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

Real-Time Systems / Real-Time Operating Systems

EE445M/EE380L.12, Spring 2018

Final Exam

Date: May 10, 2018

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 and open notes.
- No calculators or any electronic devices (turn cell phones off).
- Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space (boxes) provided.
- *Anything outside the boxes will be ignored in grading.*
- For all questions, unless otherwise stated, find the most efficient (time, resources) solution.

Problem 1	20	
Problem 2	15	
Problem 3	20 + 10	
Problem 4	25	
Problem 5	10	
Problem 6	5	
Problem 7	5	
Total	100 + 10	

Problem 1 (20 points): Miscellaneous

a) Does it make sense to give a CPU-bound thread higher priority for disk I/O than an I/O bound thread? Why? Explain your answer briefly.

b) A virtual memory system with paging can have external fragmentation, true or false? Why or why not?

c) Virtual address size must be the same as physical address size, true or false, and why? What determines the size of virtual and physical addresses?

d) Describe two situations in which spinlock (busy waiting) semaphores can be more appropriate than blocking ones.

e) Assume you have a system in which there can be a circular wait pattern in the resource-allocation graph. Is this a definite sign of a deadlock? Why or why not?

f) Can a system with all jobs having the same priority ever suffer from the priority inversion problem? Why or why not?

g) Consider an RTOS implementing a priority scheduler, where two tasks having the same priority is not allowed (such as in uCOS). Does the OS ever require a context switch in the default periodic SysTick handler?

Problem 2 (15 points): Thread Synchronization

Malek, Shailesh, and Shan go to an Indian restaurant at a busy time of the day. The waiter apologetically explains that the restaurant can provide only two pairs of spoons (for a total of four spoons) to be shared among the three people. Shailesh proposes that all four spoons be placed in an empty glass at the center of the table and that each diner should obey the following protocol (where `spoon` is a semaphore that is initialized to 4):

```
while (!had_enough_to_eat()) {
  OS_Wait(&spoon);
  OS_Wait(&spoon);
  eat();
  OS_Signal(&spoon);
  OS_Signal(&spoon);
}
```

a) Can this dining plan lead to a deadlock? Explain your answer.

b) Can there ever be a deadlock depending on the number of spoons? If so, show such a deadlock scenario. What is the minimum number of spoons for there not to be a deadlock?

c) Suppose now that instead of three there will be an arbitrary number of *D* diners. Furthermore, each diner d = 1..D may require a different number of S_d spoons to eat. For example, it is possible that one of the diners is an octopus, who for some reason refuses to begin eating before acquiring $S_{octopus}$ = 8 spoons. What is the smallest number of spoons needed to ensure that deadlock can not occur?

Problem 3 (20 + 10 points): Synchronization Primitives

In class, we discussed that atomic test-and-set and more generalized compare-and-swap operations can provide minimal canonical primitives to solve any synchronization problem, and that such operations have thus historically been directly implemented in hardware in many machines.

a) Our ARM does not provide built-in test-and-set or compare-and-swap operations. Shown below is C code for a software implementation on the ARM. This code is not atomic and thus has race conditions. Describe possible race conditions. Modify the code to make it atomic. You can assume that standard OS primitives are available:

```
void DisableInterrupts(void);
void EnableInterrupts(void);
long StartCritical(void);
void EndCritical(long);
```

```
int compare_and_swap(int* dst, int expected, int new) {

     int cur;

     cur = *dst;

     if(cur == expected) {

          *dst = new;

     }

     return cur;

}

int test_and_set(int* dst) {    // binary version

     return compare_and_swap(dst, 0, 1);

}
```

b) Implement binary spin-lock semaphores in C using only test-and-set or compare-and-swap primitives.

```
void OS_bInitSemaphore(int *semaPt) {  // initialize to be unlocked




}
void OS_bWait(int *semaPt) {






}
void OS_bSignal(int* semaPt) {






}
```

c) In addition to semaphores themselves, test-and-set and compare-and-swap primitives allow implementing so-called non-blocking or lock-free data structures that are made thread-safe without traditional coarse-grain and expensive blocking using semaphores. Shown below is the code for inserting a TCB into the OS ready queue as called from *OS_AddThread()* shown in the midterm. Modify (add/delete/replace) the code to make it thread-safe using only compare-and-swap or test-and-set.

```
void Q_Insert(struct TCB **queue, struct TCB *new) {


     new->next = *queue;


     *queue = new;


}
```

d) **(Required for graduate students, extra credit for undergraduates)** What are advantages and disadvantages of lock-free programming versus use of semaphores? When can and should it be used, when can it not be used?

e) **(Required for graduate students, extra credit for undergraduates)** Instead of native test-and-set or compare-and-swap, the ARM provides hardware support for synchronization through `LDREX`/`STREX` instructions. Implement compare-and-swap in assembly using `LDREX`/`STREX`.

```
compare_and_swap

        BX      LR
```

Problem 4 (25 points): Filesystems

Assume you have a disk with a filesystem using indexed allocation, where the first blocks on the disk store the directory information and the global index table.

a) Assuming a disk with 65536 blocks of 512 byte each, what is the size of the index table and how many blocks on disk does it occupy? What is the largest disk size supported if the index table must fit into one disk block? What if it needs to fit into half a block?

Size of index table & number of index blocks for 65536 block disk?	
Largest disk size with 1 block index table?	
Largest disk size with ½ block index table?	

b) What is the reliability of this filesystem? Can the files on disk be partially or completely recovered if i) the directory block(s), ii) the index table block(s) or iii) any other block(s) on disk are damaged? What can and cannot be recovered in each case?

Block(s) lost	Recoverable?
i) Directory	
ii) Index table	
iii) Other	

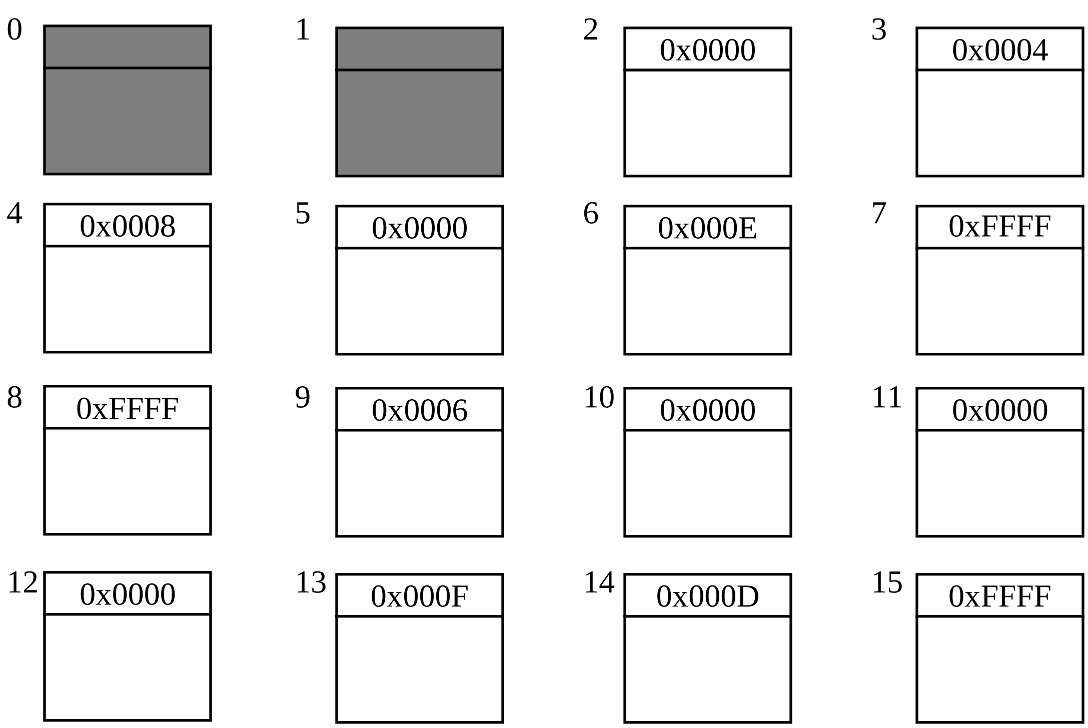
c) To improve reliability (while maintaining fast random access capabilities), indexed allocation can be combined with linked allocation. Shown below is a disk using such a combined filesystem, where the first two blocks on disk normally store the directory and index table, but have been damaged. The first two bytes of each regular block otherwise contain a pointer to the address of the next block in the file, 0xFFFF if the block is the last one, or 0x0000 if the block does not belong to a file (is empty). Assuming the directory and index blocks have both been damaged, what can be recovered from the remaining disk information? For the specific disk content below, recover as much information as possible to fill in the missing directory and index table information.

Directory

File name	Start block	Start index	Size (blocks)

Index Table

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



d) How would the recovered directory and index table be different when block 14 would also be unreadable in the example above?

Problem 5 (10 points): Relocation

Given the following simple user program to be loaded and executed by the OS:

```
// Display_Message prototype
#include "display.h"

int live = 0;
char* msg = "Hello";

void inc(void)
{
  live++;
}

int main(void)
{
  inc();
  Display_Message(0, 0, msg, live);
}
```

For each of the following two disassembled program data and code regions as generated by different compiler variants, is the code position-independent, i.e. will it be able to execute as is when loaded into an arbitrary location in memory? If not, indicate which lines of code are not and show how the code will need to be relocated by the OS at load time. You can assume that the OS will set R9 appropriately when launching the process to execute the program.

a)

```
0x20000000 0000      DCW          0x0000
0x20000004 …         DCB          "Hello",0
```

```
    Display_Message
0x00000092 F8DFC004  LDR          r12,[pc,#4]  ; @0x0000009A
0x00000096 4760      BX           r12

0x0000009A 00000000  DCD          0x00000000

    inc
0x00000110 4802      LDR          r0,[pc,#8]  ; @0x0000011C
0x00000112 6800      LDR          r0,[r0,#0x00]
0x00000114 1C40      ADDS         r0,r0,#1
0x00000116 4901      LDR          r1,[pc,#4]  ; @0x0000011C
0x00000118 6008      STR          r0,[r1,#0x00]
0x0000011A 4770      BX           lr

0x0000011C 20000000  DCD          0x20000000

    main
0x00000120 B510      PUSH         {r4,lr}
0x00000122 F7FFFFF5  BL           inc (0x00000110)
0x00000126 4804      LDR          r0,[pc,#16]  ; @0x00000138
0x00000128 4A04      LDR          r2,[pc,#16]  ; @0x0000013C
0x0000012A 2100      MOVS         r1,#0x00
0x0000012C 6803      LDR          r3,[r0,#0x00]
0x0000012E 4608      MOV          r0,r1
0x00000130 F7FFFFAF  BL           Display_Message (0x00000092)
0x00000134 2000      MOVS         r0,#0x00
0x00000136 BD10      POP          {r4,pc}

0x00000138 20000000  DCD          0x20000000
0x0000013C 20000004  DCD          0x20000004
```

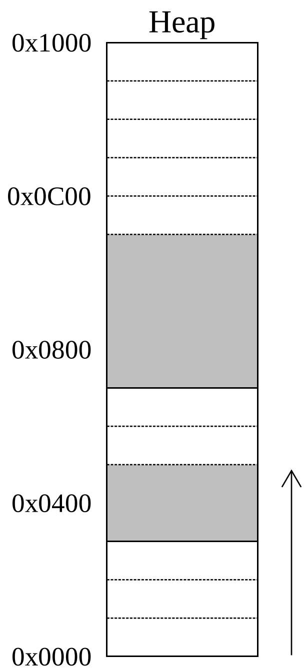
b)

```
0x20000004 0000      DCW           0x0000
0x20000008 …         DCB           “Hello”,0
```

```
    Display_Message
0x00000040 DF2A      SVC           0x2A
0x00000042 4770      BX            lr

    inc
0x000000BC 4803      LDR           r0,[pc,#12]  ; @0x000000CC
0x000000BE 4448      ADD           r0,r0,r9
0x000000C0 6800      LDR           r0,[r0,#0x00]
0x000000C2 1C40      ADDS          r0,r0,#1
0x000000C4 4901      LDR           r1,[pc,#4]  ; @0x000000CC
0x000000C6 4449      ADD           r1,r1,r9
0x000000C8 6008      STR           r0,[r1,#0x00]
0x000000CA 4770      BX            lr

0x000000CC 00000004  DCD           0x00000004

    main
0x000000D0 B510      PUSH          {r4,lr}
0x000000D2 F7FFFFF3  BL            inc (0x000000BC)
0x000000D6 4805      LDR           r0,[pc,#20]  ; @0x000000EC
0x000000D8 4448      ADD           r0,r0,r9
0x000000DA 4A05      LDR           r2,[pc,#20]  ; @0x000000F0
0x000000DC 444A      ADD           r2,r2,r9
0x000000DE 2100      MOVS          r1,#0x00
0x000000E0 6803      LDR           r3,[r0,#0x00]
0x000000E2 4608      MOV           r0,r1
0x000000E4 F7FFFFAC  BL            Display_Message (0x00000040)
0x000000E8 2000      MOVS          r0,#0x00
0x000000EA BD10      POP           {r4,pc}

0x000000EC 00000004  DCD           0x00000004
0x000000F0 00000008  DCD           0x00000008
```

Problem 6 (5 points): Heap

Consider a 4kB heap in the state as shown below after the given sequence of `malloc()` and `free()` calls has been executed. Assume that the heap manager does not require any overhead for extra meta-data, and that the heap is allocated from bottom to top, i.e. a block always ends up being placed at the bottom of its chosen free space region. What allocation strategy (first/best/worst fit) does the heap use?




```
p1 = Heap_Malloc(1792);
p2 = Heap_Malloc(1024);
Heap_Free(p1);
p3 = Heap_Malloc(768);
p4 = Heap_Malloc(512);
Heap_Free(p3);
```

Problem 7 (5 points): CAN

Consider a CAN network with 4 microcontrollers in which microcontroller M0 periodically sends messages with ID 14 to microcontroller M1 every 31ms, and microcontroller M2 periodically sends messages with ID 4 to microcontroller M3 every 11ms. What is the maximum jitter experienced by M1 and M3 in receiving their messages? Show the result as a function of the frame delay t_f (= time to complete a single message transfer).