Jonathan W. Valvano First Name: _______________ Last Name:___________________

May 13, 2017, 9am-12n Closed book part

(25) Question 1. For each definition, select the term that best describes it. Not all words will be used. No words will be used more than once. Place the corresponding numbers into the boxes.

1. Aging
2. Aliasing
3. Anti-Reset-Windup
4. Atomic
5. Bank-Switched Memory
6. Big Endian
7. Board Support Package
8. Bounded Waiting
9. Brushed DC motor
10. Burst DMA
11. Central Limit Theorem
12. Content Addressable Memory
13. Cooperative Nonpreemptive scheduler
14. Crisp Input
15. Critical Section
16. Cycle Steal DMA
17. Deadlock
18. Dual Address DMA
19. External Fragmentation
20. Firm real time
21. Flash Memory
22. Full Duplex Channel
23. Half Duplex Channel
24. Hard real time
25. Hook
26. Internal Fragmentation
27. Little Endian
28. Little's Formula
29. Minimally Intrusive
30. Mutual Exclusion
31. Nyquist Theorem
32. Path Expression
33. Preemptive scheduler
34. Priority Inversion
35. Random Access Memory
36. Reentrant function
37. Priority Scheduler
38. Pulse width modulation
39. Semaphore initialization
40. Servo
41. Simplex Channel
42. Single Address DMA
43. Slew Rate
44. Soft real time
45. Stabilization
46. Stuff Bits
47. Utilization factor
48. Velocity Factor

	A DC motor with built-in controller. The microcontroller specifies the desired position and the motor adds/subtracts power to move the shaft to that position.
	A storage device that takes as input the data, and creates as output the address at which these data are located.
	Throughput (actual number of packets per second) divided by the capacity (maximum capacity the system can handle in packets per second).
	A system that expects all critical tasks to complete on time. Once a deadline as passed, there is no value to completing the task. However, the consequence of missed deadlines is real but the overall system operates with reduced quality.
	A system that can guarantee that a process will complete a critical task within a certain specified range. There is an upper bound on the latency between when a task is supposed to be performed and when it is actually performed.
	A system that implements best effort to execute critical tasks on time, typically using a priority scheduler. Once a deadline as passed, the value of completing the task diminishes over time.

	A type of memory such that when you perform a write cycle to it, you can cause bits to go from 1 to 0, but not 0 to 1.
	A scheduler that cannot suspend execution of a thread without the thread's permission. The threads suspend themselves at times convenient for the thread.
	A set of software routines that abstract the I/O hardware such that the same high-level code can run on multiple computers.
	A software function that can be started by one thread, interrupted and executed by a second thread.
	A software technique to guarantee subfunctions within a module are executed in a proper sequence. For example, it forces the user to initialize I/O device before attempting to perform I/O.
	A technique used in priority schedulers that temporarily increases the priority of low priority treads so they are run occasionally.
	An I/O synchronization scheme that transfers an entire block of data all at once directly from an input device into memory, or directly from memory to an output device.
	Method used in CAN to synchronize in conditions when long strings of zeros are sent, or when long strings of ones are sent.
	An indirect function-call added to a software system that allows the user to attach their programs to run at strategic times. These attachments are created dynamically at run time and do not require recompiling the entire system.
	The average number of packets in the system is equal to the average arrival rate in packets per second multiplied by the average response time of a packet.
	Direct memory access that requires two bus cycles to transfer data from source to destination. The first cycle brings data from the source into the DMA controller, and the second cycle sends the data to the destination.
	Used to determine the minimum sampling rate required to faithfully represent a signal in digital form.
	A communication channel that allows bits (information, error checking, synchronization or overhead) to transfer only in one direction.
	Locations within a software module, which if an interrupt were to occur at one of these locations, then an error might occur (e.g., data lost, corrupted data, program crash, etc.)
	Mechanism for storing multiple byte numbers such that the most significant byte exists first in the smallest memory address.
	Storage that is allocated for the convenience of the operating system but contains no information. This space is wasted.
	The ratio of the speed at which information travels on a copper wire like CAN relative to the speed of light.
	Thread synchronization scheme where at most one thread at a time is allowed to enter a specified region of code at a time.
	A section of software code that once execution has started, it cannot be divided or interrupted.

(5) Question 2. What are the three necessary conditions to cause **deadlock**?

(5) Question 3. You are designing a **real-time scheduler** for this system. There are three periodic tasks that have minimal interaction with each other.

Task 1: Executes every 1000 μs, execution time varies from 5 to 100μs.

Task 2: Executes every 400 μs, execution time varies from 10 to 100μs.

Task 3: Executes every 100 μs, execution time varies from 1 to 10μs.

Without actually writing the scheduler, you can determine whether or not a real-time solution with no jitter is likely. Is it possible to schedule these tasks? If no, prove it. If yes, justify your answer.

(5) Question 4. Explain how an operating system can implement **position independent data** on the Cortex M. This means the OS can reposition the global variables of a user program without recompiling.

(5) Question 5. Give three different reasons for implementing **paging** in a multi-process OS.

(15) Question 6. Write C code for a FIFO queue that can be used to pass 8-bit data between foreground threads. None of the FIFO functions will be called from an interrupt service routine. You must write all of the FIFO code. There will be multiple producers and multiple consumers running in the foreground using a preemptive scheduler accessing this one FIFO. You can define semaphores by adding globals:

```
long semaphore=0;
```

You may call these two blocking semaphore functions without showing their implementations.

```
void Wait(long *semaPt);
void Signal(long *semaPt);
```

You must use the following private globals. Other than semaphores, you may not add any additional global variables. You may use local variables.

```
#define SIZE 10   // SIZE can be any value greater than or equal to 1
uint8_t static volatile PutI;  // index to put next
uint8_t static volatile GetI;  // index to get next
uint8_t static Fifo[SIZE];
```

Part a) List the semaphores needed. Use good semaphore names

Part b) Show the initialization code that configures the FIFO and initializes the semaphores

```
void Fifo_Init(void){
```

Part c) Show the function that stores into the FIFO. A producer thread should block on full.

```
void Fifo_Put(uint8_t data){
```

Part d) Show the function that retrieves from the FIFO. A consumer thread should block on empty.

```
uint8_t Fifo_Get(void){
```

Jonathan W. Valvano First Name: _______________ Last Name:_____________________
Open book part

Open book, open notes, calculator (no laptops, phones, devices with screens larger than a TI-89 calculator, devices with wireless communication). Please don't turn in any extra sheets.

(10) Question 7. A **barrier** for a group of threads is a place in the code where each thread must stop and cannot proceed until all other threads reach their barriers. You can define and initialize semaphores by adding globals like this.

```
long semaphore=0;
```

You may call the following two blocking semaphore functions without showing their implementations.

```
void Wait(long *semaPt);
void Signal(long *semaPt);
```

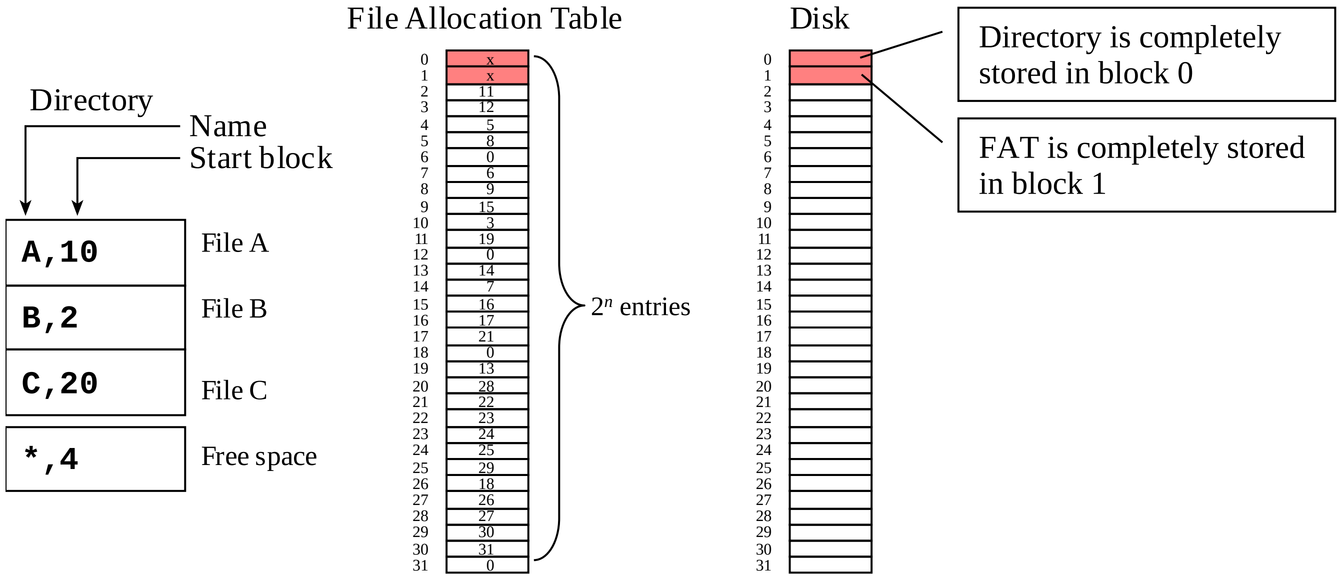
Other than semaphores, you may not add any additional global variables.

Part a) Define the semaphores needed, including their initial values.

Part b) Place a barrier between the **`start`** and **`end`** functions in each thread to implement this three-thread barrier. Basically, the threads will not execute their corresponding **`end`** functions until all threads have executed their **`start`** functions. You may assume this sequence executes just once.

`void thread1(void){`
` start1();`
 	
` end1();`
` OS_Kill();`
`}`
	`void thread2(void){`
	`  start2();`
	
	`  end2();`
	`  OS_Kill();`
	`}`
		`void thread3(void){`
		`  start3();`
		
		`  end3();`
		`  OS_Kill();`
		`}`

(10) Question 8. Consider a file system that uses a FAT. Let n be an integer greater than or equal to 5. There are 2^n entries in the FAT, and each entry is 4 bytes (32 bits). Each disk block contains $4*2^n$ bytes, meaning the entire FAT will always fit in one block. Assume the FAT entry size is always 32 bits. It is shown as n=5 in the figure, but n could be larger. The directory is in block zero, the FAT is in block 1. Each directory entry contains a file name, file size, and an index to the first FAT entry for that file. The last entry in the directory contains an index to the first FAT entry for the free space.



Part a) If each directory entry requires 16 bytes of information (file name, file size, and starting FAT index), then what is the maximum number of files that can be stored on this disk? Solve in general for any n (partial credit: solve specifically for n=5).

Part b) If there are 2^n entries in the FAT, and each block contains $4*2^n$ bytes, what is the maximum number of data bytes that can be stored on this disk? Solve in general for any n.

Part c) (required for grad students, extra credit for undergrad) Clearly, this works for n=5, but at n=64, the entire disk will not be accessible. Assuming the FAT entries remain 32 bits, and there is one block for the FAT, what is the largest value of n possible, such that the entire disk is accessible?

(20) Question 9. In this question you will implement **blocking semaphores with bounded waiting**. The OS has the following TCB structure, and it cannot be changed.

```
struct TCB {
  long *stackPointer;   // pointer to top of stack
  struct TCB *Next;     // linked list
  long *BlockPt;        // nonzero if blocked on this semaphore
  uint64_t BlockTime;   // time when this thread was blocked
};
typedef struct TCB TCBType;
typedef TCBType * TCBPtr;
TCBPtr RunPt;           // Pointer to tcb of thread currently running
```

The OS uses a signed 32-bit integer for semaphores (**`long`**), which also cannot be changed. There is an **`OS_Time`** function that returns the current time as a 64-bit unsigned integer with units of 12.5ns. You may assume this time never rolls over (i.e., the system runs for less than 664 years). The prototype is

```
uint64_t OS_Time(void);
```

This is the ISR thread switch, Program 3.11 in the book, and it cannot be modified

```
SysTick_Handler                ; 1) Saves R0-R3,R12,LR,PC,PSR
    CPSID   I                  ; 2) Prevent interrupt during switch
    PUSH    {R4-R11}           ; 3) Save remaining regs r4-11
    LDR     R0, =RunPt         ; 4) R0=pointer to RunPt, old thread
    LDR     R1, [R0]           ;    R1 = RunPt
    STR     SP, [R1]           ; 5) Save SP into TCB
    PUSH    {R0,LR}
    BL      Scheduler
    POP     {R0,LR}
    LDR     R1, [R0]           ; 6) R1 = RunPt, new thread
    LDR     SP, [R1]           ; 7) new thread SP; SP = RunPt->sp;
    POP     {R4-R11}           ; 8) restore regs r4-11
    CPSIE   I                  ; 9) tasks run with interrupts enabled
    BX      LR                 ; 10) restore R0-R3,R12,LR,PC,PSR
```

This is the scheduler, and it cannot be modified

```
void Scheduler(void){
  RunPt = RunPt->Next;    // skip at least one
  while(RunPt->BlockPt){  // do not run if blocked
    RunPt = RunPt->Next;  // find first one not blocked
  }
}
```

The function **`OS_Suspend`** will trigger a SysTick and suspend this thread

```
void OS_Suspend(void){
  NVIC_ST_CURRENT_R = 0;      // next thread gets a full time slice
  NVIC_INT_CTRL = 0x04000000; // trigger SysTick
}
```

You may call these four functions, as defined in the book and starter code.

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

Part a) Implement **`OS_Wait`**, which has the following prototype. Your solution must implement bounded waiting.

```
void OS_Wait(long *semaPt){
```

Part b) Implement **`OS_Signal`**, which has the following prototype. Your solution must implement bounded waiting

```
void OS_Signal(long *semaPt){
```