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

Real-Time Systems / Real-Time Operating Systems

EE445M/ECE380L.12, Spring 2022

Final Exam

Date: May 14, 2022

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	15	
Problem 3	5	
Problem 4	20	
Problem 5	20	
Problem 6	20	
Total	100	

Problem 1 (20 points): Context Switching

Context switching overhead can limit performance on systems with many tasks and a lot of context switches. Given the following context switch implementation:

```
PendSV_Handler
  CPSID I
  PUSH  {R4-R11}
  LDR   R0, =RunPt
  LDR   R1, [R0]
  STR   SP, [R1]
  LDR   R1, [R1, #4]
  STR   R1, [R0]
  LDR   SP, [R1]
  POP   {R4-R11}
  CPSIE I
  BX    LR
```

a) How long (in ns) does it take to perform a context switch in this implementation? Assume each Load/Store takes 3 cycles, Push/Pop are 3 cycles per register, and all other instructions take 1 cycle. Also assume a 80 MHz clock and 10 cycles to enter/exit the interrupt not including pushes/pops.

b) A way to save context switch time is to avoid the interrupt overhead and perform context switches in a cooperative manner using *OS_Suspend* calls realized as regular subroutines, as we did in the very first context switch implementation shown in class. What are the issues and limitations with such an approach? Under what conditions can context switches be executed this way (without using an interrupt and interrupt handler)?

c) Another way to reduce context switch overhead is to reduce the number of or completely avoid saving any registers on the stacks. Suppose we only PUSH and POP a subset (or none) of the registers R4-R11 in the *PendSV_Handler*, what are the issues and limitations with such an approach? Under what conditions can context switches be executed in this way?

d) An approach that combines solutions from b) and c) is called *co-routines*. In co-routines, lightweight context switches are performed as cooperative subroutine calls that save only a subset of registers, inserted at appropriate locations in the code (determined by the user or compiler) where it is guaranteed that respective conditions are satisfied. Assuming a co-routine approach where locations are chosen such that no registers need to be saved at all, implement such a co-routine variant of *OS_Suspend*. What is the context switch delay in this case?

```
OS_Suspend
```

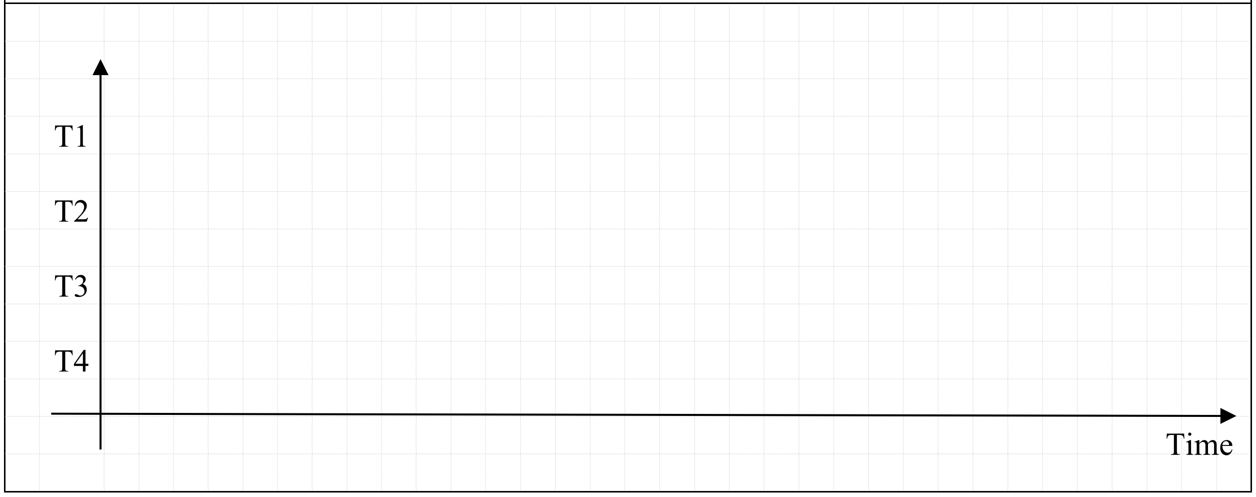
Problem 2 (15 points): Scheduling

Below is the list of tasks for those tasks from Problem 3 in the midterm. In the midterm, you examined the context switching overhead of this schedule using standard algorithms.

Task	**Execution Time**	**Period**
T1	1 ms	5 ms
T2	1 ms	6 ms
T3	2 ms	10 ms
T4	6 ms	15 ms

a) Are EDF or RMS optimal to minimize the number of context switches? Why or why not?

b) Draw a schedule that minimizes the number of context switches while meeting all deadlines. What is the maximum context switch overhead (in ms) such that the tasks are schedulable?



*Name:*______________________________

Problem 3 (5 points): SVC Handler

Assume an OS that implements *OS_Kill* called from user threads via SVC traps. Shown below is a SVC Handler and a skeleton of your *OS_Kill* function. This implementation has an error in it. What is the error, how does it manifest itself and why is it happening. How can you correct it?

```
SVC Handler
  LDR R12, [SP, #24]
  LDRH R12, [R12, #-2]
  BIC R12, #0xFF00
  LDR  SP, {R0-R3}
  …
  BL OS_Kill
  …
  STR R0, [SP]
  BX LR
```

```
OS_Kill() {
  DisableInterrupt();

  // Remove from run list
  runPt->prev->next = runPt->next;
  runPt->next->prev = runPt->prev;

  // Switch to next thread
  OS_Suspend();
  EnableInterrupts();

  // Prevent running dead thread
  for(;;);
}
```

Problem 4 (20 points): Process Loading

a) Is position-independent code and data sufficient to address relocation when dynamically loading processes like in Lab 5? Why or why not?

b) In Lab 5, the ELF loader performed patching (aka relocation) at load time. Was this necessary or could we have handled all of the patched cases via SVC traps and an *SVC_Handler*?

c) Alternatively, could we have used patching to replace all SVC traps and the *SVC_Handler?*

d) In a virtual memory system, where each process has its own private virtual address space, are position independence, patching/relocation and/or SVC traps still required?

Problem 5 (20 points): Networking

Assume you want to implement a system transfering files between two TM4Cs that are connected via Ethernet and CAN interfaces. Each TM4C has an SD card connected through the SPI protocol.

a) When your SPI bus is running at a clock rate of 4MHz, how long does it take and what is hence the effective bandwidth for reading a contiguous block of 64MB of data from the SD card. Assume zero command-response delay (NCR=0) and an immediate data start (i.e. zero SD card latency and data packet delay).

b) Assuming a CAN 2.0A bus running at a baud rate of 1Mbit/s using 11bit IDs and assuming no bit stuffing is needed, how long does it take and what is thus the effective bandwidth for sending 64MB of data?

c) Assuming an Ethernet physical layer baud rate of 10Mbit/s, how long does it take and what is this the effective bandwidth for sending 64MB of data? You can assume that there are no other machines on the Ethernet network.

d) How long does it take to first read 64MB of data from the SD card and then send it through CAN 2.0A or Ethernet, and what is the effective bandwidth for each case? Which network interface should you use for the file transfer, and why? Is there any way to reduce transfer time and achieve a higher effective bandwidth? If so, how and what is the maximum bandwidth?

e) Now, assume that there are 40 TM4Cs connected to each other using CAN and Ethernet, where half of them are sending a 64MB file to another TM4C at the same time. How long does it take for all 20 transfer to be complete and what is this the effective system bandwidth? Which network should you use to connect the TM4Cs, CAN or Ethernet? Justify your answer.

Problem 6 (20 points): Filesystem

You are asked to design a filesystem that supports SD cards of up to 32GiB (2^{35} bytes) size, where each block holds 512 bytes. The filesystem should support at least 60 files, where you can assume that each directory entry takes up 8 bytes.

a) How many SD card blocks need to be reserved for the directory?

b) You choose to use linked allocation. What is the largest file size (in bytes) you can support and why? You don't need to write out the actual number, just the expression showing how it is derived and computed is enough.

c) You choose instead to use a FAT file system. What is the largest file size (in bytes) you can support and why? Again, it is ok to express file size as formula instead of final number.

d) Assuming that you can cache the directory, one FAT and one data block in memory, given the following sequence of file system accesses, what are the best-case and worst-case number of SD card reads for the two file systems (linked, FAT). Assume that a FAT block remains in memory until it is evicted by another FAT block load. The command *Read(file* f, int X)* should return 1 byte at the *X*th byte position in file *f*.

```
// Open file with read permission
File* f = fopen("./testfile.txt", "r")

char A = Read(f,100);
char AA = Read(f,int(A));
char B = Read(f,1000);
char BB = Read(f,int(B));
char C = Read(f,2000);
char CC = Read(f,int(C));
```