EE475 Lab #8 Fall 2003

Task Creation and Management with µC/OS-II

In this lab you will use a version of the µc/OS-II operating system ported for the x86 processor of the PC. The source code is compiled and linked as a console application from the DOS command line. Although these exercises will run µc/OS-II in a command window within the Windows XP environment, the code principles and behavior apply equally well for general embedded systems. This lab uses the Example #1 code from Chapter 1 in the textbook as a starting point, so be sure to understand the essential features of the example project.

Preliminaries

1. To build this particular version of the µc/OS-II code you will need to have a vintage version of the Borland C compiler and the proper source code files. CHECK the computer to see that it has the compiler directory `c:\Borland\bc31` and the µC/OS-II source files directory `c:\Micrium`. If either or both of these directories do not exist on your machine, install the files from the course web site: unzip `turbocpp.zip` to `c:\Borland\bc31` (be sure 'use folder names' is checked before you unzip), and unzip `micrium.zip` to `c:\` (also with 'use folder names' checked).
2. Make a temporary local folder for your lab work: `c:\EEClasses\EE475\tempxxx`.
3. Unzip `ex1.zip` from the course web site into your temporary directory. These are the example #1 project files.

Exercise #1: Build the Example µC Project

Open a command window (`cmd.exe`) and `cd` to the local temp folder where you unzipped the `ex1` files. The folder contains the example code (`test.c`), header files (`includes.h`, `os_cfg.h`), and the build instructions (`test.lnk`, `test.mak`, `test.rsp`, and `maketest.bat`).

a) At the command line, run `maketest.bat`, which builds the executable `test.exe`. There may be some warnings during the build process, but if any errors appear you will need to track down and fix the problem(s).

b) Run the `test.exe` program and observe its behavior: the example code creates 10 tasks, each of which selects a random location on the screen and prints its task number (0-9). You can read the details in the textbook (section 1.01, pp. 2-10).

c) The `test.c` program creates a task called `taskStart()`, which then calls a function `TaskStartCreateTasks()` to launch `N_TASKS` instances of the `Task()` function. Find this function in the `test.c` file. Now edit the `test.c` program so that the `Task()` functions will display the letters 'A', 'B', … instead of the original '0', '1', '2',… . Save the file, run `maketest.bat`, resolve any errors, and then run `test.exe` to verify your change.

d) Now edit the `test.c` program once again so that the randomly located numbers are displayed in white (`DISP_FGND_WHITE`) on a blue background (`DISP_BGND_BLUE`) instead of the original black on light gray (see the function `Task()` in `test.c`). Save the file, again run `maketest.bat`, and run `test.exe` to verify the behavior.

→ Demonstrate your "white letters with blue background" version for the instructor.

Exercise #2: Alter the Random Positions

Observe the code to locate the random position algorithm and the way in which each instance of `Task()` knows which character to print.

Now alter the `Task()` function so that the first instance of `Task()` selects a random horizontal position, *but only uses the top line of the display*, the second instance selects a random position *only on the second line of the display*, and so forth.

It is not necessary to demonstrate this exercise for the instructor, but use this version for the following exercises.

Exercise #3: Add a New Task

Create a new task function, `myTask()`, that will run concurrently with the existing tasks. Use `OSTaskCreate()` to start your new task inside the existing `TaskStartCreateTasks()` function, and be sure to allocate stack space and assign a unique task priority number.

The `myTask()` function you create must "erase" one column of the display (write a column of blank characters), then delay for 10 OS clock ticks, then erase the next column, and so forth, modulo 80 columns.

→ Demonstrate your program now including the `myTask()` function. Be able to explain the number of tasks displayed in the status area of the screen.

Exercise #4: Create a Screen Clear Semaphore Interlock

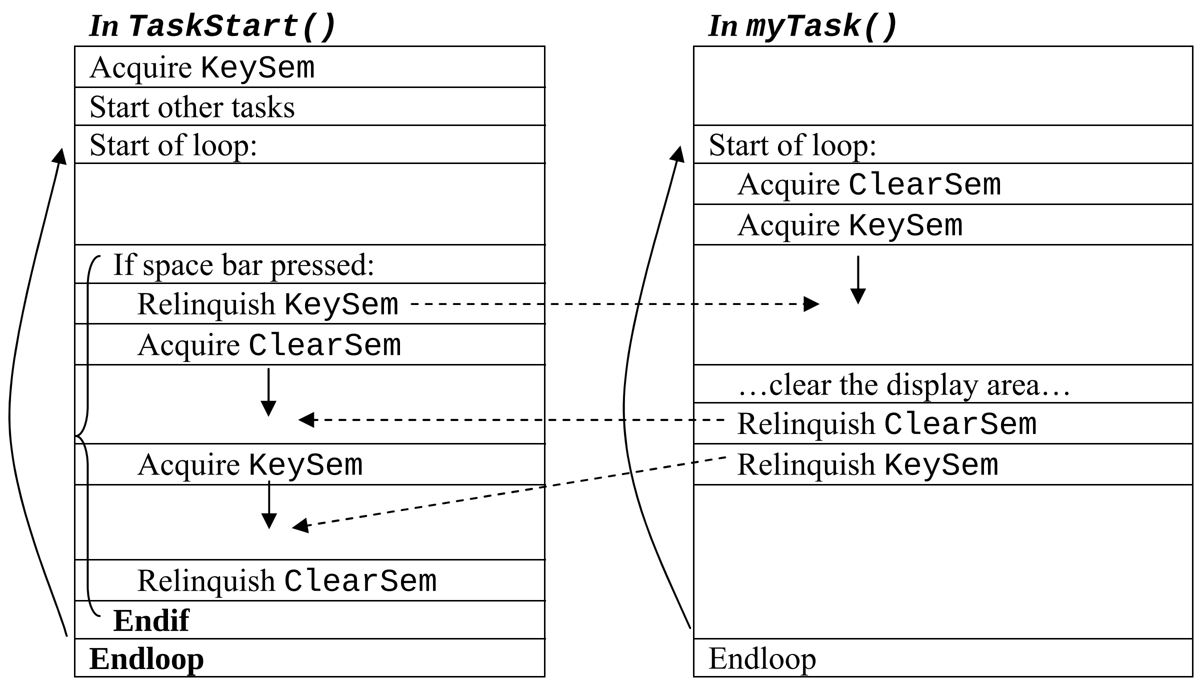
You now need to alter the `test.c` program with a pair of semaphores that control clearing the screen when the space bar is pressed.

a) First, modify your `myTask()` function so that it clears the entire display area (all 80 columns).

b) Next, figure out how to detect a press of the 'space bar' key inside the `TaskStart()` function.

c) Declare two new global semaphore pointers, `OS_EVENT *KeySem` and `*ClearSem`, and create both semaphores in `main()` using `OSSemCreate()`.

d) Devise the proper sequence of `OSSemPend()` and `OSSemPost()` calls so that the following interlock is achieved:



→ Demonstrate your program for the instructor. For your report, comment on the semaphore interlock: is a deadlock possible with this arrangement?

Lab Report

The lab report is to be written up in the Memo format. Be sure to put the *lab number* in the Memo header along with your name and date. For each exercise, explain what was done, how it was accomplished, and answer the given questions to demonstrate your understanding of the exercise. Include **commented** file excerpts related to each exercise, and the signed instructor verification sheet.

→ The lab report is due at class time in two weeks.

Instructor Verification Sheet
Lab #8   Fall 2003

Student Name: ______________________

	Instructor Signature	**Date**
#1 White letters on blue background modification		
#3 New task to clear columns sequentially		
#4 Space bar interlock		