16.070 Introduction to Computers and Programming

Problem Set 10       Spring 2001

Section A Due: 2 May 2001
Section B Due: 9 May 2001

Section A (Theory out of 100%)

Problem 1 (10%)
Explain the difference between a task and a function.

Problem 2 (20%)
Identify what implementation would be better (polled loop or interrupt-driven) for the following scenarios. Indicate why you selected your choice.

a. A temperature monitoring device in the International Space Station
b. A low oxygen indicator in the Space Shuttle crew compartment
c. A task to retrieve navigation data from a suite of sensors for display to the crew
d. A task to monitor for the presence of a refueling probe on an aircraft

Problem 3 (20%)
What makes a Real Time Operating System different from a typical desktop PC OS?

Problem 4 (20%)
If your operating system’s kernal has the ability to force a task to release a shared resource it is holding, is deadlock between tasks possible?

Problem 5 (30%)
The code below uses a global variable as a semaphore. If you had two tasks, both running this code, what sequence of events would violate the principle of mutual exclusion? What would you need to fix it?

```c
int lock;    /* global variable */

void MyTask( void )
{
    ...
    while ( lock == 1);         /* get semaphore */
    /* loop until lock becomes 0 */
    /* enter critical section*/
    lock = 1;                   /* -take semaphore*/
    do_really_important_stuff();
    lock = 0;                  /* release semaphore */
    ...
} end_task;
```

Turn In: Typed answers to Problems 1 to 5.
Section B (Project out of 100%)

Problem 1   (80%)

For your final 16.070 Project, you will create the following system to simulate a Mars lander.

System Description
The system shall be composed of a controller (implemented on a handyboard), a workstation simulation of the Mars Lander, a graphics package (which shall be supplied to the developer) and a serial interface (cable and software) to connect the controller to the workstation. The Mars Lander simulation shall begin to execute on the workstation. The simulation ends when the Mars Lander softly comes to rest on the surface of Mars, when it crashes into the planet, or when it rises above its initial altitude.

System Requirements
There are several performance requirements associated with the system as a whole. When the spacecraft is allowed to free fall (no thrust), it shall crash into the Mars surface in approximately 16 seconds. With thrust, the user should be able to land the spacecraft softly on Mars.

The system you will create is represented by the following diagram:

![Diagram of the system](image)

Controller Requirements
The controller shall operate as the user’s interface to the Mars Lander Vehicle. Upon receipt of a data packet (1 byte) from the simulation, the controller shall:
1. Display vehicle altitude and vehicle velocity on the handyboard LCD.
2. Light the LEDs if the “Fuel Out” bit is on.
3. Format a data packet and send it to the simulation. This data packet shall contain a “thrust on/off” indication. When the start button is depressed, the controller shall indicate a “thrust on” condition to the simulation. This condition will remain “on” until the stop button has been depressed (even through subsequent transmissions of the data packet to the simulation). When the stop button is depressed, the controller shall indicate a “thrust off” condition to the simulation. This condition shall remain until the start button is pressed.

Simulation Requirements
The simulation represents the Mars Lander vehicle. The simulation will also serve as your system’s main timing loop (that is, it will drive the system) that will look something like the following (shown in pseudocode). Your main loop should include a delay such that your simulation will operate in near actual time.

```
put initial conditions & declarations here
while (not_exit)
{
    /* main loop*/
```
The physics of your simulation have been largely worked out in PS8, but you will need to add a state variable for mass flow (see required constants below).

The simulation will track fuel remaining and send a flag to the controller identifying when the fuel is out. If the vehicle has run out of fuel, the simulation will not process any more thrust commands from the controller.

The simulator will have three types of output. It will:
1. send information to the controller specifying altitude, velocity, and whether there is fuel remaining;
2. call a graphics package (provided) that will display a representation of the lander as it descends. The simulator must determine the condition of the vehicle using one of four conditions defined in the Graphics Package Interface section (see "vehicle-condition" variable);
3. write altitude, velocity, thrust and fuel remaining information to a telemetry file using the same format as the simulation in PS8 (plus a fuel remaining column), and write an error message to the telemetry file if the graphics package returns an error condition.

The simulation will end when the vehicle reaches 0 meters of altitude or goes above its initial altitude.

**Simulation Conditions**

Your starting altitude for all test cases will be 500 meters above the surface of Mars. A soft landing will is considered to be reaching 0 meters of altitude with a speed of less than 3 m/s. A velocity toward the surface of Mars will be considered negative (note: your vehicle can have positive or negative velocity)

Your simulator time step will be 0.2 seconds.

The empty mass of the lander is 500 kg. The lander begins with 160 kg of fuel. The lander uses fuel at a constant rate of 4 kg/s while the thrust is on. When the thrust is off, no fuel is expended. Thrust force is 3kN.

**Simulation Test Cases**

You will run your simulation/controller system with two test cases to be turned in:

A) No thrust turned on throughout the simulation
B) A smooth landing.

**Graphics Package Interface**

The graphics package will be supplied to the developer. Instructions on linking the package will also be provided. The interface to this package will be defined as follows.

The Graphics package:
• accepts float % of screen as input to display vehicle at altitude (100% = top of screen = initial altitude of the lander);
• reads the vehicle_condition variable to know how to draw the vehicle;
• displays optional text string in simulation window.

In implementation, the interface will be via the following library function call:

```c
int SimDisplay(float percent, int vehicle_condition, char text_string[50]);
```

– percent = % display (100.00 = top of screen, 0.00 = surface of planet)
– vehicle_condition: 1 = thrust on, 2 = thrust off, 3 = landed, 4 = crashed
– text_string[50] = “You’re coming in too fast” or whatever you want.

Note: SimDisplay should check for errors (e.g., passed parameter "out-of-range" errors) and return an error code where 0 = success, 1 = fail.

**Interface Requirements**

The following is a diagram of the interface between your simulation and your controller:

The serial interface should be used to continually send data back and forth between the simulation and the controller.

**Suggestion 1**: represent the altitude sent to the controller as a series of eight discrete altitude ranges (3 bits)

**Suggestion 2**: for velocity, use the first bit to convey sign information and the other 3 bits to indicate ranges (choose your own suitable maximum velocity)

**Note**: SimDisplay will be available at [http://web.mit.edu/16.070/www/display/simdisp.h](http://web.mit.edu/16.070/www/display/simdisp.h)

**Turn In**: Printout of source code, screen dump of the 2 required test cases and also copy your workstation program as `username_PS10B_ws.c`, your Handy Board program as `username_PS10B_hb.c` and your telemetry files as `username_PS10B_telemA.dat` (B) to `\CDIO-Prime\16.070HW\ps10b\pX\`. There will also be a demo of your working system during the week of 5/14/01 to be scheduled at a later time. As always, remember to record the amount of time spent working on the problem.
Problem 2  (20%)
Test identification – Planned tests
Complete the following 6 paragraphs outlining your proposed test plan for the program in Problem 1.
The test plan should be designed to facilitate integration of the different modules and code segments of which your program comprises and should test and verify the expected outputs/results of the different modules. Note: this test plan will be very helpful for you to integrate and debug your final project. Also note that you will be writing a test report showing the results of running this test plan for Problem Set 11.

2.1 Serial I/O Test(s)
This paragraph shall describe the tests to be run to confirm operation of the serial I/O portion of the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

2.2 Thrust Controller Test(s)
This paragraph shall describe the tests to be run to confirm operation of the thrust controller portion of the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

2.3 Simulator Test(s)
This paragraph shall describe the tests to be run to confirm operation of the simulation portion of the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

2.4 Fuel Out Indicator Test (s)
This paragraph shall describe the tests to be run to confirm operation of the fuel monitor / LED display portion of the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

2.5 Altitude and Velocity Display Test(s)
This paragraph shall describe the tests to be run to confirm operation of the Altitude and Velocity display portion of the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

2.6 System Test(s)
This paragraph shall describe the tests to be run to confirm operation of the integrated system for the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

Turn In: Typed answers to 2.1 – 2.6.