CSE 522: RTES
Assignment 01

Name: Hrishikesh Dadhich
ASU_ID: 1222306930

Description of implementation:

1) **Parse and create arrays:**
    a) Use task_model.h to create the thread, mutex, semaphore, and timer arrays.
2) **The "activate" command:**
    a) I have used "conditional variable" to implement functionality to start all threads after getting the "active" command from the user.
    b) All threads will wait for conditional variable initially and upon signal/broadcast, they will start the computation.
3) **Execution time:**
    a) I have not used traditional k_msleep or k_sleep to implement <wait_for_period> Instead I am using a combination of a timer and semaphore for each thread.
    b) Each thread will start the timer at the start of the computation (i.e. first statement of while(1) with timeout equal to the period of the thread.
    c) The timer callback will give "sem_give" the semaphore.
    d) Each thread will get blocked at "sem_take" upon completion of the task_body.
4) **Deadline error message:**
    a) I am maintaining a global "compute_flag" array of size equal to NUM_THREADS.
    b) A timer callback will check the flag and throw the error message accordingly if compute_flag[thead_id] is false.
    c) Each thread will set compute_flag[thread_id] to true upon completion of the task.

System View Screenshots:

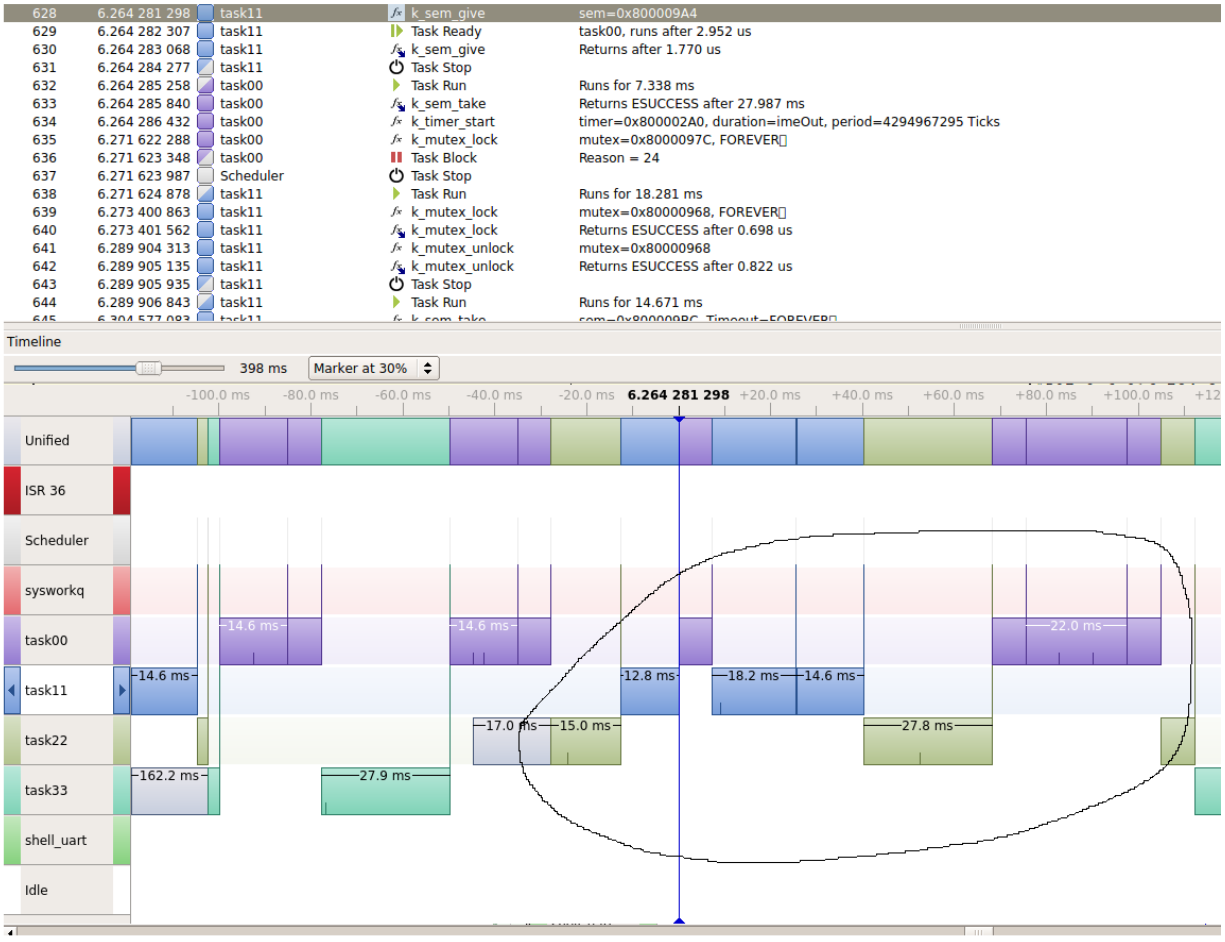
The task set used for both screenshots is given below

```
#define NUM_MUTEXES 3                   // number of mutexes
#define NUM_THREADS  4                  // number of threads
#define TOTAL_TIME 4000          // total execution time in milliseconds

#define THREAD0 {"task00", 2, 50, {400000, 400000, 400000}, 1}
#define THREAD1 {"task11", 3, 160, {800000, 900000, 800000}, 0}
#define THREAD2 {"task22", 4, 220, {200000, 2000000, 400000}, 1}
```

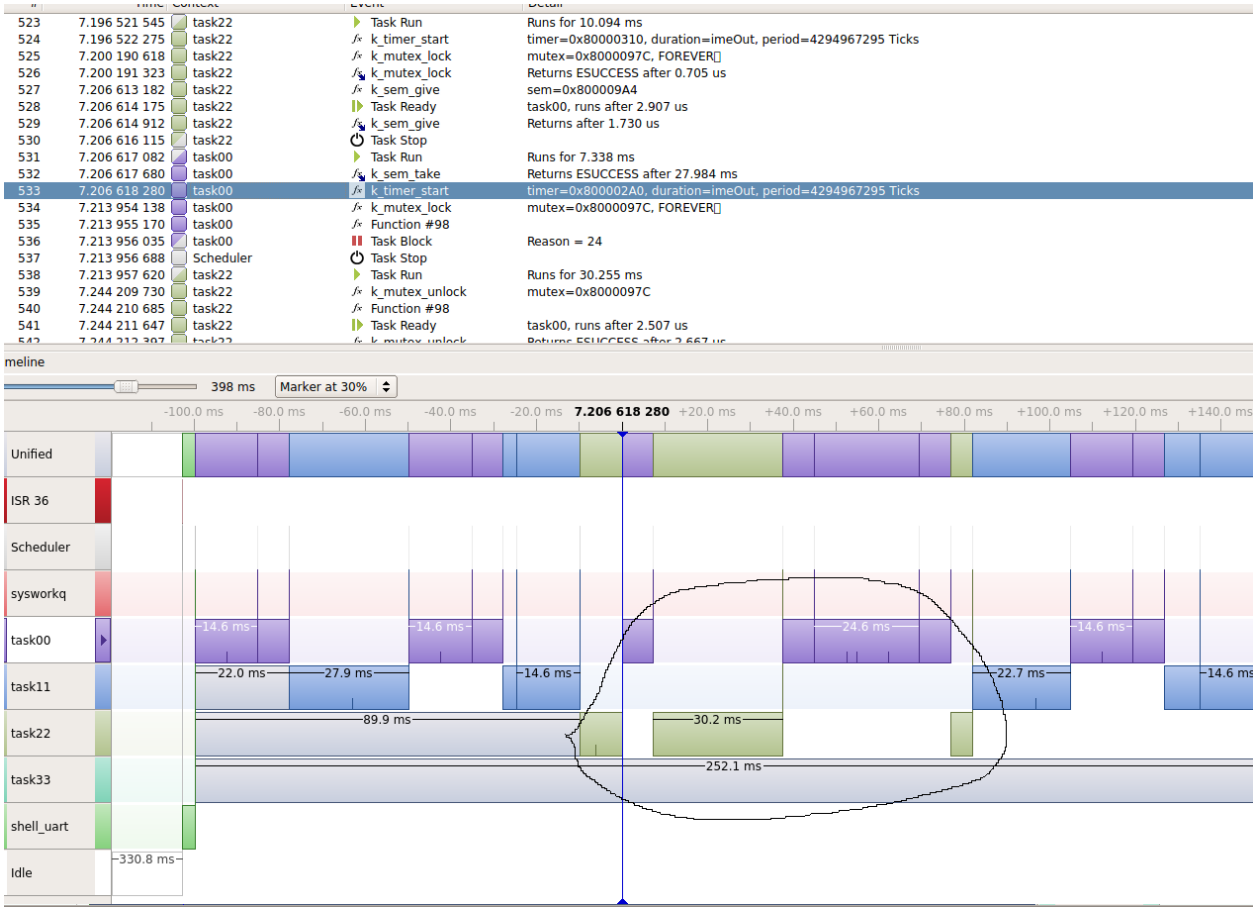
#define THREAD3 {"task33", 5, 360, {200000, 2000000, 400000}, 2}

1) CONFIG_PRIORITY_CEILING=10



628	6.264 281 298	task11	k_sem_give	sem=0x800009A4
629	6.264 282 307	task11	Task Ready	task00, runs after 2.952 us
630	6.264 283 068	task11	k_sem_give	Returns after 1.770 us
631	6.264 284 277	task11	Task Stop	
632	6.264 285 258	task00	Task Run	Runs for 7.338 ms
633	6.264 285 840	task00	k_sem_take	Returns ESUCCESS after 27.987 ms
634	6.264 286 432	task00	k_timer_start	timer=0x800002A0, duration=imeOut, period=4294967295 Ticks
635	6.271 622 288	task00	k_mutex_lock	mutex=0x8000097C, FOREVER
636	6.271 623 348	task00	Task Block	Reason = 24
637	6.271 623 987	Scheduler	Task Stop	
638	6.271 624 878	task11	Task Run	Runs for 18.281 ms
639	6.273 400 863	task11	k_mutex_lock	mutex=0x80000968, FOREVER
640	6.273 401 562	task11	k_mutex_lock	Returns ESUCCESS after 0.698 us
641	6.289 904 313	task11	k_mutex_unlock	mutex=0x80000968
642	6.289 905 135	task11	k_mutex_unlock	Returns ESUCCESS after 0.822 us
643	6.289 905 935	task11	Task Stop	
644	6.289 906 843	task11	Task Run	Runs for 14.671 ms

2) CONFIG_PRIORITY_CEILING=0



523	7.196 521 545	task22	Task Run	Runs for 10.094 ms
524	7.196 522 275	task22	k_timer_start	timer=0x80000310, duration=imeOut, period=4294967295 Ticks
525	7.200 190 618	task22	k_mutex_lock	mutex=0x8000097C, FOREVER
526	7.200 191 323	task22	k_mutex_lock	Returns ESUCCESS after 0.705 us
527	7.206 613 182	task22	k_sem_give	sem=0x800009A4
528	7.206 614 175	task22	Task Ready	task00, runs after 2.907 us
529	7.206 614 912	task22	k_sem_give	Returns after 1.730 us
530	7.206 616 115	task22	Task Stop	
531	7.206 617 082	task00	Task Run	Runs for 7.338 ms
532	7.206 617 680	task00	k_sem_take	Returns ESUCCESS after 27.984 ms
533	7.206 618 280	task00	k_timer_start	timer=0x800002A0, duration=imeOut, period=4294967295 Ticks
534	7.213 954 138	task00	k_mutex_lock	mutex=0x8000097C, FOREVER
535	7.213 955 170	task00	Function #98	
536	7.213 956 035	task00	Task Block	Reason = 24
537	7.213 956 688	Scheduler	Task Stop	
538	7.213 957 620	task22	Task Run	Runs for 30.255 ms
539	7.244 209 730	task22	k_mutex_unlock	mutex=0x8000097C
540	7.244 210 685	task22	Function #98	
541	7.244 211 647	task22	Task Ready	task00, runs after 2.507 us

Explanation of above screenshots.

Scenario:

In the above figures, the priority of task22 < task11 < task00.

Task00 and task22 share the common mutex (resource).

Task22 gets preempted by task11, task11 gets preempted by task00.

Task00 tries to get the mutex lock which is currently acquired by task22 and hence task00 goes in a waiting state.

1) When CONFIG_PRIORITY_CEILING is set to 0 the task22 now becomes a task with the highest priority and the scheduler makes task22 in a ready state to release the mutex lock as soon as possible.

2) When CONFIG_PRIORITY_CEILING is set to 10, task11 gets scheduled over task22 as the priority of task11 is greater than task22 and hence task00 gets delayed.