Contents

1. Overview

This lab is a guide to µC/OS-II in SoC design, especially focuses on Context Switch and Inter-Process Communication (IPC). The Context Switch is a process to change execution of tasks. The variables are backuped/restored and control is passed from one task back to the OS, and then passed to the task to be executed. The Inter-Process Communication is a machenism provided by the OS to allow tasks talking to each other. In multitasking systems, the tasks are isolated for many reasons, such as system protection. Thus, the tasks can only communicate through a special channel provided by the OS.

2. Background Information

2.1. Task Scheduling & Context Switch

In µC/OS-II, task scheduling is performed on following conditions:

- A task is created/deleted
- A task changes state
  - On interrupt exit
  - On post signal
  - On pending event
  - On task suspension

If the scheduler chooses a new task to run, context switch occurs. First, the context (processor registers) of current running task is saved in its stack. Next, the context of the new task is loaded into the processor. Finally, the processor continues execution.

2.2. Resource management using Semaphore

Semaphore is used to represent the status of a resource. To use a shared resource such as I/O port, hardware device or global variable, you must request the semaphore from OS and release the semaphore after access (see ***Figure 1***).

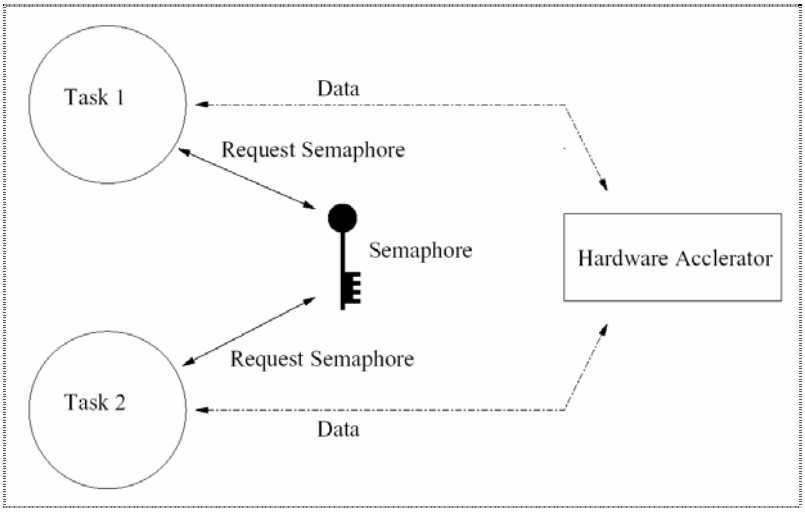

Figure 1 Resource management using Semaphore

Hiding the semaphore within device driver enables easy access to the hardware. IP vender can provide a set of API for the IP. Programmers don't need to deal with detailed IP configuration and complex OS resources management scheme (see ***Figure 2***).

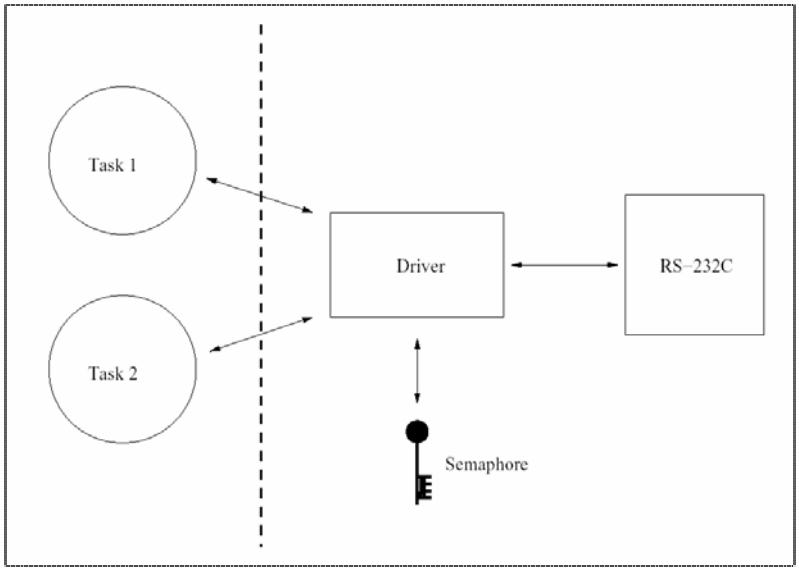

Figure 2 Hiding Semaphore from tasks

2.3. Inter-Process Communication

In µC/OS-II, tasks are not allowed to communicate to each other directly. The communication must be done under control of OS through Mailbox (***Figure 3***). A mailbox is a special channel provided by the OS, the messages are passed in the form of data structure pointer through the mailbox. If multiple messages are to be passed through the same channel, a similar channel called Queue can be used. A Queue is nothing different from the mailbox but just an array of mailbox with FIFO or FILO configuration.

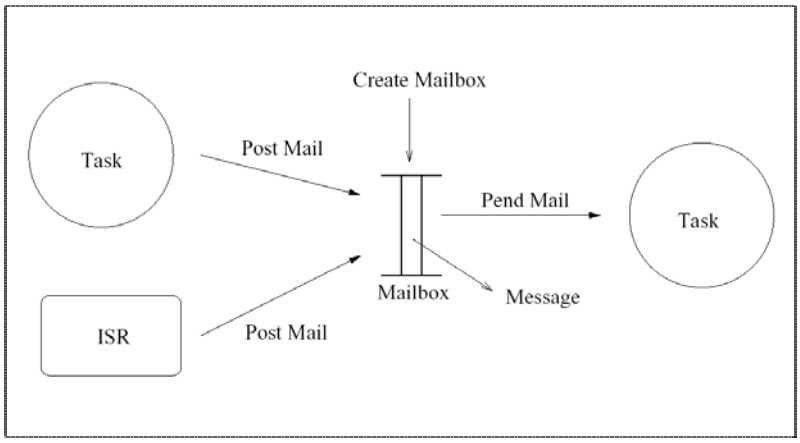

Figure 3 Inter-Process Communication using Mailbox

3. Instructions

3.1. Context Switch

Project eg2 is a context switch example. There are three tasks in this program, and only one can be executed at a time.

1. Open project "**eg2**" in C:/lab8/eg2/ with CodeWorrior.

2. Repeat step 2 ~ 7 at **Lab08 (part 1) Building Program with µC/OS-II**.

------ Note ------
Before add **µC/OS-II** as a sub-project, you should build it successfully first.
---

3. **Build** and **Run**, you can see programs running on µC/OS-II in AXD console window. (***Figure 4***).

4. Try to explain this program. Try to change the execution order into ***Task3*** -> ***Task 2*** -> ***Task1*** -> ***Task3***.

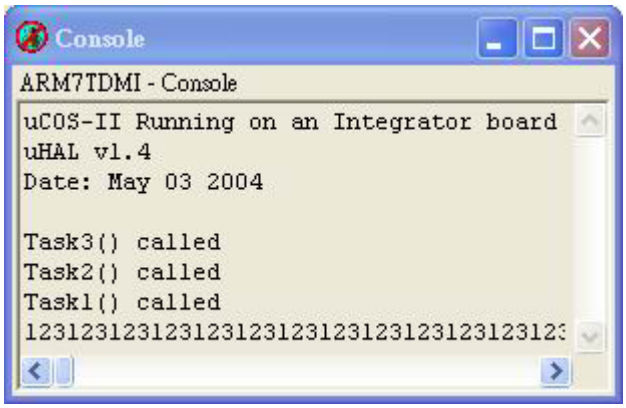

Figure 4 uC/OS-II program running in AXD

3.2. Using Inter-Process Communication

To communicate between processes, you must call OS service: **message mailbox** or **message queue**. Build and run project eg3 to learn about IPC. This program does the very same thing as eg1 in Lab08 (part-1) does. However, eg3 use one task for input and another task for output. Data is passed between tasks using Mailbox.

1. Open project "**eg3**" in C:/lab8/eg3/ with CodeWorrior.

2. Repeat step 2 ~ 7 at **Lab08 (part 1) Building Program with µC/OS-II**.

3. Because multiple values are to be passed, declare a structure to hold the values. (***Figure 5***)

```
typedef struct
{
    char name[64];
    int age;
}envelope;

envelope personal_info;
```

Figure 5 Structure for passing multiple values in a mail

4. We will trace uC/OS-II IPC mechanism, so link **Debug** target of uC/OS-II.

5. Build project and start AXD for Debug

6. Set a break point near line 74 in Task1, where Task1 posts mail to the mailbox. (see ***Figure 6***)

7. Set a break point near line 94 in Task2, where Task 2 posts mail to the mailbox.

8. Click the run button until program counter reaches the break point on line 74.

9. Use "**Step In (F8)**" to trace into the **OSMboxPost()**. The cursor should jump to **OSMboxPost()** function.

10. Next, use "**Step (F10)**" to trace how message is delivered from Task1 to Task2. You can see that the control is passed from Task1 to the OS and then passed from the OS to Task2.

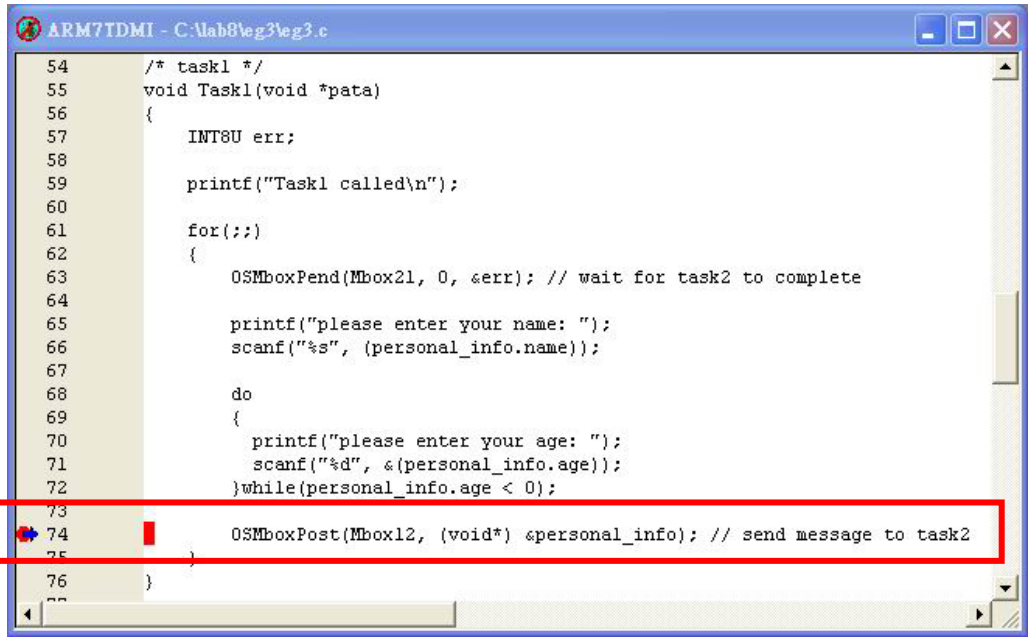

Figure 6 Set a break point at OSMboxPost()

11. Finally, you reach Task2 near line 88. Where Task2 pending on the mailbox. (see ***Figure 7***)
12. Modify Task2 of the program. In Task2, ask the gender of the user, and pass the information back to Task1 for display.

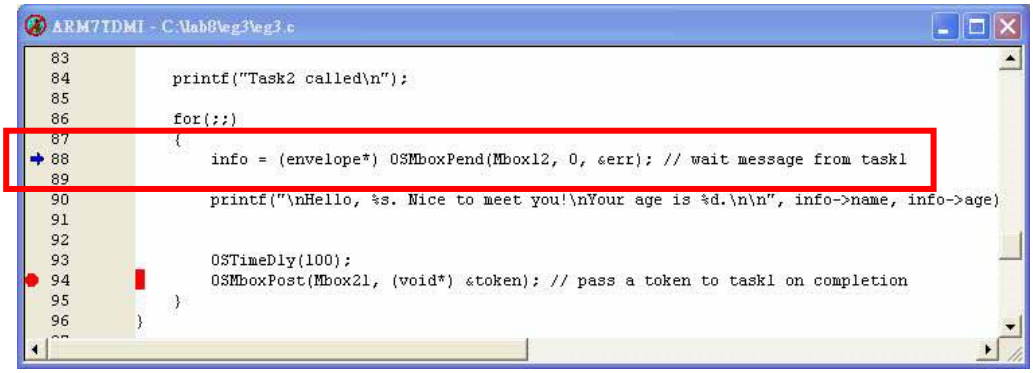

Figure 7 Program counter arrived at OSMboxPend in Task2

4. Exercise

Write an ID checking engine and a front–end interface. The front-end interface accepts **multiple sets of data** and calls the back-end checking engine to verify the correctness of ID. The checking engine can be called multiple times to verify one set of ID each time it is called. Or, it can be called once to verify all IDs. You have to trade-off between the total numbers of context switching and the total stack size required. The checking rule is in the reference section of **Lab08 (part 1**).

Requirements:

- The engine and front-end must be implemented in different tasks.
- The ID information and checking results should be passed between tasks using mailboxes.
- The front-end task can accept up to 4 IDs in each round.
- The program runs continuously round by round.

User input:

- The amount of ID to be checked. (1~4)
- The ID numbers

Program output:

- The ID numbers
- Check results

Example:

(start of round)
How many IDs to check: 2
Enter ID #1: A123456789
Enter ID #2: B987654321
=== check result ===
A123456789 valid
B987654321 invalid
(end of round)

(start of round)
How many IDs to check: 2
Enter ID #1: A123456789
A123456789 is valid
Enter ID #2: B987654321
B987654321 is invalid
(end of round)

5. References

Information about µC/OS-II

- http://www.ucos-ii.com/
- “MicroC/OS-II, the real-time kernel” (ISBN: 0-87930-543-6)