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

Real-Time Systems / Real-Time Operating Systems

EE445M/EE380L.12, Spring 2020

Final Exam

Date: May 13, 2020

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 calculators or any electronic devices other than your laptop/PC (turn cell phones off).
- 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	15	
Problem 2	15	
Problem 3	15	
Problem 4	20	
Problem 5	10	
Problem 6	15	
Problem 7	10	
Total	100	

*Name:*________________________________

Problem 1 (15 points): LDREX/STREX

Given the following semaphore implementation using LDREX/STREX:

```
OS_Wait ; R0 points to counter
   LDREX    R1, [R0] ; counter
   SUBS     R1, #1   ; counter -1,
   ITT      PL       ; ok if >= 0
   STREXPL R2,R1,[R0]; try update
   CMPPL    R2, #0   ; succeed?
   BNE      OS_Wait  ; no, try again
   BX       LR
```

```
OS_Signal ; R0 points to counter
   LDREX    R1, [R0]  ; counter
   ADD      R1, #1    ; counter + 1
   STREX    R2,R1,[R0]; try update
   CMP      R2, #0    ; succeed?
   BNE      OS_Signal ; no, again
   BX       LR
```

a) Is this a spinlock or blocking semaphore implementation?

b) Is this a cooperative or non-cooperative semaphore implementation?

c) If cooperative, how can it be made non-cooperative, and if non-cooperative, how can it be made cooperative?

d) Given the following code from the midterm, where *Dump()* can be called by multiple threads:

```
char DebugDump[DUMP_SIZE];
unsigned int DebugCnt = 0;

void Dump(char c) {
  long sr;
  sr = StartCritical();
  DebugDump[DebugCnt++] = c;
  UART_OutChar(c);
  if(DebugCnt >= DUMP_SIZE)
    DebugCnt = 0;
  EndCritical(sr);
}
```

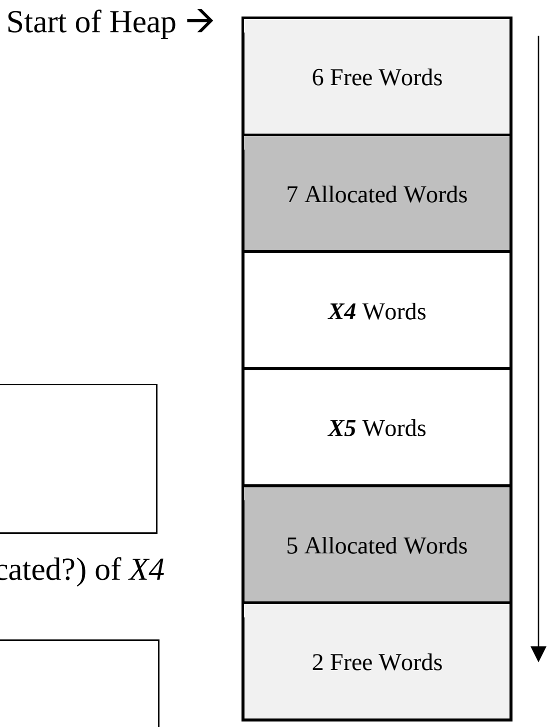
```
long UARTSema = 1;

void UART_OutChar(char c) {
  OS_Wait(&UARTSema);
  // UART code here
  ...
  OS_Signal(&UARTSema);
}
```

Is there still an issue with this code using the LDREX/STREX semaphore implementation above? If so, what is the issue? If not, why not?

Problem 2 (15 points): Heap

Given below is a partially complete series of heap accesses and the corresponding heap state (not drawn to scale). Accesses to the heap are done using *Heap_Alloc(X)*, which allocates *X* **words**. Each allocated block on the heap requires 2 additional words of metadata overhead, i.e. if *X* words are requested, *X*+2 words are actually allocated on the heap. The heap is 32 words total.

```
Heap_Init();
P1 = Heap_Alloc(4);
P2 = Heap_Alloc(7);
P3 = Heap_Alloc(8);
Heap_Free(X1);
P4 = Heap_Alloc(5);
Heap_Free(X2);
P5 = Heap_Alloc(X3);
```



a) What are the values of *X1*, *X2*, and *X3*?

b) What are the values and allocation status (free or allocated?) of *X4* and *X5*?

c) What heap algorithm is used? How do you know?

d) What is the largest *X6* that a *Heap_Alloc(X6)* can still allocate after the sequence of calls above?

Problem 3 (15 points): File system

Assume a FAT file system with 3 files currently in it. A zero in the File Allocation Table (FAT) signifies the end of a file. The disk size is 2kB, and consists of 16 blocks. Assume that all disk metadata (directory and FAT) is stored in the first two blocks.

a) Given the following Directory and FAT, fill in the missing information.

File Allocation Table

Index	Data
0	X
1	X
2	
3	6
4	12
5	0
6	7
7	
8	3
9	2
10	
11	10
12	9
13	0
14	0
15	14

Directory

File Name	Size	Start of file
A	4	
B	8	
C	1	
Free space		5

b) What is the size of each disk block?

c) What is the largest new file that can be created on the disk in its current state as above?

d) How much space does the FAT occupy on disk? What is the minimum space needed per directory entry?

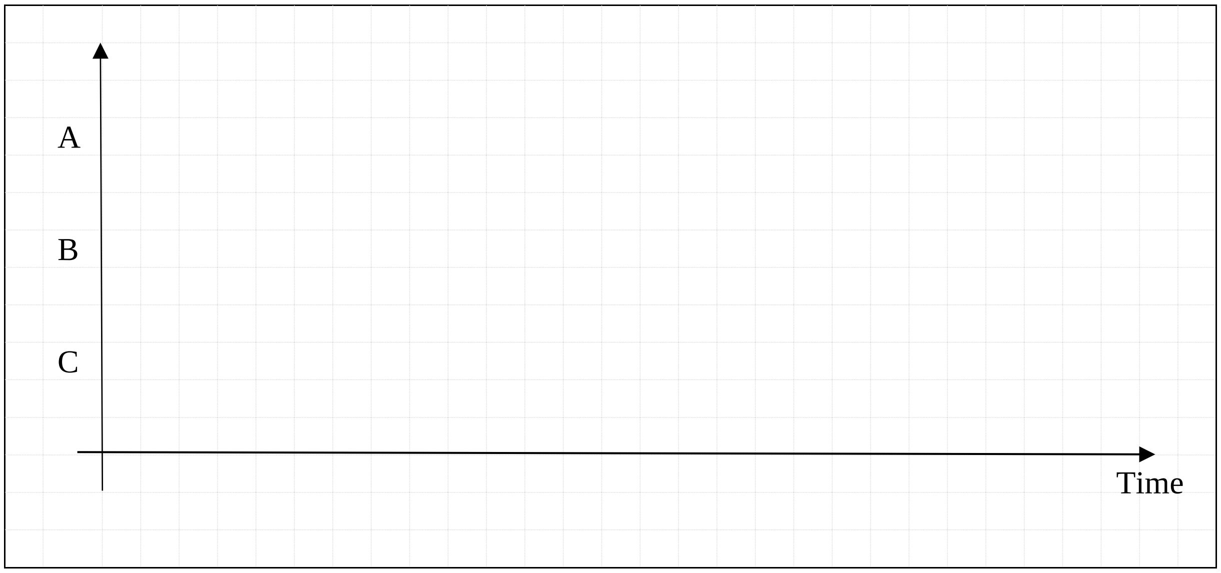
Problem 4 (20 points): Scheduling

Consider a real-time system running three periodic tasks with the following periods (= deadlines) and execution times. You can assume zero context switch and interrupt overhead.

Task	**Execution time**	**Period**
Task A	10ms	30ms
Task B	15ms	60ms
Task C	15ms	40ms

a) What is the processor utilization when executing this task set.

b) Draw the schedule of task executions over time using a round-robin scheduler with a 5ms time slice. Assume that all tasks become ready to execute, i.e. start their first period at time zero. Draw one iteration of the schedule until it starts repeating. Is the task set schedulable, i.e. do all tasks finish their execution before the start of their next period (=deadline)?



c) Assign priorities to tasks to implement a rate monotonic scheduling (RMS) strategy.

d) Draw the schedule of task executions over time with a RMS scheduler. Assume that all tasks become ready to execute, i.e. start their first period at time zero. Draw one iteration of the schedule until it starts repeating. Is the task set schedulable, i.e. do all task finish their execution before the start of their next period (=deadline)?

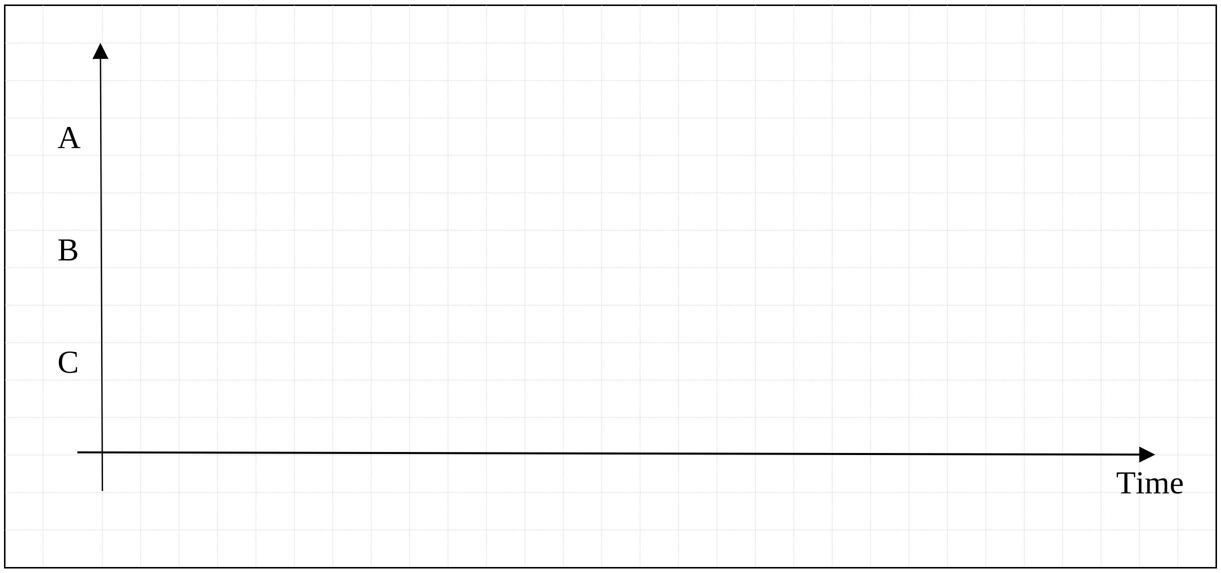
e) Finally, would an EDF scheduler be able to run this task set such that all tasks finish their execution before the start of their next period (=deadline)? Why or why not?

Problem 5 (10 points): Ethernet

Now consider a real-time system with three devices connected over an Ethernet network. Each device will need to send an Ethernet packets with given sizes (transmission times) periodically as indicated below. The system uses a real-time variant of CSMA/CD invented by a 445M student in which devices are assigned pre-determined and fixed backoff times as shown below. Ethernet collisions will occur if two devices start sending at the same time. Assume it takes 5ms for all devices to detect the collision after it has occurred.

Device	**Transmission time**	**Period**	**Backpff Time**
Device A	10ms	30ms	0ms
Device B	15ms	60ms	15ms
Device C	15ms	40ms	5ms

a) Draw the timeline of data transmission over the Ethernet bus. You can use terminology such as *S* for carrier sensing, *D* for collision detection and *O* for backoff. Draw the timeline for one hyperperiod. Do all tasks finish their transmission before the start of their next period (=deadline)?



b) Instead of fixed backoff times, would a dynamic backoff assignment, e.g. following an EDF-like strategy be ever a possibility in Ethernet?

Problem 6 (15 points): SVC Handler

Given the following assembly code of a basic SVC handler.

```
SVC_Handler

    LDR     R12,[SP,#24]   ; Return PC



    LDRH    R12,[R12,#-2]  ; SVC instruction is 2 bytes



    BIC     R12,#0xFF00    ; Decode instruction



    LDM     SP,{R0-R3}     ; Get any parameters



    BL      OS_Call



    STR     R0,[SP]        ; Store return value



    BX      LR             ; Return from exception
```

a) The code has a bug. Where is the bug and what is needed to fix it? Modify the code above to fix the bug.

b) Assume an *OS_Call* routine that expects the ID of the OS routine to call as the first parameter and then itself internally calls the correct OS routine as follows:

```
int OS_Call(int id, int para1, int para2, int para3) {
  switch(id) {
    case 0: return OS_func1(para1);
    case 1: return OS_func2((void*)para1, (short)para2);
    case 3: return OS_func3();
    …
  }
  return 0;
}
```

Show the modifications of the *SVC_Handler* code needed to correctly call an *OS_Call* with this behavior.

c) What is the maximum number of OS function parameters supported by the SVC handler from a) and your modified handler from b)? Can this approach be extended to support additional and even an arbitrary number of parameters? If so, how? Describe the changes necessary to the user code triggering the SVC traps and/or the SVC handler inside the OS.

Problem 7 (10 points): Virtual Memory

In class we discussed virtual memory and paging, and that paging effectively solves the fragmentation problem similar to how indexed allocation did for filesystems, where the page table is equivalent to the index table.

a) Could a linked allocation be used for memory systems instead? Why is this a good or bad idea? What kind of hardware support would be needed in the MMU, i.e. what would the MMU need to do on every load or store for this to work?

b) How about a FAT-like system for virtual memory management?