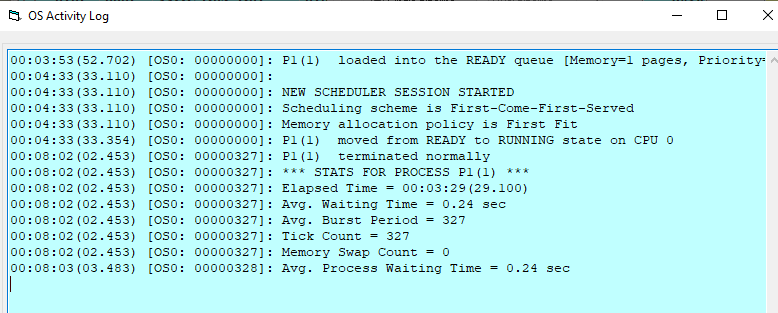
Loaded Program

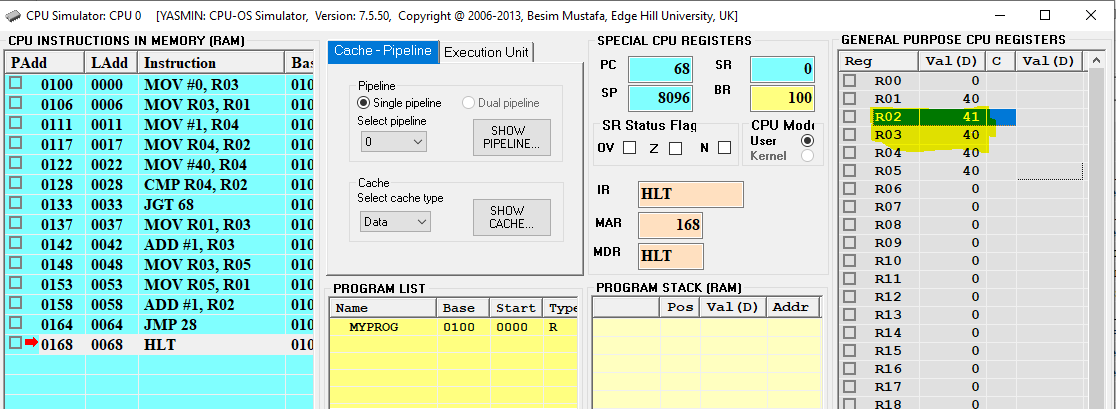


Scheduling the process instance using FCFS Schedule:-

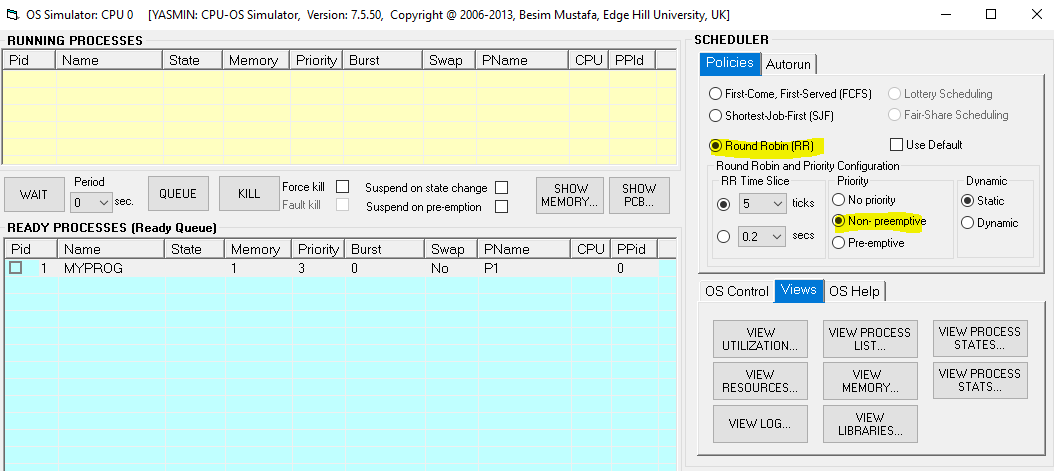


Output

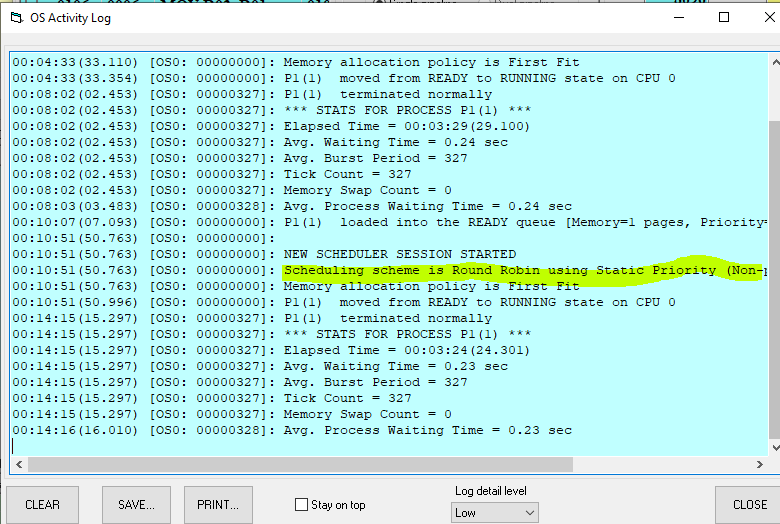




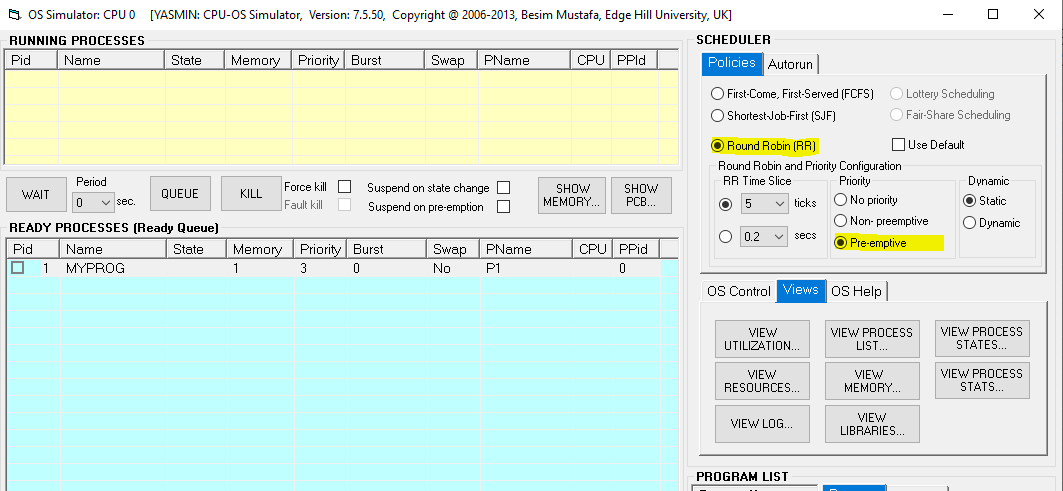
Scheduled using Round Robin method with non-preemptive

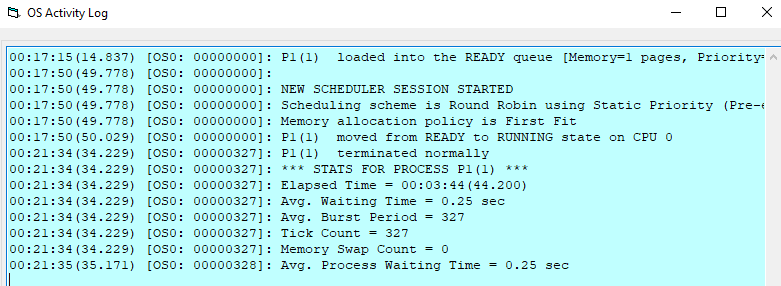


Output



Scheduling in pre-emptive





B) Explain the differences between pre‐emptive and non‐pre‐emptive scheduling.

**Preemptive Scheduling** is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switch to ready state.

Algorithms that are backed by preemptive Scheduling are round-robin (RR), priority, SRTF (shortest remaining time first).

**Non-preemptive Scheduling** is a CPU scheduling technique the process takes the resource (CPU time) and holds it till the process gets terminated or is pushed to the waiting state. No process is interrupted until it is completed, and after that processor switches to another process.

Algorithms that are based on non-preemptive Scheduling are non-preemptive priority, and shortest Job first.

**Preemptive Vs Non-Preemptive Scheduling**

| **Preemptive Scheduling** | **Non-Preemptive Scheduling** |
| --- | --- |
| Resources are allocated according to the cycles for a limited time. | Resources are used and then held by the process until it gets terminated. |
| The process can be interrupted, even before the completion. | The process is not interrupted until its life cycle is complete. |
| Starvation may be caused, due to the insertion of priority process in the queue. | Starvation can occur when a process with large burst time occupies the system. |
| Maintaining queue and remaining time needs storage overhead. | No such overheads are required. |

11) Enter the following source code in the CPU-OS simulator:

program CriticalRegion

var g integer

sub thread1 as thread

writeln("In thread1")

**enter**

g = 0

for n = 1 to 20

g = g + 1

next

writeln("thread1 g = ", g)

**leave**

writeln("Exiting thread1")

end sub

sub thread2 as thread

writeln("In thread2")

**enter**

g = 0

for n = 1 to 12

g = g + 1

next

writeln("thread2 g = ", g)

**leave**

writeln("Exiting thread2")

end sub

writeln("In main")

call thread1

call thread2

wait

writeln("Exiting main")

end

NOTE: The **enter** and **leave** keyword pair protect the program code between them. This makes sure the protected code executes exclusively without sharing the CPU with any other thread.

a)Compile the above program and load in memory.

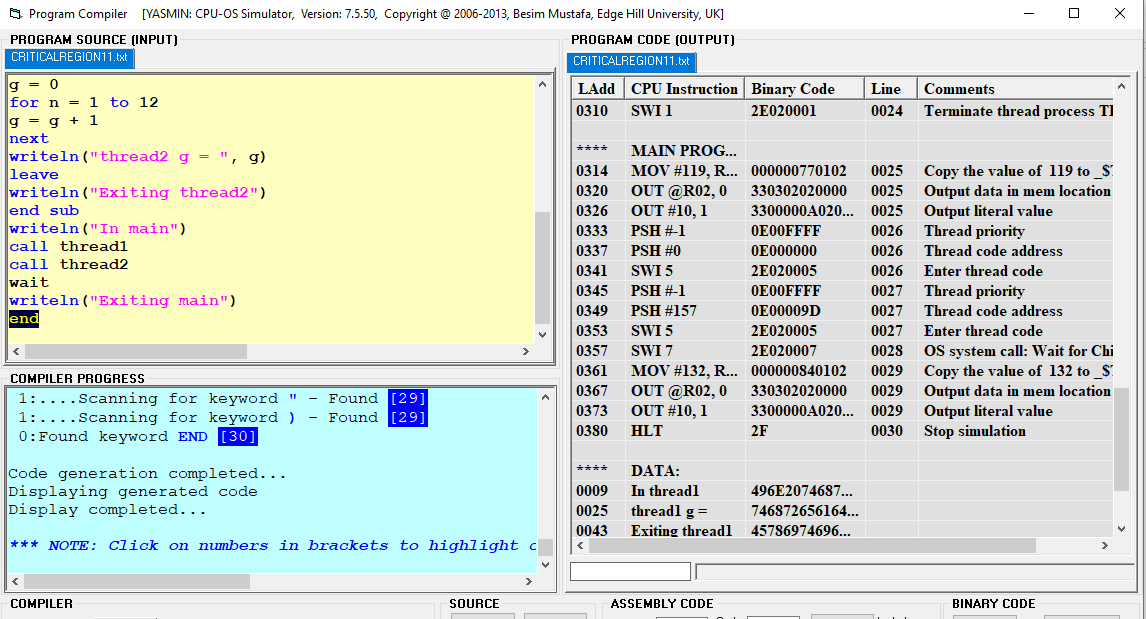
Perform the following:

1. Set time quantum for RR scheduling algorithm to 10 ticks and start the program. When the program stops, note down the value of g printed in sub thread1 and sub thread2 routines.
2. Set time quantum for RR scheduling algorithm to 5 ticks and start the program. When the program stops, note down the value of g printed in sub thread1 and sub thread2 routines.

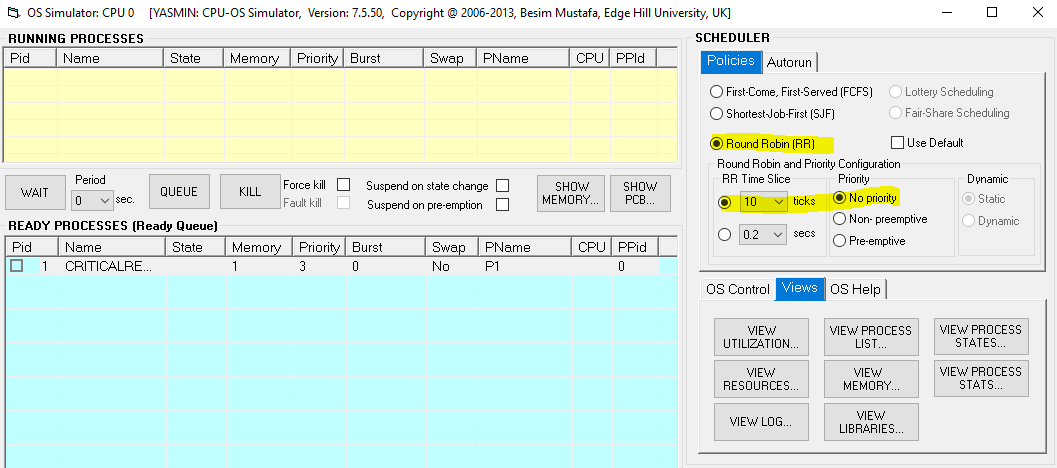
Is there any difference in values printed in **a**and**b**?

b)Write down the observation when the above keywords are added and explain their use.

Coded loaded



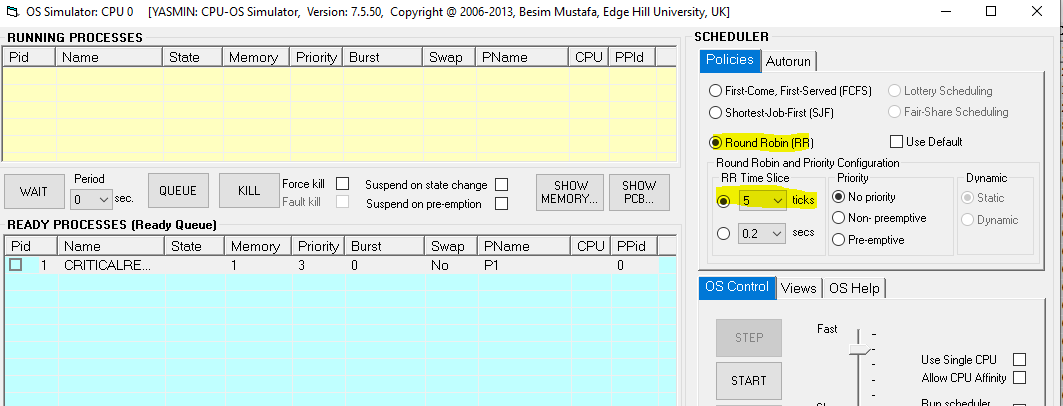
A) Set time quantum for RR scheduling algorithm to 10 ticks and start the program. When the program stops, note down the value of g printed in sub thread1 and sub thread2 routines.



Output:-



1. Set time quantum for RR scheduling algorithm to 5 ticks and start the program. When the program stops, note down the value of g printed in sub thread1 and sub thread2 routines.



**Output :**



Is there any difference in values printed in **a** and **b**?

**No there is no difference.**