Section 5: Priority Donation, Deadlock, and I/O Devices

February 28, 2020

Contents

1 Vocabulary

- **Multi-Level Feedback Queue Scheduling** - MLFQS uses multiple queues with priorities, dropping CPU-bound jobs that consume their entire quanta into lower-priority queues.

- **Priority Inversion** - If a higher priority thread is blocking on a resource (a lock, as far as you're concerned but it could be the Disk or other I/O device in practice) that a lower priority thread holds exclusive access to, the priorities are said to be inverted. The higher priority thread cannot continue until the lower priority thread releases the resource. This can be amended by implementing priority donation.

- **Priority Donation** - If a thread attempts to acquire a resource (lock) that is currently being held, it donates its effective priority to the holder of that resource. This must be done recursively until a thread holding no locks is found, even if the current thread has a lower priority than the current resource holder. (Think about what would happen if you didn't do this and a third thread with higher priority than either of the two current ones donates to the original donor.) Each thread's effective priority becomes the max of all donated priorities and its original priority.

- **Deadlock** - A case of starvation due to a cycle of waiting. Computer programs sharing the same resource effectively prevent each other from accessing the resource, causing both programs to cease to make progress.

- **Banker's Algorithm** - A resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, before deciding whether allocation should be allowed to continue.

- **I/O** In the context of operating systems, input/output (I/O) consists of the processes by which the operating system receives and transmits data to connected devices.

- **Controller** The operating system performs the actual I/O operations by communicating with a device controller, which contains addressable memory and registers for communicating the the CPU, and an interface for communicating with the underlying hardware. Communication may be done via programmed I/O, transferring data through registers, or Direct Memory Access, which allows the controller to write directly to memory.

- **Interrupt** One method of notifying the operating system of a pending I/O operation is to send a interrupt, causing an interrupt handler for that event to be run. This requires a lot of overhead, but is suitable for handling sporadic, infrequent events.

- **Polling** Another method of notifying the operating system of a pending I/O operating is simply to have the operating system check regularly if there are any input events. This requires less overhead, and is suitable for regular events, such as mouse input.

- **Memory-Mapped I/O** Memory-mapped I/O (not to be confused with memory-mapped file I/O) uses the same address bus to address both memory and I/O devices – the memory and registers of the I/O devices are mapped to (associated with) address values. So when an address is accessed by the CPU, it may refer to a portion of physical RAM, but it can also refer to memory of the I/O device. Thus, the CPU instructions used to access the memory can also be used for accessing devices.

2   Priority Inversion & Donation

2.1   All Threads Must Die

You have three threads with the associated priorities shown below. They each run the functions with their respective names. Assume upon execution all threads are initially unblocked and begin at the top of their code blocks. The operating system runs with a preemptive priority scheduler. You may assume that set_priority commands are atomic.

Tyrion : 4
Ned: 5
Gandalf: 11

Note: The following uses references to Pintos locks and data structures.

```
struct list braceYourself;    // pintos list. Assume it's already initialized and populated.
struct lock midTerm;          // pintos lock. Already initialized.
struct lock isComing;

void tyrion(){
    thread_set_priority(12);
    lock_acquire(&midTerm);
    lock_release(&midTerm);
    thread_exit();

}

void ned(){
    lock_acquire(&midTerm);
    lock_acquire(&isComing);
    list_remove(list_head(braceYourself));
    lock_release(&midTerm);
    lock_release(&isComing);
    thread_exit();
}

void gandalf(){
    lock_acquire(&isComing);
    thread_set_priority(3);
    while (thread_get_priority() < 11) {
        printf("YOU .. SHALL NOT .. PAAASS!!!!!!);
        timer_sleep(20);
    }
    lock_release(&isComing);
    thread_exit();
}
```

What is the output of this program when there is no priority donation? Trace the program execution and number the lines in the order in which they are executed.
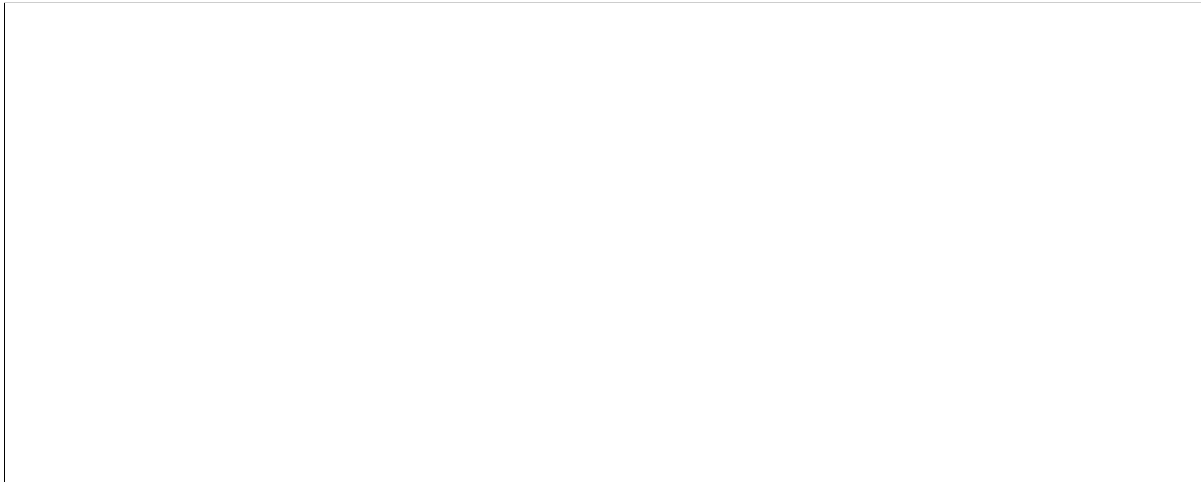
What is the output and order of line execution if priority donation was implemented? Draw a diagram of the three threads and two locks that shows how you would use data structures and struct members (variables and pointers, etc) to implement priority donation for this example.
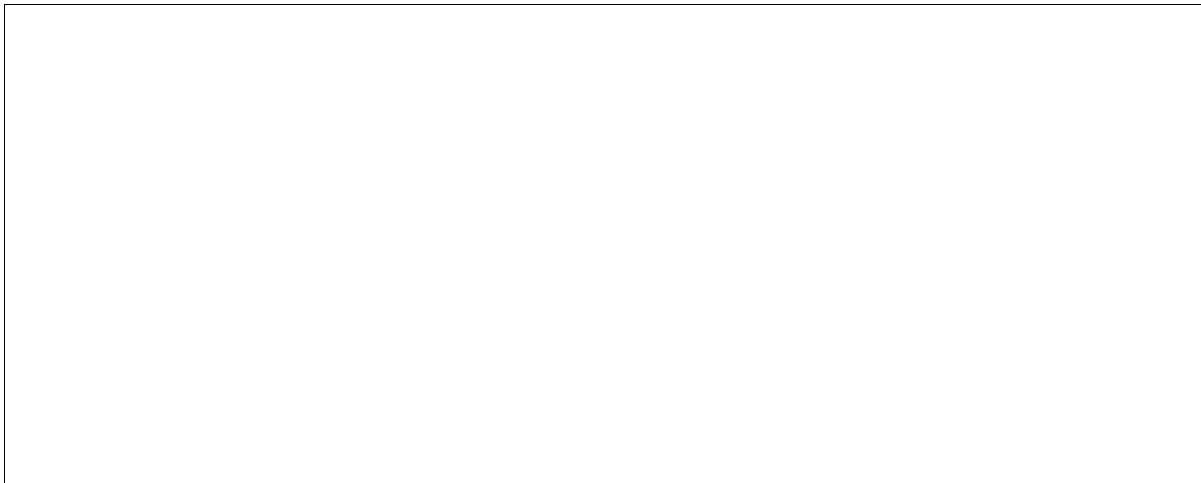
3   Deadlock

3.1   Introduction

What are the four requirements for Deadlock?

What is starvation and what is deadlock? How are they different?

3.2   Banker's Algorithm

Suppose we have the following resources: A, B, C and threads T1, T2, T3 and T4. The total number of each resource as well as the current/max allocations for each thread are as follows:

Total		
A	B	C
7	8	9

T/R	Current			Max		
	A	B	C	A	B	C
T1	0	2	2	4	3	3
T2	2	2	1	3	6	9
T3	3	0	4	3	1	5
T4	1	3	1	3	3	4

Is the system in a safe state? If so, show a non-blocking sequence of thread executions.

Repeat the previous question if the total number of C instances is 8 instead of 9.
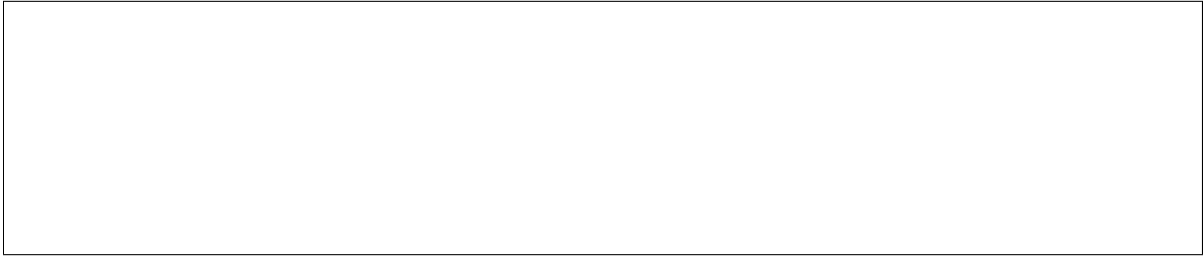
4   I/O

4.1   I/O Devices

What is a block device? What is a character device? Why might one interface be more appropriate than the other?

Why might you choose to use DMA instead of memory mapped I/O? Give a specific example where one is more appropriate than the other.

Explain what is meant by "top half" and "bottom half" in the context of device drivers.