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

Real-Time Systems / Real-Time Operating Systems

ECE445M/ECE380L.12, Spring 2023

Final Exam

Date: April 28, 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	10	
Problem 2	20	
Problem 3	30	
Problem 4	15	
Problem 5	25	
Total	100	

Problem 1 (10 points): Miscellaneous

a) Suppose you are asked to design a new robot car using your RTOS and a PID controller. How would you define the error-term for your PID controller?

b) What are two things you could do to minimize the impact of noise when using the GP2Y0A21YK IR sensors with your robot car?

Problem 2 (20 points): OS Kernel

Below is the *SVC_Handler*, *OS_Kill()*, and *OS_AddThread()* in a multithreaded, round-robin RTOS with spinlock semaphores and dynamic process loading via a heap, where applications running on the OS trigger calls to *OS_Id()*, *OS_Kill()*, *OS_Sleep()*, *OS_Time()* and *OS_AddThread()* via SVC traps. Assume that *allocateTCB()* and *allocateStack()* do all the proper allocations and initializations associated with thread TCB and stack creation. If unsure, write down any of your assumptions.

```
SVC_Handler
  LDR  R12,[SP, #24]
  LDRH R12,[R12,#-2]
  BIC  R12, #0xFF00
  LDM  SP, {R0-R3}

  PUSH {LR}
  LDR  LR, =Return
  CMP  R12, #0
  BEQ  OS_Id
  CMP  R12, #1
  BEQ  OS_Kill
  CMP  R12, #2
  BEQ  OS_Sleep
  CMP  R12, #3
  BEQ  OS_Time
  CMP  R12, #4
  BEQ  OS_AddThread

Return
  POP  {LR}
  STR  R0, [SP]
  BX   LR
```

```
// Kill thread
OS_Kill() {
  OS_bWait(&mutex); // lock OS kernel

  // Decrement threads in process
  PCB_t* parent = runPt->pcb;
  parent->numThreads--;
  if (!parent->numThreads){
     // Kill process if last thread is killed
     Heap_Free(parent->data);
     parent->id = -1; // Mark parent as unused
  }

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

  OS_Signal(&mutex);
  OS_Suspend();
}
```

```
// Add thread, return 0 if unsuccessful
int OS_AddThread(void(*task)(void), uint32_t stackSize) {
   OS_bWait(&mutex); // lock OS kernel
   TCB_t *tcb = allocateTCB();
   If (tcb == NULL) {
      OS_bSignal(&mutex);
      return 0;
   }
   uint32_t *sp = allocateStack(task);
   if (sp == NULL) {
      OS_bSignal(&mutex);
      return 0;
   }
   enqueue(runPt, tcb); // Place TCB in running linked list

   OS_bSignal(&mutex);
   return 1;
}
```

There may be bugs associated with the code above. Please list all bugs, if any, and how they might be fixed. If there are multiple different solutions, list all possible ways to fix a bug.

Problem 3 (30 points): Heap

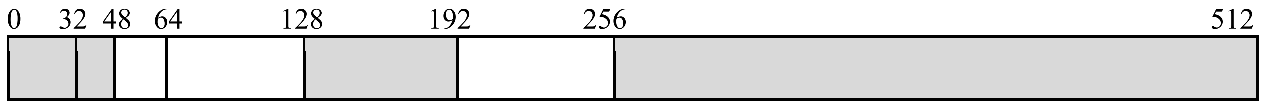
a) Assume a 512 byte heap implemented using the algorithm discussed in class, in the lecture notes and book, consisting of blocks with headers and tails for each block indicating the block size and negative sizes indicating free space. Memory is 4-byte word aligned. Given the sequence of *Heap_Malloc()* and *Heap_Free()* calls on the right, show the final state of the heap at the end of the sequence for each allocation scheme.

```
a = Heap_Malloc(14);
b = Heap_Malloc(4);
c = Heap_Malloc(3);
e = Heap_Malloc(3);
Heap_Free(a);
Heap_Free(c);
d = Heap_Malloc(4);
```

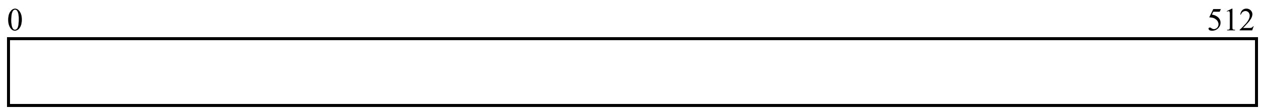
Initial Heap	First Fit	Best Fit	Worst Fit
-126			
…	…	…	…
-126			

*Name:*________________________________

b) Now assume that the 512 byte heap is implemented using the Knuth's Buddy Allocation algorithm with the smallest allocation size of 4 bytes. Given the following heap state (where grey are allocated blocks and white are free ones), list a sequence of *Heap_Malloc*/*Heap_Free* commands that could result in this heap.



c) What would Knuth's Buddy Allocation Algorithm final heap state look like for the sequence from a)? You can assume that the algorithm stores necessary meta-data to keep track of heap state outside of the heap array. Not accounting for meta-data overhead, which implementation or allocation scheme has the most and least amount of internal or external fragmentation?



*Name:*________________________________

Problem 4 (15 points): File System

a) For an SD Card with 64 GiB of memory, how many 512-byte blocks would you need to store the FAT?

b) The FAT on the right was recovered for a SD Card, but the directory was corrupted. Fill in the directory with the starting blocks and the lengths of each file and the free blocks list. You can assume that the free block list is larger than any file.

Directory:

File Name	Starting Block	File Size (in blocks)
Free		

FAT:

0	X
1	X
2	4
3	10
4	8
5	12
6	14
7	21
8	-
9	-
10	-
11	-
12	13
13	20
14	17
15	9
16	7
17	18
18	19
19	24
20	11
21	22
22	23
23	31
24	25
25	26
26	27
27	28
28	29
29	30
30	-
31	-

Problem 5 (25 points): System Design

a) Assuming you have a round robin scheduler with a time slice of 2 ms with 5 tasks. Suppose you want to log data to your SD card in Task 1. The amount of data to log is 24 MB and it takes 6000 time slices of running Task 1 to collect the data. The data is not written to disk by Task 1 until all 24 MB have been collected. Assuming a disk with 512 byte blocks and an SPI bus running at a clock rate of 4 MHz using single-block transfers only, what is the amount of time needed to finish logging (i.e. collecting and writing) the data in ms? Assume zero command-response delay (NCR=0). Please show your work.

b) Now assume that we want to do remote logging to a second TM4C connected via Ethernet. After collecting the data, the first TM4C sends the log via the network and the second TM4C writes it to their SD card. The receiver TM4C first buffers all data before writing it to disk. Assuming your SPI is running at 4 MHz and an Ethernet physical layer baud rate of 10 Mbits/s, what is the total time needed to log the data? You can assume that there are no other machines on the Ethernet network and zero SPI command response delay. Please show your work.

c) What is the total time needed to log the data if we change from Ethernet to using a CAN running at 1Mbit/s with 11 bit IDs assuming no stuffing is needed. Please show your work.

d) Now assume that the first TM4C can start collecting the next batch of data as soon as it is finished sending the previous one. What is the maximum rate at which data can be logged using Ethernet versus CAN? What is the maximum rate at which data can be logged assuming DMA is used on both TM4Cs to send and receive data, i.e. to copy the data from memory to the Ethernet or CAN controller and vice versa?