REPORT - PROJECT 4

Advanced Real Time Systems - ECE 5550G

PART 1 -

I) A Brief Description of Original Ceiling Priority Protocol (**OCPP**):

It is a resource access protocol that is used to synchronize shared resources and prevent unbounded priority inversion and mutual deadlock that arise due to improper nesting of critical sections. It consists :-

- **Scheduling Rule** : A job is scheduled in a priority driven manner where the current priority is the assigned priority.
- **Allocation Rule** : Let Job **J** request resource **R**
    - If **R** is busy, **J** is blocked and the request is denied.
    - If **R** is free :
        - If **P(J)** > **PC(S*)**, **R** is allocated to **J**.
        - If **P(J)** <= **PC(S*)**, **R** is allocated to **J** only if **J** holds some resource whose **PC(S)** = **PC(S*)**, otherwise **J** is blocked and the request is denied.
- **Priority Inheritance Rule** : If a lower priority **J** blocks a higher priority **J**, the low priority **J** inherits the high priority until it releases every resource whose priority is greater than the inherited priority.

PC(S*) : It is the semaphore S having the maximum priority ceiling C among all the semaphores currently locked by tasks. It is called the **system priority ceiling**.

PC(S) : It is the priority ceiling C of the semaphore S.

P(J) : Priority of Job **J**

II) Code Implementation of the Original Ceiling Priority Protocol (**OCPP**) with **RM**:

This project built upon the previous project which has the RM implementation.

By implementing the following edits to the scheduler.cpp, we can get OCPP to work :

- **xSchedulerCreateResource** : This function creates and returns a resource (semaphore) handle which is used to identify the resource that is requested or released by a task.

- **vSchedulerResourceUsedByTask** : This function is used for specifying all the tasks that will be accessing a certain resource. It is used for calculating the priority ceiling of each resource.

- **vSchedulerResourceWait** : This function is used by a task to request a resource before entering the critical section. This is a blocking function and will only return when the resource is free.

- **vSchedulerResourceSignal** : This function is used by a task to free the locked resource while it is exiting the critical section.

- **prvSetPriorityCeilingToResources** : This is a private function which is called before the scheduler starts. It is used to assign priority ceiling to each of the resources based on the tasks that may access it.

- **prvUpdateSystemPriorityCeiling** : This is a private function is used to update the **system priority ceiling PC(S*)** whenever a resource is locked by a task or freed by a task.

- **prvFreeAllResourcesHeldByTask** : This is a private function used to free any and all resources that were locked by a task. This function is used when a function exceeds its maximum execution time or misses a deadline.

- **prvCheckResourcesPriorityCeilingHeldByTask** : This is a private function that is used to check if a given task holds any resources that has a priority ceiling equal to the system priority ceiling. It checks if **PC(S)** = **PC(S*)** for a given task **J.**

- **prvLockResource** : This is a private function that is used to lock a resource if a requested resource is allocated to the task by updating the resource control block.

- **prvDenyResource** : This is a private function that is used to keep track of the task that is blocked directly or indirectly on a particular resource, the task which is blocked and to run the **Priority Inheritance Rule**.

- **prvBlockTask** : This is a private function that is used to keep a task that was denied a particular resource in the blocked state until the resource is available or the maximum execution time of the task is reached.

- **prvResourceWait** : This is a private function that is to perform the **Allocation Rule** check using **OCPP** on the resource requested by a task and check if the resource can be allocated or the task should be blocked.

- **prvFreeResource** : This is a private function that is used to free a locked resource once the task exits the critical section by updating the resource control block.

- **prvUpdateTaskPriority** : This is a private function used to update the priority of the task after it has freed a resource. The priority is based on whether it holds any more resources i.e **Priority Inheritance Rule**

- **prvUnblockTasks** : This is a private function that unblocks all the tasks that were blocked by another task when it exits its critical section.

- **prvResourceSignal** : This is a private function that is to the if the task that acquired the resource is the task that is giving back the resource.

III) Running Original Ceiling Priority Protocol (**OCPP**) with **RM** using the example task set from Homework 4 Problem 1 :

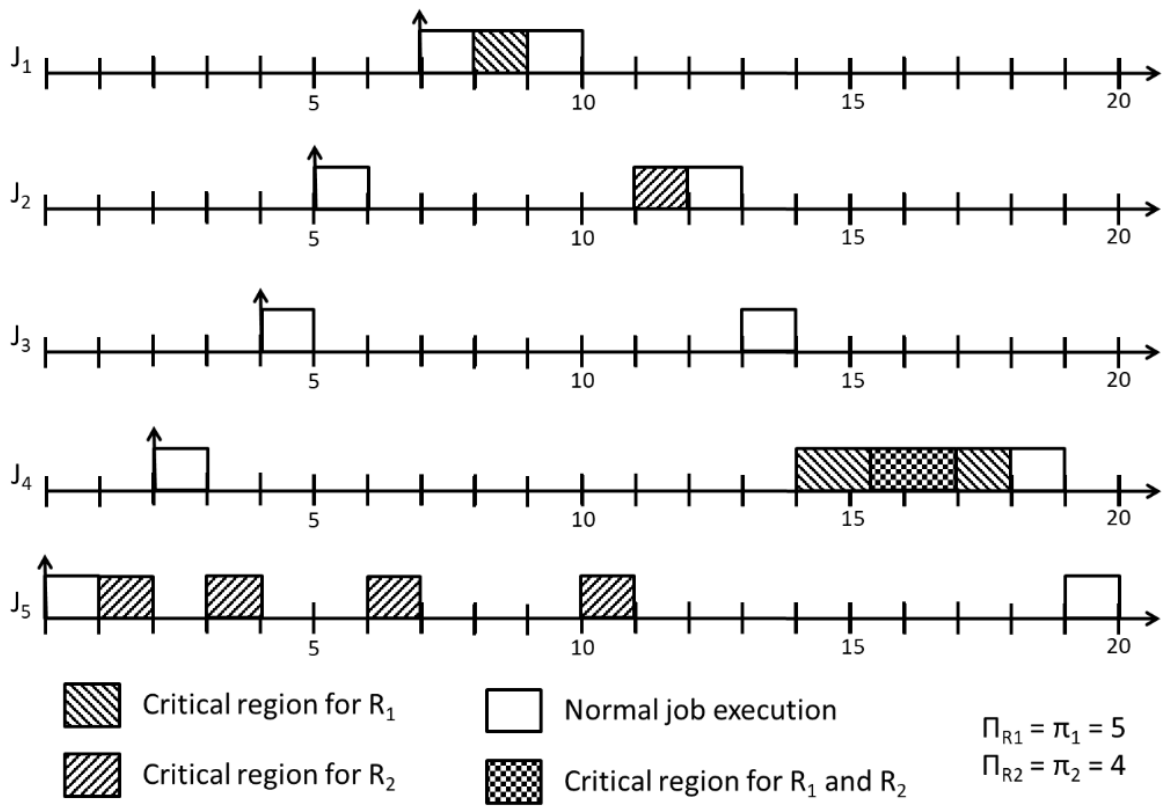
```
---- Opened the serial port COM3 ----
----- Program Started -----

FUNC: vSchedulerInit

FUNC: vSchedulerPeriodicTaskCreate
---- Task Details ----
Name                : T5
Phase Tick          : 0
Max. Execution Tick : 100
Rel. Deadline Tick  : 100
Period Tick         : 100
Priority            : 0
----------------------


FUNC: vSchedulerPeriodicTaskCreate
---- Task Details ----
Name                : T4
Phase Tick          : 0
Max. Execution Tick : 98
Rel. Deadline Tick  : 98
Period Tick         : 98
Priority            : 0
----------------------


FUNC: vSchedulerPeriodicTaskCreate
---- Task Details ----
Name                : T3
Phase Tick          : 0
Max. Execution Tick : 96
Rel. Deadline Tick  : 96
Period Tick         : 96
Priority            : 0
----------------------


FUNC: vSchedulerPeriodicTaskCreate
---- Task Details ----
Name                : T2
Phase Tick          : 0
Max. Execution Tick : 94
Rel. Deadline Tick  : 94
Period Tick         : 94
Priority            : 0
----------------------


FUNC: vSchedulerPeriodicTaskCreate
---- Task Details ----
Name                : T1
Phase Tick          : 0
Max. Execution Tick : 92
Rel. Deadline Tick  : 92
Period Tick         : 92
Priority            : 0
----------------------


FUNC: xSchedulerCreateResource

FUNC: xSchedulerCreateResource
```

```
FUNC: vSchedulerResourceUsedByTask

FUNC: vSchedulerResourceUsedByTask

FUNC: vSchedulerResourceUsedByTask

FUNC: vSchedulerResourceUsedByTask

FUNC: vSchedulerStart

FUNC: prvSetFixedPriorities
----Using RM Scheduling Algorithm----
 Task : T1, Priority : 5, Tick : 92
 Task : T2, Priority : 4, Tick : 94
 Task : T3, Priority : 3, Tick : 96
 Task : T4, Priority : 2, Tick : 98
 Task : T5, Priority : 1, Tick : 100
-------------------------------------

FUNC: prvCreateSchedulerTask
---- Scheduler Details ----
Period Tick  : 2
Priority     : 6
Overhead     : 0
---------------------------

FUNC: prvCreateAllTasks
----Using OCPP----

FUNC: prvSetPriorityCeilingToResources
---- Resource Details ----------
Name                          : R1
Priority Ceiling              : 5
Number Of Tasks Using Resource : 2
Tasks Utilizing Resource      : [ T1 T4 ]
--------------------------------

---- Resource Details ----------
Name                          : R2
Priority Ceiling              : 4
Number Of Tasks Using Resource : 3
Tasks Utilizing Resource      : [ T2 T4 T5 ]
--------------------------------


FUNC: prvPeriodicTaskCode -> TASK: T1, INIT RUN

FUNC: prvPeriodicTaskCode -> TASK: T2, INIT RUN

FUNC: prvPeriodicTaskCode -> TASK: T3, INIT RUN

FUNC: prvPeriodicTaskCode -> TASK: T4, INIT RUN

FUNC: prvPeriodicTaskCode -> TASK: T5, INIT RUN
TASK : T5, PRIORITY : 1

FUNC: prvResourceWait
R2 Locked By T5 | SPC : 4 @ T : 4
T5 : CRITICAL SECTION START
TASK : T5, PRIORITY : 1
TASK : T4, PRIORITY : 2

FUNC: prvResourceWait
```

```
FUNC: prvResourceWait
R2 Locked By T5 | SPC : 4 @ T : 4
T5 : CRITICAL SECTION START
TASK : T5, PRIORITY : 1
TASK : T4, PRIORITY : 2

FUNC: prvResourceWait
R1 Denied To T4 | T4 Blocked T5 @ T : 6
TASK : T3, PRIORITY : 3
TASK : T1, PRIORITY : 5

FUNC: prvResourceWait
R1 Locked By T1 | SPC : 5 @ T : 9
T1 : CRITICAL SECTION START
TASK : T1, PRIORITY : 5
T1 : CRITICAL SECTION END

FUNC: prvResourceSignal
R1 Freed By T1 @ T : 11
TASK : T1, PRIORITY : 5
STAT: T1, ST:0000, ET:0011, RT:11, DT: 0092
TASK : T2, PRIORITY : 4

FUNC: prvResourceWait
R2 Denied To T2 | T2 Blocked T5 @ T : 14
T5 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T5 @ T : 16
T4 Unblocked T5 @ T : 16
T2 Unblocked T5 @ T : 16

FUNC: prvResourceWait
R2 Locked By T2 | SPC : 4 @ T : 17
T2 : CRITICAL SECTION START
TASK : T2, PRIORITY : 4
T2 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T2 @ T : 18
TASK : T2, PRIORITY : 4
STAT: T2, ST:0000, ET:0018, RT:18, DT: 0094
STAT: T3, ST:0000, ET:0021, RT:21, DT: 0096

FUNC: prvResourceWait
R1 Locked By T4 | SPC : 5 @ T : 21
T4 : CRITICAL SECTION START
TASK : T4, PRIORITY : 2

FUNC: prvResourceWait
R2 Locked By T4 | SPC : 5 @ T : 24
T4 : CRITICAL SECTION START
TASK : T4, PRIORITY : 2
T4 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T4 @ T : 26
TASK : T4, PRIORITY : 5
T4 : CRITICAL SECTION END

FUNC: prvResourceSignal
R1 Freed By T4 @ T : 27
TASK : T4, PRIORITY : 2
STAT: T4, ST:0000, ET:0027, RT:27, DT: 0098
```

```
FUNC: prvResourceSignal
R2 Freed By T4 @ T : 26
TASK : T4, PRIORITY : 5
T4 : CRITICAL SECTION END

FUNC: prvResourceSignal
R1 Freed By T4 @ T : 27
TASK : T4, PRIORITY : 2
STAT: T4, ST:0000, ET:0027, RT:27, DT: 0098
TASK : T5, PRIORITY : 1
STAT: T5, ST:0000, ET:0027, RT:27, DT: 0100
TASK : T1, PRIORITY : 5

FUNC: prvResourceWait
R1 Locked By T1 | SPC : 5 @ T : 100
T1 : CRITICAL SECTION START
TASK : T1, PRIORITY : 5
T1 : CRITICAL SECTION END

FUNC: prvResourceSignal
R1 Freed By T1 @ T : 100
TASK : T1, PRIORITY : 5
STAT: T1, ST:0092, ET:0100, RT:08, DT: 0184
TASK : T5, PRIORITY : 1

FUNC: prvResourceWait
R2 Locked By T5 | SPC : 4 @ T : 100
T5 : CRITICAL SECTION START
TASK : T5, PRIORITY : 1
T5 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T5 @ T : 100
TASK : T5, PRIORITY : 1
STAT: T5, ST:0100, ET:0100, RT:00, DT: 0100
TASK : T2, PRIORITY : 4

FUNC: prvResourceWait
R2 Locked By T2 | SPC : 4 @ T : 103
T2 : CRITICAL SECTION START
TASK : T2, PRIORITY : 4
T2 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T2 @ T : 103
TASK : T2, PRIORITY : 4
STAT: T2, ST:0094, ET:0103, RT:09, DT: 0188
TASK : T3, PRIORITY : 3
STAT: T3, ST:0096, ET:0107, RT:11, DT: 0192
TASK : T4, PRIORITY : 2

FUNC: prvResourceWait
R1 Locked By T4 | SPC : 5 @ T : 107
T4 : CRITICAL SECTION START
TASK : T4, PRIORITY : 2

FUNC: prvResourceWait
R2 Locked By T4 | SPC : 5 @ T : 107
T4 : CRITICAL SECTION START
TASK : T4, PRIORITY : 2
T4 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T4 @ T : 107
```

c) OCPP



OCPP Timeline taken from Homework 4 Solution posted on canvas.

By comparing the OCPP timeline to the debug output of the program running the same task set and resources (overlooking the execution time of each task and difference in release due to scheduler) we can see that the debug output matches the timeline. Hence we can justify that the implementation is correct.

PART 2 -

I) A Brief Description of Immediate Ceiling Priority Protocol (**ICPP**):

It is a resource access protocol that is used to synchronize shared resources and prevent unbounded priority inversion and mutual deadlock that arise due to improper nesting of critical sections. It consists :-

- **Scheduling Rule** : A job is scheduled in a priority driven manner where the current priority is the assigned priority.
- **Allocation Rule** : Let Job **J** request resource **R**
    - If **R** is busy, **J** is blocked and the request is denied.
    - If **R** is free :
        - If **P(J)** > **PC(S)**, **R** is allocated to **J**.
        - If **P(J)** < **PC(S)**, **R** is allocated to **J** and **P(J)** = **PC(S)**.

PC(S) : It is the priority ceiling C of the semaphore S.

P(J) : Priority of Job **J**

II) Code Implementation of the Original Ceiling Priority Protocol (**OCPP**) with **RM**:

This project built upon the previous project which has the RM implementation

By implementing the following edits to the scheduler.cpp, we can get ICPP to work. Most of the functions are retained from OCCP implementation and only a few are modified. The modified functions are :

- **prvResourceWait** : This is a private function that is to perform the **Allocation Rule** check for **ICPP** on the resource requested by a task and check if the resource can be allocated or the task should be blocked.

- **prvDenyResource** : This is a private function that is used to keep track of the task that is blocked directly or indirectly on a particular resource, the task which is blocked and to run the **Priority Inheritance Rule**.

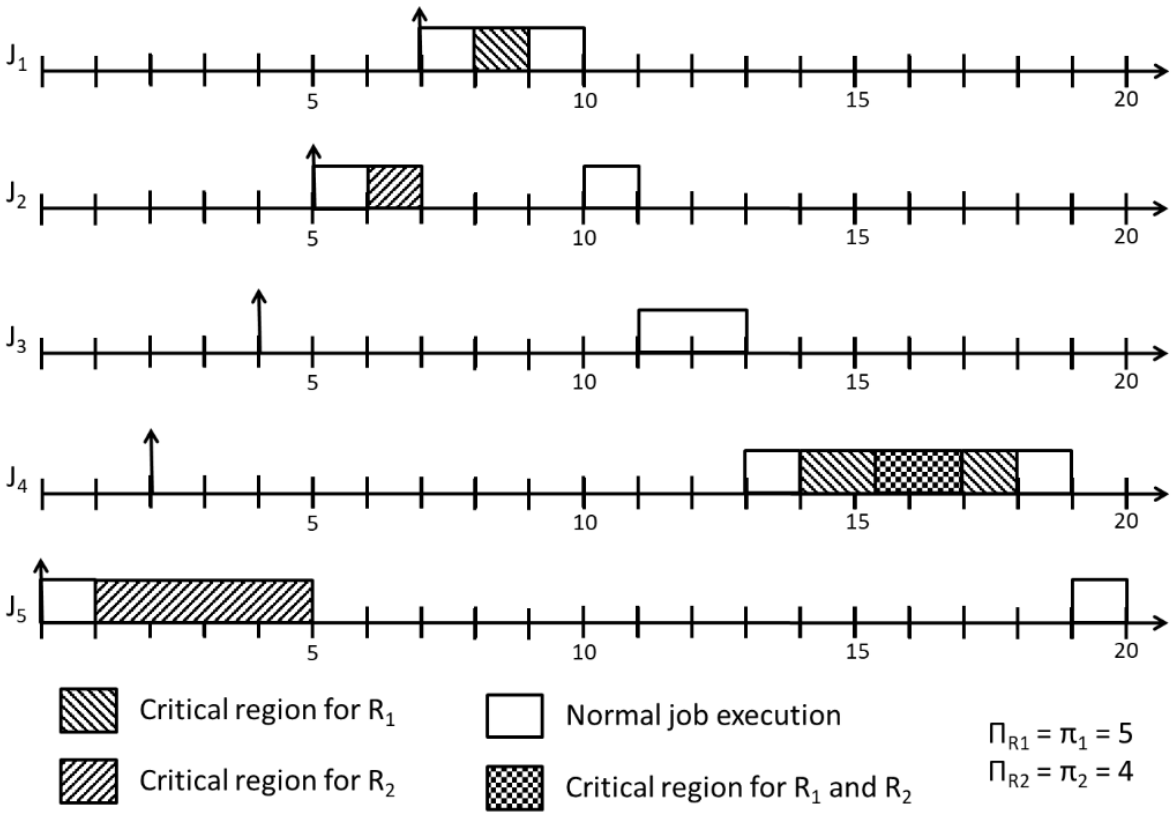
III) Running Original Ceiling Priority Protocol (**OCPP**) with **RM** using the example task set from Homework 4 Problem 1 :

```
---- Opened the serial port COM3 ----
----- Program Started -----

FUNC: vSchedulerInit

FUNC: vSchedulerPeriodicTaskCreate
---- Task Details ----
Name                : T5
Phase Tick          : 0
Max. Execution Tick : 100
Rel. Deadline Tick  : 100
Period Tick         : 100
Priority            : 0
----------------------


FUNC: vSchedulerPeriodicTaskCreate
---- Task Details ----
Name                : T4
Phase Tick          : 0
Max. Execution Tick : 98
Rel. Deadline Tick  : 98
Period Tick         : 98
Priority            : 0
----------------------


FUNC: vSchedulerPeriodicTaskCreate
---- Task Details ----
Name                : T3
Phase Tick          : 0
Max. Execution Tick : 96
Rel. Deadline Tick  : 96
Period Tick         : 96
Priority            : 0
----------------------


FUNC: vSchedulerPeriodicTaskCreate
---- Task Details ----
Name                : T2
Phase Tick          : 0
Max. Execution Tick : 94
Rel. Deadline Tick  : 94
Period Tick         : 94
Priority            : 0
----------------------


FUNC: vSchedulerPeriodicTaskCreate
---- Task Details ----
Name                : T1
Phase Tick          : 0
Max. Execution Tick : 92
Rel. Deadline Tick  : 92
Period Tick         : 92
Priority            : 0
----------------------


FUNC: xSchedulerCreateResource

FUNC: xSchedulerCreateResource
```

```
FUNC: vSchedulerResourceUsedByTask

FUNC: vSchedulerResourceUsedByTask

FUNC: vSchedulerResourceUsedByTask

FUNC: vSchedulerResourceUsedByTask

FUNC: vSchedulerStart

FUNC: prvSetFixedPriorities
----Using RM Scheduling Algorithm----
 Task : T1, Priority : 5, Tick : 92
 Task : T2, Priority : 4, Tick : 94
 Task : T3, Priority : 3, Tick : 96
 Task : T4, Priority : 2, Tick : 98
 Task : T5, Priority : 1, Tick : 100
-------------------------------------

FUNC: prvCreateSchedulerTask
---- Scheduler Details ----
Period Tick  : 2
Priority     : 6
Overhead     : 0
---------------------------

FUNC: prvCreateAllTasks
----Using ICPP----

FUNC: prvSetPriorityCeilingToResources
---- Resource Details ----------
Name                          : R1
Priority Ceiling              : 5
Number Of Tasks Using Resource : 2
Tasks Utilizing Resource      : [ T1 T4 ]
--------------------------------

---- Resource Details ----------
Name                          : R2
Priority Ceiling              : 4
Number Of Tasks Using Resource : 3
Tasks Utilizing Resource      : [ T2 T4 T5 ]
--------------------------------


FUNC: prvPeriodicTaskCode -> TASK: T1, INIT RUN

FUNC: prvPeriodicTaskCode -> TASK: T2, INIT RUN

FUNC: prvPeriodicTaskCode -> TASK: T3, INIT RUN

FUNC: prvPeriodicTaskCode -> TASK: T4, INIT RUN

FUNC: prvPeriodicTaskCode -> TASK: T5, INIT RUN
TASK : T5, PRIORITY : 1

FUNC: prvResourceWait
R2 Locked By T5
T5 : CRITICAL SECTION START
TASK : T5, PRIORITY : 4
T5 : CRITICAL SECTION END

FUNC: prvResourceSignal
```

```
FUNC: prvPeriodicTaskCode -> TASK: T4, INIT RUN

FUNC: prvPeriodicTaskCode -> TASK: T5, INIT RUN
TASK : T5, PRIORITY : 1

FUNC: prvResourceWait
R2 Locked By T5
T5 : CRITICAL SECTION START
TASK : T5, PRIORITY : 4
T5 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T5 @ T : 7
TASK : T3, PRIORITY : 3
TASK : T1, PRIORITY : 5

FUNC: prvResourceWait
R1 Locked By T1
T1 : CRITICAL SECTION START
TASK : T1, PRIORITY : 5
T1 : CRITICAL SECTION END

FUNC: prvResourceSignal
R1 Freed By T1 @ T : 11
TASK : T1, PRIORITY : 5
STAT: T1, ST:0000, ET:0011, RT:11, DT: 0092
TASK : T2, PRIORITY : 4

FUNC: prvResourceWait
R2 Locked By T2
T2 : CRITICAL SECTION START
TASK : T2, PRIORITY : 4
T2 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T2 @ T : 16
TASK : T2, PRIORITY : 4
STAT: T2, ST:0000, ET:0016, RT:16, DT: 0094
STAT: T3, ST:0000, ET:0020, RT:20, DT: 0096
TASK : T4, PRIORITY : 2

FUNC: prvResourceWait
R1 Locked By T4
T4 : CRITICAL SECTION START
TASK : T4, PRIORITY : 5

FUNC: prvResourceWait
R2 Locked By T4 @ T : 24
T4 : CRITICAL SECTION START
TASK : T4, PRIORITY : 5
T4 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T4 @ T : 27
TASK : T4, PRIORITY : 5
T4 : CRITICAL SECTION END

FUNC: prvResourceSignal
R1 Freed By T4 @ T : 27
T4 : CRITICAL SECTION END

FUNC: prvResourceSignal
R1 Freed By T4 @ T : 27
TASK : T4, PRIORITY : 2
STAT: T4, ST:0000, ET:0027, RT:27, DT: 0098
TASK : T5, PRIORITY : 1
STAT: T5, ST:0000, ET:0027, RT:27, DT: 0100
TASK : T1, PRIORITY : 5

FUNC: prvResourceWait
R1 Locked By T1
T1 : CRITICAL SECTION START
TASK : T1, PRIORITY : 5
T1 : CRITICAL SECTION END

FUNC: prvResourceSignal
R1 Freed By T1 @ T : 100
TASK : T1, PRIORITY : 5
STAT: T1, ST:0092, ET:0100, RT:08, DT: 0184
TASK : T5, PRIORITY : 1

FUNC: prvResourceWait
R2 Locked By T5
T5 : CRITICAL SECTION START
TASK : T5, PRIORITY : 4
T5 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T5 @ T : 100
TASK : T5, PRIORITY : 1
STAT: T5, ST:0100, ET:0100, RT:00, DT: 0100
TASK : T2, PRIORITY : 4

FUNC: prvResourceWait
R2 Locked By T2
T2 : CRITICAL SECTION START
TASK : T2, PRIORITY : 4
T2 : CRITICAL SECTION END

FUNC: prvResourceSignal
R2 Freed By T2 @ T : 103
TASK : T2, PRIORITY : 4
STAT: T2, ST:0094, ET:0103, RT:09, DT: 0188
TASK : T3, PRIORITY : 3
STAT: T3, ST:0096, ET:0107, RT:11, DT: 0192
TASK : T4, PRIORITY : 2

FUNC: prvResourceWait
R1 Locked By T4
T4 : CRITICAL SECTION START
TASK : T4, PRIORITY : 5

FUNC: prvResourceWait
R2 Locked By T4 @ T : 107
T4 : CRITICAL SECTION START
TASK : T4, PRIORITY : 5
T4 : CRITICAL SECTION END

FUNC: prvResourceSignal
```

d) ICPP



ICPP Timeline taken from Homework 4 Solution posted on canvas.

By comparing the ICPP timeline to the debug output of the program running the same task set and resources(overlooking the execution time of each task and difference in release due to scheduler) we can see that the debug output matches the timeline. Hence we can justify that the implementation is correct.

REFERENCES

1. Please read the README.md before getting started with the code
2. Help on Reading the Output ➖
    a. FUNC : <Function Name> refers to the function called
    b. TASK : <Task Name>, PRIORITY <num> refers to the task that is executing and its priority.
    c. STAT : <Task Name>, ST : <num>, ET : <num>, RT : <num>, DT : <num> shows the stats of the task that finished executing: -
        i. ST is the tick at which the task starts executing.
        ii. ET is the tick at which the task finished executing.
        iii. RT is the number of ticks that task took to complete (including how long the task was preempted, if it was preempted)
        iv. DT is the absolute deadline of the task in ticks.
    d. SPC stands for System Priority Ceiling.
    e. <TASK> : CRITICAL SECTION START marks the beginning of the critical section.
    f. <TASK> : CRITICAL SECTION END marks the end of the critical section.