The University of Texas at Austin
Department of Electrical and Computer Engineering

# Real-Time Systems / Real-Time Operating Systems

**ECE445M/ECE380L.12, Spring 2023**

---

## Midterm Exam

**Date:** March 2, 2023

UT EID: _____

Printed Name: _____
Last, First

Your signature is your promise that you have not cheated and will not cheat on this exam, nor will you help others to cheat on this exam:

Signature: _____

**Instructions:**

- Open book, open notes and open web.
- No electronic devices other than your laptop/PC (cell phones off and stowed away).
- You are allowed to access any resource on the internet, but no electronic communication other than with instructors.
- Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space (boxes) provided.
- For all questions, unless otherwise stated, find the most efficient (time, resources) solution.

| | | |
|---|---|---|
| **Problem 1** | 20 | |
| **Problem 2** | 20 | |
| **Problem 3** | 20 | |
| **Problem 4** | 25 | |
| **Problem 5** | 15 | |
| **Total** | 100 | |

**Problem 1 (20 points): Miscellaneous**

a) What are two advantages of preemptive schedulers over cooperative schedulers? What are some advantages of cooperative schedulers?

b) Describe what happens in your Lab 2/Lab 3 OS when you call *OS_Sleep()* from a background thread.

c) Describe what happens when you call *OS_Wait()* from a background thread with spinlock semaphores? What about with blocking semaphores?

d) There are several ways to make critical sections atomic. What is the difference between using *Disable*/*EnableInterrupts*, *Start*/*EndCritical*, and semaphores for synchronization?

**Problem 2 (20 points): Context Switching**

Given the following TCB structure:

```
typedef struct TCB TCB_t;
struct TCB {
    uint32_t  id;         /* thread ID */
    uint32_t* sp;         /* stack pointer */
    TCB_t* next;          /* pointer to the next TCB */
    TCB_t* previous;      /* pointer to the previous TCB */
};
TCB_t RunPt;
```

a) Write assembly code for a context switch not done in an interrupt handler (i.e. not using *PendSV* or any other interrupt) using the TCB defined above assuming a round-robin scheduler.

```
ContextSwitch
```

b) Write assembly code for a context switch in the *PendSV Handler* using the TCB defined above assuming a round-robin scheduler.

```
PendSV_Handler
```

**Problem 3 (20 points): Synchronization**

Given the application below that runs on an OS using a strict priority scheduler with blocking sempahores:

```
uint32_t Count1 = 0;
uint32_t Count2 = 0;
uint32_t Count3 = 0;
```

```
sema_t mutex;
sema_t lock;
```

```
void Thread1(void) {
  OS_Sleep(50);
  while(1){
    OS_Wait(&mutex);
    OS_Wait(&lock);
    Count1++;
    OS_Sleep(10);
    OS_Signal(&lock);
    OS_Signal(&mutex);
  }
}

void Thread2(void) {
  while(1) {
     OS_Wait(&lock);
     OS_Wait(&mutex);
     Count2++;
     OS_Sleep(50);
     OS_Signal(&mutex);
     OS_Signal(&lock);
  }
}
```

```
void Thread3(void) {
  OS_Sleep(10);
  while(1) {
    Count3++;
  }
}

void Idle(void) {
  while(1) {
    WaitforInterrupt();
}

main {
  OS_Init()
  OS_InitSemaphore(&mutex, 1);
  OS_InitSemaphore(&lock, 1);
  OS_AddThread(Thread1,0);
  OS_AddThread(Thread2,2);
  OS_AddThread(Thread3,1);
  OS_AddThread(Idle,3); // lowest prio
  OS_Launch(2MS_TIME);
}
```

a) The code above has a bug. Find and describe the bug, and describe possible solutions.

b) Now assume that a student removes *Thread3*. This code still has a bug. Explain what issue remains and how it might be fixed. Is there any solution that fixes both a) and b) without changing the application?

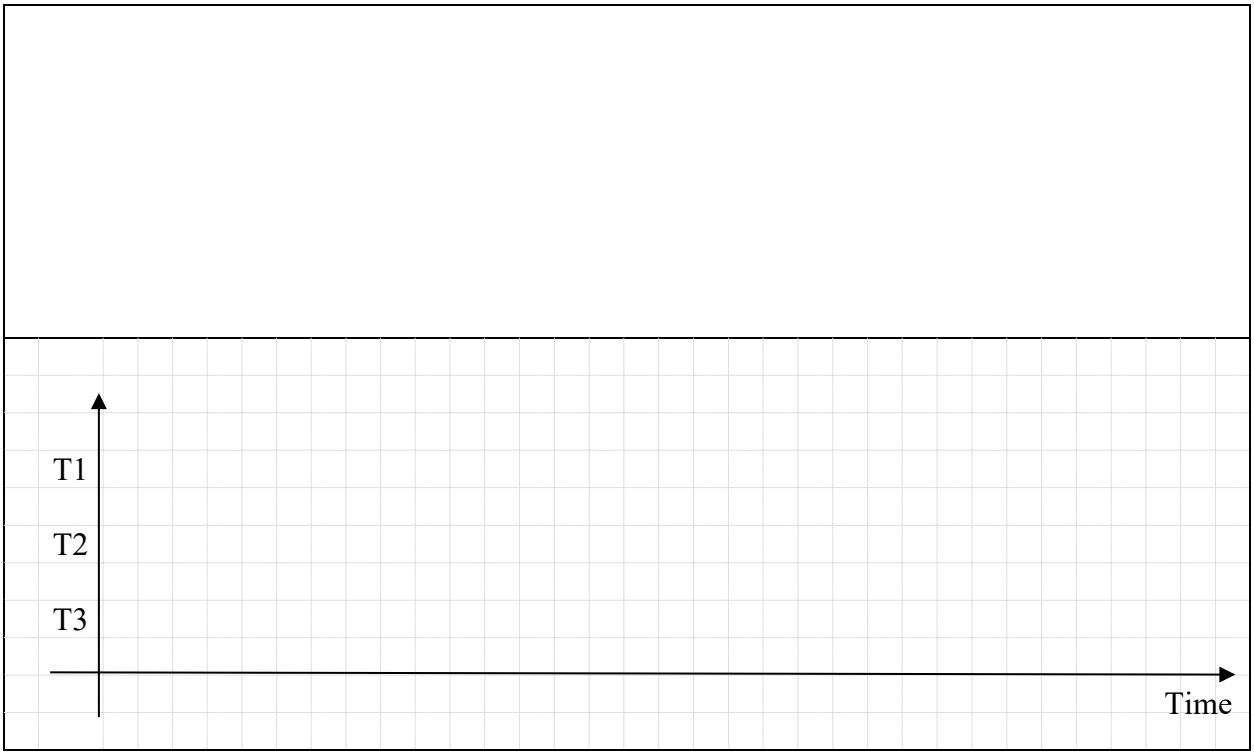
**Problem 4 (25 points): Scheduling**

You are given a system with 3 tasks listed below.

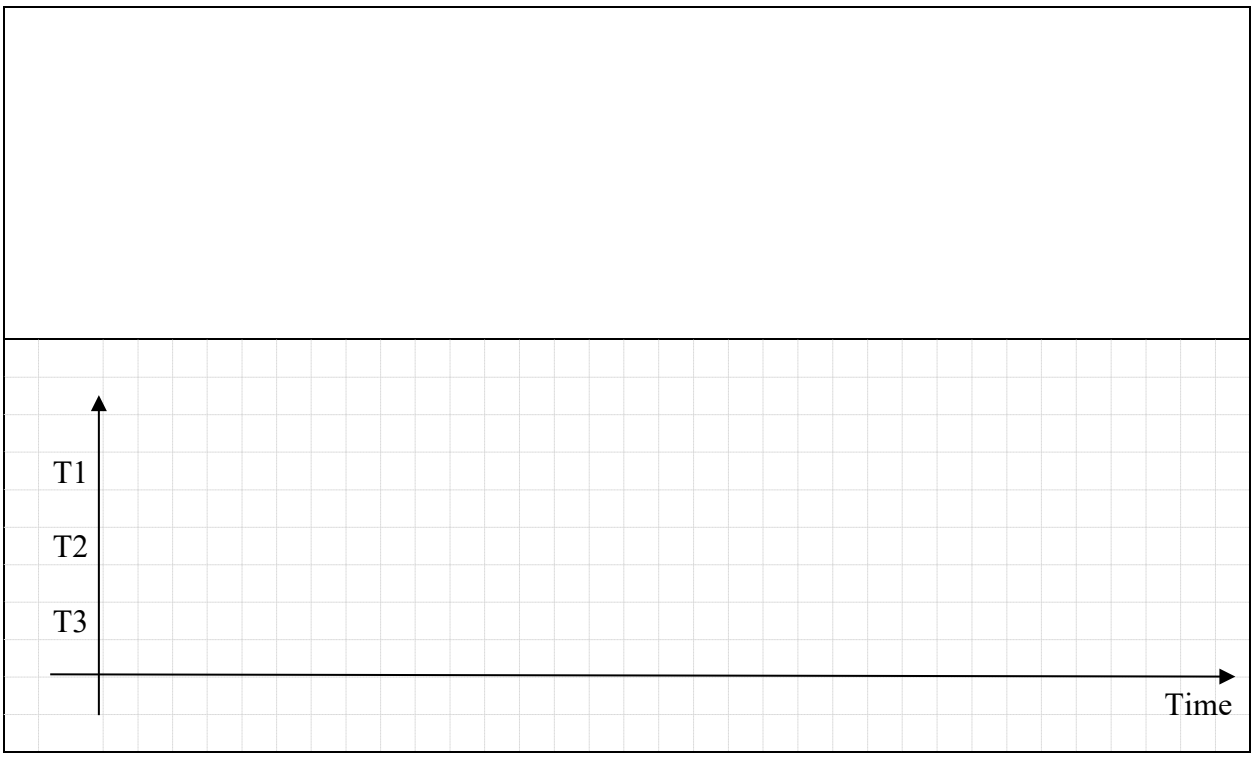
| Task | Execution Time | Period |
|---|---|---|
| T1 | 2ms | 5ms |
| T2 | 3ms | 10ms |
| T3 | 4ms | 15ms |

a) What is the CPU utilization running these tasks? Is a system with this CPU utilization guaranteed to be schedulable by RMS and/or EDF?

b) Show the EDF schedule on the diagram below. Is this task set schedulable by EDF? If so, what is the maximum overhead that context switching can take. If not, specify what deadline is missed.




c) Show the RMS schedule on the diagram below. Is this task set schedulable by RMS? If so, what is the maximum overhead that context switching can take. If not, specify what deadline is missed.



**Problem 5 (15 points): Semaphores**

a) Implement a spinlock semaphore that will call *OS_DeadlockResart* after a certain amount of time (DEAD_LOCK_TIME, given in ms) if the semaphore can not be acquired before. You may call *OS_MsTime*, which returns the system time in ms counting up. Multiple threads can call *OS_Wait* on the same or different semaphores. Do not add anything to the TCB.

```
void OS_Wait(Sema4Type *semaPt) {




}
```

b) What would you need to do to implement a similar deadlock checker for a blocking semaphore? Assume that the blocking semaphore functions are implemented using a linked list per semaphore. (You do not need to code this, but simply explain/write some pseudo code.)