Investigating Deadlocks

Introduction

Objectives

At the end of this lab you should be able to:

- Construct a resource allocation graph for a deadlock condition and verify using the simulator.
- Use two methods of resolving a deadlock condition.
- Use two methods of preventing a deadlock condition.
- Explain and use the “total ordering” method to prevent a deadlock.

Basic Theory

Process deadlocks which are created when processes require access to more than one resource (not-sharable) are frequent occurrences in a modern operating system (OS). The OS can employ various techniques for avoiding, preventing or resolving process deadlocks which will allow the system to run as efficiently as possible.

Lab Exercises - Investigate and Explore

1. Four processes are running. They are called **P1** to **P4**.There are also four resources available (only one instance of each). They are named **R0** to **R3**. At some point of their existence each process allocates a different resource for use and holds it for itself forever. Later each of the processes request another one of the four resources. Draw the resource allocation graph for a four process deadlock condition. Do not continue until you do this and get it verified by the tutor.

2. In the compiler window, enter the following source code in the compiler source editor area (under **PROGRAM SOURCE** frame title).

```
program DeadlockP<b>N</b>
     resource(<b>X</b>, allocate)
     wait(3)
     resource(<b>Y</b>, allocate)
     for n = 1 to 20
     next
end
```

The above code creates a program which attempts to allocate two resources for itself. After the first allocation it waits for 3 seconds and tries to allocate another resource. Finally it counts from 1 to 20 in a loop and then terminates. Other than that it does nothing sensible or useful!

Now follow the instructions below as faithfully as you can:

a. Copy the above code and paste it in three more edit windows so that you have a total of four pieces of source code.
b. In each case change **N** in the program name to 1 to 4, e.g. DeadlockP**1,** DeadlockP**2,** etc.
c. Look at your graph you constructed in (1) above and using that information fill in the values for each of the **X**s and **Y**s in the four pieces of source code.
d. Compile each one of the four source code.
e. Load in memory the four pieces of code generated.
f. Now switch to the OS simulator.
g. Create a single instance of each of the programs. You can do this by double-clicking on each of the program names in the **PROGRAM LIST** frame under the **Program Name** column.
h. In the **SCHEDULER** frame select **Round Robin (RR)** scheduling policy in the **Policies** tab.
i. In OS Control tab, push the speed slider up to the fastest speed.
j. Select the **Views** tab and click on the **VIEW RESOURCES...** button.
k. Select **Stay on top** check box in the displayed window.
l. Back in the **OS Control** tab use the **START** button to start the OS scheduler and observe the changing process states for few seconds.
m. Have you got a deadlock condition same as you constructed in (1) above? If you haven't then check and if necessary re-do above. Do not proceed to (n) or (3) below until you get a deadlock condition.
n. If you have a deadlock condition then click on the **SHOW DEADLOCKED PROCESSES...** button in the **System Resources**

window. Does the highlighted resource allocation graph look like yours?

3. Now that you created a deadlock condition let us try two methods of getting out of this condition:

    a. In the **System Resources** window, there should be four resource shapes that are in red colour indicating they are both allocated to one process and requested by another.
    b. Select one of these resources and click on the **Release** button next to it.
    c. Observe what is happening to the processes in the OS Simulator window.
    d. Is the deadlock situation resolved? Explain briefly why this helped resolve the deadlock.
    e. Re-create the same deadlock condition (steps in 2 above should help).
    f. Once the deadlock condition is obtained again do the following: In the OS Simulator window, select a process in the waiting queue in the **WAITING PROCESSES** frame.
    g. Click on the REMOVE button and observe the processes.
    h. Has this managed to resolve the deadlock? Explain briefly why this helped resolve the deadlock.

    This part of the exercises was about two methods of **recovering** from a deadlock condition **after** it happens.

4. We now look at two methods of **preventing** a deadlock condition **before** it happens.

    a. In the **System Resources** window select the **Disallow hold and wait** check box in the **Prevent** frame.
    b. Try to re-create the same deadlock condition as before. Have you been successful? What happened? Click on the **SHOW DEADLOCKED PROCESSES...** button and observe the displayed information in the text window for potential clues.
    c. Next, uncheck the **Disallow hold and wait** check box and check the **Disallow circular wait** check box.
    d. Try to re-create the same deadlock condition as before. Have you been successful? What happened? Click on the **SHOW DEADLOCKED PROCESSES...** button and observe the displayed information in the text window for potential clues.

5. We are now going to try a third method of preventing deadlocking before it happens. It is called “total ordering” method. Here the resources are allocated in increasing resource id numbers only. So, for example, resource R3 must be allocated after resources R0 to R2 and resource R1 cannot be

allocated after resource R2 is allocated. Looking at your resource allocation graph can you see how this ordering can prevent a deadlock? Comment.

a. In the **System Resources** window select the **Use total ordering** check box in the **Prevent** frame. The other options should be unchecked.
b. Try to re-create the same deadlock condition as before. Have you been successful? What happened? Click on the **SHOW DEADLOCKED PROCESSES...** button and observe the displayed information in the text window for potential clues. What happened? Comment.