Operating Systems Workbench V2.1
Threads (Introductory) Activities and Experiments

Richard Anthony January 2005

This laboratory sheet accompanies the '***Threads – Introductory***' application within the Operating Systems Workbench.

1. Prerequisite knowledge

You should have a basic understanding of the following concepts: Operating System, Scheduling, Process, Thread.

2. Introduction

This simulation has been designed to introduce you to the fundamentals of multi-threading and thread scheduling whilst keeping the complexity low. Up to three threads can be executed simultaneously. Each thread is associated with a colour (red, green and blue). Each thread will try to increase its colour intensity of randomly chosen pixels.

You can investigate the differences between executing a single thread and executing multiple threads. You can investigate the differences between synchronous scheduling and asynchronous scheduling of threads. You can investigate the effect of priority on the scheduling.

Figure 1 shows the Threads (Introductory) interface during an emulation of asynchronous scheduling of three threads each with normal priority.

The display is divided into a number of sections. Each section is briefly explained:

- **Schedule Threads Asynchronously** (under Operating System control) – The emulation enables configuration of up to three threads to be scheduled for concurrent execution. Threads can be created, started and stopped (destroyed) independently.
- **Priority** – The priority can be set independently for each thread. The options provided are similar to those typically available to software developers. HIGHEST priority would usually be only used if the thread had a real-time processing requirement. LOWEST priority would be used for threads that run in the background and do not need to be responsive to users. Most threads would be scheduled with the NORMAL priority (the default).
- **Schedule Threads Synchronously** (under control by the application's Main thread) – The emulation enables the three threads to be scheduled for synchronous execution (The threads each execute for a short time, in sequence. The overall effect is that all threads get regular turns to run – in a round-robin fashion, but this much coarser-grained than the round-robin scheduling associated with asynchronous scheduling). Threads are created (and started) and stopped (destroyed) collectively.
- **Interpretation of priority for synchronous threads** – The operating system's internal mechanism can be used (however, note that with synchronous scheduling only one thread ever executes at a time as far as the operating system is concerned). Alternatively, priority can be imposed externally (within the application). In both cases the actual priority values for each thread are taken from the priority settings discussed earlier.
- **Drawing canvas** – This area of the display contains a large number of pixels that the threads compete to colour. The real-time thread-scheduling behaviour can be evidenced in terms of the relative success of the threads (a thread that is given a greater share of CPU time will manage to colour more of the pixels).
- **Reset drawing canvas button** – This paints the entire drawing canvas black, without changing other settings. It is useful when experiments need to be repeated.
- **Cancel button** – This button exits the application immediately without preserving statistics or configuration settings. Control is returned to the top-level menu.

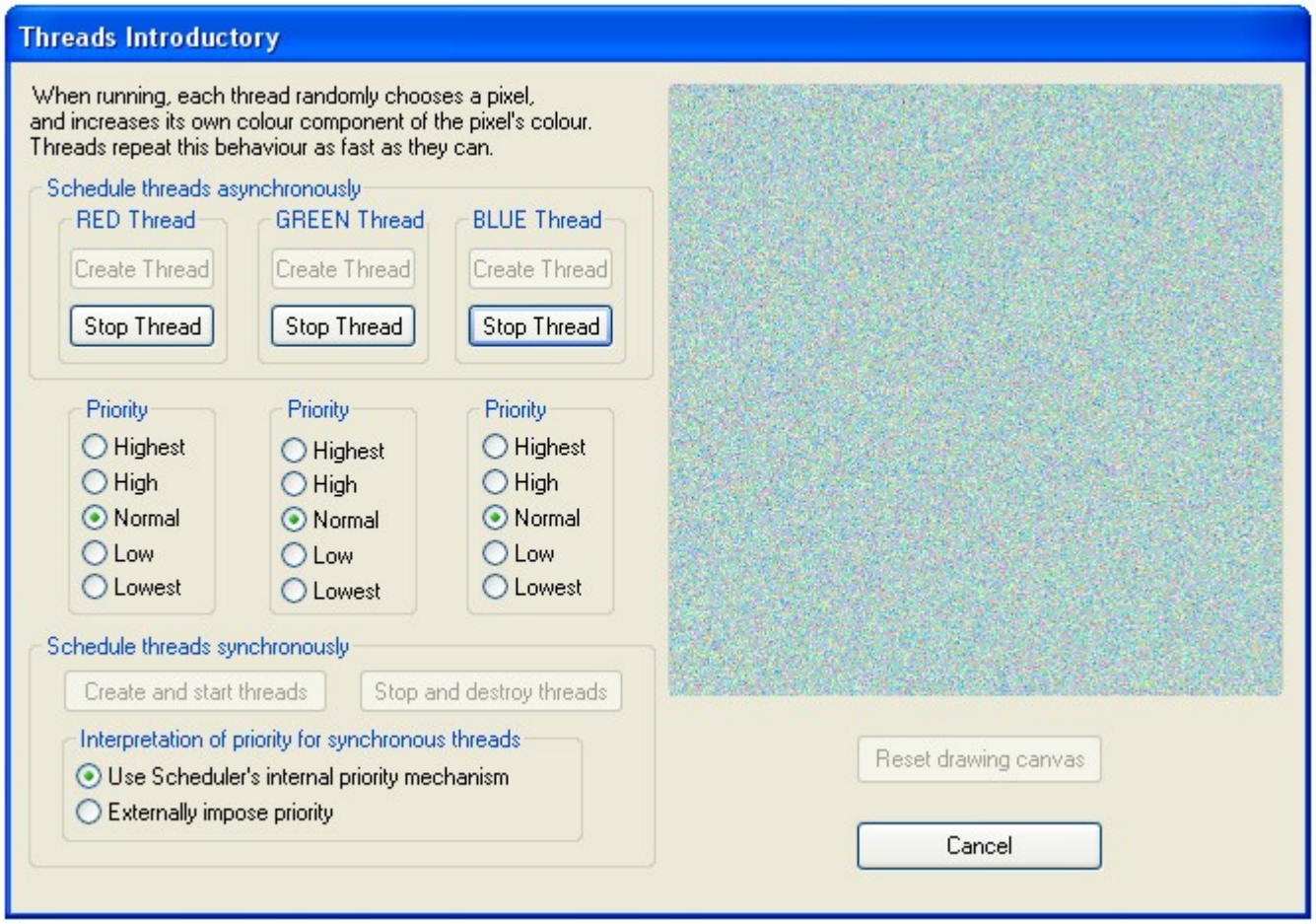

Figure 1. The Threads (Introductory) interface.

The following 'lab activities' sections describe specific experiments and investigations to help you maximise the benefit of the software.

Please take care to read the instructions carefully for each step and try to follow instruction sequences carefully. Try to predict the outcome of experiments in advance if possible. If the outcome is not as expected try to determine why not. You can repeat experiments as often as required and can work at your own pace. Try repeating experiments with slight changes in the parameters – sometimes just changing one parameter slightly can lead to big differences in the results and can shed light on the relative importance of a particular aspect of the emulation.

For each activity the configuration settings are explained. In each case it is assumed that you have already started the 'Introductory Scheduling Algorithms' simulation application. To do this:
1. Start the **Operating Systems Workbench**.
2. From the main menu bar, select **Threads**.
3. From the drop-down menu, select **Introductory**.

Lab Activity: Threads: Introductory: Single Thread
Introduction to the Introductory Threads application

1. In the ‘Schedule Threads Asynchronously’ section press the Create Thread button for the RED thread.

2. Press the Start Thread button for the RED thread.

Q1. What happens?

3. When the drawing canvas has turned completely red, press the Stop Thread button for the RED thread.

Q2. Why were the pixels not coloured evenly (why did some take longer to be coloured red than others)?

Lab Activity: Threads: Introductory: Two Threads
Introduction to the Introductory Threads application

A pixel can be any one of a very large number of colours. This is because the density of each primary colour is represented as a number in the range 0 – 255. Thus, a pixel's colour is actually described using three numbers in the format:

pixel colour = (R, G, B)

Where R is the density of the Red colour, G is the density of the Green colour and B is the density of the Blue colour.
For example (255,0,0) gives a bright red, (0,0,0) is black and (150,150,0) is orange.

The threads each choose a pixel at random and increase the density of their own colour. They also reduce the density of the other colours.

Q1. What do you think will happen if two threads run at the same time?

1. In the 'Schedule Threads Asynchronously' section press the Create Thread button for the RED thread.

2. In the 'Schedule Threads Asynchronously' section press the Create Thread button for the GREEN thread.

3. Press the Start Thread button for the RED thread.

4. After a short time (about three seconds) press the Start Thread button for the GREEN thread.

Q2. What happens?

4. After about twenty seconds, press the Stop Thread button for the RED thread.

Q3. What happens?

Lab Activity: Threads: Introductory: Three Threads
Introduction to the Introductory Threads application

Q1. What do you think will happen if three threads run at the same time? What colour will the drawing canvas appear to be?

1. In the 'Schedule Threads Asynchronously' section press the Create Thread button for the RED thread.

2. In the 'Schedule Threads Asynchronously' section press the Create Thread button for the GREEN thread.

3. In the 'Schedule Threads Asynchronously' section press the Create Thread button for the BLUE thread.

4. In quick succession, Press the Start Thread button for each of the threads.

Q2. What happens?

5. After about ten seconds, press the Stop Thread button for the RED thread.

Q3. What happens?

6. After another ten seconds, press the Stop Thread button for the GREEN thread.

Q4. What happens?

7. You may, up until this point, have been answering the questions in terms of what is happening visually. Now go back over this activity and consider the questions in terms of the CPU time. Think about how the CPU time is divided between the threads that are running – how much of a share do they each get?

8. Think about the scheduling of the threads – which scheduling algorithm best describes what you see happening when all three threads are running (FCFS, SJF, RR or SRJN)? (Make sure you can justify your answer).

Lab Activity: Threads: Introductory: Asynchronous vs Synchronous Thread Scheduling
Understanding the differences between Asynchronous and Synchronous Thread Scheduling

There are two main ways in which multi-threading can be supported:

- **The first approach is to provide support within the Operating System (OS) (so that the OS is aware of the different threads and allocates processing time to them as appropriate). In the Introductory Threads application this is referred to as Asynchronous Thread Scheduling because the interleaving of the threads is fine-grained (as with process scheduling).**

- **The second approach is to manage the thread scheduling within applications themselves (this is really a simulation of threading rather than the real thing, but it can have a similar effect). The OS is not aware that the process contains several threads. In this case the OS simply controls the amount of processing time allocated to the process overall. Within the application, decisions must be made as to when to start and stop each 'thread'. In the Introductory Threads application this is referred to as Synchronous Thread Scheduling because the interleaving of the threads is more coarse-grained (the threads appear to run in turns rather than all at once).**

Experiment with the two different modes of scheduling provided. Use three threads in each case and do not alter the priority settings.

Q1. Can you clearly see the difference between the two modes of scheduling in terms of the thread behaviours? To confirm this, think how you would clearly explain what is happening to a friend?

Q2. In which mode is it easiest to see that there are separate threads running?

Q3. Which mode would be suitable for an application in which each thread dealt with a different query, such as a web-server? Why?

Lab Activity: Threads: Introductory: Thread Priority
Understanding Thread Priority in both Asynchronous and Synchronous Thread Scheduling

The mechanism of priority is handled differently in each of the two thread scheduling modes:

- **In Asynchronous Thread Scheduling the Operating System (OS) is in control. The priority value is assigned to threads by the OS. This means that the OS will take the priority value into account automatically when deciding how much processing time to give to each thread.**
- **In Synchronous Thread Scheduling the OS only knows about the process itself (only one thread per process is really ever running at a time). Because of this the OS assigned priority values operate at the process level and have no effect on the relative priority of the threads within the process. To get around this, it is possible to externally alter the priority of specific threads within the application itself.**

Asynchronous Thread Scheduling.
1. Create and Start three threads. Experiment with various priority configurations.

Q1. What is the visual effect of increasing a thread's priority?

Q2. What actually causes this visual effect (in terms of the thread scheduling)?

Q3. What is the effect of uniformly increasing the priority of ALL threads to HIGHEST? What is the effect uniformly decreasing the priority of ALL threads to LOWEST? What is the effect of setting one thread's priority to HIGH and setting the other two threads' priority to LOW? (try also other configurations).

Q4. Based on your evaluation of the answers to Q3, do you think the priority settings have an:
relative, **absolute** or **relative and absolute**
effect on the threads within a single process?

Synchronous Thread Scheduling.
2. Create and Start the threads. Experiment with various priority configurations, USING THE SCHEDULERS INTERNAL PRIORITY MECHANISM.

Q5. What is the visual effect of increasing a thread's priority? Why is this?

3. Create and Start the threads. Experiment with various priority configurations, USING EXTERNALLY IMPOSED PRIORITY.

Q6. What is the visual effect of increasing a thread's priority? Why is this?

Q7. What actually causes this visual effect (in terms of the thread scheduling)?

Q8. What is the effect of uniformly increasing the priority of ALL threads to HIGHEST? What is the effect uniformly decreasing the priority of ALL threads to LOWEST? What is the effect of setting one thread's priority to HIGH and setting the other two threads' priority to LOW? (try also other configurations).

Q9. Based on your evaluation of the answers to Q8, do you think the priority settings have an:
relative, **absolute** or **relative and absolute**
effect on the threads within a single process?