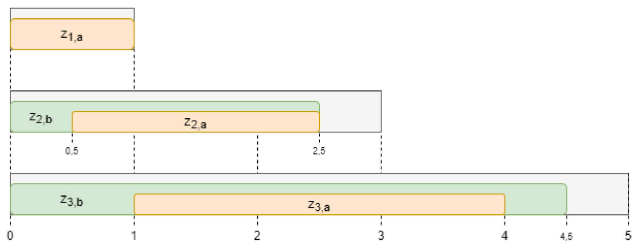
Real TIme Operating System Assignment

The application proposed is composed of three harmonic periodic tasks $\Gamma = \{\tau_1, \tau_2, \tau_3\}$ which share 2 different, non-preemptable resources R_a and R_b; The considered application of three tasks is scheduled considering just a single-processor machine, applying the Priority Inheritance Protocol (PIP) to solve the priority inversion problem. The graphical representation of the critical sections in Γ is:

	a_i	T_i	C_i	$t(R_a)$	δ_a	$t(R_b)$	δ_b
τ_1	0	4	1	0	1	-	-
τ_2	0	8	3	0,5	2	0	2,5
τ_3	0	16	5	1	3	0	4,5



From the above scheme it can be seen that the detail of accesses to share resources :

τ_1 : the first task, th one with the smaller period, starts executing with the shared resource R_a with a critical section long 1 time unit. After executing the critical section then it terminates.

τ_2 : the second task, starts executing with the shared resource R_b with a critical section long 2,5 time units. At 0,5 time unit of computation inside the critical section of R_b, the second task require to have access also to the resource R_a for 2 time units. Then the task will release both the resources, execute the last 0,5 time unit and terminate itself.

τ_3 : as in τ_2, in this third task would require a nested access of the two resources. This task starts executing accessing to the resource R_b and releasing it after 4,5 time units. At 1 time unit of computation, the third task would also require the access to resource $R_a$with a critical section long 3 time units. Then, after the release of resource R_b the task would compute the final 0.5 time unit and terminates.

Feasibility Analysis

Blocking term B_i:

Referring to the critical section table, then the maximum blocking time for each task can be found computing the highest combination of critical sections of lower priority tasks:

	δ_a	δ_b	B_i
T_1	1	-	6,5
T_2	2	2,5	4,5
T_3	3	4,5	0

- **Extended Static Priority Ordering Analysis**

$$\tau_1 : \frac{C_1 + B_1}{T_1} = \frac{1 + 6.5}{4} = 1.875 \quad (> 1)$$ ●

$$\tau_2 : \frac{C_1}{T_1} + \frac{C_2 + B_2}{T_2} = \frac{1}{4} + \frac{3 + 4,5}{8} = 1,1875 \quad (> 1\ !!)$$ ●

$$\tau_3 : \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3 + B_3}{T_3} = \frac{1}{4} + \frac{3}{8} + \frac{5 + 0}{16} = 0.9375 \quad (\leq 1)$$

- **Extended Response Time Analysis**

$\tau_1 : R_1^{(0)} = R_1^{(1)} = C_1 + B_1 = 1 + 6.5 = 7.5 \quad (> T_1 = 4)$ ●

$\tau_2 : R_2^{(0)} = C_1 + C_2 + B_2 = 1 + 3 + 4{,}5 = 8{,}5 \quad (> T_2 = 8{,}5 \,!!)$ ●

$\tau_3 : R_3^{(0)} = C_1 + C_2 + C_3 + B_3 = 1 + 3 + 5 + 0 = 9$

$$R_3^{(1)} = C_3 + \left\lceil \frac{R_3^{(0)}}{T_1} \right\rceil + \left\lceil \frac{R_3^{(0)}}{T_2} \right\rceil + B_3 = 5 + 3 + 6 + 0 = 14$$

$$R_3^{(2)} = C_3 + \left\lceil \frac{R_3^{(1)}}{T_1} \right\rceil + \left\lceil \frac{R_3^{(1)}}{T_2} \right\rceil + B_3 = 5 + 4 + 6 + 0 = 15$$

$$R_3^{(3)} = C_3 + \left\lceil \frac{R_3^{(3)}}{T_1} \right\rceil + \left\lceil \frac{R_3^{(3)}}{T_2} \right\rceil + B_3 = 5 + 4 + 6 + 0 = 15 \quad (\leq T_3 = 16)$$

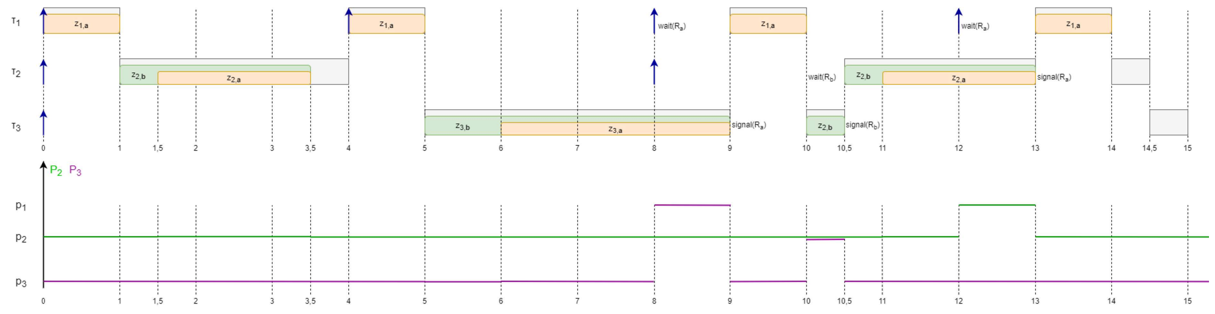
Scheduling Diagrams

The Hyper-period of task set :

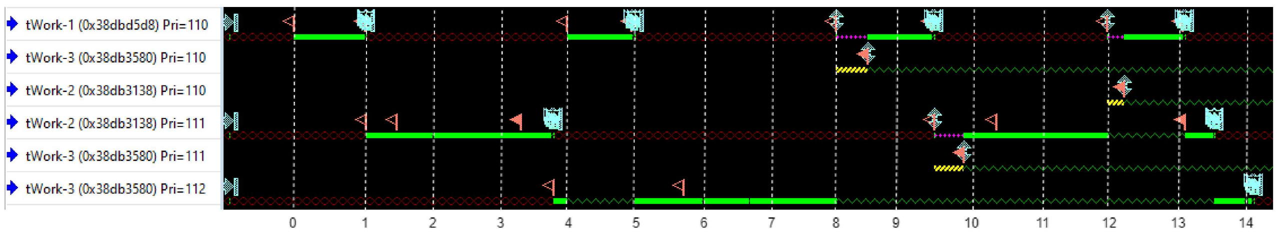
$l.c.m.\{T_1, T_2, T_3\} = l.c.m.\{4, 8, 16\} = 16$

actually being a simply periodic task set, all the tasks in harmonic relation, and synchronized, the can be identified in the period of the task T_3.

By hand:



From VxWorks:


tWork-1 (0x38dbd5d8) Pri=110
tWork-3 (0x38db3580) Pri=110
tWork-2 (0x38db3138) Pri=110
tWork-2 (0x38db3138) Pri=111
tWork-3 (0x38db3580) Pri=111
tWork-3 (0x38db3580) Pri=112
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

The proposed task set Γ can be seen to be deadlock-free, as by looking the tasks τ_2, τ_3 (having nested access to the two resources), both tasks have a total ordering where the resource R_b is always the resource with critical section higher than the one of resource R_a;

From the scheduling above shown, it can be noticed that both the tasks τ_2, τ_3, would block τ_1, at time unit 12 and 8 respectively; And thus at those time units the two tasks would inherit the priority of τ_1 for the time required to finish the computation in the critical section of resource R_a; Furthermore, also the task τ_2 is seen to be blocked therefore task τ_3 would assume priority p2 at time unit 10 (approximately) to be able to execute the remaining half time unit in the critical section of resource R_b.

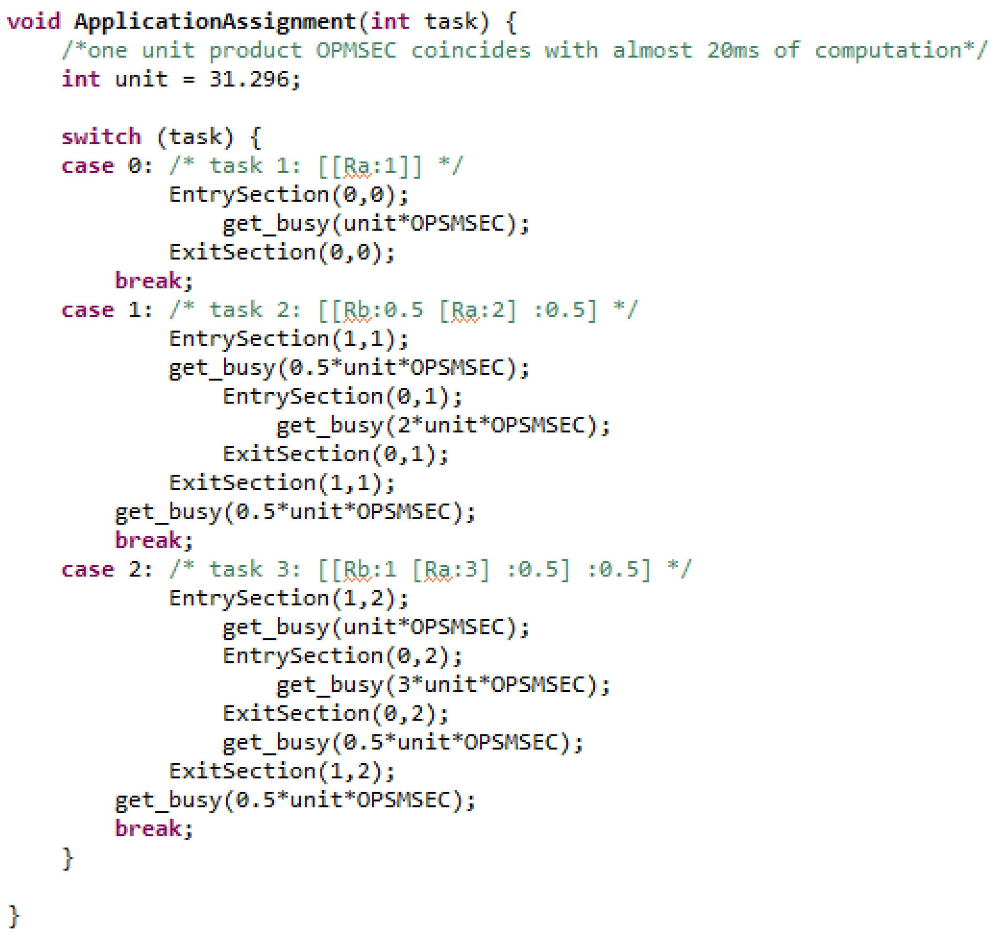
From the feasibility analysis the task set cannot be proven to be feasible; As both methods, the extended Static Priority Ordering and the extended Response Time Analysis, do not verify the feasibility of the task set. However, through the direct scheduling of Γ, no overruns have occurred and thus the feasibility of the task set scheduled with PIP has been proved. This type of result is not unexpected as regarding resource access scheduling the feasibility analysis as a validity only of sufficient condition and not as necessary, meaning that in case the feasibility test fails in assessing the feasibility of a task set this does not mean that the task would result to be unfeasible.

The tasks are found to be synchronous, 0 phase, which is the worst possible case, in terms of feasibility; Thus considering the same Γ but whit τ_1, τ_2, τ_3 having some phase shift between each other, it can be said a priory that the task set would be found to be still feasible.

The Code

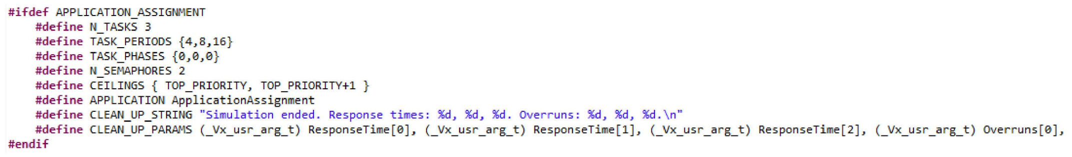
Referring to the files provided by the Laboratories Sections 13 (metascheduler.c, metascheduler.h, simulation.c, simulation.h, resources.c, resources.h, applications.c, applications.h, dummyTask.c, dummyTask.h)The following section of code have been modified/added in order to build the Application properly.

Definition of the application in the file application.c



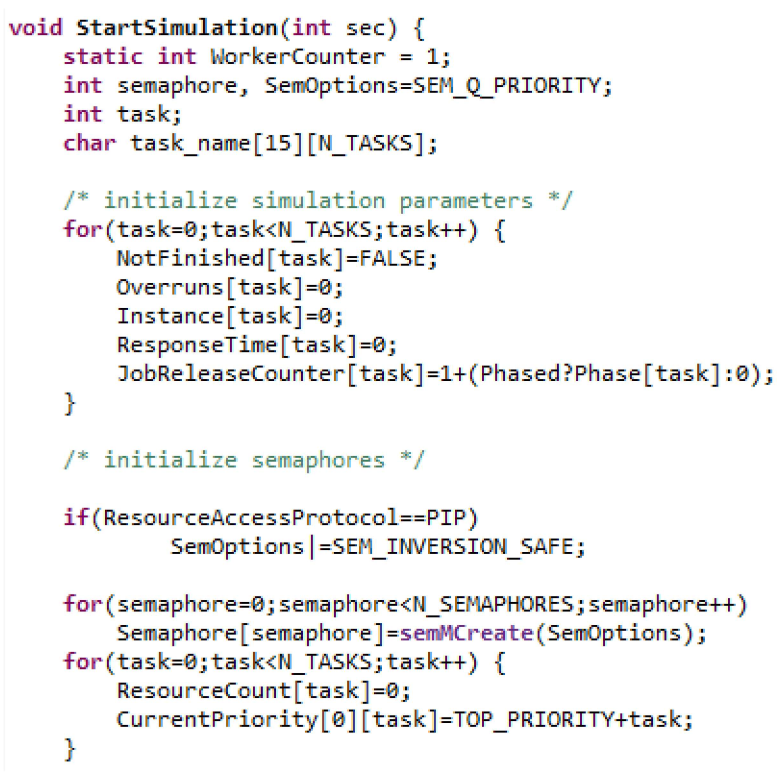
```
void ApplicationAssignment(int task) {
    /*one unit product OPMSEC coincides with almost 20ms of computation*/
    int unit = 31.296;

    switch (task) {
    case 0: /* task 1: [[Ra:1]] */
            EntrySection(0,0);
                get_busy(unit*OPSMSEC);
            ExitSection(0,0);
        break;
    case 1: /* task 2: [[Rb:0.5 [Ra:2] :0.5] */
            EntrySection(1,1);
            get_busy(0.5*unit*OPSMSEC);
                EntrySection(0,1);
                    get_busy(2*unit*OPSMSEC);
                ExitSection(0,1);
            ExitSection(1,1);
        get_busy(0.5*unit*OPSMSEC);
        break;
    case 2: /* task 3: [[Rb:1 [Ra:3] :0.5] :0.5] */
            EntrySection(1,2);
                get_busy(unit*OPSMSEC);
                EntrySection(0,2);
                    get_busy(3*unit*OPSMSEC);
                ExitSection(0,2);
                get_busy(0.5*unit*OPSMSEC);
            ExitSection(1,2);
        get_busy(0.5*unit*OPSMSEC);
        break;
    }

}
```

Definition of the application specifics in application.h



```
#ifdef APPLICATION_ASSIGNMENT
    #define N_TASKS 3
    #define TASK_PERIODS {4,8,16}
    #define TASK_PHASES {0,0,0}
    #define N_SEMAPHORES 2
    #define CEILINGS { TOP_PRIORITY, TOP_PRIORITY+1 }
    #define APPLICATION ApplicationAssignment
    #define CLEAN_UP_STRING "Simulation ended. Response times: %d, %d, %d. Overruns: %d, %d, %d.\n"
    #define CLEAN_UP_PARAMS (_Vx_usr_arg_t) ResponseTime[0], (_Vx_usr_arg_t) ResponseTime[1], (_Vx_usr_arg_t) ResponseTime[2], (_Vx_usr_arg_t) Overruns[0],
#endif
```

Initialization of the Simulations parameters and semaphores (keeping particularly attention to the definition of SemOptions for the PIP algorithm)



```c
void StartSimulation(int sec) {
    static int WorkerCounter = 1;
    int semaphore, SemOptions=SEM_Q_PRIORITY;
    int task;
    char task_name[15][N_TASKS];

    /* initialize simulation parameters */
    for(task=0;task<N_TASKS;task++) {
        NotFinished[task]=FALSE;
        Overruns[task]=0;
        Instance[task]=0;
        ResponseTime[task]=0;
        JobReleaseCounter[task]=1+(Phased?Phase[task]:0);
    }

    /* initialize semaphores */

    if(ResourceAccessProtocol==PIP)
            SemOptions|=SEM_INVERSION_SAFE;

    for(semaphore=0;semaphore<N_SEMAPHORES;semaphore++)
        Semaphore[semaphore]=semMCreate(SemOptions);
    for(task=0;task<N_TASKS;task++) {
        ResourceCount[task]=0;
        CurrentPriority[0][task]=TOP_PRIORITY+task;
    }
```