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

Real-Time Systems / Real-Time Operating Systems

EE445M/EE380L.6, Spring 2016

Midterm

Date: March 24, 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	20	
Problem 2	20	
Problem 3	25	
Problem 4	20	
Problem 5	15	
Total	100	

Problem 1 (20 points): Critical Sections and Deadlock

a) Given the following two routines that can be called from any user thread. Does this code have any critical sections or reentrancy issues? Justify, and list all such cases in the code.

```
void Dbg_Out(int v) {
  int save;
  save = GPIO_PORTF_DATA_R;
  GPIO_PORTF_DATA_R = v;
  Dbg_Record(v);
  GPIO_PORTF_DATA_R = save;
}
```

```
int Dbg_Dump[MAX_DUMP];
unsigned Dbg_Count = 0;

void Dbg_Record(int v) {
  if(Dbg_Cnt==MAX_DUMP) return;
  Dbg_Dump[Dbg_Cnt] = v;
  Dbg_Cnt = Dbg_Cnt + 1;
}
```

b) Now consider the following code. Does this code have any critical sections, reentrancy or deadlock issues? Justify your answer, and list all such cases in the code.

```
void Dbg_Out(int v) {
  int save;
  DisableInterrupts();
  save = GPIO_PORTF_DATA_R;
  GPIO_PORTF_DATA_R = v;
  Dbg_Record(v);
  GPIO_PORTF_DATA_R = save;
  EnableInterrupts();
}
```

```
int Dbg_Dump[MAX_DUMP];
unsigned Dbg_Count = 0;

void Dbg_Record(int v) {
  if(Dbg_Cnt==MAX_DUMP) return;
  DisableInterrupts();
  Dbg_Dump[Dbg_Cnt] = v;
  Dbg_Cnt = Dbg_Cnt + 1;
  EnableInterrupts();
}
```

c) Now consider the following code. Does this code have any critical sections, reentrancy or deadlock issues? Justify your answer, and list all such cases in the code.

```
void Dbg_Out(int v) {
  int save;
  DisableInterrupts();
  save = GPIO_PORTF_DATA_R;
  GPIO_PORTF_DATA_R = v;
  Dbg_Record(v);
  GPIO_PORTF_DATA_R = save;
  EnableInterrupts();
}
```

```
int Dbg_Dump[MAX_DUMP];
unsigned Dbg_Count = 0;
sema_t Dbg_Mutex = 1;

void Dbg_Record(int v) {
  if(Dbg_Cnt==MAX_DUMP) return;
  OS_bWait(&Dbg_Mutex);
  Dbg_Dump[Dbg_Cnt] = v;
  Dbg_Cnt = Dbg_Cnt + 1;
  OS_bSignal(&Dbg_Mutex);
}
```

d) Provide another solution that avoids all critical sections and deadlocks. Prove that your solution is deadlock-free. List all deadlock conditions and show that at least one of them is violated.

Problem 2 (20 points): Priority Scheduling

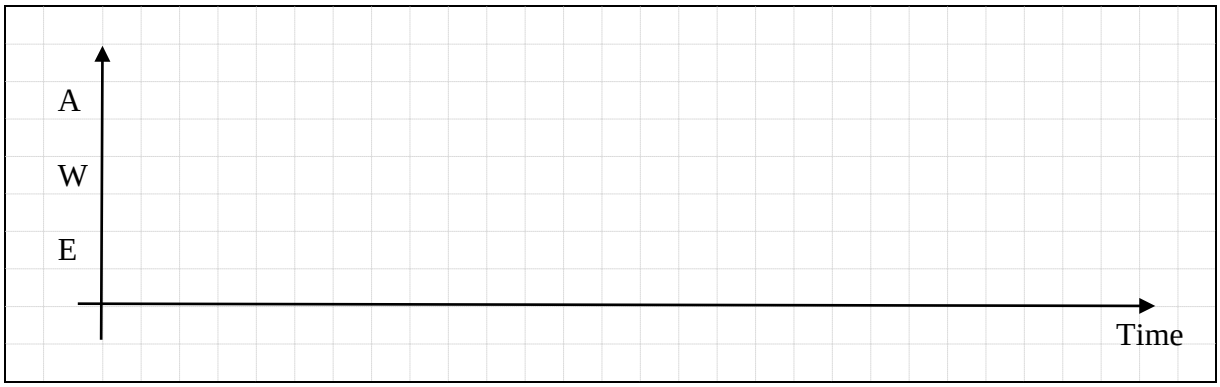
Consider a real-time system running three periodic tasks with the following periods (= deadlines) and execution times. You can assume zero context switch and interrupt overhead.

Task	Execution Time	Period
Airbag (A)	10ms	30ms
Warning (W)	20ms	40ms
Engine (E)	10ms	60ms

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

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

c) Draw the schedule of task executions over time. 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)?



Problem 3 (25 points): OS Sleep Support

a) Given the basic round-robin OS kernel code below, show the necessary modifications (insertions and/or deletions) to add *OS_Sleep()* functionality. Keep your implementation simple, i.e. you are not required to optimize for performance.

```
struct tcb {
  long *sp;
  struct tcb *next;


}


struct tcb* RunPt;
```

```
#define ContextSwitch() (NVIC_INT_CTRL_R=0x10000000) // trigger PendSV
void DisableInterrupts(void);
void EnableInterrupts(void);
long StartCritical(void);
void EndCritical(long sr);

void OS_Sleep(unsigned long delay) {









}

void SysTick_Handler(void) {









  ContextSwitch();



}
```

```
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
```

b) How does your OS implementation behave when all threads are sleeping, i.e. what happens when all but one thread currently sleep and the last active/running thread calls *OS_Sleep()*?

Problem 4 (20 points): Thread Exit and Kill

Assume a basic round-robin OS kernel (as shown in Problem 3) with the following *OS_AddThread()* implementation:

```
long* SetInitialStack(long *sp, void (*entry)(void)) {
  *(sp)   = (long)0x01000000L;  /* xPSR */
  *(--sp) = (long)entry;        /* PC  */
  *(--sp) = (long)0x14141414L;  /* R14 */
  *(--sp) = (long)0x12121212L;  /* R12 */
  *(--sp) = (long)0x03030303L;  /* R3  */
  *(--sp) = (long)0x02020202L;  /* R2  */
  *(--sp) = (long)0x01010101L;  /* R1  */
  *(--sp) = (long)0x000000000;  /* R0  */
  *(--sp) = (long)0x11111111L;  /* R11 */
  *(--sp) = (long)0x10101010L;  /* R10 */
  *(--sp) = (long)0x09090909L;  /* R9  */
  *(--sp) = (long)0x08080808L;  /* R8  */
  *(--sp) = (long)0x07070707L;  /* R7  */
  *(--sp) = (long)0x06060606L;  /* R6  */
  *(--sp) = (long)0x05050505L;  /* R5  */
  *(--sp) = (long)0x04040404L;  /* R4  */
  return sp;
}

int OS_AddThread(void(*task)(void)) { long sr;
  struct tcb* newPt;

  sr = StartCritical();


  newPt = AllocTcb();  // get TCB & stack, set SP to top of stack
  if(!newPt) { EndCritical(sr); return 0; }


  newPt->sp = SetInitialStack(newPt->sp, task);

  newPt->next = RunPt->next;
  RunPt->next = newPt;


  EndCritical(sr);
  return 1;
}
```

Given the following user code:

```
void main(void) {
  int i;
  OS_Init();
  OS_AddThread(Thread1);
  ...
  OS_Launch();
}
```

```
Thread1
    ...

    BX  LR      <--- PC
```

a) What happens when *Thread1* exits normally (without calling *OS_Kill()*)? In other words, if the current PC points to the last `BX LR` return instruction in *Thread1*, where will the branch go to and what line of code will be executed next? Hint: think about the value that will be in the `LR` register during execution of *Thread1* and thus when `BX LR` is executed.

b) Modify the OS kernel code shown above such that *OS_Kill()* will be automatically executed whenever a thread added to the OS exits normally. You are only allowed to make modifications to the kernel but not the user code, i.e. the *OS_xxx* interface must remain as is and must work for an arbitrary number of threads. If you need additional code, show it in the box below.

Problem 5 (15 points): Synchronization and Deadlock

a) Given the two threads below, can any deadlock occur? If yes, why (show an execution sequence leading to deadlock)? If not, why not (prove that there is no deadlock)?

```
void Thread1(void) {
  OS_bWait(&file_mutex);
  ...
  OS_bWait(&memory_mutex);
  ...
  OS_bSignal(&memory_mutex);
  ...
  OS_bSignal(&file_mutex);
}
```

```
void Thread2(void) {
  OS_bWait(&memory_mutex);
  ...
  If(debug) {
    OS_bWait(&file_mutex);
    ...
    OS_bSignal(&file_mutex);
  }
  ...
  OS_bSignal(&memory_mutex);
}
```

b) Given the two threads below, can any deadlock occur? If yes, why (show an execution sequence leading to deadlock)? If not, why not (prove that there is no deadlock)?

```
void Thread1(void) {
  OS_bWait(&file_mutex);
  ...
  OS_Wait(&available);
  ...
  OS_bSignal(&file_mutex);
}
```

```
void Thread2(void) {
  OS_bWait(&file_mutex);
  ...
  If(new_data) {
    OS_Signal(&available);
  }
  ...
  OS_bSignal(&file_mutex);
}
```

c) Assume that the code is running on top of an OS that uses priority scheduling and a priority ceiling protocol for all semaphores. For each of the examples above, can any deadlock still occur? If yes, why (show an execution sequence leading to deadlock)? If not, why not (prove that there is no deadlock)?

Example a)
Example b)