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

Real-Time Systems / Real-Time Operating Systems

EE445M/EE380L.6, Spring 2016

Final Exam

Date: May 12, 2016

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

Problem 1 (10 points): Synchronization

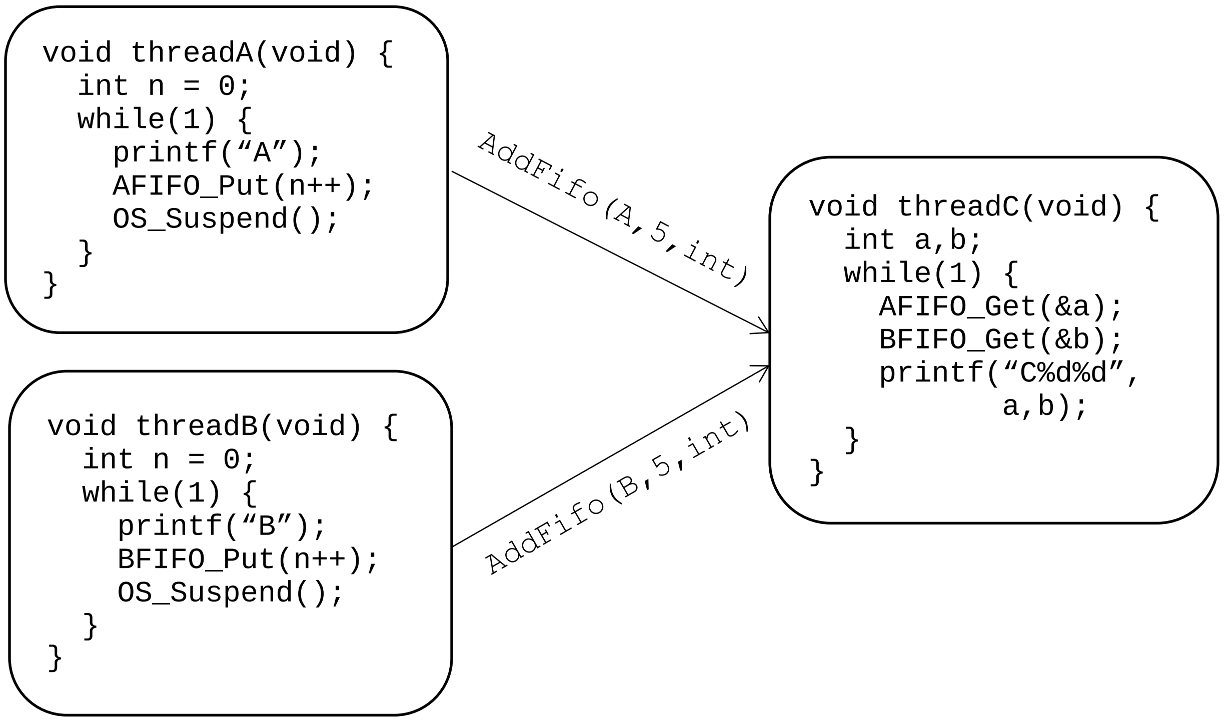
Given the macro-based basic FIFO implementation below, insert mutexes to make the FIFO fully thread-safe, i.e. resolve all critical sections against multiple concurrent foreground threads calling *Get*() and *Put*() concurrently. Also insert counting semaphores for `roomLeft` and `dataAvailable` to wait and block calling threads until there is room in the FIFO on *Put*() and data available on *Get*():

```
#define AddFifo(NAME,SIZE,TYPE)                                 \

unsigned long volatile PutI ## NAME;                            \
unsigned long volatile GetI ## NAME;                            \


TYPE static Fifo ## NAME [SIZE];                                \


void NAME ## Fifo_Init(void){                                   \


  PutI ## NAME= GetI ## NAME = 0;                               \


}                                                               \


void NAME ## Fifo_Put (TYPE data){                              \


  Fifo ## NAME[ PutI ## NAME &(SIZE-1)] = data;                 \


  PutI ## NAME ## ++;                                           \


}                                                               \


void NAME ## Fifo_Get (TYPE *datapt){                           \


  *datapt = Fifo ## NAME[ GetI ## NAME &(SIZE-1)];              \


  GetI ## NAME ## ++;                                           \


}
```

*Name:*______________________________

Problem 2 (10 points): Scheduling

Given the following KPN-like application code running on top of your OS and using blocking FIFOs as described and implemented in Problem 1. All FIFOs have a size of 5 integers. Assume that *threadA* is the first thread to be launched by the OS:



```
void threadA(void) {
  int n = 0;
  while(1) {
    printf("A");
    AFIFO_Put(n++);
    OS_Suspend();
  }
}
```

```
void threadB(void) {
  int n = 0;
  while(1) {
    printf("B");
    BFIFO_Put(n++);
    OS_Suspend();
  }
}
```

```
void threadC(void) {
  int a,b;
  while(1) {
    AFIFO_Get(&a);
    BFIFO_Get(&b);
    printf("C%d%d",
           a,b);
  }
}
```

a) What will be the output printed by the program if it is running under a non-preemptive, i.e. cooperative round-robin scheduler? Will any of the FIFOs ever be full?

b) What will be the output of the program under a strict priority scheduler with threads *A*, *B* and *C* having high, medium and low priority, respectively? Will any of the FIFOs ever be full?

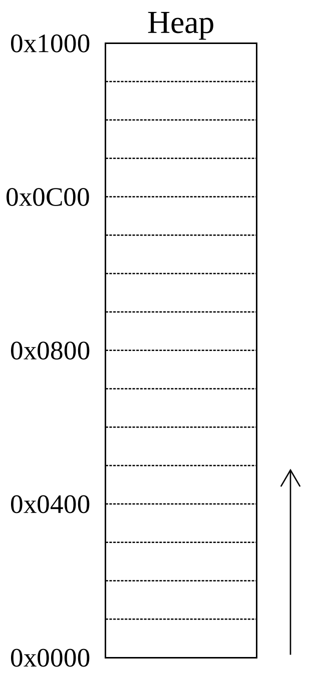
Problem 3 (15 points): Memory Management

Consider a 4kB heap and the following sequence of heap allocations and de-allocations triggered, for example, by different programs being loaded/launched and exiting:

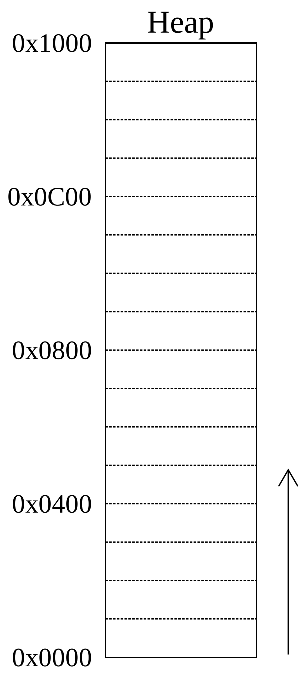
Program 1 launches:	`p1 = Heap_Malloc(512);`
Program 2 launches:	`p2 = Heap_Malloc(1024);`
Program 3 launches:	`p3 = Heap_Malloc(1024);`
Program 2 exits:	`Heap_Free(p2);`
Program 4 launches:	`p4 = Heap_Malloc(768);`
Program 1 exits:	`Heap_Free(p1);`

For each of the different heap allocation strategies below, show the state of the heap at the end of this sequence. Mark the memory regions allocated to each program that is loaded into memory at this point. Is the heap fragmented? What is the amount of free space and what is the largest block/program size that can be allocated/loaded then? 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.

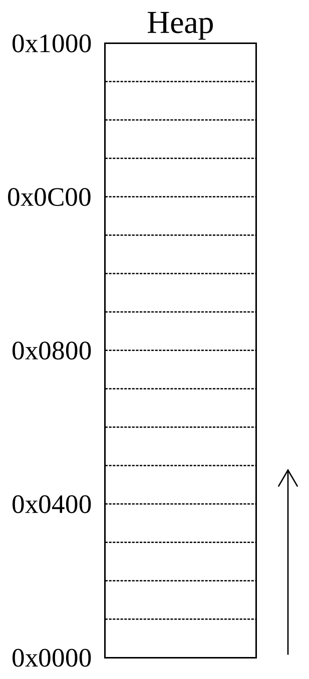
a) First fit



b) Best fit



c) Worst fit



Problem 4 (10 points): Supervisor Call

a) Given below is the code template for an *SVC_Handler* given in class. Complete the code fragment to realize a supervisor call to *OS_AddThread* whenever a `SVC #42` instruction is executed.

```
SVC_Handler

    LDR  R12,[SP,#24]       ; Get return address from stack

    LDRH R12,[R12,#-2]      ; Load SVC instruction (2 bytes)

    BIC  R12,#0xFF00        ; Extract ID in R12

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










    STR  R0,[SP]            ; Store return value

    BX   LR                 ; Return from exception
```

b) In earlier lectures we had discussed that invocation of OS kernel routines via supervisor calls (SVCs) will also be necessary when running an OS that uses the Process Stack Pointer (PSP) for all user code. Would your handler above also work in such a setup? If not, show the necessary modifications in the code above for the handler to work in a PSP setup.

Problem 5 (20 points): Networking

a) Assume a CAN 2.0A bus running at a baud rate of 1Mbit/s (=1,000,000 bit/s) and using 11-bit IDs, what is the maximum bandwidth achievable for CAN data transfers? You don't have to compute the actual result, just showing the expression is ok.

b) What is the bandwidth when (continuously) transmitting 32-bit data values of 0x00FF00FF? Hint: Don't forget about bit stuffing. Again, just the expression not final number is ok.

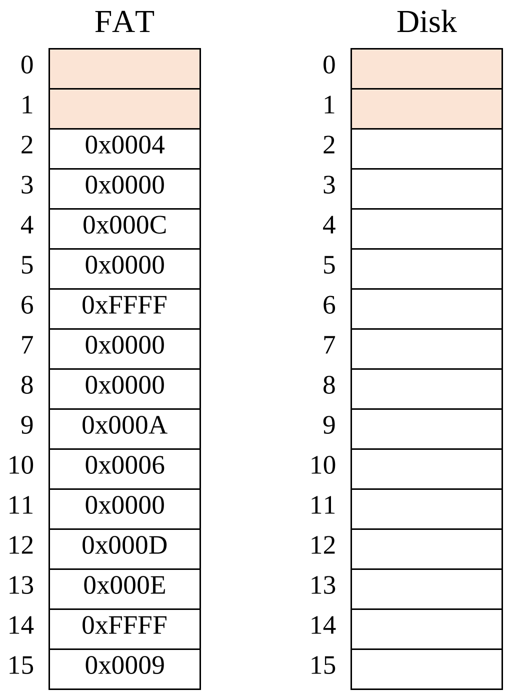
c) Assuming a CAN network with 3 nodes. At time 0, node 0 wants to send a message with ID 42 to node 1, node 1 wants to send a message with ID 14 to node 2 and node 2 wants to broadcast a message with ID 4 to nodes 0 and 1. At what time does each of the three messages reach their destination(s)? Just show the result as a function of the frame delay t_f (= time to complete a single message transfer).

d) Now consider an SPI bus running at a clock rate of 8MHz (=8,000,000 bit/s). What is the maximum achievable bandwidth for (continuously) reading single data blocks of 512 bytes assuming zero command-response delay (NCR=0) and an immediate data start (i.e. zero data packet delay)? How does that compare to the SD card bandwidth you measured in Lab 4?

Problem 6 (25 points): Filesystem

The FAT16 filesystem standard uses a file allocation table (FAT) scheme with 16-bit FAT entries. The first two blocks on disk and corresponding FAT entries are reserved to store the boot sector, the root directory and one or more redundant copies of the FAT itself. FAT entries of zero indicate an empty block, while any value larger than 0xFFF8 is used as end-of-file marker. Each directory entry uses 32 bytes. Given the following datastructures for a FAT16-compliant filesystem (directory and FAT):

```
// directory w/ 32 byte entries
struct dir_entry {
  char   name[8];  // file name
  char   ext[3];   // extension
  uint8  attr[11]; // attributes
  uint16 date;     // modification
  uint16  time;    //  date & time
  uint16 start;    // start block
  uint32 size;     // size in byte
} dir[DIR_SIZE];

uint16 fat[FAT_SIZE]; // FAT
```



	FAT	Disk
0	▨	▨
1	▨	▨
2	0x0004	
3	0x0000	
4	0x000C	
5	0x0000	
6	0xFFFF	
7	0x0000	
8	0x0000	
9	0x000A	
10	0x0006	
11	0x0000	
12	0x000D	
13	0x000E	
14	0xFFFF	
15	0x0009	

a) For the specific instance of a FAT with 16 entries as shown in the figure above, how many files are there on disk and what is the size of each of these files assuming 512 byte disk blocks. Mark the blocks belonging to each file and each file's starting block in the disk layout on the right. How much free space is there left on the disk?

b) What is the maximum file size supported by FAT16, and what part of the FAT16 datastructure determines the maximum file size?

c) Assuming 512 byte disk blocks, what is the largest disk size supported by FAT16, and how many entries does the FAT maximally have? What determines the maximum disk size? How large does the FAT need to be for a disk with 128kB?

Largest disk size & why?	
Max. `FAT_SIZE`	
`FAT_SIZE` for 128kB disk	

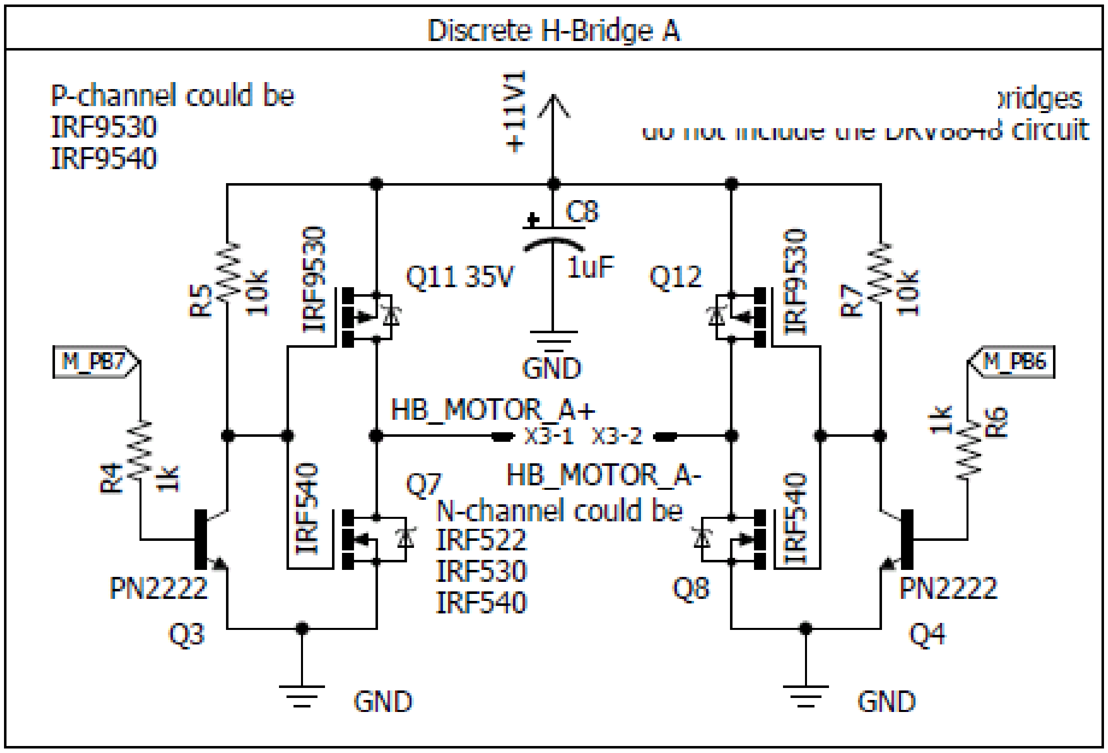
d) To support larger disk sizes, the FAT16 standard in reality uses clustering, where the disk is partitioned into fixed-sized clusters and each FAT entry refers to a cluster of contiguous blocks/sectors on disk. What is the cluster size needed for a disk with 2^{27} bytes? What is the disadvantage of clustering?

e) Assuming a FAT16-formatted disk with 4kB cluster size and sectors of 512 bytes, show the code of a filesystem routine that reads the *n*-th byte from the file *handle* pointing to its directory entry. You can assume that the `dir` and `fat` datastructures have already been read and populated from disk, where `start` directory fields points to a file's first cluster.

```
// Return 0 if successful, 1 on failure
int FAT16_ReadByte(struct dir *handle, long n, char* pt)
{
  char buf[512];











  if(eDisk_ReadBlock(buf,               ) return 1;





  return 0;
}
```

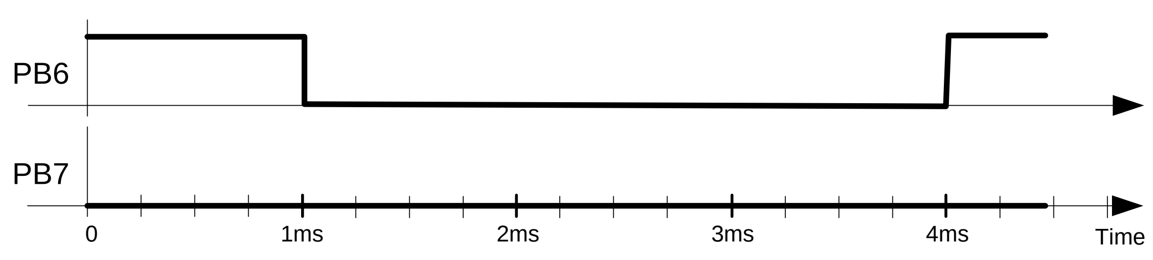
Problem 7 (10 points): Motor Driver

Shown below is the schematic of the H-bridge on our motor board. Assume that the motor runs in "forward" direction if current is flowing from A+ to A-:

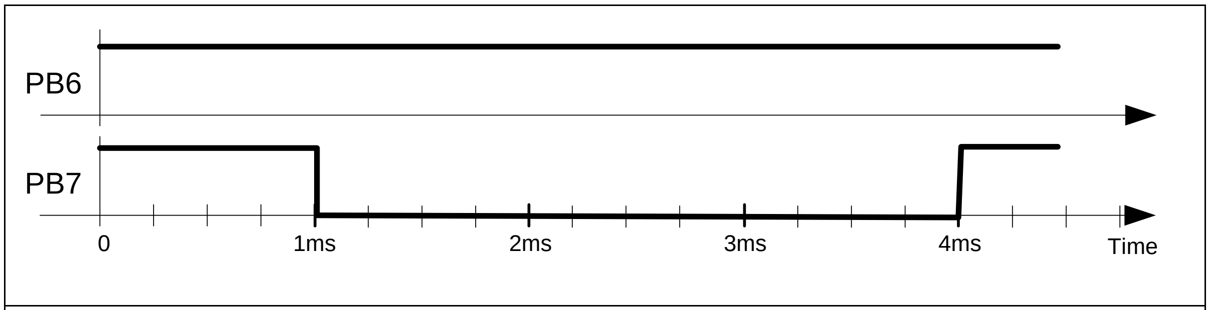


For each of the following waveforms (with 4ms period), describe precisely what will happen if they are applied to the PB6 and PB7 inputs? Is the motor running? If so, in what direction and at what speed? Etc.

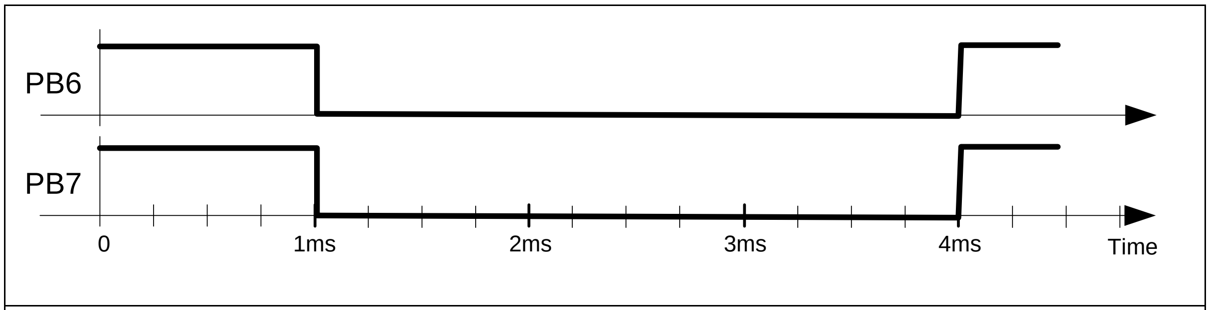
a)



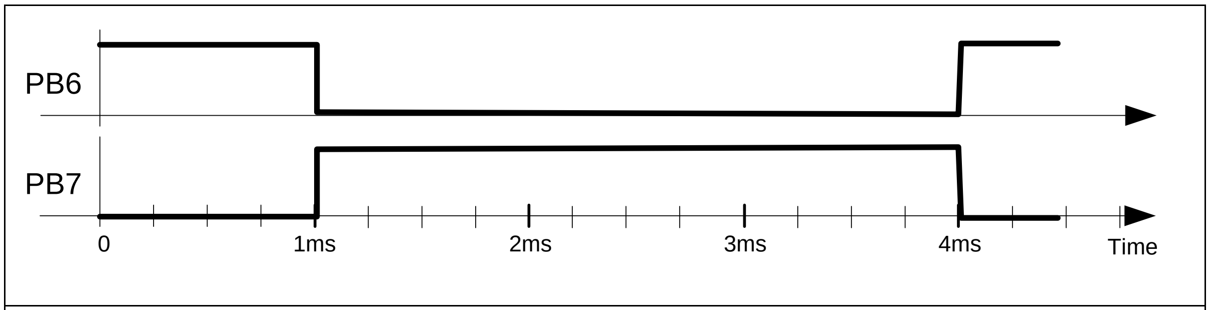
b)


PB6
PB7
0
1ms
2ms
3ms
4ms
Time

c)


PB6
PB7
0
1ms
2ms
3ms
4ms
Time

d)


PB6
PB7
0
1ms
2ms
3ms
4ms
Time