Homework 5: Low Level Software (Practice Problems)
Suggested Solutions

Problem P1 [15 points]:
Is the following statement true or false: If \( T_i = k_i \cdot T_{i-1} \) where \( k_i = 2, 3, ..., \) then a set of \( n \) tasks is RM-schedulable if \( C_1/T_1 + C_2/T_2 + ... + C_n/T_n \leq 1 \). If you believe it to be true, prove it. If you think it is false, provide a counter example. (Hint: N&S conditions are given by the response time test.)

The statement is TRUE. The sufficient condition for RM schedulability is given by the upper bound of \( N \cdot (2^{1/N} - 1) \). However, the necessary condition is given by the response time test for all tasks. Taking the response time test (bound on the number of interrupts of a tasks by lower priority tasks), you can prove by induction that the task utilization bound can be increased to 1. (You already know that by an example in the class.)

Problem P2 [10,5 points]
Is the following task set schedulable when it is required that the computation represented by task A be given the highest priority since they have the highest criticality. If not, how would you transform this task set to make it schedulable?

<table>
<thead>
<tr>
<th>Task</th>
<th>T</th>
<th>C</th>
<th>Criticality</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>60</td>
<td>10</td>
<td>High</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>3</td>
<td>Low</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>2</td>
<td>Low</td>
</tr>
</tbody>
</table>

The task set is not schedulable. It must have a period transformation applied to it. A is transformed to a task that has a period of 6 and computation time of 1. RM scheduling now gives A the highest priority. The utilization of the new tasks set is: 0.1666+0.333+0.25 which meets the utilization bound.

Problem P3 [25 points]
What is a transaction in Mars operating system? Describe the mechanisms and assumptions used by the Mars operating system to guarantee time of the real-time transactions. Explain how the time-stamping in Mars operating system controls the measurement error in reading the time of one component by another component.

A transaction is a single execution of a specified set of tasks described using a DAG. Mars provides guarantees based on a static construction of the TDMA schedule.

Problem P4 [15, 15, 15]: Max Data Rate on Polled Waiting Loops
Polled waiting loops test the status of device before transferring a data byte. For either input or output, the time required to poll a device determines the minimum time per
transfer and thus the maximum data rate. To speed up this rate, the polled waiting loop functions can be programmed in assembly, as shown below:

```c
#define SDP 0x2F8 /* Serial Data Port */
#define SSP 0x2FD /* Serial Status Port */
#define RX_READY (l <<0) /* bit 0: 1 = input data ready */

Serial_Input (void) {
    While ((inportb(SSP) & RX_READY) == 0) {;
        Return inportb(SDP);
    }
}
```

Since the function _Serial_Input makes no access to memory except to fetch instructions and to pop 4 bytes of return address information off the stack. And it does to IO transfers. Assume:

a. the IO device is ready (so that no looping occurs)
b. all opcodes occupy a single byte of memory
c. all instructions are read off the cache
d. data bus to memory is 4 bytes wide, we retrieve 4 bytes of data per memory read

Answer the following:

a. How many bytes of data is needed from memory (including opcodes, operands, stack info) for each iteration of the _Serial_Input? How many (minimum) memory cycles are needed to fetch this data? Assuming a memory cycle time of 60 ns, what is the total time needed for memory transfers, exclusive of IO transfers?

b. Estimate additional time needed for IO transfers. Assume IO device resides on an adapter card installed on a 33-MHz PCI bus (i.e., 30 ns per IO read)

c. What is the maximum data-transfer rate supported by our function?

Before we begin, we note that Intel x86 ISA provides explicit IO instructions to read from the ports. Many IO port numbers are in the range 0-255 and thus fit into a single byte for addressing. (Details: The IO port address bus is 12 bits wide and thus valid port numbers can go as high as 4095; in these cases the port number must be preloaded into register DX in order to access the port.)

Let us now examine the routine _Serial_Input below for how much memory traffic each instruction creates first for fetching the instruction, any stack modifications and IO operations. For instance, MOV DX, 02FDh has one instruction byte (for MOV), and 2 bytes for the operand. Each invocation of the function determines the minimum time per transfer and thus the maximum data rate. Latency, however, is unpredictable since there is no guarantee that the program will have arrived at the waiting loop before the device becomes ready.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Bytes</th>
<th>Stack Bytes</th>
<th>IO Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV DX, 02FDh</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ST1: IN AL, DX</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST AL, 00000001b</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Memory bandwidth is typically the bottleneck on IO performance. So, in principle we should be able to obtain a rough estimate of the execution time by counting the number of cycles required to execute a function. (Keep in mind that most processors will have the driver code in the cache, thus our estimates are pessimistic or worst case.) The _Serial_Input code’s memory accesses consist of instruction fetches, and to pop 4 bytes of return information off the stack. In addition, it performs two IO operations.

From the table above, we transfer 18 bytes of data (8 instruction, 6 operands, 4 stack). This will require minimum of 5 memory cycles or 0.3 microseconds. (Of course, actual number of memory cycles may be higher due to alignment issues. To be accurate we should count memory cycles for code, data and stack separately.)

Execution time of this function will include memory access time, IO operations. So, execution time is 0.3 + 2x0.03 = 0.36 microseconds. This gives a maximum data transfer rate of 2.78 MB/second.

For comparison, the fastest serial device operates at 115,200 bits per second or just over 10 KB/second.

FOLLOW UP: Consider Interrupt-Driven IO operation. The sequence of events is as follows: interrupt causes CPU to finish current instruction and then initiate an interrupt response cycle; as a part of interrupt response CPU pushes EFlags and the return address (held in CS and EIP registers), disables further interrupts and jumps to the appropriate ISR (interrupt service routine). The maximum latency in case of interrupt driven transfer therefore consists of time to execute longest instruction, hardware response time and time to execute the ISR.

Problem P5 [20 points]: Real-Time Memory Management
Suppose three processes are running in an interrupt-only system where a single interrupt based on three prioritized interrupts is generated. Let T1, T2 and T3 be three tasks as follows:

procedure T1;
begin
  application1;
  application2;
end;
procedure T2;
begin
  application2;
end;
application3;
end;
procedure T3;
begin
  application3;
  application4;
end;

Suppose T1 is running when it is interrupted by T2 during application2. Later T2 is then
interrupted by T3 during application3. Assuming runtime stacks are used, that is, each task
has its own runtime stack. Show the contents and their pointers for the various stacks.

*Hint: Use a stack for main tasks whose contents will list tasks. Runtime stacks of the tasks
will list applications.*

*The three stacks for T1, T2 and T3 have application 2, application 3 and application 3 as
top of stack respectively.*